## Eindhoven University of Technology

## BACHELOR

## On the efficient calculation of PageRank

Lukas, K.A.Y.

Award date:
2014

Link to publication

## Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

## General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain


## BACHELORS THESIS

# On the efficient calculation of PageRank 

By
K.A.Y. Lukas

Supervisor:
dr. M.E. Hochstenbach

Eindhoven, September 2014

## Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This thesis is the result of my own work and includes nothing that is the outcome of work done in collaboration, except where specifically indicated in the text.
K.A.Y. Lukas

September 2014

## Acknowledgements

I would like to acknowledge dr. M.E. Hochstenbach for supervising this project and being a source for new ideas. Special thanks go to D.J.A. Creemers, A. Pieterse, R. Scharrenberg and W.M. Sonke for proofreading this thesis.


#### Abstract

The importance of web pages, while the definition of importance can vary, can be assessed by different techniques. One of these techniques is PageRank. In this thesis the basics behind PageRank are explained, some important properties are given and the algorithm to calculate the PageRank vector is given.

Calculating the PageRank of a set of pages can be done in multiple ways: power iteration combined with some extrapolation techniques, Arnoldi iteration, minimal polynomial extrapolation (MPE) and reduced rank extrapolation (RRE). The performance of these methods is compared in this paper.

Furthermore, a web graph is built of the TU/e website, used in the comparison of the various algorithms and a basic search engine is built using the results of the computation.


## Contents

Declaration ..... iii
Acknowledgements ..... v
Abstract ..... vi
Contents ..... ix
1 Introduction ..... 1
1.1 Random Surfer Model ..... 2
1.1.1 Personalized PageRank ..... 3
1.2 Applications ..... 4
2 Theory ..... 5
2.1 Power iteration ..... 11
2.1.1 Optimization for PageRank ..... 11
2.1.2 Complexity ..... 12
2.2 The second eigenvector and eigenvalue ..... 13
2.2.1 Influence of $p$ on the eigengap ..... 13
2.2.2 The multiplicity of the second eigenvalue ..... 15
2.3 Krylov ..... 16
2.3.1 Lanczos and Arnoldi ..... 16
2.4 Extrapolation ..... 20
2.4.1 Aitken extrapolation ..... 20
2.4.2 Quadratic extrapolation ..... 21
2.4.3 MPE and RRE ..... 23
3 Bright: an indexation of the tue.nl website ..... 29
3.1 Web crawling process ..... 29
3.2 Indexing keywords ..... 30
3.3 Result ..... 31
3.4 Distance between departments ..... 32
4 Performance analysis ..... 35
4.1 Extrapolation methods ..... 35
4.2 Krylov methods ..... 38
4.3 Performance Comparison ..... 40
5 Conclusion ..... 43
5.1 Future work ..... 44
5.2 Discussion ..... 44
Appendix A Matlab Code ..... 47
A. 1 Power iteration ..... 47
A. 2 Linear extrapolation ..... 48
A. 3 Quadratic extrapolation ..... 49
A. 4 Arnoldi Iteration ..... 50
A. 5 Minimal polynomial extrapolation ..... 52
A. 6 Reduced rank extrapolation ..... 53
Appendix B Theorems ..... 55
References ..... 59
Index ..... 61

## Chapter 1

## Introduction

Web search engines are ubiquitous in the everyday life of students, researchers and nearly any person that uses the Internet[1]. Although the feature set of search engines has increased over the years, its basic objective remains the same: to provide users with a usable list of search results most related to the given keywords. Although many search results can be related to a given keyword, not all of them are useful for the user. For example, a web page for a high school project about the current economic situation in the U.S. will most likely be far less interesting than the web page of the Bureau of Economic Analysis. To distinguish the quality of each web page, the two PhD students Larry Page and Sergey Brin at the Stanford University developed an algorithm named PageRank ${ }^{1}$. Therefore, PageRank can be seen as a quality ranking algorithm.

Early search engines used a technique called backlink counting to determine the quality of a web page. This means that the importance of a web page is determined by the number of links referring to the web page. This simple technique works fine when each web page has a similar quality. For example, this technique works pretty well for academic papers. However, web pages vary on a much wider scale in quality than academic papers.

An illustration of the issue can be seen in figure 1.1. Each node in this graph can be seen as a web page and each outgoing edge of a node can be seen as a link. Using a simple backlink counting algorithm the web page TheWorldIsFlat has the same quality as NASA, however TheWorldIsFlat is cited by a low quality web page (George's blog), while NASA is cited by a high quality website: The Times.

Another important factor in a quality ranking algorithm is the susceptibility to manipulation. A simple backlink counting algorithm is very susceptible to manipulation: creating a lot of simple web pages (no real quality is needed for these) linking to a web page, will cause the algorithm to assign a high quality rank to the last mentioned web page. PageRank addresses this problem and as we see later, also methods could be developed to detect manipulation.

[^0]

Figure 1.1 A graph modelling a part of the Internet. Each node is a web page and each edge is a link.

### 1.1 Random Surfer Model

There are different interpretations of the PageRank algorithm, in this thesis we use a stochastic approach by considering a Random Surfer Model.

The Random Surfer Model models in its simplest form considers a surfer that simply keeps clicking on successive links at random. The PageRank value of web page $p$ is then the proportion of the time the random surfer is at web page $p$. We assume in this model that the amount of time between each click is constant.

This simple model can be seen as a finite state Markov chain on a graph similar to figure 1.1, from which we will derive in chapter 2 the PageRank algorithm. A Markov chain is a mathematical process that undergoes transitions from one state to another state. The transition is randomly chosen in each state according to a discrete distribution. This discrete distribution is memoryless, the probability distribution does only depend on the current state. For more information about Markov chains the book Introduction to Probability Models[3], especially chapter 4 can be of service. Basic knowledge about Markov chains and probability theory is assumed in this thesis.

Each state of this Markov chain is a web page and each link of the web page introduces a $1 / n$ transition probability to the web page that is linked to ( $n$ is the total number of links in the current web page, such that the total transition probability is 1). We call this the Simplified Random Surfer Model.

In Markov chains each state can be either recurrent or transient. A state is recurrent if the probability that starting in state $i$ the process will ever reach state $i$ again is equal to 1 ; in any other
case the state is transient. In the current model transient states can exist, for example in figure 1.1 every state except TheWorldIsFlat and nasa is transient. Formally, recurrent states can be defined as follows:

State $i$ is recurrent if and only if, starting in state $i$, the expected number of time periods that the process is in state $i$ is infinite.[3, p. 196]

This means that the proportion of the time that the random surfer is at web page $p$ is non-zero if and only if, $p$ is recurrent. Moreover, the random surfer can get stuck in a trap when the rest of the graph is not reachable from a subgraph of the complete graph (for instance when two web pages point to only each other). This is called a rank sink.

To solve this problem a rank source is introduced: the random surfer has a small chance of jumping to a page at random, instead of clicking a link. This models the behavior of a random surfer that "gets bored". For the Markov chain representation this means that each state has a non-zero transition probability for every state. The probability that the random surfer "gets bored" is assumed to be constant in the Random Surfer Model, but the distribution of the random jump can be varied, this distribution is modelled by a vector $\mathbf{e}$ of probabilities. The addition of this vector $\mathbf{e}$ gives us the Random Surfer Model.

Another problem are dangling links: web pages with no outgoing links. Multiple ways exists to handle these dangling links. One option is to ignore the dangling links in the initial calculation. After the PageRank values of all non dangling links are determined, we calculate the PageRank values of the dangling links using the PageRank values of the linking web pages. Another option, which we actually used in our calculation, is to let all dangling links assume the transition probabilities of vector $\mathbf{e}$ (i.e. the surfer always gets bored when it reaches a dangling link).

### 1.1.1 Personalized PageRank

The vector e can be used to fine-tune the algorithm aside from solving the problem of rank sinks. In a basic scenario e has an equal value for each component. This, however, makes the algorithm more susceptible to manipulation: because each web page receives a little PageRank from the vector e, creating a lot of web pages that link to a web page $p$, gives web page $p$ a higher PageRank. Effectively, this introduces the same problems seen in the backlink counting technique, albeit in a lesser degree. As an alternative we can let e point to the home pages of a selection of important domains. Another alternative would be to experimentally record web traffic and use the number of visits to each web page to fill $\mathbf{e}$.

The vector e can also be chosen as the distribution of the web pages visited by the user. This basically boosts all web pages related to the web pages that the user finds interesting. Obviously, this is very expensive to calculate for every user, so to implement this idea it is needed to group the users into interest groups.

### 1.2 Applications

Many applications for PageRank have been found outside the normal search engine applications. In protein-to-protein interactions it's useful to find a well-connected protein with the personalized PageRank algorithm, making the algorithm useful in bioinformatics[4]. web pages are, in the view of PageRank, very similar to academic publications as they both can reference another source: CiteRank implements this algorithm[5]. The similarity could also be seen in early search engines, the backlink counting techniques were borrowed from academic article ranking algorithms[6].

Another interesting approach is InvestorRank: it finds the leading investors by considering who invested first in a company $[7]$. If investment company $A$ invested before investment company $B$ then this is seen in the algorithm as $B$ references to $A$. Therefore, the investment company that invests first in a good opportunity has a high InvestorRank.

Finally the personalized PageRank can be used for various recommendation systems: finding friends in social networks $[8]$, purchase recommendations $[9]$, etc.

## Chapter 2

## Theory

As we have seen in chapter 1, the PageRank of a web page $p$ is defined as the proportion of the time that the random surfer is at web page $p$. To formalize the Markov chain that represents the Random Surfer Model we first have to define stochastic matrices and vectors.

Definition 1 (Stochastic vector). A stochastic vector is a vector of nonnegative real numbers whose elements sum to 1.

Definition 2 (Stochastic matrix). A (left) stochastic matrix is a square matrix with stochastic vectors as columns.

Basically, a stochastic vector models a discrete distribution. It can be seen that multiplication of an $n \times n$ stochastic matrix $A$ and an $n$-dimensional stochastic vector $\mathbf{v}$ yields an $n$-dimensional stochastic vector $\mathbf{w}$ :

$$
\begin{aligned}
\mathbf{w}_{\mathbf{i}} & =\sum_{j=1}^{m} A_{i j} \mathbf{v}_{\mathbf{j}} \geq 0 . \\
\sum_{i=1}^{n} \mathbf{w}_{\mathbf{i}} & =\sum_{i=1}^{n} \sum_{j=1}^{m} A_{i j} \mathbf{v}_{\mathbf{j}} \\
& =\sum_{j=1}^{m} \mathbf{v}_{\mathbf{j}} \sum_{i=1}^{n} A_{i j} \\
& =\sum_{j=1}^{m} \mathbf{v}_{\mathbf{j}} \cdot 1 \\
& =1
\end{aligned}
$$

Now let us introduce stochastic matrix $A$ and stochastic vector $\mathbf{e}$.
Let $n$ be the number of web pages (excluding dangling links) in our data set. Then we define $A$ as the $n \times n$ matrix giving the transition probabilities of the Simplified Random Surfer Model:

$$
A=\left(\begin{array}{lllll}
\mathrm{a}_{\mathbf{0}} & \ldots & \mathrm{a}_{\mathbf{i}} & \ldots & \mathrm{a}_{\mathbf{n}-\mathbf{1}}
\end{array}\right)
$$

Let arbitrary web pages $i$ and $j$ in our dataset be given. Denote by $N_{i}$ the number of outgoing links of web page $i$ to websites inside our dataset.

Then:

$$
\left(\mathbf{a}_{\mathbf{i}}\right)_{j}= \begin{cases}\frac{1}{N_{i}} & i \text { has an outgoing link to } j \\ 0 & \text { otherwise }\end{cases}
$$

Obviously, if web page $i$ is not a dangling link (i.e. $N_{i} \neq 0$ ) then $\mathbf{a}_{\mathbf{i}}$ is a stochastic vector. So $A$ is a stochastic matrix, assuming no dangling links exist. Let $\mathbf{e}$ be an $n$-dimensional vector, such that $\|\mathbf{e}\|_{1}=1$. This vector defines the random jump distribution as described in section 1.1. Also let $p$ be the probability of a random jump using vector $\mathbf{e}$. The vector of transition probabilities in web page $i$ in the Random Surfer Model is defined as:

$$
(1-p) \mathbf{a}_{\mathbf{i}}+p \mathbf{e}
$$

Which gives the transition probability matrix:

$$
P=(1-p) A+p\left(\mathbf{e} \cdot \mathbf{1}^{T}\right)
$$

Where $\mathbf{1}$ is a column vector consisting of all ones. Now suppose $\mathbf{v}$ is a vector of probabilities, describing the probability the Markov chain is in each state. Then $P \mathbf{v}$ is a vector of probabilities giving the probability that the Markov state is in each state after one transition. For a more detailed explanation of Markov chains we again refer to Introduction to Probability Models[3].

The PageRank value of a state is defined by the proportion of the time the random surfer is in each state. This PageRank value is equal to the limiting probabilities of $P$, where the limiting probabilities are defined by ${ }^{1}$ :

Definition 3. (cf. [3, p. 205]) The limiting probabilities of a Markov chain with transition matrix $A$ is a stochastic vector $\boldsymbol{\pi}$ such that $A \boldsymbol{\pi}=\boldsymbol{\pi}$.

It is easy to see why the limiting probabilities are equal to the proportion of the time the random surfer is in each state: if we start in a random state chosen using the distribution of $\boldsymbol{\pi}$, then after every transition the probability we are in each state is still $\boldsymbol{\pi}$.

Note that the limiting probabilities of a Markov chain do not necessarily exist and multiple limiting probabilities can exist.

Definition 4. Let $A$ be a matrix representing a set of web pages and their outgoing links. Let $\mathbf{e}$ be some vector over the web pages that corresponds to a source of rank and $\|\mathbf{e}\|_{1}=1$. Let $p$ be the probability of a random jump. Then, the PageRank of a set of web pages is the limiting probabilities $\boldsymbol{\pi}$ of the Markov chain of the set of web pages with transition probability matrix:

$$
(1-p) A+p\left(\mathbf{e} \cdot \mathbf{1}^{T}\right)
$$

Note that $\boldsymbol{\pi}$ is a column vector. Our definition differs somewhat from the definition given by Page and Brin:

[^1]Definition 5. ([6, p. 4]) Let $B_{u}$ be the set of pages that point to $u$. Let $\widetilde{\mathbf{e}}(u)$ be some vector over the web pages that corresponds to a source of rank. Then, the PageRank of a set of web pages is an assignment, $\widetilde{R}$, to the web pages which satisfies:

$$
\widetilde{R}(u)=c \sum_{v \in B_{u}} \frac{\widetilde{R}(v)}{N_{v}}+c \widetilde{\mathbf{e}}(u) .
$$

such that $c$ is maximized and $\|\widetilde{R}\|_{1}=1$.
Note: in this definition $\widetilde{\mathbf{e}}(u)$ does not have to satisfy the property $\|\widetilde{\mathbf{e}}\|_{1}=1$.
We will show that these definitions are indeed equal. Let $\boldsymbol{\pi}$ be the limiting probabilities of our Random Surfer Model with transition probability matrix $P$. Note that $\|\boldsymbol{\pi}\|_{1}=1$. Then [3, p. 212]:

$$
P \pi=\pi .
$$

Which gives:

$$
\begin{aligned}
\boldsymbol{\pi}_{\boldsymbol{u}} & =\left((1-p) A+p\left(\mathbf{e} \cdot \mathbf{1}^{T}\right)\right) \boldsymbol{\pi} \\
& =(1-p) A \boldsymbol{\pi}+p\left(\mathbf{e} \cdot \mathbf{1}^{T}\right) \boldsymbol{\pi} \\
& =(1-p) \sum_{v \in B_{u}} \frac{\boldsymbol{\pi}_{\boldsymbol{v}}}{N_{v}}+p\left(\mathbf{e} \cdot \mathbf{1}^{T}\right) \boldsymbol{\pi}
\end{aligned}
$$

$$
=(1-p) \sum_{v \in B_{u}} \frac{\boldsymbol{\pi}_{v}}{N_{v}}+p \mathbf{e} \quad \text { using: }\|\boldsymbol{\pi}\|_{1}=1
$$

Now let $c=1-p$ and set $\widetilde{\mathbf{e}}=\frac{p}{1-p} \mathbf{e}$ :

$$
\boldsymbol{\pi}_{\boldsymbol{u}}=c \sum_{v \in B_{u}} \frac{\boldsymbol{\pi}_{\boldsymbol{v}}}{N_{v}}+c \widetilde{\mathbf{e}}
$$

To understand how the limiting probabilities can be calculated we first have to define some properties of Markov chains.

Definition 6. (cf. [3, p. 204]) Let a Markov chain with stochastic matrix $A$ be given. Then state $i$ is aperiodic if and only if there exists an $m$ such that:

$$
\forall m^{\prime} \geq m: A_{j, j}^{m^{\prime}}>0
$$

Furthermore, a Markov chain is aperiodic if and only if all states are aperiodic.
Definition 7. (cf. [3, p. 194]) Let a Markov chain with stochastic matrix A be given. Then the Markov chain is irreducible if and only if it is possible to reach any state from any state, or mathematically:

$$
\forall i, j:\left(\exists m: A_{i, j}^{m} \geq 0\right)
$$

Now to be able to calculate the PageRank vector we will prove some properties.

Lemma 1. Let a stochastic $n \times n$ matrix $A$ of a Markov chain be given. Let a vector $\mathbf{v}$ be given. Then:

$$
\|A \mathbf{v}\|_{1} \leq\|\mathbf{v}\|_{1} .
$$

Moreover, when $\mathbf{v}$ contains at least one positive and one negative value and the Markov chain is irreducible and aperiodic, then there is a $k$ such that:

$$
\left\|A^{k} \mathbf{v}\right\|_{1}<\|\mathbf{v}\|_{1}
$$

Proof. For the first equation we can write:

$$
\begin{aligned}
\|A \mathbf{v}\|_{1} & =\sum_{i}\left|\sum_{j} A_{i, j} \mathbf{v}_{\mathbf{j}}\right| \\
& \leq \sum_{i} \sum_{j} A_{i, j}\left|\mathbf{v}_{\mathbf{j}}\right| \quad A \text { is nonnegative } \\
& =\sum_{j}\left|\mathbf{v}_{\mathbf{j}}\right| \sum_{i} A_{i, j} \\
& =\sum_{j}\left|\mathbf{v}_{\mathbf{j}}\right| \\
& =\|\mathbf{v}\|_{1}
\end{aligned}
$$

So the first identity holds. Now suppose $\mathbf{v}$ has at least one positive and one negative value.
Consider $A_{i, j}^{l}$ for any $l>0, i$ and $j$. Then by definition of a stochastic matrix of a Markov chain we know:
$A_{i, j}^{l}$ is the probability that when starting in $i$ we are in $j$ after exactly $l$ steps.
As our Markov chain is irreducible, $j$ must be reachable from $i$, so there must exist an $l$ depending on $i$ and $j$, such that $A_{i, j}^{l} \neq 0$. Now by the definition of aperiodic, an $m$ must exist (depending on $j$ ) such that:

$$
\forall m^{\prime} \geq m: A_{j, j}^{m^{\prime}}>0
$$

Combine these two results to get for all $m^{\prime} \geq m$ :

$$
\begin{array}{rlr}
A_{i, j}^{l+m^{\prime}} & =\left(A^{m^{\prime}} A^{l}\right)_{i, j} & \\
& \geq A_{j, j}^{m^{\prime}} A_{i, j}^{l} & A^{m^{\prime}} \text { is nonnegative } \\
& >0 . &
\end{array}
$$

Now let $k$ be the maximum of $l+m$ over all $i$ and $j$. Then we have $A_{i, j}^{k}>0$ for all $i$ and $j$. And
so:

$$
\begin{aligned}
\left\|A^{k} \mathbf{v}\right\|_{1} & =\sum_{i}\left|\sum_{j} A_{i, j}^{k} \mathbf{v}_{\mathbf{j}}\right| \\
& <\sum_{i} \sum_{j} A_{i, j}^{k}\left|\mathbf{v}_{\mathbf{j}}\right| \quad A_{i, j}^{k}>0 \text { and } \mathbf{v} \text { has a negative and positive element. } \\
& =\sum_{j}\left|\mathbf{v}_{\mathbf{j}}\right| \sum_{i} A_{i, j}^{k} \\
& =\sum_{j}\left|\mathbf{v}_{\mathbf{j}}\right| \\
& =\|\mathbf{v}\|_{1} .
\end{aligned}
$$

It is important to note that every vector with at least one negative and one positive element is independent of the set of stochastic vectors. This leads to the following corollary:

Corollary 1. Every eigenvector of an irreducible and aperiodic stochastic matrix that is not a multiple of a stochastic eigenvector has an eigenvalue in magnitude smaller than 1.

Lemma 2. Let a stochastic matrix $A$ of an irreducible and aperiodic Markov chain be given. Then there exists exactly one stochastic eigenvector with eigenvalue 1.

Proof. By definition of stochastic matrix and stochastic vector, we know that if a stochastic eigenvector exists, it must have eigenvalue 1. So we only need to show that a stochastic eigenvector exists. We will prove this by contradiction. Suppose no stochastic eigenvector exist. Take an arbitrary stochastic vector $\mathbf{v}$. Then define the sequence of stochastic vectors $\mathbf{v}^{(\mathbf{n})}$ and the difference between each element $\mathbf{d}^{(\mathbf{n})}$ as follows:

$$
\begin{aligned}
& \mathbf{v}^{(\mathbf{n})}=A^{n} \mathbf{v} \\
& \mathbf{d}^{(\mathbf{n})}=\mathbf{v}^{(\mathbf{n})}-\mathbf{v}^{(\mathbf{n}-\mathbf{1})} .
\end{aligned}
$$

Because we have no stochastic eigenvectors $\mathbf{d}^{(\mathbf{n})}$ is never $\mathbf{0}$. By Lemma 1 we have a $k$ such that:

$$
\begin{aligned}
\left\|\mathbf{d}^{(\mathbf{n}+\mathbf{k})}\right\|_{1} & =\left\|\mathbf{v}^{(\mathbf{k}+\mathbf{n})}-\mathbf{v}^{(\mathbf{k}+\mathbf{n}-\mathbf{1})}\right\|_{1} \\
& =\left\|A^{k} \mathbf{v}^{(\mathbf{n})}-A^{k} \mathbf{v}^{(\mathbf{n}-\mathbf{1})}\right\|_{1} \\
& =\left\|A^{k}\left(\mathbf{v}^{(\mathbf{n})}-\mathbf{v}^{(\mathbf{n}-\mathbf{1})}\right)\right\|_{1} \\
& <\left\|\mathbf{v}^{(\mathbf{n})}-\mathbf{v}^{(\mathbf{n}-\mathbf{1})}\right\|_{1}=\left\|\mathbf{d}^{(\mathbf{n})}\right\|_{1} .
\end{aligned}
$$

So $\mathbf{d}^{(\mathbf{n})} \rightarrow 0$ as $n \rightarrow \infty$. Therefore, $\mathbf{v}^{(\mathbf{n})}$ converges to, say, $\mathbf{v}^{*}$. Because $\mathbf{v}^{(\mathbf{n})}$ is a sequence of stochastic vectors $\mathbf{v}^{*}$ is also a stochastic vector.

Now by Lemma 1 we have:

$$
\begin{aligned}
\left\|A \mathbf{v}^{*}-\mathbf{v}^{(\mathbf{n}+\mathbf{1})}\right\|_{1} & =\left\|A \mathbf{v}^{*}-A \mathbf{v}^{(\mathbf{n})}\right\|_{1} \\
& =\left\|A\left(\mathbf{v}^{*}-\mathbf{v}^{(\mathbf{n})}\right)\right\|_{1} \\
& \leq\left\|\mathbf{v}^{*}-\mathbf{v}^{(\mathbf{n})}\right\|_{1} .
\end{aligned}
$$

Now let $n \rightarrow \infty$ :

$$
\left\|A \mathbf{v}^{*}-\mathbf{v}^{*}\right\|_{1}<\left\|\mathbf{v}^{*}-\mathbf{v}^{*}\right\|_{1}=0
$$

Thus $A \mathbf{v}^{*}=\mathbf{v}^{*}$. This contradicts the assumption that no eigenvector exists.
Suppose two distinct stochastic eigenvectors $\mathbf{v}$ and $\mathbf{w}$ exist with eigenvalue 1 . Set $\mathbf{y}$ to $\mathbf{v}-\mathbf{w}$. Then:

$$
\begin{aligned}
\sum_{i} \mathbf{y}_{\mathbf{i}} & =\sum_{i}\left(\mathbf{v}_{\mathbf{i}}-\mathbf{w}_{\mathbf{i}}\right) \\
& =\sum_{i} \mathbf{v}_{\mathbf{i}}-\sum_{i} \mathbf{w}_{\mathbf{i}} \\
& =1-1=0 .
\end{aligned}
$$

Because $\mathbf{y} \neq \mathbf{0}, \mathbf{y}$ must have at least one negative value and at least one positive value. As $\mathbf{y}$ is a non-zero linear combination of two eigenvectors with the same eigenvalue, $\mathbf{y}$ must be an eigenvector with eigenvalue 1 . But by Lemma 1 we have a $k$ such that:

$$
\|\mathbf{y}\|_{1}=\left\|A^{k} \mathbf{y}\right\|_{1}<\|\mathbf{y}\|_{1} .
$$

which is a contradiction.

Note that Lemma 1 and Lemma 2 are also consequences of the Perron-Frobenius theorem:
Theorem 1. (See [10]) (Shortened version of the Perron-Frobenius theorem) Let $A$ be an $n \times n$ positive matrix: $A_{i j}>0$ for $1 \leq i, j \leq n$. Then the following statements hold.

- There is a positive real number $r$, called the Perron root, such that $r$ is an eigenvalue of $A$ and any other eigenvalue $\lambda$ (possibly, complex) is strictly smaller than $r$ in magnitude.
- There exists an eigenvector $\mathbf{v}$ of $A$ with eigenvalue $r$ such that all components of $\mathbf{v}$ are positive: $A \mathbf{v}=r \mathbf{v}$,
- There are no other positive (moreover non-negative) eigenvectors except positive multiples of $\mathbf{v}$, i.e., all other eigenvectors must have at least one negative or non-real component.

Note that the Perron-Frobenius theorem only applies to positive matrices. So we still have to use the irreducibility and aperiodicity of the transition matrix to be able to apply the theorem. A special theorem for irreducible matrices exists that shows the same properties.

We are now ready to show why we can find the PageRank vector using eigenvector methods.

Theorem 2. Let $A$ be a stochastic matrix of an irreducible and aperiodic Markov chain. Then the limiting probabilities of $A$ can be found by calculating the eigenvector with largest eigenvalue (in magnitude). Moreover, the limiting probabilities are unique.

Proof. By Lemma 1, we have that every eigenvector independent of all stochastic eigenvectors has an eigenvalue smaller (in magnitude) than 1. By Lemma 2 follows that every stochastic eigenvector has eigenvalue 1. So the largest eigenvalue must be 1 and must correspond to a stochastic eigenvector. Furthermore, the limiting probabilities are unique as there is exactly one stochastic eigenvector.

Now we will discuss the methods to calculate this eigenvector.

### 2.1 Power iteration

One of the most simple methods to calculate an eigenvector $\mathbf{v}=\mathbf{v}_{\mathbf{0}}$ with the largest eigenvalue of matrix $P$ is the power iteration algorithm. This algorithm applies $P$ multiple times to a starting vector $\mathbf{b}$. More formally speaking, suppose a starting vector $\mathbf{b}_{\mathbf{0}}$ is given with $\mathbf{b}_{\mathbf{0}}$ not independent of v. Then define:

$$
\mathbf{b}_{\mathbf{i}}=\frac{P \mathbf{b}_{\mathbf{i}-\mathbf{1}}}{\left\|P \mathbf{b}_{\mathbf{i}-\mathbf{1}}\right\|}
$$

Then $\mathbf{b}_{\mathbf{i}}$ will converge to an eigenvector with the largest eigenvalue of matrix $P$. Other methods build upon this idea. When $\mathbf{b}_{\mathbf{0}}$ can be completely decomposed in a set of eigenvectors $\mathbf{b}=c_{0} \mathbf{v}_{\mathbf{0}}+c_{1} \mathbf{v}_{\mathbf{1}}+\ldots$ it is easily shown why this works:

$$
\begin{aligned}
P^{i} \mathbf{b}_{\mathbf{0}} & =c_{0} P^{i} \mathbf{v}_{\mathbf{0}}+c_{1} P^{i} \mathbf{v}_{\mathbf{1}}+\ldots \\
& =c_{0} \lambda_{0}^{i} \mathbf{v}_{\mathbf{0}}+c_{1} \lambda_{1}^{i} \mathbf{v}_{\mathbf{1}}+\ldots \\
& =\lambda_{0}^{i}\left(c_{0} \mathbf{v}_{\mathbf{0}}+c_{1}\left(\frac{\lambda_{1}}{\lambda_{0}}\right)^{i} \mathbf{v}_{\mathbf{1}}+\ldots\right) .
\end{aligned}
$$

Now the expression between the parentheses converges to $c_{0} \mathbf{v}_{\mathbf{0}}$, because $\frac{\lambda_{1}}{\lambda_{0}}<1$. Therefore $\mathbf{b}_{\mathbf{i}}$ converges linearly to $\mathbf{v}$. If $\mathbf{b}_{\mathbf{0}}$ cannot be decomposed into eigenvectors $P$ must be written in Jordan canonical form and nearly the same reasoning can be applied[11].

### 2.1.1 Optimization for PageRank

The matrix $P$ is most likely a dense matrix, as $\mathbf{e} \mathbf{1}^{T}$ will most likely be dense. So keeping the whole matrix $P$ in memory is not feasible, as this requires $\mathcal{O}\left(n^{2}\right)$ with $n$ being the number of web pages. However, $A$ is sparse as each web page has a limited number of outgoing edges, which limits the number of non-zero elements in each row. Suppose the average number of links per website is $m_{p}$ then $A$ requires $\mathcal{O}\left(n \cdot m_{p}\right)$ memory. Calculating the PageRank with power iteration can now be implemented by following the pseudocode in algorithm 1.

Note that $\mathbf{r}_{\mathbf{0}}$ can be any vector, $\mathbf{e}$ is just an intelligent guess for $\boldsymbol{\pi}$. Obviously, only the last two $r_{i}$ 's need to be saved in this algorithm. This is also the method described in the original paper[6, p. 6].

If a web page is a dangling link, then $A$ will not be a valid Markov matrix. Instead of removing the dangling link, as described earlier, we can also modify the algorithm as seen in algorithm 2[12].

```
Algorithm 1 Calculating PageRank using power iteration
    \(i \leftarrow 0\)
    \(\mathbf{r}_{\mathrm{i}} \leftarrow \mathbf{e}\)
    do
        \(\mathbf{r}_{\mathbf{i}+\mathbf{1}} \leftarrow(1-p) A \mathbf{r}_{\mathbf{i}}+p \mathbf{e}\)
        \(\delta \leftarrow\left\|\mathbf{r}_{\mathbf{i}+\mathbf{1}}-\mathbf{r}_{\mathbf{i}}\right\|_{1}\)
        \(i \leftarrow i+1\)
    while \(\delta>\epsilon\)
```

```
Algorithm 2 Calculating PageRank using power iteration, while handling dangling links
    \(i \leftarrow 0\)
    \(\mathbf{r}_{\mathrm{i}} \leftarrow \mathbf{e}\)
    do
        \(\mathbf{y} \leftarrow(1-p) A \mathbf{r}_{\mathbf{i}}\)
        \(w \leftarrow\left\|\mathbf{r}_{\mathbf{i}}\right\|_{1}-\|\mathbf{y}\|_{1}\)
        \(\mathbf{r}_{\mathbf{i}+\mathbf{1}} \leftarrow \mathbf{y}+w \mathbf{e}\)
        \(\delta \leftarrow\left\|\mathbf{r}_{\mathbf{i}+\mathbf{1}}-\mathbf{r}_{\mathbf{i}}\right\|_{1}\)
        \(i \leftarrow i+1\)
    while \(\delta>\epsilon\)
```


### 2.1.2 Complexity

As described in section 2.1.1, the amount of memory required for our algorithm is $\mathcal{O}\left(n \cdot m_{p}\right)$. To determine the complexity of the algorithm we should first note the following things:

- Since the matrix multiplication $A \mathbf{r}_{\mathbf{1}}$ only requires $\mathcal{O}\left(n \cdot m_{p}\right)$ operations, we can leave out any zero multiplications.
- Scalar multiplication with a vector and vector addition requires $\mathcal{O}(n)$.
- Calculating the norm requires only $\mathcal{O}(n)$.
- One iteration thus requires $\mathcal{O}\left(n \cdot m_{p}\right)$ operations.

The $\delta$ value in the algorithm is determined by the value of $\left(c_{1} \frac{\lambda_{1}}{\lambda_{0}}\right)^{i} \mathbf{v}_{\mathbf{1}}+\ldots$. If we assume that the other eigenvalues are sufficiently small, the magnitude of error term becomes approximately $\left(c_{1} \frac{\lambda_{1}}{\lambda_{0}}\right)^{i}\left\|\mathbf{v}_{\mathbf{1}}\right\|$. So we terminate if:

$$
\begin{equation*}
\left(\frac{\lambda_{1}}{\lambda_{0}}\right)^{i}\left\|c_{1} \mathbf{v}_{\mathbf{1}}\right\|_{1} \leq \epsilon \tag{2.1}
\end{equation*}
$$

and by calculation:

$$
\begin{aligned}
& i \log \left(\frac{\lambda_{1}}{\lambda_{0}}\left\|c_{1} \mathbf{v}_{\mathbf{1}}\right\|_{1}\right) \leq \log \epsilon \\
\Rightarrow & \frac{\log (\epsilon)}{\log \left(\frac{\lambda_{1}}{\lambda_{0}}\left\|c_{1} \mathbf{v}_{\mathbf{1}}\right\|_{1}\right)}<i .
\end{aligned}
$$

So the number of iterations are approximately:

$$
\left\lceil\frac{\log (\epsilon)}{\log \left(\frac{\lambda_{1}}{\lambda_{0}}\left\|c_{1} \mathbf{v}_{\mathbf{1}}\right\|_{1}\right)}\right\rceil
$$

In our case we apply power iteration to a stochastic matrix, so we can simplify this to:

$$
\left\lceil\frac{\log (\epsilon)}{\log \left(\lambda_{1}\left\|c_{1} \mathbf{v}_{\mathbf{1}}\right\|_{1}\right)}\right\rceil .
$$

### 2.2 The second eigenvector and eigenvalue

So far we have considered the first eigenvector only. An interesting question is: do the other eigenvectors have any importance? The proof of Theorem 2 does bring up some ideas. What happens when the Markov chain is reducible? Well, it can be seen that the Markov chain will have multiple limiting probabilities, one for each separated set of nodes in the Markov chain.

So if the Markov chain induced by transition matrix $A$ is reducible, $A$ will have multiple eigenvectors with eigenvalue 1. Logically we could ask ourselves, does this property somehow carry over to our PageRank matrix?

Indeed, a weak version of this property does carry over to our PageRank matrix. Recall the definition of the PageRank matrix:

$$
P=(1-p) A+p\left(\mathbf{e} \cdot \mathbf{1}^{T}\right) .
$$

Recall that $A$ is sparse and is not necessarily irreducible. If $p$ is really small it is intuitive to think that eigenvectors and their eigenvalues $A$ will also be close to the eigenvectors and eigenvalues of $P$. Therefore, $p$ most likely has some influence on the distance between the first eigenvalue and the second eigenvalue. This is useful to investigate as the distance between the first eigenvalue and second eigenvalue (called the eigengap [13, p. 5]) determines the convergence rate of algorithms like power iteration. In the following sections we will formalize this.

An important property when studying the second eigenvector is the irreducibility of $A$. Related to the concept of irreducibility is the concept of communicating classes. A communicating class $C$ is a class or set of states in the state space of $A$ (thus, in our case, web pages). For each pair of states $u, v \in C$ there must be a non-zero probability of ever reaching $u$ from $v$ and reaching $v$ from $u$. Furthermore, no state outside of $C$ must be reachable from any state in $C$. If $A$ is irreducible, it must have only one communicating class[3, p. 194]. As we will see later on, the second eigenvectors seem to be induced by the communicating classes of $A$.

### 2.2.1 Influence of $p$ on the eigengap

Suppose $\mathbf{v}$ is the second eigenvector associated with the second eigenvalue $\lambda_{1}$. We will prove the following theorem:

Theorem 3. (cf. [13, Theorem 1]) The second eigenvector of the PageRank matrix P:

$$
P=(1-p) A+p\left(\mathbf{e} \mathbf{1}^{T}\right)
$$

has an eigenvalue of (in magnitude) at most $1-p$.
We assume that $0<p<1$, however both properties can also easily be proven when $p=1$ or $p=0[13]$. First we need to prove some basic properties of the second eigenvector.

Lemma 3. The second eigenvector of the PageRank matrix $P$ is orthogonal to $\mathbf{1}$ if $P$ is irreducible and aperiodic.

Proof. Let $\lambda_{1}$ be the second eigenvalue of $P$ and $\mathbf{v}_{\mathbf{1}}$ be a corresponding eigenvector. By Lemma 2 we must have $\left|\lambda_{1}\right|<1$. Now we also have:

$$
\begin{aligned}
\left(P^{T} \mathbf{1}\right)_{i} & =\sum_{j} P_{i, j} \mathbf{1}_{j}=1 \\
& \Rightarrow P^{T} \mathbf{1}=\mathbf{1}
\end{aligned}
$$

So 1 is an eigenvector of $P^{T}$ with eigenvalue 1.
But now calculate:

$$
\begin{aligned}
\mathbf{1} \cdot \mathbf{v}_{\mathbf{1}} & =\frac{\mathbf{1} \cdot \lambda_{1} \mathbf{v}_{\mathbf{1}}}{\lambda_{1}} \\
& =\frac{P^{T} \mathbf{1} \cdot P \mathbf{v}_{\mathbf{1}}}{\lambda_{1}} \\
& =\frac{\mathbf{1}^{T} P P \mathbf{v}_{\mathbf{1}}}{\lambda_{1}} \\
& =\frac{\mathbf{1}^{T} \lambda_{1}^{2} \mathbf{v}_{\mathbf{1}}}{\lambda_{1}} \\
& =\lambda_{1}\left(\mathbf{1} \cdot \mathbf{v}_{\mathbf{1}}\right)
\end{aligned}
$$

As $\lambda_{1} \neq 1$ this can only hold if $\mathbf{1} \cdot \mathbf{v}_{\mathbf{1}}=0$. Thus $\mathbf{v}_{\mathbf{1}}$ is orthogonal to $\mathbf{1}$.
Lemma 4. The second eigenvector $\mathbf{v}_{\mathbf{1}}$ of the PageRank matrix $P$ is also an eigenvector of the stochastic matrix of the simplified random surfer model $A$. Moreover, if $\lambda_{1}$ is the eigenvalue of $\mathbf{v}_{\mathbf{1}}$ with matrix $P$ and $\widetilde{\lambda}$ is the eigenvalue of $\mathbf{v}_{\mathbf{1}}$ with matrix $A$. Then:

$$
\widetilde{\lambda}=\frac{\lambda_{1}}{1-p}
$$

Proof. Let $\mathbf{v}_{\mathbf{1}}$ be the second eigenvector of the PageRank matrix $P$. Using Lemma 3 we can write:

$$
\begin{aligned}
\lambda_{1} \mathbf{v}_{\mathbf{1}} & =P \mathbf{v}_{\mathbf{1}} \\
& =(1-p) A \mathbf{v}_{\mathbf{1}}+p\left(\mathbf{e} \mathbf{1}^{T}\right) \mathbf{v}_{\mathbf{1}} \\
& =(1-p) A \mathbf{v}_{\mathbf{1}}+p\left(\mathbf{e}\left(\mathbf{1} \cdot \mathbf{v}_{\mathbf{1}}\right)\right) \\
& =(1-p) A \mathbf{v}_{\mathbf{1}} .
\end{aligned}
$$

So we have:

$$
\frac{\lambda_{1}}{1-p} \mathbf{v}_{\mathbf{1}}=A \mathbf{v}_{\mathbf{1}}
$$

Now we are able to prove Theorem 3.
Proof of Theorem 3. Let the second eigenvector $\mathbf{v}_{\mathbf{1}}$ be given. Let $\lambda_{1}$ be the eigenvalue corresponding to $\mathbf{v}_{\mathbf{1}}$. By Lemma 4 we have that $A \mathbf{v}_{\mathbf{1}}=\frac{\lambda_{1}}{1-p} \mathbf{v}_{\mathbf{1}}$. By Lemma 1 we must have:

$$
\frac{\lambda_{1}}{1-p} \leq 1
$$

Thus $\lambda_{1} \leq 1-p$.
So we know that the eigengap is at most $1-p$. Therefore $p$ could be increased for faster convergence (of course, this influences the resulting vector).

### 2.2.2 The multiplicity of the second eigenvalue

As we have already seen in Lemma 4, the second eigenvector of the PageRank matrix is also an eigenvector of $A$. As discussed $p$ controls the influence of multiple communicating classes in $A$. If $p$ increases, the probability to 'jump' to another communicating class in $A$ increases. Most likely $A$ is reducible, thus cycles will exist in the web graph. Therefore, $A$ will have a stochastic eigenvector with eigenvalue 1 for each communicating class.

If the second eigenvalue is $1-p$, it is possible to derive a lower bound on the multiplicity of the second eigenvalue.

Theorem 4. Let $D=\left\{D_{1}, \ldots, D_{k}\right\}$ be the set of aperiodic communicating classes in $A$, and $k=$ $|D|>2$, that is $k$ is the number of communicating classes. Then the multiplicity of the eigenvalue $1-p$ is at least $k-1$.

Proof. Because each communicating class $D_{i}$ in $D$ is aperiodic, we must have a limiting probability in $A$ when considering only the states in $D_{i}$. So, let $\mathbf{w}_{\mathbf{i}}$ be the limiting probability of $A$ when only considering $D_{i}$. Because communicating classes must be disjoint, we have that $\left\langle\mathbf{w}_{\mathbf{1}}, \ldots, \mathbf{w}_{\mathbf{k}}\right\rangle$ must be independent of each other.

Now we can generate $k-1$ unique eigenvectors. Define for $1 \leq i \leq k-1$ :

$$
\mathbf{w}_{\mathbf{i}}^{*}=\mathbf{w}_{\mathbf{1}}-\mathbf{w}_{\mathbf{i}+\mathbf{1}} .
$$

Then we can write:

$$
\begin{aligned}
P \mathbf{w}_{\mathbf{i}}^{*} & =P \mathbf{w}_{\mathbf{1}}-P \mathbf{w}_{\mathbf{i}+\mathbf{1}} \\
& =(1-p) A \mathbf{w}_{\mathbf{1}}-(1-p) A \mathbf{w}_{\mathbf{i}+\mathbf{1}}+p\left(\mathbf{e} \mathbf{1}^{T}\right) \mathbf{w}_{\mathbf{1}}-p\left(\mathbf{e} \mathbf{1}^{T}\right) \mathbf{w}_{\mathbf{i}+\mathbf{1}} \\
& =(1-p) A \mathbf{w}_{\mathbf{1}}-(1-p) A \mathbf{w}_{\mathbf{i}+\mathbf{1}}+p \mathbf{e}-p \mathbf{e} \\
& =(1-p)\left(\mathbf{w}_{\mathbf{1}}-\mathbf{w}_{\mathbf{i}+\mathbf{1}}\right)=(1-p) \mathbf{w}_{\mathbf{1}}^{*} .
\end{aligned}
$$

So $\mathbf{w}_{\mathbf{i}}^{*}$ is an eigenvector for eigenvalue $1-p$. The $\mathbf{w}_{\mathbf{i}}^{*}$ eigenvectors are linearly independent, as we can generate $\left\langle\mathbf{w}_{\mathbf{1}}, \ldots, \mathbf{w}_{\mathbf{k}}\right\rangle$ using $\left\langle\mathbf{w}_{\mathbf{1}}^{*}, \ldots \mathbf{w}_{\mathbf{k}-\mathbf{1}}^{*}, \mathbf{w}_{\mathbf{1}}\right\rangle$.

Thus, we have $k-1$ independent eigenvectors for the eigenvalue $1-p$.

In conclusion, the second eigenvalue of $P$ is smaller than or equal to $1-p$. Equality holds if $A$ is reducible. The eigenvector is probably a combination of the eigenvectors of $A$ and so, the negative elements in the eigenvector will not communicate with the positive values in the eigenvector. This could be used to detect manipulation.

### 2.3 Krylov

In the section 2.1 we have seen a way to calculate the PageRank using power iteration. In this section and the upcoming sections we will improve the power iteration, while still keeping the objective the same: calculating the eigenvector with the greatest (in magnitude) eigenvalue.

The power iteration method calculates the sequence $\mathbf{v}, A \mathbf{v}, A^{2} \mathbf{v} \ldots A^{i} \mathbf{v}$ with matrix $A$ and vector $\mathbf{v}$ of the appropriate dimension, and returns $A^{i} \mathbf{v}$ as estimation of the requested eigenvector. Krylov methods try to enhance this algorithm by also using the information of the space $\left\langle\mathbf{v}, A \mathbf{v}, A^{2} \mathbf{v} \ldots A^{i} \mathbf{v}\right\rangle$ to determine a better estimation.

Suppose we want to find the second largest eigenvalue (in magnitude) of $A$ (see why this is useful for PageRank in section 2.2.2). The most naive method is using power iteration: first calculate the largest eigenvector, then perform power iteration while keeping the vector orthogonal with the first eigenvector. This has two difficulties: you need to perform the power iteration twice and, even worse, the first calculated eigenvector nearly always has a nonzero error, making it impossible to orthogonalize a vector on the largest eigenvector.

By using the previously calculated iterates, we can solve this problem efficiently. Consider the matrix $A$ with eigenvectors $\mathbf{v}_{\mathbf{0}}, \mathbf{v}_{\mathbf{1}}, \ldots$ and the space $S_{k}=\left\langle A^{k-1} \mathbf{v}, A^{k} \mathbf{v}\right\rangle . S_{k}$ is called a Krylov subspace and is often written as $\mathcal{K}_{2}\left(A^{k-1} \mathbf{v}, A\right)$. For large $k$, the subspace $S_{k}$ will be approximately $\left\langle\mathbf{v}_{\mathbf{0}}, \mathbf{v}_{\mathbf{1}}\right\rangle$ as these are the most dominant eigenvectors. This can be seen by considering $A^{k} \mathbf{v}$ and $A^{k} \mathbf{v}-\lambda_{0} A^{k-1} \mathbf{v}$ for large $k$ :

$$
\begin{aligned}
\frac{A^{k} \mathbf{v}}{\lambda_{0}^{k}} & =c_{0} \mathbf{v}_{\mathbf{0}}+c_{1} \frac{\lambda_{1}^{k}}{\lambda_{0}^{k}} \mathbf{v}_{\mathbf{1}}+\ldots \\
& \rightarrow c_{0} \mathbf{v}_{\mathbf{0}}, \\
\frac{A^{k} \mathbf{v}-\lambda_{0} A^{k-1} \mathbf{v}}{\lambda_{1}^{k-1}} & =c_{1}\left(\lambda_{1}-\lambda_{0}\right) \mathbf{v}_{\mathbf{1}}+c_{2}\left(\lambda_{2}-\lambda_{0}\right) \frac{\lambda_{2}^{k-1}}{\lambda_{1}^{k-1}} \mathbf{v}_{\mathbf{2}}+\ldots \\
& \rightarrow c_{1}\left(\lambda_{1}-\lambda_{0}\right) \mathbf{v}_{\mathbf{1}} .
\end{aligned}
$$

So we have $\lim _{k \rightarrow \infty} S_{k}=\left\langle\mathbf{v}_{\mathbf{0}}, \mathbf{v}_{\mathbf{1}}\right\rangle$. Thus by extension the space $\left\langle\mathbf{v}, A \mathbf{v}, A^{2} \mathbf{v} \ldots A^{i} \mathbf{v}\right\rangle=\mathcal{K}_{i}(\mathbf{v}, A)$ will be roughly spanned by the $i$ most dominant eigenvectors of $A$. Krylov methods mostly use this property by projecting $A$ on the Krylov subspace. The largest eigenvectors of the projection will correspond to the largest eigenvector of $A$.

### 2.3.1 Lanczos and Arnoldi

Lanczos and Arnoldi are two very similar algorithms which calculate the largest $k$ eigenvectors of a matrix $A$. Lanczos iteration only handles symmetric matrices, while Arnoldi is more generic and handles almost every type of matrix.

The Arnoldi iteration uses the Gram-Schmidt process to orthogonalize the Krylov subspace into so-called Arnoldi vectors $\left\langle\mathbf{q}_{\mathbf{0}}, \mathbf{q}_{\mathbf{1}}, \ldots\right\rangle$.

Then using these vectors a matrix $H$ is calculated such that:

$$
H=Q^{T} A Q
$$

with $Q$ the matrix formed by the Arnoldi vectors. $H$ will be in upper Hessenberg form, which is a matrix of the following form:

$$
H=\left(\begin{array}{ccccc}
h_{1,1} & h_{1,2} & h_{1,3} & \ldots & h_{1, n} \\
h_{2,1} & h_{2,2} & h_{2,3} & \ldots & h_{2, n} \\
0 & h_{3,2} & h_{3,3} & \ldots & h_{3, n} \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \ldots & 0 & h_{n, n-1} & h_{n, n}
\end{array}\right)
$$

Because $H$ is in upper Hessenberg form we can calculate the eigenvectors of $H$ using the $Q R$ method. The eigenvectors of $H$ then can be transformed to eigenvectors of $A$ by multiplication of the eigenvector with $Q$.

The Arnoldi vectors are defined as follows:

- $\mathbf{q}_{0}=\mathbf{v}$;
- $\mathbf{q}_{\mathbf{i}}$ is the orthonormal component of $A \mathbf{q}_{\mathbf{i}-\mathbf{1}}$ relative to the basis $\left\langle\mathbf{q}_{\mathbf{0}}, \ldots, \mathbf{q}_{\mathbf{i}-\mathbf{1}}\right\rangle$.

The algorithm terminates when an $i$ is found such that $\mathbf{q}_{\mathbf{i}+\mathbf{1}}=\mathbf{0}$ (or to correct for rounding errors: sufficiently small).

First we will show that the Arnoldi vectors of $A$ with start vector $\mathbf{v}$ indeed span $\mathcal{K}_{i}$.

Lemma 5. The Arnoldi vectors $\left\langle\mathbf{q}_{\mathbf{1}}, \ldots, \mathbf{q}_{\mathbf{i}}\right\rangle$ span $\mathcal{K}_{i}(A, \mathbf{v})=\left\langle\mathbf{v}, A \mathbf{v}, \ldots, A^{i-1} \mathbf{v}\right\rangle$.

Proof. By induction.
For $i=1$ we have $\mathbf{q}_{\mathbf{1}}=\mathbf{v}$ and thus $\left\langle\mathbf{q}_{\mathbf{1}}\right\rangle=\langle\mathbf{v}\rangle$. Now suppose $i>1$, then because $\mathbf{q}_{\mathbf{i}}$ is the orthonormal component of $A \mathbf{q}_{\mathbf{i}-\mathbf{1}}$ on the basis $\left\langle\mathbf{q}_{\mathbf{1}}, \ldots \mathbf{q}_{\mathbf{i}-\mathbf{1}}\right\rangle$, we can find $c_{1}, \ldots, c_{i}$ such that $\left(c_{i} \neq 0\right)$ :

$$
A \mathbf{q}_{\mathbf{i}-\mathbf{1}}=c_{1} \mathbf{q}_{\mathbf{1}}+\ldots+c_{i} \mathbf{q}_{\mathbf{i}}
$$

Now find $d_{1}, \ldots, d_{i-1}$ such that (by induction):

$$
\mathbf{q}_{\mathbf{i}-\mathbf{1}}=d_{1} \mathbf{v}+d_{2} A \mathbf{v}+d_{i-1} A^{i-2} \mathbf{v}
$$

So that:

$$
\begin{align*}
\mathbf{q}_{\mathbf{i}} & =\frac{A \mathbf{q}_{\mathbf{i}-\mathbf{1}}-\sum_{k=1}^{i-1} c_{k} \mathbf{q}_{\mathbf{k}}}{c_{i}} \\
& =\frac{A\left(d_{1} \mathbf{v}+d_{2} A \mathbf{v}+\ldots+d_{i-1} A^{i-2} \mathbf{v}\right)-\sum_{k=1}^{i-1} c_{k} \mathbf{q}_{\mathbf{k}}}{c_{i}} \\
& =\frac{d_{1} A \mathbf{v}+d_{2} A^{2} \mathbf{v}+\ldots+d_{i-1} A^{i-1} \mathbf{v}-\sum_{k=1}^{i-1} c_{k} \mathbf{q}_{\mathbf{k}}}{c_{i}} \\
& =\frac{d_{i-1} A^{i-1} \mathbf{v}}{c_{i}}+\frac{d_{1} A \mathbf{v}+d_{2} A^{2} \mathbf{v}+\ldots+d_{i-2} A^{i-2} \mathbf{v}-\sum_{k=1}^{i-1} c_{k} q_{k}}{c_{i}} . \tag{2.2}
\end{align*}
$$

By induction we have $\mathbf{q}_{\mathbf{1}}, \ldots, \mathbf{q}_{\mathbf{i}-\mathbf{1}} \in\left\langle\mathbf{v}, A \mathbf{v}, \ldots, A^{i-2} \mathbf{v}\right\rangle$ and so:

$$
\begin{equation*}
\frac{d_{1} A \mathbf{v}+d_{2} A^{2} \mathbf{v}+\ldots+d_{i-2} A^{i-2} \mathbf{v}-\sum_{k=1}^{i-1} c_{k} \mathbf{q}_{\mathbf{k}}}{c_{i}} \in\left\langle\mathbf{v}, A \mathbf{v}, \ldots, A^{i-2} \mathbf{v}\right\rangle \tag{2.3}
\end{equation*}
$$

Furthermore we have obviously:

$$
\begin{equation*}
\frac{d_{i-1} A^{i-1} \mathbf{v}}{c_{i}} \in\left\langle\mathbf{v}, A \mathbf{v}, \ldots, A^{i-1} \mathbf{v}\right\rangle . \tag{2.4}
\end{equation*}
$$

Combining 2.2, 2.3 and 2.4 gives $\mathbf{q}_{\mathbf{i}} \in\left\langle\mathbf{v}, A \mathbf{v}, \ldots, A^{i-1} \mathbf{v}\right\rangle$. This gives together with the induction hypothesis

$$
\left\langle\mathbf{q}_{1}, \ldots, \mathbf{q}_{\mathbf{i}}\right\rangle \subseteq\left\langle\mathbf{v}, A \mathbf{v}, \ldots, A^{i-1} \mathbf{v}\right\rangle
$$

Because the dimensions of both $\left\langle\mathbf{q}_{1}, \ldots, \mathbf{q}_{\mathbf{i}}\right\rangle$ and of $\left\langle\mathbf{v}, A \mathbf{v}, \ldots, A^{i-1} \mathbf{v}\right\rangle$ are $i$, we conclude that:

$$
\left\langle\mathbf{q}_{\mathbf{1}}, \ldots, \mathbf{q}_{\mathbf{i}}\right\rangle=\left\langle\mathbf{v}, A \mathbf{v}, \ldots, A^{i-1} \mathbf{v}\right\rangle .
$$

We can also prove that $H$ is in Hessenberg form:

Lemma 6. If $Q$ is the orthogonal transformation matrix generated by the Arnoldi vectors $\mathbf{q}_{\mathbf{1}}, \mathbf{q}_{\mathbf{2}}, \ldots, \mathbf{q}_{\mathbf{n}}$ of $A$ with some initial vector $\mathbf{v}$, then

$$
H=Q^{T} A Q
$$

is a matrix in upper Hessenberg form.

Proof. Let $e_{i-1}$ be the canonical basis vector with a 1 on coordinate $i-1$. Suppose $i<n$. Because $\mathbf{q}_{\mathbf{i}+\mathbf{1}}$ is the orthonormal component of $A \mathbf{q}_{\mathbf{i}}$ on the basis $\left\langle\mathbf{q}_{\mathbf{1}}, \ldots \mathbf{q}_{\mathbf{i}}\right\rangle$, we can take $c_{1}, \ldots, c_{i+1}$ such that $A \mathbf{q}_{\mathbf{i}}=c_{1} \mathbf{q}_{\mathbf{1}}+\ldots+c_{i+1} \mathbf{q}_{\mathbf{i}+\mathbf{1}}$.

Now let $H=\left(\begin{array}{llll}\mathbf{h}_{\mathbf{1}} & \mathbf{h}_{\mathbf{2}} & \ldots & \mathbf{h}_{\mathbf{n}}\end{array}\right)$. We have that:

$$
\begin{aligned}
\mathbf{h}_{\mathbf{i}} & =H \mathbf{e}_{\mathbf{i}} \\
& =H Q^{T} \mathbf{q}_{\mathbf{i}} \\
& =Q^{T} A \mathbf{q}_{\mathbf{i}} \\
& =Q^{T}\left(c_{1} \mathbf{q}_{\mathbf{1}}+c_{2} \mathbf{q}_{\mathbf{2}}+\ldots+c_{i+1} \mathbf{q}_{\mathbf{i}+\mathbf{1}}\right) \\
& =\left(\begin{array}{lllllll}
c_{1} & c_{2} & \ldots & c_{i+1} & 0 & \ldots & 0
\end{array}\right)^{T} .
\end{aligned}
$$

If $i=n$, we can apply the same reasoning but because $\mathbf{q}_{\mathbf{i}+\mathbf{1}}=0$ we have $A \mathbf{q}_{\mathbf{i}}=c_{1} \mathbf{q}_{\mathbf{1}}+\ldots+c_{i} \mathbf{q}_{\mathbf{i}}$, so that $\mathbf{h}_{\mathbf{n}}$ can be expressed in an $n$-dimensional vector.

In algorithm 3 the Arnoldi algorithm is given in its explicit form.

```
Algorithm 3 Arnoldi iteration
    \(\mathbf{q}_{1} \leftarrow \mathbf{v}\)
    \(k \leftarrow 1\)
    while \(\left|\mathbf{q}_{\mathbf{k}}\right|>\delta\) do
        \(k \leftarrow k+1\)
        \(\mathbf{q}_{\mathbf{k}} \leftarrow A \mathbf{q}_{\mathbf{k}-\mathbf{1}}\)
        for \(j=1 \ldots k-1\) do
            \(h_{j, k-1} \leftarrow \mathbf{q}_{\mathbf{j}}^{\mathbf{T}} \mathbf{q}_{\mathbf{k}}\)
            \(\mathbf{q}_{\mathbf{k}} \leftarrow \mathbf{q}_{\mathbf{k}}-h_{j, k-1} \mathbf{q}_{\mathbf{j}}\)
        end for
        \(h_{k, k-1} \leftarrow\left\|\mathbf{q}_{\mathbf{k}}\right\|\)
        \(\mathbf{q}_{\mathbf{k}} \leftarrow \frac{\mathbf{q}_{\mathbf{k}}}{h_{k, k-1}}\)
    end while
```

Now suppose $A$ is symmetric. Then $H^{T}=\left(Q^{T} A Q\right)^{T}=Q^{T} A^{T} Q=Q^{T} A Q=H$. So $H$ is symmetric. As $H$ is also in upper Hessenberg form, $H$ must be in a tridiagonal form:

$$
H=\left(\begin{array}{cccccc}
\alpha_{1} & \beta_{1} & 0 & \ldots & & 0 \\
\beta_{1} & \alpha_{2} & \beta_{2} & & & \vdots \\
0 & \beta_{2} & \alpha_{3} & \ddots & & \\
& & \ddots & \ddots & \beta_{n-2} & 0 \\
\vdots & & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\
0 & & \ldots & 0 & \beta_{n-1} & \alpha_{n}
\end{array}\right)
$$

Lanczos optimizes Arnoldi iteration by eliminating the calculation of all $h_{j, k}$ 's and replaces it by the calculation of $\alpha_{i}$ and $\beta_{i}$. Because our matrix will not be symmetric, Lanczos iteration is purely given as illustration of Krylov methods.

In practice, for large matrices algorithm 3 is not used, because it can take too long before $\left|\mathbf{q}_{\mathbf{k}}\right|<\delta$ (or $\left|\mathbf{q}_{\mathbf{k}}\right|$ is unpredictable). Therefore, we mostly limit the dimension of the Krylov subspace by
stopping whenever the number of iterations exceeds a limit. If the resulting vector is not accurate enough after this number of iterations, we can restart the Arnoldi process. This method is called Restarted Arnoldi.

As seen in section 2.1.2, the convergence of power iteration depends on the second eigenvalue. It is known that the convergence of Arnoldi iteration depends on the $(k+1)^{\text {th }}$ eigenvalue. This is formalized in Theorem 5. The proof of this theorem is out of the scope of this thesis.

Theorem 5. (See [14, Theorem 3.1 and Theorem 5.4]) Apply a $k$-dimensional Arnoldi iteration repeatedly on matrix $A$. Let $n$ be the number of Arnoldi iterations performed. Let $\mathbf{w}$ be the approximation of the largest eigenvector $\mathbf{v}_{\mathbf{0}}$ according to the Arnoldi iteration. Then:

$$
\left\|\mathbf{w}-\mathbf{v}_{\mathbf{o}}\right\|=\mathcal{O}\left(\left|\frac{\lambda_{k+1}}{\lambda_{0}}\right|^{n}\right)
$$

As we have seen in section 2.2 .2 the second eigenvalue $(1-p)$ has a multiplicity of at least the number of aperiodic communicating classes minus one. When applying to real web graphs, the number of communicating web graphs can be considerably large. This leads to the hypothesis that Arnoldi will have no asymptotic influence on the convergence.

Other Krylov techniques exist. In the upcoming section we will elaborate on two techniques that are closely related to Krylov Methods: Reduced Rank Extrapolation (RRE) and Minimal Polynomial Extrapolation (MPE). These two methods are actually Krylov methods when applied to power iteration, but can be applied to a wide range of vector sequences outside of the vector sequence produced by power iteration. For more information about Arnoldi iteration one could have a look at [15].

### 2.4 Extrapolation

Instead of transforming the PageRank matrix into a smaller matrix, it is also possible to directly improve the performance of power iteration. The performance can be improved by using Aitken extrapolation alike methods. These extrapolation techniques use the last three or four iterates and produce an extrapolated approximation of the largest eigenvector. This is very much alike the onedimensional variants: Richardson extrapolation and Aitken extrapolation[16].

Aside the Aitken extrapolation technique, two other extrapolation methods are also shown: Reduced Rank Extrapolation and Minimal Polynomial Extrapolation. These techniques focus more directly on all forms of vector sequences: the extrapolation technique is not limited to power iteration. However, as will be seen in the upcoming sections, this technique is a Krylov method when applied to power iteration.

### 2.4.1 Aitken extrapolation

Recall, that in power iteration we calculate $\mathbf{r}_{\mathbf{i}}=A^{i} \mathbf{r}_{\mathbf{0}}$ for $i$ large enough. Now for our Aitken extrapolation method we assume that for large enough $i, r_{i}$ can be expressed - with a small error margin - in terms of the first two largest eigenvectors $\mathbf{v}_{\mathbf{0}}$ and $\mathbf{v}_{\mathbf{1}}$. The original description of this technique can be read in Extrapolation Methods for Accelerating PageRank Computations[12]. We
write:

$$
\mathbf{r}_{\mathbf{i}-\mathbf{2}}=\mathbf{v}_{\mathbf{0}}+\alpha_{1} \mathbf{v}_{\mathbf{1}} .
$$

And so:

$$
\begin{align*}
\mathbf{r}_{\mathbf{i}-\mathbf{1}} & =P \mathbf{r}_{\mathbf{i}-\mathbf{2}} \\
& =\mathbf{v}_{\mathbf{0}}+\alpha_{1} \lambda_{1} \mathbf{v}_{\mathbf{1}} ; \\
\mathbf{r}_{\mathbf{i}} & =P \mathbf{r}_{\mathbf{i}-\mathbf{1}} \\
& =\mathbf{v}_{\mathbf{0}}+\alpha_{1} \lambda_{\mathbf{1}}^{2} \mathbf{v}_{\mathbf{1}} . \tag{2.5}
\end{align*}
$$

We can now solve for $\mathbf{v}_{\mathbf{0}}$, to this end let $\mathbf{r}_{\mathbf{i}}{ }^{(k)}$ be the $k^{\text {th }}$ component of $\mathbf{r}_{\mathbf{i}}$. Then:

$$
\begin{aligned}
\left(\mathbf{r}_{\mathbf{i}-\mathbf{1}}{ }^{(k)}-\mathbf{r}_{\mathbf{i}-\mathbf{2}}{ }^{(k)}\right)^{2} & =\alpha_{1}^{2}\left(\lambda_{1}-1\right)^{2}\left(\mathbf{v}_{\mathbf{1}}{ }^{(k)}\right)^{2} \\
{\mathbf{\mathbf { r } _ { \mathbf { i } }}}^{(k)}-2 \mathbf{r}_{\mathbf{i}-\mathbf{1}}{ }^{(k)}+\mathbf{r}_{\mathbf{i}-\mathbf{2}}{ }^{(k)} & =\alpha_{1}\left(\lambda_{1}-1\right)^{2} \mathbf{v}_{\mathbf{1}}{ }^{(k)}
\end{aligned}
$$

So we have:

$$
\mathbf{v}_{\mathbf{0}}{ }^{(k)}=\mathbf{r}_{\mathbf{i}-\mathbf{2}}{ }^{(k)}-\frac{\left(\mathbf{r}_{\mathbf{i}-\mathbf{1}^{(k)}}-\mathbf{r}_{\mathbf{i}-\mathbf{2}}(k)\right)^{2}}{\mathbf{r}_{\mathbf{i}}^{(k)}-2 \mathbf{r}_{\mathbf{i}-\mathbf{1}} \mathbf{1}^{(k)}+\mathbf{r}_{\mathbf{i}-\mathbf{2}}{ }^{(k)} .}
$$

It is also possible to subtract the error term from $\mathbf{r}_{\mathbf{i}}$ instead of $\mathbf{r}_{\mathbf{i}-\mathbf{2}}$, as $\lambda_{1}$ is known.

$$
\begin{aligned}
\mathbf{v}_{\mathbf{0}}{ }^{(k)} & =\mathbf{r}_{\mathbf{i}}{ }^{(k)}-\lambda_{1}^{2} \frac{\left(\mathbf{r}_{\mathbf{i} \mathbf{1}}{ }^{(k)}-\mathbf{r}_{\mathbf{i}-\mathbf{2}}{ }^{(k)}\right)^{2}}{\mathbf{r}_{\mathbf{i}}{ }^{(k)}-2 \mathbf{r}_{\mathbf{i} \mathbf{1}}{ }^{(k)}+\mathbf{r}_{\mathbf{i} \mathbf{-}}{ }^{(k)}} \\
& =\mathbf{r}_{\mathbf{i}}{ }^{(k)}-(1-p)^{2} \frac{\left(\mathbf{r}_{\mathbf{i}-\mathbf{1}}{ }^{(k)}-\mathbf{r}_{\mathbf{i}-\mathbf{2}}{ }^{(k)}\right)^{2}{ }^{(k)}-2 \mathbf{r}_{\mathbf{i} \mathbf{- 1}} \mathbf{1}^{(k)}+\mathbf{r}_{\mathbf{i}-\mathbf{2}}{ }^{(k)}}{}
\end{aligned}
$$

Furthermore, it can be noted that $(1-p)$ can still be pretty close to 1 (Kamvar[12] seems to use this approximation in its method).

It is not possible to apply this kind of extrapolation in each step, our assumptions in 2.5 would not hold. Furthermore, note that the number of operations is $\mathcal{O}(n)$, the same as one power iteration. Moreover, note that the precision of the extrapolation depends on the multiplicity of the second eigenvalue. If the second eigenvalue has a multiplicity of 1 , the performance will be optimal.

### 2.4.2 Quadratic extrapolation

Instead of assuming that $\mathbf{r}_{\mathbf{i}}$ can be expressed in two eigenvectors, we will now assume that $P$ has only three eigenvectors. Using this we will approximate the largest eigenvector, using $\mathbf{r}_{\mathbf{i}-\mathbf{3}}, \ldots, \mathbf{r}_{\mathbf{i}}$. The original description of this technique can again be read in Extrapolation Methods for Accelerating PageRank Computations[12]. This method is based upon the Cayley-Hamilton Theorem:

Theorem 6. (cf. [17, p. 340]) Let $A$ be an $n \times n$ matrix and $I_{n}$ the $n \times n$ identity matrix. Then the characteristic polynomial defined by:

$$
p(\lambda)=\operatorname{det}\left(\lambda I_{n}-A\right)
$$

satisfies the identity:

$$
p(A)=0 .
$$

Proof. See appendix B.

We assume, similar to section 2.4.1:

$$
\begin{equation*}
\mathbf{r}_{\mathbf{i}-\mathbf{3}}=\mathbf{v}_{\mathbf{0}}+\alpha_{1} \mathbf{v}_{\mathbf{1}}+\alpha_{2} \mathbf{v}_{\mathbf{2}} . \tag{2.6}
\end{equation*}
$$

Because we assumed $P$ has only three eigenvectors, the characteristic polynomial will be of the form:

$$
p_{P}(\lambda)=\alpha_{0}+\alpha_{1} \lambda+\alpha_{2} \lambda^{2}+\alpha_{3} \lambda^{3} .
$$

Because $P$ 's largest eigenvector is 1 :

$$
\begin{equation*}
p_{P}(1)=\alpha_{0}+\alpha_{1}+\alpha_{2}+\alpha_{3}=0 \tag{2.7}
\end{equation*}
$$

By the Cayley-Hamilton theorem we have:

$$
p_{P}(P)=\alpha_{0} I+\alpha_{1} P+\alpha_{2} P^{2}+\alpha_{3} P^{3}=0 .
$$

And so:

$$
\begin{aligned}
p_{P}(P) \mathbf{r}_{\mathbf{i}-\mathbf{3}} & =\alpha_{0} I+\alpha_{1} P+\alpha_{2} P^{2}+\alpha_{3} P^{3} \\
& =\alpha_{0} \mathbf{r}_{\mathbf{i}-\mathbf{3}}+\alpha_{1} \mathbf{r}_{\mathbf{i}-\mathbf{2}}+\alpha_{2} \mathbf{r}_{\mathbf{i}-\mathbf{1}}+\alpha_{3} \mathbf{r}_{\mathbf{i}} \\
& =0
\end{aligned}
$$

Using equation 2.7:

$$
\begin{aligned}
\alpha_{0} \mathbf{r}_{\mathbf{i}-\mathbf{3}}+\alpha_{1} \mathbf{r}_{\mathbf{i}-\mathbf{2}}+\alpha_{2} \mathbf{r}_{\mathbf{i}-\mathbf{1}}+\alpha_{3} \mathbf{r}_{\mathbf{i}} & =\left(-\alpha_{1}-\alpha_{2}-\alpha_{3}\right) \mathbf{r}_{\mathbf{i}-\mathbf{3}}+\alpha_{1} \mathbf{r}_{\mathbf{i}-\mathbf{2}} \alpha_{2} \mathbf{r}_{\mathbf{i}-\mathbf{1}}+\alpha_{3} \mathbf{r}_{\mathbf{i}} \\
& =\alpha_{1}\left(\mathbf{r}_{\mathbf{i}-\mathbf{2}}-\mathbf{r}_{\mathbf{i}-\mathbf{3}}\right)+\alpha_{2}\left(\mathbf{r}_{\mathbf{i}-\mathbf{1}}-\mathbf{r}_{\mathbf{i}-\mathbf{3}}\right)+\alpha_{3}\left(\mathbf{r}_{\mathbf{i}}-\mathbf{r}_{\mathbf{i}-\mathbf{3}}\right) \\
& =0
\end{aligned}
$$

For convenience we define:

$$
\begin{aligned}
& y_{0}=r_{i}-r_{i-3} ; \\
& \mathrm{y}_{1}=\mathrm{r}_{\mathrm{i}-1}-\mathrm{r}_{\mathrm{i}-\mathbf{3}} \text {; } \\
& \mathbf{y}_{2}=\mathbf{r}_{\mathrm{i}-\mathbf{2}}-\mathrm{r}_{\mathrm{i}-3} \text {. }
\end{aligned}
$$

So we have

$$
\alpha_{1} \mathbf{y}_{\mathbf{2}}+\alpha_{2} \mathbf{y}_{\mathbf{1}}+\alpha_{3} \mathbf{y}_{\mathbf{0}}=0
$$

Because we are only interested in the zero's of the equation we may set $\alpha_{3}=1$ :

$$
\begin{aligned}
\alpha_{1} \mathbf{y}_{\mathbf{2}}+\alpha_{2} \mathbf{y}_{\mathbf{1}} & =-\mathbf{y}_{\mathbf{0}} \\
\left(\begin{array}{ll}
\mathbf{y}_{\mathbf{1}} & \mathbf{y}_{\mathbf{2}}
\end{array}\right)\binom{\alpha_{1}}{\alpha_{2}} & =-\mathbf{y}_{\mathbf{0}} .
\end{aligned}
$$

This is an overdetermined system of equations: we have two variables and $n$ equations. So we solve using the least-squares method.

Now that we have calculated the value of $\alpha_{0}, \ldots, \alpha_{3}$ values, we need to use these values to calculate an approximation of $\mathbf{v}_{\mathbf{0}}$. This is again done using the characteristic polynomial. We know that the characteristic polynomial should have a root at $\lambda=1$ of complexity 1 . We create a new characteristic polynomial that reduces the influence of the other eigenvalues. We can define:

$$
\begin{aligned}
q_{P}(\lambda) & =\frac{p_{P}(\lambda)}{\lambda-1} \\
& =\alpha_{3} \lambda^{2}+\left(\alpha_{3}+\alpha_{2}\right) \lambda+\alpha_{3}+\alpha_{2}+\alpha_{1}+\frac{\alpha_{3}+\alpha_{2}+\alpha_{1}+\alpha_{0}}{\lambda-1}
\end{aligned}
$$

And so, as $\lambda=1$ is a root we have:

$$
q_{P}(\lambda)=\beta_{2} \lambda^{2}+\beta_{1} \lambda+\beta_{0}
$$

With:

$$
\begin{aligned}
& \beta_{2}=\alpha_{3} \\
& \beta_{1}=\alpha_{3}+\alpha_{2} \\
& \beta_{0}=\alpha_{3}+\alpha_{2}+\alpha_{1} .
\end{aligned}
$$

So we have:

$$
(P-I) q_{P}(P)=p_{P}(P)=0
$$

Thus $q_{P}(P) \mathbf{z}$ must be either $\mathbf{0}$ or an eigenvector with eigenvalue 1 of $P$, for some vector $\mathbf{z}$. For $\mathbf{z}$ we can take $\mathbf{r}_{\mathbf{i}-\mathbf{2}}$ as $\mathbf{r}_{\mathbf{i}-\mathbf{2}}$ will be probably already close to $\mathbf{v}_{\mathbf{0}}$.

Furthermore, note that these equations only hold if $P$ only has three eigenvectors. But assuming the other eigenvalues are small enough we can extrapolate $\mathbf{v}_{\mathbf{0}}$ using the following equation:

$$
\begin{aligned}
\mathbf{v}_{\mathbf{0}} & \approx \beta_{0} \mathbf{r}_{\mathbf{i}-\mathbf{2}}+\beta_{1} A \mathbf{r}_{\mathbf{i}-\mathbf{2}}+\beta_{2} A^{2} \mathbf{r}_{\mathbf{i}-\mathbf{2}} \\
& =\beta_{0} \mathbf{r}_{\mathbf{i}-\mathbf{2}}+\beta_{1} \mathbf{r}_{\mathbf{i}-\mathbf{1}}+\beta_{2} \mathbf{r}_{\mathbf{i}}
\end{aligned}
$$

### 2.4.3 MPE and RRE

Minimal Polynomial Extrapolation (MPE) and Reduced Rank Extrapolation (RRE) work by considering the difference between each subsequent vector of the given iterates. As explained in the previous sections, MPE and RRE do not need to be applied to power iteration, but can be applied to a wide type of vector sequences.

Let $\mathbf{x}_{\mathbf{i}}$ be the $i^{\text {th }}$ given iterate and define the following vectors:

$$
\begin{array}{r}
\Delta_{i}=\mathbf{x}_{\mathbf{i}+\mathbf{1}}-\mathbf{x}_{\mathbf{i}} ; \\
\Delta_{i}^{2}=\Delta_{i+1}-\Delta_{i} .
\end{array}
$$

and define the following matrices:

$$
\begin{aligned}
D_{n} & =\left(\begin{array}{llll}
\Delta_{0} & \Delta_{1} & \ldots & \Delta_{n-1}
\end{array}\right) ; \\
V_{n} & =\left(\begin{array}{llll}
\Delta_{0}^{2} & \Delta_{1} & \ldots & \Delta_{n-1}^{2}
\end{array}\right) .
\end{aligned}
$$

Furthermore, in this section we suppose that the sequence $\mathbf{x}_{\mathbf{i}}$ converges to $\mathbf{x}^{*}$.
Roughly speaking, the idea behind MPE and RRE is: treat $\Delta_{i}$ as the vector difference to the limit of the vector series and produce a linear combination of the given iterates. All-in-all RRE is a more complex algorithm than MPE. Both algorithms produce a combination of scalars $\gamma_{i}$. These scalars create a new extrapolation:

$$
\mathbf{s}=\sum_{i=0}^{n-1} \gamma_{i} \mathbf{x}_{\mathbf{i}}
$$

When applied to the sequence of vectors generated by power iteration, the combination is contained in the Krylov subspace $\mathcal{K}_{n}$ generated by $P$ and $x_{0}$, with $n$ being the number of iterates used to extrapolate. Therefore, this basically is a Krylov method.

In the upcoming sections we will elaborate on how the $\gamma$ values are determined in both algorithms. We will assume that MPE and RRE extrapolate power iteration, but the method will still work when the iterates are not a power sequence. For more information about MPE and RRE and their implementation, see [18][19].

## Minimal Polynomial Extrapolation

As seen in Theorem 6 in section 2.4.2, we have that the characteristic polynomial $p(\lambda)$ of $A$ satisfies the identity $p(A)=O$. The method of MPE constructs a minimal polynomial relative to $\Delta_{0}$ such that $p(A) \Delta_{0} \approx \mathbf{0}$ and $p(A)$ must have a degree equal to $n$. First note the following identity:

$$
\begin{align*}
A \Delta_{i} & =A \mathbf{x}_{\mathbf{i}+\mathbf{1}}-A \mathbf{x}_{\mathbf{i}} \\
& =\mathbf{x}_{\mathbf{i}+\mathbf{2}}-\mathbf{x}_{\mathbf{i}+\mathbf{1}} \\
& =\Delta_{i+1} \tag{2.8}
\end{align*}
$$

We have:

$$
p(\lambda)=\sum_{i=0}^{n} c_{i} \lambda^{i} .
$$

And also using equation 2.8 :

$$
\begin{align*}
p(A) \Delta_{0} & =\sum_{i=0}^{n} c_{i} A^{i} \Delta_{0} \\
& =\sum_{i=0}^{n} c_{i} \Delta_{i} . \tag{2.9}
\end{align*}
$$

We can force a degree of $k$ by setting $c_{k}$ to 1 . As written before, we determine the $c_{i}$ values such that $p(A) \Delta_{0} \approx \mathbf{0}$, this is done using least squares. Note that $p$ is also the minimal polynomial relative to all $\Delta_{i}$ 's:

$$
\begin{aligned}
p(A) \Delta_{i} & =p(A) A^{i} \Delta_{0} \\
& =A^{i} p(A) \Delta_{0} \approx A^{i} \mathbf{0}=\mathbf{0}
\end{aligned}
$$

Also it is important to mark that $i$ in this case does not have to be smaller than $n$. Now we apply $p(A)$ to $\mathbf{x}_{\mathbf{0}}$ :

$$
\begin{aligned}
p(A) \mathbf{x}_{\mathbf{0}} & =p(A)\left(\mathbf{x}_{\mathbf{0}}-\mathbf{x}^{*}+\mathbf{x}^{*}\right) \\
& =p(A)\left(\sum_{i=0}^{\infty}\left(\mathbf{x}_{\mathbf{i}}-\mathbf{x}_{\mathbf{i}+\mathbf{1}}\right)+\mathbf{x}^{*}\right) \\
& =p(A)\left(\sum_{i=0}^{\infty} \Delta_{i}+\mathbf{x}^{*}\right) \\
& =p(A) \mathbf{x}^{*} \\
& =\sum_{i=0}^{n} c_{i} A^{i} \mathbf{x}^{*} \\
& =\sum_{i=0}^{n} c_{i} \mathbf{x}^{*}=\mathbf{x}^{*} \sum_{i=0}^{n} c_{i}
\end{aligned}
$$

In the following paragraph we consider the calculation of the MPE.

MPE determines the $\gamma$ values in a computationally simple manner. By equation 2.9 we have a minimization problem:

$$
\operatorname{minimize}\left|\sum_{i=0}^{n} c_{i} \Delta_{i}\right|, \text { subject to } c_{n}=1
$$

This problem can be written as the following linear least squares problem:

$$
\sum_{i=0}^{n-1} c_{i} \Delta_{i} \approx-\Delta_{n}
$$

Now set $c_{n}$ to 1 . Then we have with $\mathbf{c}=\left(c_{0}, \ldots, c_{n}\right)$ :

$$
U_{n+1} \mathbf{c}=\sum_{i=0}^{n-1} c_{i} \Delta_{i}+\Delta_{n} \approx \Delta_{n}-\Delta_{n}=0
$$

The $\gamma$ values can be found by normalizing $\mathbf{c}$ :

$$
\gamma_{i}=\frac{\mathbf{c}}{\sum_{i=0}^{n} c_{i}} .
$$

And then:

$$
\mathbf{s}=\sum_{i=0}^{n-1} \gamma_{i} \mathbf{x}_{\mathbf{i}}
$$

Alternatively, this normalization step can be skipped for our purposes, by normalizing the output vector instead of the scalars:

$$
\begin{aligned}
\mathbf{s} & =\sum_{i=0}^{n-1} c_{i} \mathbf{x}_{\mathbf{i}} \\
\mathbf{s}^{+}{ }_{\mathbf{i}} & =\max \left\{\mathbf{s}_{\mathbf{i}}, 0\right\} ; \\
\mathbf{s}^{*} & =\frac{\mathbf{s}^{+}}{\left\|\mathbf{s}^{+}\right\|_{1}} .
\end{aligned}
$$

This is useful, because the extrapolated value is not necessarily non-negative.

## Reduced Rank Extrapolation

RRE is derived from Full Rank Extrapolation. We will first show Full Rank Extrapolation and then we derive rre from this method. The idea behind Full Rank Extrapolation is to derive $(I-A)^{-1}$ from the iterates. Using $(I-A)^{-1}$ we can derive the limit $\mathbf{x}^{*}$. First write:

$$
\begin{aligned}
\Delta_{i} & =\mathbf{x}_{\mathbf{i}+\mathbf{1}}-\mathbf{x}_{\mathbf{i}} \\
& =A \mathbf{x}_{\mathbf{i}}-\mathbf{x}_{\mathbf{i}} \\
& =A \mathbf{x}_{\mathbf{i}}-\mathbf{x}_{\mathbf{i}}+\mathbf{x}^{*}-\mathbf{x}^{*} \\
& =A \mathbf{x}_{\mathbf{i}}-\mathbf{x}_{\mathbf{i}}+\mathbf{x}^{*}-A \mathbf{x}^{*} \\
& =(I-A)\left(\mathbf{x}^{*}-\mathbf{x}_{\mathbf{i}}\right) .
\end{aligned}
$$

And so:

$$
\mathbf{x}_{\mathbf{j}}-(I-A)^{-1} \Delta_{j}=\mathbf{x}^{*}
$$

Note that in the general case $I-A$ is not invertible, because $\mathbf{x}^{*}$ will be in the null space. This could be solved by using the generalized inverse of $I-A$. Unfortunately, $I-A$ is not known or is too large
to inverse. So we will derive $(I-A)^{-1}$ from $\Delta^{2}$ :

$$
\begin{align*}
\Delta_{i}^{2} & =\Delta_{i+1}-\Delta_{i} \\
& =A \Delta_{i}-\Delta_{i} \\
& =(A-I) \Delta_{i} \\
\Rightarrow V & =(A-I) D . \tag{2.10}
\end{align*}
$$

And if $I-A$ would be invertible:

$$
\begin{gathered}
-D V^{-1}=(I-A)^{-1} \\
\Rightarrow \mathbf{x}_{\mathbf{j}}-D V^{-1} \Delta_{j}=\mathbf{x}^{*}
\end{gathered}
$$

Due to resource constraints it will not be feasible to calculate enough iterates such that $V$ is invertible (not to mention whether this is possible at all). So the idea behind RrE is to apply this method with a reduced rank of $V$. If we assume $(I-A)$ is full rank, then we can write:

$$
\begin{align*}
\mathbf{x}_{\mathbf{j}}-D \boldsymbol{\xi} & =\mathbf{x}^{*}  \tag{2.11}\\
V \boldsymbol{\xi} & =\Delta_{j} . \tag{2.12}
\end{align*}
$$

In RRE we solve 2.12 using the least-squares method and it is then straightforward to calculate $\mathbf{x}^{*}$ using equation 2.11.

Lastly, the value for $j$ can be chosen arbitrarily. Some papers extrapolate $\mathbf{x}_{\mathbf{0}}$. It does not really influence the result.

## Chapter 3

## Bright: an indexation of the tue.nl website

Before we can generate the PageRank of a web page, we need to determine the transition matrix. Various datasets exist, a well-used dataset is the Stanford web graph or WebBase. The data of Stanford web graph was collected in 2002, consists of web pages of the domain of stanford.edu only, and contains 281903 nodes. WebBase contains around 80 million nodes of a large number of domains.

Both datasets are somewhat dated, however, and in the current rapid-changing Internet structure the data may also be less realistic. Therefore, we will crawl the tue.nl domain (and subdomains) and collect a new dataset. We have chosen to collect only data from on the tue.nl domain as collecting a dataset like WebBase would not only demand tremendous amount of time and resources, but calculating the PageRank vector for such a dataset would pose difficulties at our available time and hardware.

### 3.1 Web crawling process

Crawling a web page is usually done in a breadth-first search manner. That is, web pages are visited in a First In First Out order. This prevents indexing only specific parts of the web graph very thoroughly, by crawling all parts of the web graph with the same depth. The algorithm to crawl a website is straightforward and can be seen in algorithm 4.

We have used the framework $\operatorname{SCRAPY}[20]$ to handle most of the web crawling process; it provides a FIFO queue and a way to extract links from HTML. Moreover, as $S$ and $Q$ can become very large in algorithm 4 , the framework also provides a disk-based FIFO queue.

The HREF property of the A tag and SRC of the FRAME and IFRAME are used to extract the links from each web page.

```
Algorithm 4 Web Crawling TuE.NL
    \(S \leftarrow\) \{'www.tue.nl','w3.tue.nl','owinfo.tue.nl','onderwijs.tue.nl','www.tue.nl/en/'\}
    \(Q \leftarrow \operatorname{FifoQueue}(S)\)
    while \(Q \neq\{ \}\) do
        address \(\leftarrow\) Q.dequeue()
        \(h t m l \leftarrow G e t H t m l C o n t e n t s(a d d r e s s)\)
        links \(\leftarrow\) ExtractLinks \((\mathrm{html})\)
        links \(\leftarrow\) FilterByDomain('tue.nl',links)
        links \(\leftarrow\) links \(\backslash S\)
        \(S \leftarrow S \cup\) links
        Q.enqueue(links)
        for link \(\in\) links do
            Q.enqueue(link)
            output address \(\rightarrow\) link
        end for
    end while
```


### 3.2 Indexing keywords

Crawling a web page is mostly I/O bound; most time is spend retrieving a web page. To lower the time spent waiting on I/O, multiple web pages can be retrieved at the same time. Unfortunately, the performance increase is limited when crawling a single domain, as we are still communicating with a single server.

To optimize the resource usage we have chosen to index the keywords of each web page, when waiting on a response of the server, as well as generating a web graph. The aim is to build a small search engine to compare search results with and without using PageRank. Because the crawler is I / O bound this does not slow down the crawl. Using Scrapy we extract all text in the following tags: $\langle\mathrm{P}\rangle,\langle\mathrm{H} 1\rangle,<\mathrm{H} 2\rangle,\langle\mathrm{H} 3\rangle,<\mathrm{H} 4\rangle,<\mathrm{H} 5\rangle,<\mathrm{H} 6\rangle$ and $<$ TITLE $\rangle$.

Text is processed using The Natural Language Toolkit (NLTK)[21]. This toolkit provides various Computational linguistics algorithms. We use a part-of-speech tagger to extract all nouns, foreign words, and the gerund form of verbs ${ }^{1}$ in each sentence. We limit the number of keywords per web page by filtering the keywords using 'stopwords', these contain the most common words used in the language we are processing. A crude language detection algorithm is used, that means that the words included in the dictionary of each language are simply counted. The language that contained the most words in the web page is the identified language.

The keywords we have extracted are very specific, unfortunately. For example: 'arguing' and 'argue' do not match. As a solution we use a stemming algorithm that extracts the root of a given word. This stemming algorithm maps 'arguing' and 'argue' both to 'argu'. These algorithms are, however, heuristically implemented, and as a consequence the right root is not always found. In Dutch, for example, "spitsen" (Peak hours) maps correctly to "spits" (Peak hour), but "spits" maps to "spit" (Lower back pain). This incorrect mapping is not a big concern, as searching on spits will still give results for spits, but searching on spitsen will not result in web pages containing spits. Applying the stemming algorithm until nothing changes, effectively calculating the transitive closure of the

[^2]stemming algorithm, solves this.

### 3.3 Result

The program described in the previous sections was executed and left to run for a long time. Because of the dynamic nature of today's websites (multiple URLS can be mapped to the same resource, making it possible to have an infinite number of web pages), the program was halted after 10 days. At that moment the 250000 unique links in the tue. nl domain have been extracted and 50000 web pages had been processed. Most unprocessed links contain calendars or other less important dynamic pages.

Finally, the resulting dataset was saved in an SQL database; the end size of the database was 1.2 gigabyte. The matrix exported to matlab ASCII format had a size of 122 megabytes.

The dataset has some differences from the datasets like stanford. edu. The stanford.edu dataset contains around 8 links per web page; the tue.nl dataset contains around 120 links per web page. This can be attributed to the dynamic nature of the current websites: not all links in the source code are shown; some are shown only when hovering on a button. Moreover, in 2002, frames were used thoroughly. This caused a table of contents frame to be located on a separate page. Nowadays, the table of contents of a website is often located in the HTML of every web page. This also causes the content to have a much higher PageRank than any other web page. This behavior can of course be questioned, especially when the link is only shown when hovering on a small button.

After the dataset has been processed the PageRank had been calculated using power iteration with $p=0.01$. The dangling links are not removed, but handled by using algorithm 2 . A comparison of the different methods and their performance is given in the next chapter. Some interesting results arise when indexing the website. One of the most interesting results is the most important web page according to the PageRank algorithm: http://cso.tue.nl/webph. This web page contains a search engine that can be used to search for the email address for each student.

This arises question 'Why is this the important web page?'. To explain this we first need to have a look at the structure of the tue.nl domain. This domain contains multiple subdomains:

- www.tue.nl The 'main' web page of the TU/e, contains mostly data for external visitors and future students. This web page is redesigned to offer usability and attractiveness, but offers less detailed data. The website links to the w3.tue.nl site for more elaborate information.
- w3.tue.nl The 'old' web page of the TU/e, this website is aimed on students and staff members. Contains a lot of elaborated information.
- owinfo.tue.nl The 'old' web interface providing information for students on courses and grading.
- onderwijs.tue.nl The 'new' web interface providing information for students on courses and grading.
- Multiple small subdomains per department.

As the w3.tue.nl seems to contain the most information and pages, it is the most important part according to PageRank. The w3.tue.nl subdomain has 8 different front ends. An English front end, a Dutch front end, a front end with enhanced contrast and a front end with enlarged text. These last two options can be mixed with all other options giving a total of 8 different front ends, with different urls. All internal links in a specific front end always stay within the same front end. This effectively splits the PageRank between these 8 front ends.

Each front end contains a contact link, causing this link to have a very high ranking. The contact web page contains a lot of links, but most of these stay within the front end. However, the web page also contains an absolute link to cso.tue.nl/webph and owinfo.tue.nl, causing these web pages to inherit the PageRank from all 8 front ends. To top it of cso.tue.nl/webph is a spider trap, the web graph is a complete graph with 3 nodes. Combining this gives us a PageRank of around 0.015 for each of the 3 nodes in the spider trap.

This result gives us some concerns about the algorithm. Increasing the $p$-value to $p=0.1$ solves the problem, but makes it easier to set up empty links to artificially influence the PageRank algorithm. Obviously, no real manipulation has taken place in our web graph. In table 3.1 the ten most important pages are shown according to the PageRank algorithm with $p=0.1$. The table shows us that the results are better in line with our expectations.

| Description | URL | PageRank |
| :--- | :--- | :--- |
| Home page of internal website | w3.tue. $\mathrm{nl} / \mathrm{nl} /$ | $3.64 \times 10^{-3}$ |
| University Library | w3.tue.nl/nl/diensten/bib/ | $3.39 \times 10^{-3}$ |
| About the internal website | w3.tue.nl/nl/over_de_site/ | $3.37 \times 10^{-3}$ |
| Internal website log in | w3.tue.nl/nl/inloggen/ | $3.35 \times 10^{-3}$ |
| FAQ | w3.tue.nl/nl/veelgestelde_vragen/ | $3.35 \times 10^{-3}$ |
| Services | w3.tue.nl/nl/diensten/ | $3.33 \times 10^{-3}$ |
| Links to all departmental pages | w3.tue.nl/nl/faculteiten/ | $3.31 \times 10^{-3}$ |
| The University | w3.tue.nl/nl/de_universiteit/ | $3.31 \times 10^{-3}$ |
| Alphabetical listing of web pages | w3.tue.nl/nl/tue_a_tot_z/ | $3.31 \times 10^{-3}$ |
| Home page of the external website | www.tue.nl/ | $2.80 \times 10^{-3}$ |

Table 3.1 The ten most important pages according to the PageRank algorithm with $p=0.1$; It should be noted that all pages in this top ten are written in Dutch.

### 3.4 Distance between departments

As seen in chapter 1, it is possible to personalize the PageRank algorithm on a specific subject by changing the $\mathbf{e}$ vector. The $\mathbf{e}$ vector then becomes a distribution describing the importance of each web page according to the chosen subject. This can also be applied to determine the distance between two web pages or even websites.

We have tried to determine the importance of each department web page according to each other department. Our process can be described as follows:

- $\operatorname{DEP}(\mathrm{i})$ is the set of web pages belonging to department i .
- Define $\mathbf{e}^{(\mathbf{i})}$ as follows:

$$
\mathbf{e}_{\mathbf{j}}^{(\mathbf{i})}= \begin{cases}\frac{1}{|D E P(i)|} & \text { if } j \in D E P(i) \\ 0 & \text { otherwise }\end{cases}
$$

- Calculate the PageRank $\mathbf{p r}{ }^{(j)}$ for each source of rank vector $\mathbf{e}_{\mathbf{j}}^{(\mathbf{i})}$.
- Calculate the non-personalized PageRank vector pr*.
- Set the combined PageRank distance of department $i$ to department $j$ as

$$
\operatorname{dist}(i \rightarrow j)=\frac{\sum_{k \in D E P(j)} \mathbf{p r}_{k}^{(i)}}{\sum_{k \in D E P(j)} \mathbf{p r}_{\mathbf{k}}^{*}}
$$

Using this process $\operatorname{dist}(i \rightarrow j)$ basically becomes a ratio of the importance of the department $j$ relative to department $i$ to the importance of the department $j$ in the whole website. Obviously, dist is not a metric function, as it does not conform to the definition. For example, $\operatorname{dist}(i \rightarrow i) \neq 0$. For our analysis this does not matter.

The following departments of the TU/e are considered:

- Biomedical Engineering (BMT);
- Built Environment (bwk);
- Chemical Engineering and Chemistry (CHEM);
- Electrical Engineering (ELE);
- Industrial Design (ID);
- Industrial Engineering \& Innovation Sciences (IE);
- Applied Physics (PHYS);
- Mathematics and Computer Science (win);
- Mechanical Engineering (wTB).

Each department has an informational/recruiting website and a website for students and/or staff. The resulting values of dist can be seen in table 3.2.

The first thing that becomes clear from table 3.2, is that the departments are not well interconnected: the combined PageRank decreases when a personalized source of rank is chosen. This means that it is more likely to reach the department from a random web page than from a random web page in the chosen department. Furthermore, we see that the Applied Physics department is the most interconnected and the Built Environment department is the second best interconnected department. Industrial Engineering \& Innovation Sciences is the least connected department. It should be noted that the connectedness to itself $(\operatorname{dist}(i \rightarrow i))$ has no real influence on the interconnectedness to other departments.

|  | Department $(j)$ | BMT | BWK | CHEM | ELE | ID | IE | PHYS | WIN | WTB |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| Personalization $(i)$ |  |  | 0.18 | 0.13 | 0.13 | 0.12 | 0.15 | 0.16 | 0.12 | 0.14 |
| BMT | 0.20 | 97 | 0.21 | 0.22 | 0.12 | 0.31 | 0.27 | 0.22 | 0.18 |  |
| BWK | 0.21 | 0.31 | 46 | 0.23 | 0.14 | 0.18 | 0.25 | 0.19 | 0.18 |  |
| CHEM | 0.19 | 0.29 | 0.20 | 44 | 0.12 | 0.17 | 0.27 | 0.22 | 0.17 |  |
| ELE | 0.18 | 0.20 | 0.14 | 0.15 | 42 | 0.12 | 0.19 | 0.15 | 0.12 |  |
| ID | 0.09 | 0.13 | 0.09 | 0.09 | 0.06 | 32 | 0.11 | 0.09 | 0.08 |  |
| IE | 0.22 | 0.34 | 0.24 | 0.25 | 0.15 | 0.24 | 48 | 0.21 | 0.20 |  |
| PHYS | 0.16 | 0.22 | 0.15 | 0.17 | 0.09 | 0.26 | 0.21 | 35 | 0.14 |  |
| WIN | 0.17 | 0.25 | 0.17 | 0.19 | 0.11 | 0.26 | 0.21 | 0.17 | 51 |  |
| WTB |  |  |  |  |  |  |  |  |  |  |

Table 3.2 The values of $\operatorname{dist}(i \rightarrow j)$ for each starting department ( $i$, the rows) to each ending department ( $j$, the columns).

## Chapter 4

## Performance analysis

To compare the algorithms described in the chapter 2, we have implemented an algorithm and profiled the performance on the dataset of chapter 3 . A low $p$-value of 0.01 is taken to be able to better distinguish the difference in performance. The algorithms are implemented in matlab. We aim to determine the PageRank vector with an absolute error of at most $10^{-8}$. Using double precision, the maximum attainable precision is $1.24 \cdot 10^{-16}$.

Dangling links are handled by using the method of algorithm 2. The reference method, power iteration, requires 1098 iterations to reach an absolute error of $10^{-8}$. The code used for power iteration can be seen in appendix A.1.

### 4.1 Extrapolation methods

The first class of methods is extrapolation methods. These methods require a fixed number of observations to be able to extrapolate. However, the extrapolation methods cannot be applied too often, as the result will be instable. The methods MPE and rre are not included in this class, as these methods can be applied directly without power iteration steps in between. Moreover, MPE and RRE can use different numbers of observations to extrapolate the vector sequence. Therefore, we view mpe and Rre as Krylov methods for the scope of this chapter. Extrapolations are done for the first time at the $10^{\text {th }}$ iteration. The distance between the extrapolations is varied to determine the ideal value of this parameter. In figure 4.1 the number of iterations needed to reach an error value of at most $10^{-8}$ can be seen. We first see that convergence is not guaranteed for a small interval between each extrapolation.

To guarantee convergence we apply some logic to limit the number of extrapolations by noting:

- The linear extrapolation depends on three pure power iteration results.
- The quadratic extrapolation depends on four pure power iteration results.
- The last iteration should not be extrapolated, there is no guarantee that an extrapolated value is a correct value.
- The extrapolation methods do not necessarily converge, but power iteration does.


Figure 4.1 Number of iterations needed to reach an absolute error of at most $10^{-8}$

We combine this to find the following rules to guarantee convergence:

- A minimum of 3 or 4 power iterations between each extrapolation
- After an extrapolation, do not terminate until 4 or 3 power iterations have occurred.
- Only extrapolate if the absolute error has decreased since the last extrapolation step.

These variants are marked with '(safe)'. The matlab code used to implement this variant of linear extrapolation and the quadratic extrapolation can be seen in appendix A. 2 and A. 3 respectively.

We also see some unstable behavior at the interval size of 358,495 , etc. After carefully observing the behavior of a linear extrapolation step, we observe that the extrapolated values are very infrequently completely negative. When considering the deduction of these extrapolation methods this should not occur and it is unclear to me why this happens. Because the matrix multiplication we use assumes a completely positive input vector, multiplication behaves erroneously until the power method reaches a positive vector (we do in fact reach the positive vector, because the matrix multiplication behaves erroneously when the vector is negative). This problem can be fixed by taking the absolute value of each element of the vector after an extrapolation step. Applying this fix results in the graph of figure 4.2.

We can see in figure 4.2, that for an extrapolation gap larger than 200, the number of iterations required increases linearly with the interval size. We see that for a small number of iterations between a linear extrapolation, the results are very unstable, whereas the results of the quadratic extrapolation are much more stable. A close up for less iterations between each extrapolation can be seen in figure 4.3.

In figure 4.3 we see that a value between 120 and 200 results in a stable and optimum number of iterations. The best performance is attained by the linear extrapolation method, but this seems to be based on sheer luck. If we take an extrapolation interval of 120 , the absolute error behaves as in figure


Figure 4.2 Number of iterations needed to reach an absolute error value of at most $10^{-8}$, avoiding negative values


Figure 4.3 Number of iterations needed to reach an absolute error of at most $10^{-8}$ for smaller extrapolation intervals


Figure 4.4 The absolute error per iteration. The extrapolation interval is 120 .
4.4. We see with overwhelming evidence that the quadratic extrapolation is superior to the linear extrapolation method and power iteration. Using quadratic extrapolation we reach the absolute error of $10^{-8}$ in 165 iterations, $16 \%$ of the number of iterations needed by power iteration! This is even more dramatic than the $31 \%$ showed in the reference paper. We should however note, that this is also largely due to the low value of $p$, this causes a very slow convergence of power iteration, leaving more time to apply an extrapolation.

Furthermore, it should be noted that quadratic extrapolation reaches the maximum precision attainable in only 200 steps.

### 4.2 Krylov methods

The second class of methods we can use to determine the limiting probability are the Krylov methods. The methods we examine are Arnoldi, mpe and rre. The Arnoldi method is somewhat different than the algorithm given in algorithm 3. As described earlier, instead of stopping when the $\delta$ value is low enough, we force an upper bound on the dimension of $H$, or equivalently the number of iterations. All Krylov methods are repeated until the $\delta$ value is below the given bound. For each of the three methods we varied the Krylov subspace dimension. The time spent calculating is observed as well as the number of matrix multiplications needed (only considering the transition matrix). The matlab code used to implement the Arnoldi iteration, MPE and RRE can be seen in appendix A.4, A. 5 and A. 6 respectively.

Please note that the Arnoldi method's performance only depends on the $k^{\text {th }}$ eigenvalue, so when the dimension of the Krylov subspace has exceeded the multiplicity of the second eigenvalue, the method should reach its peak performance. It might be interesting to see if this same phenomenon occurs in the other two Krylov methods.


Figure 4.5 The number of Matrix multiplications with the transition matrix $(P)$ needed to reach an absolute error of at most $10^{-8}$.

As Krylov methods bundle a high number of calculations in one iteration, especially when a high Krylov dimension size is chosen, the $\delta$ values between each iteration are generally high compared to power iteration. This causes very accurate vectors to still be rejected, while the vectors would be sufficient if power iteration was used. To level the field and provide a fair comparison with the extrapolation methods, we perform one step of power iteration between each iteration of the Krylov method. The $\delta$ value of this calculation is used to determine if the stopping criteria is fulfilled. The Krylov methods would even encounter defective matrices if this method was not used, as the numerical precision became an issue before the stopping criteria was fulfilled.

In the graph of figure 4.5 we see the number of matrix multiplications with the transition matrix. This measurement gives an idea of the overhead and the number of extrapolations each method performs. If the overhead of the Krylov methods is low enough this should be a good measure of performance. For lower Krylov dimension the overhead is low. It should be noted MPE and Rre methods encountered defective matrices for larger Krylov dimensions ( $k \gtrsim 100$ ). We did not stop these methods (i.e. we did not use a lower Krylov dimension) when this occurred.

In figure 4.6 the amount of CPU time spent calculating is shown for different number of Krylov dimensions. This graph is useful to compare to figure 4.5 and assess the overhead of each method. We see that for a large dimension size the overhead becomes more apparent, as the CPU time needed for larger dimensions increases super-linearly, while the number of matrix multiplications increases linearly.

The Arnoldi method should have a complexity determined by the $k^{\text {th }}$ eigenvalue, where $k$ is the dimension size of the Krylov subspace. As we achieve a dramatic increase in performance when using Arnoldi, the multiplicity of the second eigenvalue is probably low. A low multiplicity of the second


Figure 4.6 The amount of CPU time spent calculating until an absolute error of at most $10^{-8}$ is reached. The subgraph in the upper-left corner zooms in to show more detail for lower number of dimensions.
eigenvalue is plausible, as this would indicate that the transition matrix contains a limited number of communicating classes. As our web graph is build from a highly connected website, all web pages are most likely highly connected with all other web pages.

The MPE and RRE method seem to exhibit the same behavior in their performance curve as the Arnoldi method. Therefore, combined with the fact that both are Krylov methods, the complexity could be very similar to power iteration. Further research is obviously needed to verify this. The MPE method seems to be somewhat more reliable than the RRE method but the performance is very comparable. It seems that MPE and RRE are comparable in performance to the linear and quadratic extrapolation.

The Arnoldi iteration seems slow in calculation time, it has a lot of overhead. Even compared to power iteration, as we will see in the next section, it does not seem fast. The high overhead may be due to slow interpretation of the matLab script: the Arnoldi function uses less build-in functions, while the other methods are composed out of high-performance and compiled functions, like least squares. It can be interesting to implement Arnoldi in a compiled language and test its performance.

### 4.3 Performance Comparison

From the previous sections we have seen that Arnoldi Iteration was slow, MPE was the best method out of all Krylov methods and that quadratic extrapolation was the most stable and fastest extrapolation technique.

We analyze the performance in three situations. The first situation is the Bright web graph with
a $p$-value of 0.01 . This situation heavily favors extrapolation techniques; power iteration has a hard time converging in this case. The second situation is the Bright web graph with $p$-value of 0.1 . This situation is more realistic and does not allow many extrapolation moments. The last situation features a very unrealistic $p$-value of 0.001 . In all cases we calculate the PageRank vector with an absolute error of at most $10^{-8}$.

The extrapolation methods all extrapolate with an interval between each extrapolation of 120 matrix multiplications. In section 4.1 it can be seen that this value produces a stable result and achieves a good performance. The Krylov subspace has been chosen to have a maximum dimension of 30 . In section 4.2 it can be seen that this is a relatively stable and optimum choice.

In table 4.1 we see the performance of the various algorithms. It is clear that the quadratic extrapolation is the fastest method. It converges more than 6 times faster than power iteration! Furthermore, we see that MPE and RRE need the least number of matrix multiplications, but the overhead needed causes the quadratic extrapolation to be more performant.

Lastly, the Arnoldi method is only marginally faster than power iteration. This is very disappointing as the environment heavily favors any extrapolation technique.

| Algorithm | Parameters | Matrix Multiplications | CPU time |
| :--- | :--- | :--- | ---: |
| Power iteration | - | 1098 | 20.15 s |
| Linear extrapolation (Safe) | Interval $=120$ | 365 | 7.01 s |
| Quadratic extrapolation (Safe) | Interval $=120$ | 165 | 3.31 s |
| Restarted Arnoldi | Dimension $=30$ | 180 | 19.98 s |
| Minimal Polynomial Extrapolation | Dimension $=30$ | 93 | 5.75 s |
| Reduced Rank Extrapolation | Dimension $=30$ | 93 | 6.17 s |

Table 4.1 Performance of various algorithms on the Bright data set with $p=0.01$

In table 4.2 the performance of the various algorithms is given when $p=0.1$. This case is more realistic and does not unfairly favor the extrapolation techniques. This can be seen directly, all Krylov methods are slower - possibly due to the overhead - than power iteration. It should be noted that the Krylov methods probably eliminate the impact of the second eigenvalue. Therefore, it can easily be seen that this optimization is not useful when $p$ is sufficiently large. Actually, the Krylov methods did not significantly perform better, because the convergence of these methods does not depend on the eigengap caused by the $p$ value.

It has only been proven that Arnoldi iteration does not depend on the eigengap caused by the $p$ value, but this result indicates that MPE and RRE could also be independent of the second eigenvalue.

The linear and quadratic extrapolation techniques did enjoy a speedup, but are only barely faster than power iteration.

As we have seen, the performance RRE and MPE methods is largely independent of the $p$-value, we like to verify this claim by choosing $p=0.001$. This is a very impractical situation, but this should substantiate this claim.

In table 4.3 we see the results of this test. Indeed, the Krylov methods are not heavily influenced by the $p$-value, but are certainly not independent. Interestingly, the quadratic extrapolation is still the fastest method and also seems somewhat independent of the eigengap. The linear extrapolation is however, dependent on the $p$-value and the performance degrades even faster (relative to linear extrapolation on higher $p$-values) than power iteration.

| Algorithm | Parameters | Matrix Multiplications | CPU time |
| :--- | :--- | :--- | ---: |
| Power iteration | - | 117 | 2.29 s |
| Linear Extrapolation (Safe) | Interval $=120$ | 97 | 1.94 s |
| Quadratic Extrapolation (Safe) | Interval $=120$ | 108 | 2.23 s |
| Restarted Arnoldi | Dimension $=30$ | 150 | 17.11 s |
| Minimal Polynomial Extrapolation | Dimension $=30$ | 62 | 3.91 s |
| Reduced Rank Extrapolation | Dimension $=30$ | 93 | 6.07 s |

Table 4.2 Performance of various algorithms on the Bright data set with $p=0.1$

| Algorithm | Parameters | Matrix Multiplications | CPU time |
| :--- | :--- | :--- | ---: |
| Power iteration | - | 5389 | 97.07 s |
| Linear extrapolation (Safe) | Interval $=120$ | 2856 | 51.30 s |
| Quadratic extrapolation (Safe) | Interval $=120$ | 255 | 5.26 s |
| Restarted Arnoldi | Dimension $=30$ | 240 | 26.55 s |
| Minimal Polynomial Extrapolation | Dimension $=30$ | 155 | 9.56 s |
| Reduced Rank Extrapolation | Dimension $=30$ | 155 | 9.68 s |

Table 4.3 Performance of various algorithms on the Bright data set with $p=0.001$

In conclusion we note that quadratic extrapolation is probably the most reliable and fastest method to use. For a large eigengap (i.e., in this case, large $p$ value) the linear extrapolation performs similar to or better than the quadratic extrapolation, but is not as reliable.

## Chapter 5

## Conclusion

To assess the importance of a web page, we can use the PageRank algorithm developed by Larry Page and Sergey Brin. This algorithm assigns a PageRank value to each web page by using the pages that link to the web page and their PageRank value. So to calculate the PageRank of a web page it requires the PageRank of all linking web pages. Therefore, the algorithm is recursive and requires a large web graph. In chapter 3 we have seen that the web graph can become large for even a single website. Therefore, the PageRank algorithm is not ideal when assessing the importance of a single web page, but is ideal when assessing the importance of a large set of web pages.

To avoid manipulation a source of rank vector e was introduced and a $p$-value that can increase the influence of the source of rank. In chapter 4 we saw that the performance increases when a higher $p$-value was chosen. The performance of the basic algorithm used to calculate the PageRank vector (the power iteration) was inversely related to the $p$-value. But by using extrapolation or Krylov subspace techniques the influence of the $p$-value was diminished. The power iteration was not significantly slower then the extrapolation techniques and even faster than the Krylov methods when a high $p$-value was chosen.

The $p$-value determines how interconnected web pages are and reduces the irreducibility of the Markov chain of our web graph. It was seen that the $p$-value determines the value of the second eigenvalue, which is very important in the performance of power iteration. Arnoldi iteration and other Krylov methods work by projecting the PageRank matrix on a much smaller space, and eliminating the influence of the second eigenvalue on the convergence rate. The two extrapolation techniques MPE and RRE seemed to work in a similar manner, but no theoretical analysis proofs this yet.

In addition to increasing the difficulty of manipulation, the influence of cycles was also reduced by choosing a high $p$-value. In our analysis of the TU/e website, we saw that for a $p$-value of 0.01 a meaningless web page was marked as the most important web page. This could be blamed on cycles in close affinity of the web page. The drawback is that higher $p$-values make the PageRank vector more uniform. Moreover, when a uniform source of rank is taken, a too high $p$-value actually decreases the difficulty of manipulation, as each empty web page created will always receive a high PageRank. Therefore, the source of rank can be chosen to only add PageRank to already well-connected web pages.

A real web-graph (Bright) was found by crawling the $\mathrm{TU} / \mathrm{e}$ website for 10 days. It was seen that
crawling took far longer than actually generating the PageRank - as this was done within 20 seconds. Of course, when multiple sites are crawled at once (so the crawling speed is not limited by a single server) and the crawler is connected with a high speed, low latency Internet connection this problem might be reduced. A small text search engine was build using the PageRank vector and a natural language processing tool kit (NLTK).

The TU/e aims to have a strong collaboration between their departments. Using the Bright web graph we have performed an analysis, from which we can conclude that the websites of the departments were not strongly interconnected.

### 5.1 Future work

The Bright web-graph was explored using a basic technique that did not distinguish table of content links and links in the text of the web page itself. This made each node have a large number of outgoing links. In future research, this could be optimized by distinguishing links inside text tags $(<\mathrm{P}>,<\mathrm{H} 1>, \ldots,<\mathrm{H} 6>,<$ ARTICLE $>)$. This is harder than it seems, as not all web pages are structured the same and false positives in distinguishing these tags can occur.

The source of rank vector e used in our algorithms is a basic uniform vector. This makes manipulation of websites easier. As a solution an e can be chosen such that $\mathbf{e}$ only has positive values for nodes that are definitely not created for manipulation purposes. An example can be to generate e such that only the home page of each website has positive values. The manipulator would have to create a new domain for each manipulation node it creates, making the process time consuming and costly.

The analysis of the algorithms was done thoroughly, however a wide variations of options can still be explored. For example, instead of repeating a Krylov method directly after the required precision is not reached, a number of power iterations can also be performed before restarting the Krylov method (Arnoldi, MPE and RRE).

The Arnoldi method performed poorly. It might be possible to blame this upon the large number of manual matLab operations used to implement this, while other methods could be implemented by only performing a pair of matLab commands. As these matlab commands are heavily optimized and compiled to native code, this would cause a major speed-up. To perform a more equal comparison, Arnoldi could be implemented in a programming language that can be compiled.

Furthermore, the algorithm should be tested on other web graphs and a comparison between our web graph and other web graphs could uncover some interesting properties.

Lastly, an algorithm like PageRank can be developed that assesses the PageRank of a single web page without needing to calculate the PageRank of all web pages. Major applications for such an algorithm are Search Engine Optimizers (SEO) and on the fly ranking of persons in social networks.

### 5.2 Discussion

In this thesis a lot of work was spent exploring a real web graph and testing the various algorithms on the web graph with different kinds of parameters. As this web graph was more recent than many known web graphs this option was advantageous. However, as we only had limited resources to explore
and generate a web graph, the web graph was limited to only one website and had a limited number, though not necessarily a low number, of nodes. Furthermore, the web graph was well interconnected as only one website was indexed, which reduced the number of communicating classes and thus the multiplicity of the second eigenvalue. Therefore, calculating the PageRank vector of the Bright web graph might be easier than a real web graph. Therefore, it would be interesting to test the algorithms on other web graphs.

Testing the algorithms showed a connection with the theoretical analysis. This combination of theoretical analysis and numerical experimentation was well balanced and through the numerical experimentation some hypotheses could be formulated for further research. The theoretical analysis showed a clear connection between the stochastic side of the problem and the numerical side. This made this research project widely oriented and made it summarize some major building blocks of the PageRank algorithm. These major building blocks are in other research projects and papers often omitted, they only consider the numerical side.

The performance of the quadratic extrapolation technique surprised me, especially for low $p$ values, as it could even beat a 30-dimensional extrapolation technique that was optimal for low $p$-values. The performance of the Arnoldi method negatively surprised me, as it was slower than the power iteration in some cases, and in the best case only marginally faster.

All in all, the thesis has shown me some great connections between different kinds of mathematics: stochastic processes, numerical analysis, linear algebra and graph theory. I hope that this thesis can be useful for anyone researching this topic.

## Appendix A

## Matlab Code

## A. 1 Power iteration

```
load transition_matrix.dat
% Convert text to a sparse matrix
A = spconvert(transition_matrix);
% find the matrix dimension
n = max(size(A));
% split into 3 vectors indicating all non-zero parts
[i,j,s] = find(A);
% rebuild the matrix with correct dimensions
A = sparse(i,j,s,n,n);
% Create source of rank
e =ones([n,1]) * 1.0/n;
% --- Power iteration (algorithm 2) --- %
i = 0;
% saving all r values is pointless, as we only need the last 2
r = e;
p = 0.1;
delta = Inf;
epsilon = 0.00000001;
while delta > epsilon
    y = (1-p)* (A*r);
    % norm(r,1) should be 1
    % Note: w > 0, because p > 0 (should trump any numerical errors)
    w = 1 - norm(y,1);
    % w > 0, so r_new > 0
    r_new = y + w*e;
    delta = norm(r_new - r, 1);
    i = i + 1;
    r = r_new;
end
csvwrite('PowerIteration_PageRank.dat', r);
```


## A. 2 Linear extrapolation

```
load transition_matrix.dat
% Convert text to a sparse matrix
A = spconvert(transition_matrix);
% find the matrix dimension
n = max(size(A));
% split into 3 vectors indicating all non-zero parts
[i,j,s] = find(A);
% rebuild the matrix with correct dimensions
A = sparse(i,j,s,n,n);
% create source of rank
e =ones([n,1]) * 1.0/n;
% --- Power iteration (algorithm 2) with linear extrapolation --- %
extrapolate_gap = 120;
i = 0;
% saving all r values is pointless, as we only need the last 2
prev_r = e;
prevprev_r = e;
r = e;
p = 0.1;
extrapolate_delta = Inf;
extrapolate_i = 9 - extrapolate_gap;
restarttime = 11;
delta = Inf;
epsilon = 0.00000001;
while delta > epsilon || i - 3 < extrapolate_i
    y = (1-p)* (A*r);
    % norm(r,1) should be 1
    % using norm(r,1) will lead to an unstable calculation... norm(r,1) will
    % decrease or increase.
    % Note: w > 0, because p > 0 (should trump any numerical errors)
    w = 1 - norm(y,1);
    % w > 0, so r_new > 0
    r_new = y + w*e;
    delta = norm(r_new - r, 1);
    prevprev_r = prev_r;
    prev_r = r;
    r = r_new;
    if delta > epsilon && i - extrapolate_gap > extrapolate_i && delta < ...
            extrapolate_delta
            r = LinearExtrapolate(r, prev_r, prevprev_r, p);
            extrapolate_delta = delta;
            extrapolate_i = i;
        end
        i = i + 1;
        if i > 10000
            % Not converging...
            i = Inf;
            break;
```

```
49 end
50 end
    csvwrite('PowerIterationLinearExtrapolated_PageRank.dat', r);
```

```
function [ v ] = LinearExtrapolate( r, prev_r, prevprev_r, p )
    v =r - (1 - p ) ^ 2 * ((prev_r - prevprev_r).^2)./(r - prev_r - prev_r + ...
        prevprev_r);
    v = v ./ norm(v , 1);
    v = sign(sum(v)) * v;
end
```


## A. 3 Quadratic extrapolation

```
load transition_matrix.dat
% Convert text to a sparse matrix
A = spconvert(transition_matrix);
% find the matrix dimension
n = max(size(A));
% split into 3 vectors indicating all non-zero parts
[i,j,s] = find(A);
% rebuild the matrix with correct dimensions
A = sparse(i,j,s,n,n);
% Create source of rank
e =ones([n,1]) * 1.0/n;
% --- Power iteration (algorithm 2) with quadratic extrapolation --- %
extrapolate_gap = 120;
i = 0;
% saving all r values is pointless, as we only need the last 2
prev_r = e;
prevprev_r = e;
prevprevprev_3 = e;
r = e;
p = 0.1;
extrapolate_delta = Inf;
extrapolate_i = 9 - extrapolate_gap;
restarttime = 11;
delta = Inf;
epsilon = 0.00000001;
while delta > epsilon || i - 3 < extrapolate_i
    y = (1-p)* (A*r);
    % norm(r,1) should be 1
    % using norm(r,1) will lead to an instable calculation... norm(r,1) will
    % decrease or increase.
    % Note: w > 0, because p > 0 (should trump any numerical errors)
    w = 1 - norm(y,1);
    % w > 0, so r_new > 0
    r_new = y + w*e;
    delta = norm(r_new - r, 1);
```

```
    prevprevprev_r = prevprev_r;
    prevprev_r = prev_r;
    prev_r = r;
    r = r_new;
    if delta > epsilon && i - extrapolate_gap > extrapolate_i && delta < ...
        extrapolate_delta
        r = QuadraticExtrapolation(r, prev_r, prevprev_r, prevprevprev_r);
        extrapolate_delta = delta;
        extrapolate_i = i;
    end
    i = i + 1;
    if i > 10000
        % Not converging...
        i = Inf;
        break;
    end
end
csvwrite('PowerIterationQuadraticExtrapolated_PageRank.dat', r);
```

```
function [ v ] = QuadraticExtrapolation( r, r_1, r_2, r_3 )
    y_2 = r_2 - r_3;
    y_1 = r_1 - r_3;
    y_0 = r - r_3;
    Y = [y_2 y_1];
    % perform least squares
    x = -Y \ y_0;
    gamma_3 = 1;
    gamma_1 = x(1);
    gamma_2 = x(2);
    beta_0 = gamma_1 + gamma_2 + gamma_3;
    beta_1 = gamma_2 + gamma_3;
    beta_2 = gamma_3;
    v = beta_0 * r_2 + beta_1 * r_1 + beta_2 * r;
    % normalize v again
    v = v ./ norm(v , 1);
    v = sign(sum(v)) * v;
end
```


## A. 4 Arnoldi Iteration

```
load transition_matrix.dat
% Convert text to a sparse matrix
A = spconvert(transition_matrix);
% find the matrix dimension
n = max(size(A));
6 % split into 3 vectors indicating all non-zero parts
7 [i,j,s] = find(A);
8 % rebuild the matrix with correct dimensions
```

```
A = sparse(i,j,s,n,n);
% Create source of rank
e =ones([n,1]) * 1.0/n;
% --- Repeated Arnoldi iteration --- %
% Arnoldi saves the r values
m = 30;
p = 0.1;
delta = Inf;
epsilon = 0.00000001;
i = 0;
r = e;
while delta > epsilon
    [Q, H] = Arnoldi(A, r, e, p, m, epsilon);
    % note H is projection of A on the basis of Q
    % note <q_1, q_2,...> has a smaller dimension, so converting back does
    % only approximate eigenvector H
    % calculate largest eigenvector of H
    [~, ~, U] = svd(H - eye(size(H)));
    % the lowest eigenvector of H - I is located at the end of U
    eigenvector = U(:,end);
    r_new = Q * eigenvector;
    % r_new may be negative
    r_new = sign(sum(r_new)) * r_new;
    r_new = r_new *( 1 / norm(r_new, 1));
    % transform eigenvector back to default basis
    r = r_new;
    % one iteration of the power method
    y = (1-p)* (A*r);
    % norm(r,1) should be 1
    % using norm(r,1) will lead to an unstable calculation... norm(r,1) will
    % decrease or increase.
    % Note: w > 0, because p > 0 (should trump any numerical errors)
    w = 1 - norm(y,1);
    % w > 0, so r_new > 0
    r_new = y + w*e;
    delta = norm(r - r_new, 1);
    i = i + 1;
    r = r_new;
end
csvwrite('ArnoldiIteration_PageRank.dat', r);
```

```
function [q, H] = Arnoldi( A, q_0, e, p, maxit, threshold )
% --- algorithm 3 --- %
n = max(size(e));
q = zeros(n,maxit - 1);
q(:,1) = q_0;
H = zeros(maxit,maxit - 1);
for i = 2:maxit
    % our custom matrix multiplication:
```

```
    z = (1-p)* (A*q(:,i-1));
    % p > 0 so w > 0
    % ||q(i-1)||_1 is not necessarily 1 (||q(i-1)||_2 = 1, though!)
    w = norm(q(:,i-1), 1) - norm(z,1);
    % w > 0 so q(i) >0
    z = z + w*e;
    % perform Arnoldi - Gram-Schmidt
    for j = 1:i-1
    H(j,i-1) = dot(q(:,j), z);
    z = z - H(j,i-1)*q(:,j);
    end
    H(i,i-1) = norm(z);
    if(i ~= maxit)
        q(:,i) = z ./ H(i, i - 1);
    end
    if H(i,i-1) == 0
        H = reshape(H, i, i);
        q = reshape(q, n, i);
        break;
    end
end
end
```


## A. 5 Minimal polynomial extrapolation

```
load transition_matrix.dat
% Convert text to a sparse matrix
A = spconvert(transition_matrix);
% find the matrix dimension
n = max(size(A));
% split into 3 vectors indicating all non-zero parts
[i,j,s] = find(A);
% rebuild the matrix with correct dimensions
A = sparse(i,j,s,n,n);
% create source of rank
e =ones ([n,1]) * 1.0/n;
% --- Repeated MPE iteration --- %
m = 30;
p = 0.1;
delta = Inf;
epsilon = 0.00000001;
i = 0;
r = e;
while delta > epsilon
    [v] = MPE(A, r, e, p, m);
    r_new = v;
    % r_new may be negative
    r_new = sign(sum(r_new)) * r_new;
    r_new = r_new *( 1 / norm(r_new, 1));
```

```
    % transform eigenvector back to default basis
    r = r_new;
    % one iteration of the power method
    y = (1-p)* (A*r);
    % norm(r,1) should be 1
    % Note: w > 0, because p > 0 (should trump any numerical errors)
    w = 1 - norm(y,1);
    % w > 0, so r_new > 0
    r_new = y + w*e;
    delta = norm(r - r_new, 1);
    i = i + 1;
    r = r_new;
end
csvwrite('MPEIteration_PageRank.dat', r);
```

```
function [ v ] = MPE(A, r, e, p, maxit)
    n = max(size(e));
    R = zeros(n,maxit);
    % perform power iteration maxit times
    for i = 1:maxit
        y = (1-p)* (A*r);
        w = 1 - norm(y,1);
        r = y + w*e;
        R(:,i) = r;
    end
    U=R (:, 2:end-1)-R (:, 1:end-2);
    C= U \ - (R(:,end) - R(:,end-1));
    c}(end+1,1)=1
    v=(R (:, 2: end)*C);
    v = bsxfun(@max, v, 0);
    v = v ./(norm(v, 1));
end
```


## A. 6 Reduced rank extrapolation

```
load transition_matrix.dat
% Convert text to a sparse matrix
A = spconvert(transition_matrix);
% find the matrix dimension
n = max(size(A));
% split into 3 vectors indicating all non-zero parts
[i,j,s] = find(A);
% rebuild the matrix with correct dimensions
A = sparse(i,j,s,n,n);
% create source of rank
e =ones([n,1]) * 1.0/n;
% --- Repeated RRE iteration --- %
```

```
% Arnoldi saves the r values
m = 30;
p = 0.1;
delta = Inf;
epsilon = 0.00000001;
i = 0;
r = e;
while delta > epsilon
    [v] = RRE(A, r, e, p, m);
    r_new = v;
    % r_new may be negative
    r_new = sign(sum(r_new)) * r_new;
    r_new = r_new *( 1 / norm(r_new, 1));
    % transform eigenvector back to default basis
    r = r_new;
    % 1 iteration of the power method
    y = (1-p)* (A*r);
    % norm(r,1) should be 1
    % using norm(r,l) will lead to an instable calculation... norm(r,1) will
    % decrease or increase.
    % Note: w > 0, because p > 0 (should trump any numerical errors)
    w = 1 - norm(y,1);
    % W > 0, so r_new > 0
    r_new = y + w*e;
    delta = norm(r - r_new, 1);
    i = i + 1;
    r = r_new;
end
csvwrite('RRE Iteration_PageRank.dat', r);
```

```
function [ v ] = RRE (A, r, e, p, maxit)
    n = max(size(e));
    R = zeros(n,maxit);
    % perform power iteration maxit times
    for i = 1:maxit
        y = (1-p)* (A*r);
        w = 1 - norm(y,1);
        r = y + w*e;
        R(:,i) = r;
    end
    D=R (:, 2:end) - R (:, 1:end-1);
    V=D (:,2:end)-D (:,1:end-1);
    sigma = V \ D (:, end);
    v = R(:,end) - D(:,1:end-1) * sigma;
    v = bsxfun(@max, v, 0);
    v = v ./(norm(v, 1));
end
```


## Appendix B

## Theorems

Lemma 7. Let a Jordan block $J$ with $\lambda$ on the diagonal and of dimension $m \times m$ be given. Then for $i>0, j>0$ and $k \geq 0$ we have:

$$
\left(J^{j}\right)_{i, i+k}=\frac{d^{k}}{d \lambda^{k}} \lambda^{j}
$$

All other cases are 0.

Proof. Let a Jordan block $J$ with $\lambda$ on the diagonal and of dimension $m \times m$ be given. Then $\left(J^{j}\right)_{i, k}$ with $k<i$ is obviously equal to 0 . Now suppose $i>0, k \geq 0$. Then for $j=1$ we satisfy the condition by definition of a Jordan block.

When $k=0$ we satisfy the property obviously:

$$
\begin{aligned}
\left(J^{j}\right)_{i, i} & =\sum_{l=0}^{m}\left(J^{j-1}\right)_{i, l}(J)_{l, i} \\
& =\left(J^{j-1}\right)_{i, i} J_{i, i}
\end{aligned}
$$

Now suppose $j>1$ and $k>0$. Then:

$$
\begin{aligned}
\left(J^{j}\right)_{i, i+k} & =\sum_{l=0}^{m}\left(J^{j-1}\right)_{i, l}(J)_{l, i+k} \\
& =\sum_{l=i}^{i+k}\left(J^{j-1}\right)_{i, l}(J)_{l, i+k} \\
& =\sum_{l=0}^{k}\left(J^{j-1}\right)_{i, i+l}(J)_{i+l, i+k} \\
& =\sum_{l=k-1}^{k}\left(J^{j-1}\right)_{i, i+l}(J)_{i+l, i+k} \\
& =\sum_{l=k-1}^{k}\left(\frac{d^{l}}{d \lambda} \lambda^{j-1}\right)(J)_{i+l, i+k} \\
& =\left(\frac{d^{k}}{d \lambda} \lambda^{j-1}\right) * \lambda+\frac{d^{k-1}}{d \lambda} \lambda^{j-1} \\
& =\frac{d^{k}}{d \lambda} \lambda^{j}
\end{aligned}
$$

Theorem (6). Let $A$ be an $n \times n$ matrix and $I_{n}$ the $n \times n$ identity matrix. Then the characteristic polynomial defined by:

$$
p(\lambda)=\operatorname{det}\left(\lambda I_{n}-A\right) .
$$

satisfies the identity:

$$
p(A)=0 .
$$

Proof. (See [22]) Let $A \in \mathbb{R}^{n \times n}$ be given. Let $J$ be the Jordan normal form of matrix $P$. Then for some matrix $P \in \mathbb{R}^{n \times n}$ we have:

$$
A=P J P^{-1}
$$

With J as block matrix:

$$
J=\left(\begin{array}{llll}
J_{0} & & & \\
& J_{1} & & \\
& & \ddots & \\
& & & J_{n-1}
\end{array}\right)
$$

Multiplication and addition of block diagonal matrices with the same dimension of its blocks trivially distribute to its blocks. This gives us:

$$
p(J)=\left(\begin{array}{cccc}
p\left(J_{0}\right) & & & \\
& p\left(J_{1}\right) & & \\
& & \ddots & \\
& & & p\left(J_{n-1}\right)
\end{array}\right)
$$

Now let $m$ be such that $J_{0} \in \mathbb{R}^{m \times m}$ :

$$
J_{0}=\left(\begin{array}{ccccc}
\lambda_{0} & 1 & & & \\
& \lambda_{0} & 1 & & \\
& & \ddots & \ddots & \\
& & & \lambda_{0} & 1 \\
& & & & \lambda_{0}
\end{array}\right)
$$

Now by an induction argument we have (see Lemma 7):

$$
\left(J_{0}^{j}\right)_{i, i+k}=\frac{d^{k}}{d \lambda} \lambda^{j}
$$

Thus the elements at the right of the diagonal are the derivatives of the performed multiplication. Because derivation is linear we get:

$$
p\left(J_{0}\right)=\left(\begin{array}{ccccc}
p\left(\lambda_{0}\right) & p^{\prime}\left(\lambda_{0}\right) & p^{\prime \prime}\left(\lambda_{0}\right) & \ldots & \\
& p\left(\lambda_{0}\right) & p^{\prime}\left(\lambda_{0}\right) & \ldots & \\
& & \ddots & & \\
& & & p\left(\lambda_{0}\right) & p^{\prime}\left(\lambda_{0}\right) \\
& & & & p\left(\lambda_{0}\right)
\end{array}\right)
$$

Because the block size of $J_{0}$ corresponds to the multiplicity of $\lambda_{0}$ in $p(\lambda)$ we have:

$$
p\left(J_{0}\right)=O
$$

And because $A^{k}=P J^{k} P^{-1}$, we can factor out $P$ and $P^{-1}$ in the equation of $p$ :

$$
p(A)=p\left(P J P^{-1}\right)=P p(J) P^{-1}=P O P^{-1}=O
$$

## References

[1] D. Sullivan. Google Still World's Most Popular Search Engine By Far, But Share Of Unique Searchers Dips Slightly, February 2013. URL http://searchengineland.com/ google-worlds-most-popular-search-engine-148089.
[2] Inc. Google. Google Fun Facts. URL http://web.archive.org/web/20090424093934/http://www. google.com/press/funfacts.html.
[3] S. Ross. Introduction to Probability Models, Ninth Edition. Academic Press, Inc., Orlando, FL, USA, 2006. ISBN 0125980620.
[4] G. Ivan and V. Grolmusz. When the Web meets the cell: using personalized PageRank for analyzing protein interaction network. Bioinformatics, 27(3), 2011.
[5] D. Walker, H. Xie, K. Yan, and S. Maslov. Ranking Scientific Publications Using a Simple Model of Network Traffic. Journal of Statistical Mechanics: Theory and Experiment, 2007.
[6] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web, 1999.
[7] H. Bhat and B. Sims. InvestorRank and an Inverse Problem for PageRank. Under review, 2012. URL http://faculty.ucmerced.edu/hbhat/BhatSims2012.pdf.
[8] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. WTF: The Who to Follow Service at Twitter. In Proceedings of the 22Nd International Conference on World Wide Web, WWW '13, pages 505-514, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-2035-1. URL http://dl.acm.org/citation.cfm?id= 2488388.2488433.
[9] R. Andersen, C. Borgs, J. Chayes, U. Feige, A. Flaxman, A. Kalai, V. Mirrokni, and M. Tennenholtz. Trust-based recommendation systems: an axiomatic approach. In Proceedings of the 17th international conference on World Wide Web, pages 199-208. ACM, 2008. URL http://research.microsoft.com/ en-us/um/people/borgs/Papers/trust.pdf.
[10] A. Elhashash and D. Szyld. Perron-Frobenius Properties of General Matrices. Technical report, 2007.
[11] J. Gubernatis and T. Booth. Multiple extremal eigenpairs by the power method. J. Comput. Physics, 227(19):8508-8522, 2008. URL http://dblp.uni-trier.de/db/journals/jcphy/jcphy227. html\#GubernatisB08.
[12] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Extrapolation Methods for Accelerating PageRank Computations. In Proceedings of the 12th International Conference on World Wide Web, WWW '03, pages 261-270, New York, NY, USA, 2003. ACM.
[13] T. Haveliwala and S. Kamvar. The Second Eigenvalue of the Google Matrix. Technical Report 2003-20, Stanford InfoLab, 2003.
[14] A. Sidi. Application of Vector-Valued Rational Approximations to the Matrix Eigenvalue Problem and Connections with Krylov Subspace Methods. SIAM Journal on Matrix Analysis and Applications, 16 (4):1341-1369, 1995. URL http://dx.doi.org/10.1137/S089547989224316X.
[15] R. Morgan. On Restarting the Arnoldi Method for Large Nonsymmetric Eigenvalue Problems. Mathematics of Computation, 65(215):pp. 1213-1230, 1996. ISSN 00255718. URL http://www.jstor.org/ stable/2153800.
[16] A. Sidi. Practical Extrapolation Methods Theory and Applications. Cambridge University Press, 2003. ISBN 0521661595.
[17] G. Birkhoff. A Survey of Modern Algebra. The Macmillan Comp., 1977.
[18] A. Sidi. Efficient implementation of minimal polynomial and reduced rank extrapolation methods. Journal of Computational and Applied Mathematics, pages 305-337, 1991.
[19] Avram Sidi. Vector extrapolation methods with applications to solution of large systems of equations
and to PageRank computations. Computers $\mathcal{E}$ Mathematics with Applications, 56(1):1-24, 2008. ISSN 0898-1221. doi: http://dx.doi.org/10.1016/j.camwa.2007.11.027. URL http://www.sciencedirect.com/ science/article/pii/S0898122107008188.
[20] Scrapy. Scrapy, a fast high-level screen scraping and web crawling framework for Python. http://scrapy. org, 2008-2014.
[21] S. Bird. Nltk: The natural language toolkit. In In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics, 2002.
[22] MathPages. The Cayley-Hamilton Theorem. http://www.mathpages.com/home/kmath640/ kmath640.htm.
[23] G. Wu and Y. Wei. A Power-Arnoldi algorithm for computing PageRank. Numer. Linear Algebra Appl., pages 521-546, 2007.
[24] A. Sidi. Approximation of Largest Eigenpairs of Matrices and Applications to PageRank Computation. September 2004.

## Index

## Aperiodic

Markov chain, 7
state, 7
Arnoldi
iteration, 16
vectors, 17

Backlink counting, 1
Breadth-first search, 29
Communicating class, 13

Dangling links, 3
Dense matrix, 11

Eigengap, 13
Hessenberg form, 17

InvestorRank, 4
Irreducible
Markov chain, 7

Limiting probabilities, 6
Markov chain, 2
Memoryless
process, 2

PageRank
definition with assignment, 7 definition with Markov chains, 6 personalized, 3
part-of-speech tagger, 30
Perron root, 10

Random Surfer Model, 3
simplified, 2
Rank sink, 3
Rank source, 3

Recurrent states, 3
Restarted Arnoldi, 20

Sparse matrix, 11
Stochastic
matrix, 5
vector, 5
Stopwords, 30
Transient states, 3


[^0]:    ${ }^{1}$ In contrary to what is often assumed PageRank is not named after web pages, but after its creator: Larry Page[2]

[^1]:    ${ }^{1}$ Note that the limiting probability exists if the Markov chain is irreducible, that is: each web page is reachable from every web page, and aperiodic. e must be chosen such that this property holds.

[^2]:    ${ }^{1}$ The gerund participle of a verb has the same function as a noun and can be seen. An example is painting in "She is good at painting."

