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

The PageRank Problem

Thesis submitted on behalf of the
Delft Institute for Applied Mathematics
in partial fulfilment
of the requirements

for the degree of

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

M. (Rien) den Besten

Delft, The Netherlands
August 2010

BSc verslag TECHNISCHE WISKUNDE

"Het PageRank probleem"
(Engelse titel: "The PageRank Problem")

M. (Rien) den Besten

Technische Universiteit Delft

Begeleider

Dr.ir. M.B. van Gijzen

Overige commissieleden

Prof.dr.ir. C. Vuik	Dr.ir. H.X. Lin
Dr. J.G. Spandaw	

Augustus, 2010 Delft

Preface

This thesis is the result of a bachelor research project about Google's PageRank. In this project, an analysis of the hyperlink structure of the World Wide Web was made and a model for web surfing studied. Based on this model, the usual method to compute the PageRank of web pages was investigated. Special attention was given to computing PageRanks by using linear systems. In this respect the IDR(s) method was applied to achieve an efficient computation. Several numerical experiments were performed in order to compare the efficiency of different methods in computing PageRanks.

I kindly want to thank Martin van Gijzen for his helpful and pleasant supervision on this project.

Rien den Besten
Delft, 20 August 2010

Contents

Chapter 1

Introduction

The successful advent of search engine Google has drawed a broad attention to the way Google determines the relevance of web pages in order to sort search results. Google introduced a new approach: its search engine assigns a so called *PageRank* to web pages, which is a measure of the importance of these pages. We call finding the PageRanks of web pages in (a part of) the World Wide Web *the PageRank problem*. The calculation of the PageRanks of web pages is principally based on the hyperlink structure of the World Wide Web. Finding a solution to the PageRank problem is the main subject of the thesis ad hand.

This thesis forms the conclusion of a bachelor project titled 'How does Google work?' and is one of the bachelor projects in numerical simulations. The description of this bachelor project can be found at *http://ta.twi.tudelft.nl/nw/users/vuik/wi3606/pagerank.pdf* (in Dutch only). The stated assignments in the description comprise (1) modelling the hyperlink structure of the World Wide Web, (2) calculating PageRanks by approaching the PageRank problem as an eigenvalue problem and (3) eventually adapting the model.
In working out this bachelor project, the assignments are extended to a broader context of proving the existence and uniqueness of a solution of the PageRank problem, using a linear systems approach to solve the PageRank problem, describing several methods to compute PageRank and proving whether the conditions under which these method generate a solution, are fulfilled.

1.1 Research questions

The following research questions are being discussed in the thesis at hand:

1. *Considering the PageRank problem, what is an adequate model of the hyperlink structure of the World Wide Web?*
2. *For the given model:*
   (a) *does a unique solution of the PageRank problem exist?*
   (b) *how can PageRanks be computed efficiently?*

1.2 Contents

Chapter 2 contains a general introduction to the history and working of search engines as well as the concepts that search engine Google uses. Chapter 3 describes a model of the hyperlink structure of the World Wide Web which is extended in a model for web surfing. For this model, the existence and uniqueness of a solution to the PageRank problem is proved. Chapter 3 answers the first research question. The fourth chapter shows how and under which conditions we can compute PageRanks by approaching the PageRank problem as an eigenvalue problem. Chapter 5 provides a linear systems approach to the PageRank problem and presents several methods to compute PageRank. The chapters 4 and 5 give an answer to research question 2a. Chapters 6 and 7 describe respectively the performance and results of numerical experiments in order to find an answer research questions 2b. Finally, conclusions are drawn and recommendations made in chapter 8.

Chapter 2

Search engines

2.1 Searching on the World Wide Web

The content of this section is mainly taken from [9] and [11].
Since the very first beginning of the existence of the World Wide Web the need of indexing the larger-growing amount of available data was clear. This was done in different ways: a list of web servers was available at the CERN web server (at the end of the 1980s) and several individuals maintained a list of websites sorted by topic. The Internet growed too fast, however, and maintaining personal lists became soon inadequate. This was the reason for the development of internet search engines.

The first search engine that helped users to find computer files in internet databases was developed in Montreal at the McGill University in 1990. This engine, called *Archie*, visited at night all known archives and listed the names of the files in these archives into a searchable database. The content of these files was not indexed by Archie.

The first search engine with which one could search for words or phrases in files and web pages was *WebCrawler*, developed in 1994. As its name says, the engine was crawler-based and it had three main tasks: web crawling, indexing and searching. What these mean will be explained in the next sections. Many search engines were launched shortly after WebCrawler appeared, AltaVista, for example, among them. Almost every one of them was based on the same principles as WebCrawler had and these became standard for the major search engines since then. Even the search engine Google, which became soon very popular since its appearance in 1998, is based on these principles.

2.2 Web crawling and indexing

A web search engine crawls and indexes the world wide web, which means that it stores information about web pages on the Internet in a large database. The web crawler follows every link that it finds on a web page. Meanwhile, the content of every single web page that the crawler visits is analysed and indexed. Key words of each web page are stored in a database. The crawler also stores in a database which pages are connected to each other via a hyperlink.

2.3 Sorting search results

If a search engine receives a query, it starts a search in the database for all pages that satisfy the query. The number of found pages can be very large. A search engine's success depends on the way is sorts the pages that satisfy a query: relevant pages should be listed first and the least relevant ones last. The search engine Google has become very popular for the way it determines the relevance of web pages: it assumes in general that the relevance of a web page can be measured by its importance depending on the hyperlink structure of the web that this page is part of. This principle will be explicated in the next section.

2.4 Google's principle: concepts and definitions

At first we will formulate some limitations and assumptions:

- The range of a search engine is limited to the World Wide Web consisting of hypertext documents.
- A citation is a hyperlink on a web page to another web page.
- Search results are the collection of web pages that satisfy a query sent to a search engine.
- The PageRank $pr(w)$ of a web page w determines the importance of that web page.
- The importance of web pages determines the sorting of search results: the most important page is the first result.

Now, PageRanks can be defined as follows [15]: the PageRank of a page w, $pr(w)$, is a function of the PageRanks of all other pages:

1. A page w gets a higher ranking as more pages cite w.
2. The higher the ranking of pages citing page w is, the higher $pr(w)$ is.

2.5 Remarks

Although PageRank as defined above clearly forms a basis of the search engine Google and of many other search engines by now, there is much more Google does to sort search results. Google uses "overall importance and query-specific relevance" with "Hypertext-Matching Analysis"[8] besides its PageRank technology. In this thesis, however, we focus on PageRanking only.

Further notice that the way of sorting search results as mentioned, does not guarantee that the most relevant pages will be listed first. Google *assumes* that important (and consequently relevant) pages are much cited pages [8, *p. 3*]. Although this assumptions seems to be useful in practice, it principally gives no certainty about relevance, importance or usefulness of pages in general or in particular. A further discussion on this topic goes beyond the scope of this research project.

Chapter 3

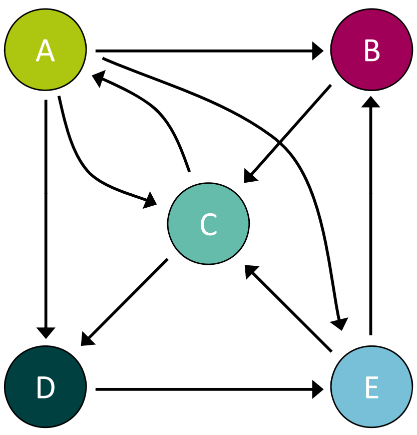
Developing models

In this chapter we present a simple model of the hyperlink structure of the World Wide Web. This model will be extended afterwards to a model for web surfing. The Google matrix will be defined and some properties of it will be mentioned. Several parts of this chapter are based on or summarised from [11], [13] and [15].

3.1 A simple model of the web structure

The World Wide Web can be represented as a directed graph in which each web page is a node and where the web page's hyperlinks are the edges. It is useful to distinguish two kinds of links: *outlinks* and *inlinks*. From web page i's perspective, an *outlink* is a hyperlink on i directing to web page j while a hyperlink on j directing to web page i is called *inlink*.

In this section this representation of the web is used to explain Google's definition of a page's ranking.



	A	B	C	D	E
1	0.2	0.2	0.2	0.2	0.2
2A	0	0.05	0.05	0.05	0.05
2B	0	0.05	0.25	0.05	0.05
⋮			...		
2	0.175	0.15	0.3	0.2	0.175
⋮			...		
∞	0.148	0.148	0.296	0.185	0.222

Figure 3.1: A simple example of a small web of html-pages with a table of the recursive process for calculating the PageRank for these pages.

In figure 3.1 a small web $\mathbb{S} = \{A, B, C, D, E\}$ is given. According to Google's definition, see section 2.4, the ranking of each page now depends on the number of inlinks and outlinks. This recursive definition suggests a recursive process to obtain the PageRank of the pages A,B,C,D and E. According to the definition, a page contributes equally to the PageRank of all the pages it has outlinks to. This leads to the following recursive process:

1. Start with a uniformly distributed ranking of all pages.

2. In each step, distribute the current rank of each page equally to all pages the page has outlinks to.

The first step in the application of this process is taking the PageRank of all pages equal to $\frac{1}{5}$. In the second step we distribute the rank of each page equally to all pages the page has outlinks to. This means, for example, that page A contributes 0.2 to pages B,C,D and E and page B contributes its total ranking to page C (see figure 3.1). Executing this for all pages completes the second step (see the table in figure 3.1).

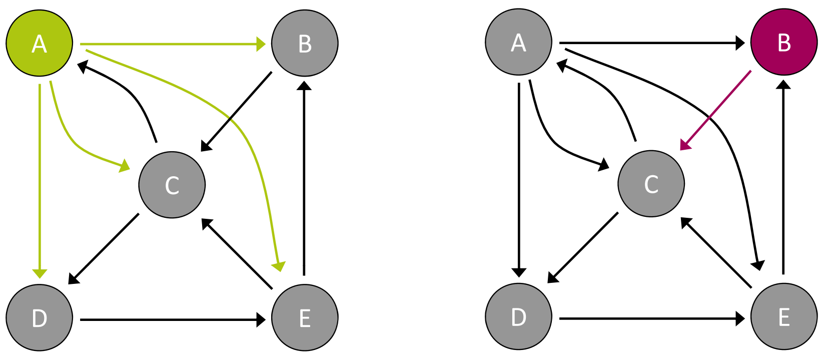


Figure 3.2: Contribution of the PageRank of pages A and B to the other pages.

Repeating the process several times gives a stationary distribution, see the table in figure 3.1. This distribution gives the PageRank of each pages.

The model given in this section is somewhat loose: we have ignored several characteristics of the World Wide Web so far. In the next sections we will define a more appropriate model by approaching web surfing as a Markov chain.

3.2 Modeling web surfing as a Markov chain

In the model we will describe in this section, we focus on the behaviour of a surfer on the World Wide Web. We assume that this surfer will start at a certain web page of his preference and surf the web by clicking on hyperlinks at the web page he is currently visiting.
In the previous section we saw that each page in a web graph contributes its rank equally to all pages it has outlinks to. We will approach web surfing as a Markov chain. To this end we assume that the probability that a surfer will move from page i to another page by clicking on a hyperlink is uniformly distributed. Figure 3.3 illustrates this: each edge in the simple web graph of figure 3.1 denotes the probability of surfing from the page it directs from to the page it directs to.

The probability matrix belonging to this graph is

$$S = \begin{pmatrix} 0 & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{4} & 1 & 0 & 0 & \frac{1}{2} \\ \frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.1}$$

The first column of S denotes the probability distribution for moving from page A to the other pages, the second column denotes the probability distribution for moving from page B to the other pages, etcetera. Notice that each column of the matrix is stochastic, i.e. each element of the column lies between 0 and 1 and the column sums up to one.

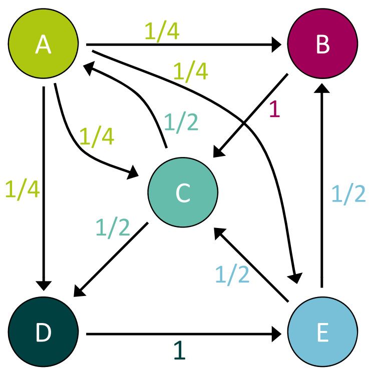


Figure 3.3: The small web of html-pages.

In general, matrix $S^{N\times N}$ can be defined for a web $\mathbb{W} = \{1, 2, 3, \ldots, N\}$ consisting of $N \in \mathbb{N}_{>0}$ pages as follows:

$$S(i,j) = \begin{cases} \frac{1}{l_i} & \text{if there is an outlink from } i \text{ to } j \\ 0 & \text{if there is no outlink from } i \text{ to } j \end{cases} \tag{3.2}$$

where $l_i \in \mathbb{N}_{>0}$ denotes the number of outlinks on page $i \in \mathbb{W}$.

The surfer will start at the web page of his preference and surf the web by clicking on hyperlinks at the web page he is currently visiting. The probability vector $\boldsymbol{\pi}_k \in [0,1]^N$, $k \in \mathbb{N}_{\geq 0}$ denoting the probability that a surfer is currently visiting a website can be found as follows:

$$\boldsymbol{\pi}_k = S\boldsymbol{\pi}_{k-1} \tag{3.3}$$

Now consider a *general preference probability vector* $\boldsymbol{p} \in [0,1]^N$, with only non-negative components adding up to one, representing the general preference of web surfers for all web pages in $\mathbb{W}$. Then the surfer starts at

$$\boldsymbol{\pi}_0 = \boldsymbol{p}. \tag{3.4}$$

We thus have defined a discrete Markov-chain describing the behaviour of a web surfer at the WWW. The limit state of this Markov chain can be interpreted as the relative frequency with which the surfer will visit pages and denotes the PageRank vector $\boldsymbol{\pi}$.

$$\boldsymbol{\pi} = \lim_{k\to\infty} \boldsymbol{\pi}_k = \lim_{k\to\infty} S^k \boldsymbol{p}. \tag{3.5}$$

In section 3.4 we will see that this steady state vector $\boldsymbol{\pi}$ indeed exists and satisfies the relation $\boldsymbol{\pi} = S\boldsymbol{\pi}$.

3.3 The PageRank model

The model described in the last section, does not deal yet with some complexities that can occur by surfing the World Wide Web. These will be added to the model in this section. After that, we present the complete PageRank model.

3.3.1 Dangling pages

Recall the example web graph of section 3.1. All the pages in that graph direct to at least one other page. Suppose now there is a page without outlinks to other pages (see figure 3.4). Pages

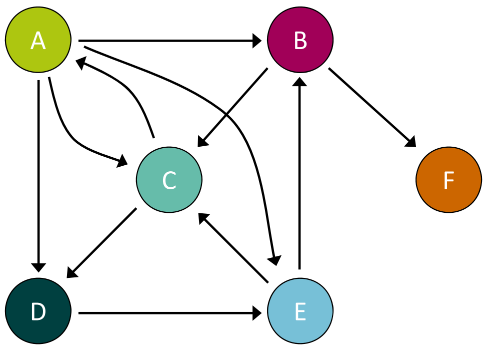


Figure 3.4: Example of a dangling page.

with no outlinks are called *dangling nodes*. These pages are not rare: many html-pages direct to files, such as pdf-files, log files etcetera. These files are all dangling pages. If a surfer visits a dangling page, we suppose that he starts surfing again by moving to a certain arbitrary page of his preference.
The matrix corresponding to the example in figure 3.4 is

$$S = \begin{pmatrix} 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ \frac{1}{4} & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\ \frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.6}$$

We now replace the *dangling column* containing only entries equal to zero, by the general preference vector $\boldsymbol{p}$. In general the dangling columns can be replaced by $\boldsymbol{p}$ by adding to S a matrix containing the desired dangling columns: $S + \boldsymbol{p}\boldsymbol{d}^T$, $\boldsymbol{d} \in \{0,1\}^N$ with

$$\boldsymbol{d}(i) = \begin{cases} 1 & \text{if the } i\text{-th column of } S \text{ is a dangling column} \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

3.3.2 Teleportation

Our model assumes that a surfer can move only from one page to another by clicking on one of the hyperlinks of the web page he is currently visiting. This seems not to be realistic: the surfer can also go to a certain webpage by entering the address of that page in the address bar of the browser. This is called *teleportation*. Neglecting teleportation can confuse the page ranking. The example in figure 3.5 explains this: a surfer can never reach pages F to K if he is currently at page A, B, C, D or E. As a result of that, the PageRank of pages A to E have no relation with the PageRank of pages G to K.
We add teleportation to the model by supposing that a surfer moves to a page by clicking with probability α (with $0 < \alpha < 1$) and by teleportation with probability $1 - \alpha$. In the case of teleportation, the probability distribution for moving to a page is that of the general preference vector $\boldsymbol{p}$ (see [6, *p. 350*]). Our model can now be described as a convex combination of the original matrix $S + \boldsymbol{p}\boldsymbol{d}^T$ and the teleportation matrix $\boldsymbol{p}\boldsymbol{1}^T$, existing only of columns equal to $\boldsymbol{p}$, as follows:

$$G = \alpha\left(S + \boldsymbol{p}\boldsymbol{d}^T\right) + (1-\alpha)\,\boldsymbol{p}\boldsymbol{1}^T. \tag{3.8}$$

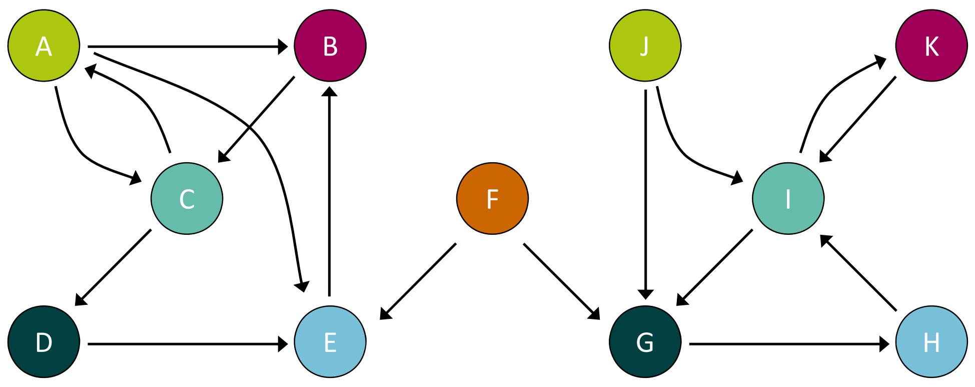


Figure 3.5: Example of a web graph containing pages that cannot be reached.

3.3.3 The Google matrix

We now have completed our model that describes web surfing as a homogeneous discrete Markov chain with transition matrix $G^{N \times N}$ and state vectors $\{\boldsymbol{\pi}_k \in [0,1]^N, k \in \mathbb{N}_{\geq 0}\}$ which satisfy the relation

$$\boldsymbol{\pi}_{k+1} = G\boldsymbol{\pi}_k. \tag{3.9}$$

Given the state vector $\boldsymbol{\pi}_0 = \boldsymbol{p}$ (c.f. (3.4)), denoting the probability distribution for the surfer's initial location, the state vector at time i will be

$$\boldsymbol{\pi}_k = G^k \boldsymbol{\pi}_0. \tag{3.10}$$

The limit state of the Markov chain is the PageRank vector $\boldsymbol{\pi}$ satisfying

$$\boldsymbol{\pi} = \lim_{k\to\infty} \boldsymbol{\pi}_k = \lim_{k\to\infty} G^k \boldsymbol{p}. \tag{3.11}$$

The matrix G of (3.8) is called *Google matrix* [11]. For notational reasons we will write the Google matrix as follows:

$$G = \alpha H + (1-\alpha)\, T \tag{3.12}$$

where $H = S + \boldsymbol{p}\boldsymbol{d}^T$ denotes moving to a page via a hyperlink and $T = \boldsymbol{p}\boldsymbol{1}^T$ denotes moving to a page via teleportation.

A choice for $\alpha = 0.85$ is originally made by Google (see [2]), the value which we will use henceforth. An explanation of this choice is given in section 4.2. In our model we assume that the general preference vector is uniformly distributed, i.e.

$$\boldsymbol{p}(i) = \frac{1}{N} \;\forall i \in 1, \ldots, N. \tag{3.13}$$

Other choices are possible, but only under the conditions

$$\boldsymbol{p}(i) \in (0,1)^N \;\forall i \in 1, \ldots, N, \quad \|\boldsymbol{p}\|_1 = 1. \tag{3.14}$$

We need these conditions to guarantee important properties of the Google matrix, as we will see in the next section.

3.4 Existence and uniqueness of the PageRank vector

The Google matrix has some important particular properties which are mentioned in the next two lemma's.

Lemma 3.4.1. *G is a column stochastic[1] Matrix.*

Proof. Recall G of (3.12). Let us first consider matrix $H = S + \boldsymbol{p}\boldsymbol{d}^T$. We have seen in section 3.2 and 3.3.1 that, according to its definition (3.2), S is a non-negative matrix of which each column sums up to one, except for the dangling columns. To avoid these dangling columns, we added $\boldsymbol{p}\boldsymbol{d}^T$ to matrix S, with $\boldsymbol{p}$ being stochastic, see (3.14). So, H is column stochastic. Furthermore, the matrix $\boldsymbol{p}\boldsymbol{1}^T$ exists only of columns equal to $\boldsymbol{p}$, so T is column stochastic too. Finally, scaling the matrices H and T by α and $1-\alpha$, with $0 < \alpha < 1$, guarantees us that every column of G is stochastic. We conclude that G is column stochastic. □

Lemma 3.4.2. *Each entry of G is strictly positive and smaller than one.*

Proof. From lemma 3.4.1 we know that $\forall i, j \in [1, n]\ 0 \leq H(i,j) \leq 1$. Because of (3.14), we also know that $\forall i, j\ 0 < T(i,j) < 1$. Then, since $0 < \alpha < 1$ (see section 3.3.2), $\forall i, j\ 0 < G(i,j) = \alpha H(i,j) + (1-\alpha)T(i,j) < 1$ holds. □

We are now in the position to prove the existence and uniqueness of the PageRank vector. In the proof we use the spectral radius of G. The spectral radius of a square matrix is defined as the maximum of the magnitudes of all eigenvalues of that matrix, i.e. $\rho(A) = \max\{|\lambda| : \lambda \in \lambda(A)\}$ [7, *p. 511*].

Theorem 3.4.3. *The PageRank vector $\boldsymbol{\pi}$ is the unique stationary distribution of a Markov chain with transition matrix G and state vectors $\{\boldsymbol{\pi}_k, k \in \mathbb{N}_{\geq 0}\}$, i.e. $\boldsymbol{\pi}$ is the eigenvector corresponding to the simple dominant eigenvalue 1 of G.*

Proof. The largest eigenvalue of a row stochastic matrix is one [1, *p. 49*], so $\rho(G^T) = 1$. But since the eigenvalues of a matrix and its transpose are equal, $\rho(G) = 1$. But then $G\boldsymbol{\pi} = \boldsymbol{\pi}$ holds: $\boldsymbol{\pi}$ is the eigenvector corresponding to the eigenvalue 1 of G and $\boldsymbol{\pi}$ is a stationary distribution of G. Now the Perron-Frobenius theorem [1, *p. 27*] ensures us that since G is a strictly positive matrix (lemma 3.4.2), $\rho(G) = 1$ is a simple eigenvalue of G. This means that the (algebraic) multiplicity of this eigenvalue is equal to 1 [7, *p. 316*] and therefore $\boldsymbol{\pi}$ is the unique eigenvector of G corresponding to the dominant eigenvalue 1 and consequently the unique stationary distribution of G. □

The sensitivity of the PageRank vector to changes in matrix S, damping factor α, general preference vector $\boldsymbol{p}$ and to adding inlinks and outlinks to web pages is discussed in [11].

[1] A real square matrix is column stochastic if all its columns are stochastic and row stochastic if all its rows are stochastic.

Chapter 4

Computing the PageRank vector: an eigenvalue problem

We have seen in section 3.4 that the PageRank vector $\boldsymbol{p}$ is the eigenvector of G corresponding to eigenvalue 1. Therefore, finding this PageRank vector can be regarded as an eigenvalue problem. A natural choice for computing the eigenvector of G is the Power method, because moving to a next state in the Markov chain, $\boldsymbol{\pi}_{k+1} = G\boldsymbol{\pi}_k$ (3.9), is equal to the Power recursion. This recursion was used by Sergey Brin and Larry Page to develop the PageRank algorithm for their search engine Google [15, *pp. 6-8*].

4.1 Power method

The Power method produces a sequence of vectors that converge to the eigenvector corresponding to the dominant eigenvalue, under the following assumptions [7]:

- The square matrix concerned has a dominant eigenvalue.
- The initial vector has a non-zero component in the direction of the eigenvector corresponding to the dominant eigenvalue.

The Google matrix has a dominant eigenvalue equal to one (see theorem 3.4.3). The state vector $\boldsymbol{\pi}_0 \in [0,1]^N$ denoting the probability distribution for the surfer's initial location should meet the second assumption. A first algorithm of the Power method for computing the PageRank vector is given below.

```
POWER METHOD
   Input:  $G \in (0,1)^{N\times N}$; $\boldsymbol{\pi}_0 \in [0,1]^N$; $\alpha \in (0,1)$; tolerance $\varepsilon \in (0,1)$.
  Output:  $\boldsymbol{\pi}^*$, approximation of PageRank vector such that $\|\boldsymbol{\pi}^* - G\boldsymbol{\pi}^*\| \leq \varepsilon$.
       1:  $\boldsymbol{\pi}^* \leftarrow \boldsymbol{\pi}_0$
       2:  while $\|\boldsymbol{\pi}^* - G\boldsymbol{\pi}^*\| > \varepsilon$
       3:      $\boldsymbol{\pi}^* \leftarrow G\boldsymbol{\pi}^* / \|G\boldsymbol{\pi}^*\|$
       4:  end while
```

We can make several improvements to the algorithm for this particular problem. The first concerns the matrix multiplication. The Power method works efficiently for (large) sparse matrices [7, *p. 332*]. Because matrix G has not a single entry equal to zero, the given algorithm will not be very fast. Besides that, there is an important reason why the implementation of the current algorithm is not desirable: storing matrix G requires a lot of memory space, because N^2 elements has to be stored. A sparser matrix would require less memory space. So, we have to avoid multiplications with matrix G and use a sparser matrix instead. To that end we take a closer look to the multiplication $G\boldsymbol{\pi}^*$:

$$G\boldsymbol{\pi}^* = \alpha S\boldsymbol{\pi}^* + \alpha \boldsymbol{p}\boldsymbol{d}^T\boldsymbol{\pi}^* + (1-\alpha)\,\boldsymbol{p}\mathbf{1}^T\boldsymbol{\pi}^* \tag{4.1}$$

Matrix S, defined in section 3.2, is very sparse: S may contain a lot of dangling pages, and, more important, a single web page cites only a very small part of the total web graph. Therefore, the matrix multiplication $S\boldsymbol{\pi}^*$ suits the Power method very well. The inner product $\boldsymbol{d}^T\boldsymbol{\pi}^*$ (to prevent dangling pages) is a simple calculation. Furthermore, note that $\mathbf{1}^T\boldsymbol{\pi}^* = 1$, because $\boldsymbol{\pi}^*$ sums up to one. Applying this yields

$$G\boldsymbol{\pi}^* = \alpha S\boldsymbol{\pi}^* + \left(\alpha\boldsymbol{d}^T\boldsymbol{\pi}^* + 1 - \alpha\right)\boldsymbol{p} \tag{4.2}$$

The initial vector should not have a non-zero element in the direction of eigenvector corresponding to the dominant eigenvalue. If we choose as initial vector the general preference vector, defined as $\boldsymbol{p}(i) = \frac{1}{n}\ \forall i \in 1,\ldots,n$ in section 3.4, we cannot guarantee that this condition is fulfilled. It is however plausible to assume that this assumption will be met in practice [7, *pp. 331-332*], [16, *p. 312*].

The normalisation in step 3 of the algorithm is meant to prevent $\boldsymbol{\pi}^*$ from growing too large [16, *p. 308*]. But because the initial vector as well as matrix G are stochastic, $\boldsymbol{\pi}^*$ will remain stochastic too. Normalisation is therefore unnecessary. The improved algorithm is given below.

```
POWER METHOD: IMPROVED ALGORITHM FOR THE PAGERANK PROBLEM
 Input:  $S \in [0,1]^{N\times N}$; $\boldsymbol{p} \in (0,1)^N$; $\boldsymbol{d} \in \{0,1\}^N$; $\alpha \in (0,1)$; tolerance $\varepsilon \in (0,1)$.
Output:  $\boldsymbol{\pi}^*$, approximation of PageRank vector such that $\|\boldsymbol{\pi}^* - G\boldsymbol{\pi}^*\| \le \varepsilon$.
     1:  $\boldsymbol{\pi}_c^* \leftarrow \boldsymbol{p}$; $\boldsymbol{\pi}_p^* \leftarrow \mathbf{1}N$
     2:  while $\left\|\boldsymbol{\pi}_c^* - \boldsymbol{\pi}_p^*\right\| > \varepsilon$
     3:      $\boldsymbol{\pi}_p^* \leftarrow \boldsymbol{\pi}_c^*$
     4:      $\boldsymbol{\pi}_c^* \leftarrow \alpha S\boldsymbol{\pi}_p^* + \left(\alpha\boldsymbol{d}^T\boldsymbol{\pi}_p^* + 1 - \alpha\right)\boldsymbol{p}$
     5:  end while
     6:  $\boldsymbol{\pi}^* \leftarrow \boldsymbol{\pi}_c^*$
```

Note that the while loop condition $\|\boldsymbol{\pi}^* - G\boldsymbol{\pi}^*\| > \varepsilon$ in the second algorithm is replaced by the equivalent condition $\left\|\boldsymbol{\pi}_c^* - \boldsymbol{\pi}_p^*\right\| > \varepsilon$ to avoid multiplication with G. Here, $\boldsymbol{\pi}_c^*$ denotes the *current* value of $\boldsymbol{\pi}^*$ as $\boldsymbol{\pi}_p^*$ denotes the *previous* value of $\boldsymbol{\pi}^*$. In this research project, for the norms in the Power method algorithms and algorithms in next chapters, the 2-norm is chosen. However, for the Power method the use of this norm is put into discussion in [22], where another (efficient) criterion is presented that can guarantee correct ranking.

4.2 Convergence

Recall from section 3.4 that the Google matrix $G = \alpha H + (1-\alpha)\,T$ with $H = S + \boldsymbol{p}\boldsymbol{d}^T$ and $T = \boldsymbol{p}\mathbf{1}^T$ has a unique largest eigenvalue equal to one. Consider the sequence of eigenvalues of G, $|\lambda_1| > |\lambda_2| \ge \ldots \ge |\lambda_N|$. Now, the convergence speed of the power method depends on the rate of convergence $|\lambda_2(G)|\,/\,|\lambda_1(G)| = |\lambda_2(G)|$ [7, *p. 331*]. The smaller this rate, the faster the convergence; a smaller second eigenvalue therefore gives a faster convergence. The next theorem gives a specific bound to the second eigenvalue and consequently on the minimum convergence speed.

Theorem 4.2.1. *The second eigenvalue of the Google matrix satisfies $|\lambda_2(G)| \leq \alpha$.*

Proof. This proof is based on [10] and [21, *pp. 3-4*]. Consider the second eigenvalue of G with the corresponding eigenvector

$$G\boldsymbol{x}_2 = \lambda_2 \boldsymbol{x}_2. \tag{4.3}$$

If a matrix is row stochastic then $\mathbf{1}$ is an eigenvector of that matrix corresponding to the dominant eigenvalue one [1, *p. 49*]. This implies that $G^T\mathbf{1} = 1 \cdot \mathbf{1}$, or, equivalently

$$\mathbf{1}^T G = \lambda_1 \mathbf{1}^T. \tag{4.4}$$

Now, left multiplication of (4.3) with $\mathbf{1}^T$ yields

$$\mathbf{1}^T G\boldsymbol{x}_2 = \lambda_2 \mathbf{1}^T \boldsymbol{x}_2 \tag{4.5}$$

and right multiplication of (4.4) with $\boldsymbol{x}_2$ yields

$$\mathbf{1}^T G\boldsymbol{x}_2 = \lambda_1 \mathbf{1}^T \boldsymbol{x}_2. \tag{4.6}$$

By subtracting (4.6) from (4.5) we obtain

$$\lambda_1 \mathbf{1}^T \boldsymbol{x}_2 = \lambda_2 \mathbf{1}^T \boldsymbol{x}_2. \tag{4.7}$$

Since $\lambda_1 = 1$ is the simple dominant eigenvalue according to theorem 3.4.3, $|\lambda_2| < \lambda_1$ and therefore $\lambda_2 \neq \lambda_1$. This together with (4.7) implies

$$\mathbf{1}^T \boldsymbol{x}_2 = 0. \tag{4.8}$$

Now, recalling (3.12), we write (4.3) as

$$\alpha H\boldsymbol{x}_2 + (1-\alpha)\, T\boldsymbol{x}_2 = \lambda_2 \boldsymbol{x}_2. \tag{4.9}$$

Because $T = \boldsymbol{p}\mathbf{1}^T$ (see section 3.3.3 and (4.8)), $T\boldsymbol{x}_2 = \boldsymbol{p}\mathbf{1}^T\boldsymbol{x}_2 = 0$, so (4.9) equals

$$\alpha H\boldsymbol{x}_2 = \lambda_2 \boldsymbol{x}_2 \tag{4.10}$$

and

$$H\boldsymbol{x}_2 = \frac{\lambda_2}{\alpha}\boldsymbol{x}_2. \tag{4.11}$$

This means that λ_2/α is an eigenvalue of H, satisfying $|\lambda_2/\alpha| \leq 1$, since $\rho(H) = 1$ (see lemma 3.4.1 and theorem 3.4.3). We conclude that $|\lambda_2(G)| \leq \alpha$. $\square$

We found the convergence rate of the power method directly related to the value of α. Because $0 < \alpha < 1$ (see section 3.3.2) is not bounded to a specific value, we could choose it to be very small in order to achieve a fast convergence. However, every choice of α directly influences the PageRank vector. Because $T = \boldsymbol{p}\mathbf{1}^T$, a very small value of α would result in a PageRank vector almost equal to $\boldsymbol{p}$, which is not realistic, since we then simply neglect the web graph's link structure. Further, remember the reason of adding T to G (section 3.3.2): it denotes the probability that a surfer moves to a page by teleportation. This probability cannot be large. In general, α is chosen equal to 0.85 [15]. Experimental comparison for different values of α is done in [5].

Chapter 5

A linear systems approach

5.1 Purpose of rewriting the model

The Power method is useful to approximate the Pagerank vector, as we have seen in the last chapter. However, there are more and even faster ways to calculate $\boldsymbol{\pi}$. One way that opens up several possibilities, is writing our model as a linear system. This allows us to use iterative methods, such as Jacobi and Gauss-Seidel to approximate $\boldsymbol{\pi}$.

We obtain a linear system of our model as follows [3, *pp. 254-255*]: recall from theorem 3.4.3 the relation

$$\boldsymbol{\pi} = G\boldsymbol{\pi} \tag{5.1}$$

with $G = \alpha\left(S + \boldsymbol{p}\boldsymbol{d}^T\right) + (1-\alpha)\,\boldsymbol{p}\boldsymbol{1}^T$. Now, as we saw in section 4.1, we can simplify (5.1) to

$$\boldsymbol{\pi} = \alpha\left(S + \boldsymbol{p}\boldsymbol{d}^T\right)\boldsymbol{\pi} + (1-\alpha)\,\boldsymbol{p} \tag{5.2}$$

because $\boldsymbol{1}^T\boldsymbol{\pi} = 1$. We use this to rewrite (5.2) as a linear system:

$$\left(I - \alpha\left(S + \boldsymbol{p}\boldsymbol{d}^T\right)\right)\boldsymbol{\pi} = (1-\alpha)\,\boldsymbol{p}. \tag{5.3}$$

For reasons of notation, we will write (5.3) as

$$A\boldsymbol{\pi} = (1-\alpha)\,\boldsymbol{p} \tag{5.4}$$

with

$$A = (I - \alpha H)\,, \quad H = S + \boldsymbol{p}\boldsymbol{d}^T. \tag{5.5}$$

5.2 Basic iterative methods

This section is partly based on [7, *pp. 509-511*]. Iterative methods generate a sequence of approximate solutions $\{\boldsymbol{x}_k\}$ of a linear system $A\boldsymbol{x} = \boldsymbol{b}$. The basic iterative methods are based on a certain splitting of matrix A in a matrix M for which the system $M\boldsymbol{y} = \boldsymbol{c}$ is 'easy' to solve and the remaining matrix

$$R = M - A, \tag{5.6}$$

which changes the linear system into

$$M\boldsymbol{x} = R\boldsymbol{x} + \boldsymbol{b}. \tag{5.7}$$

The corresponding iteration is

$$M\boldsymbol{x}_{k+1} = R\boldsymbol{x}_k + \boldsymbol{b} \quad k \in \mathbb{N}_{\geq 0} \tag{5.8}$$

5.2.1 Convergence

The error in the kth iterate is $\boldsymbol{e}_k = \boldsymbol{x}_k - \boldsymbol{x}$. Because of (5.8) $M\boldsymbol{e}_{k+1} = R\boldsymbol{e}_k$ and so the error in $\boldsymbol{x}_{k+1}$ is

$$\boldsymbol{e}_{k+1} = M^{-1}R\boldsymbol{e}_k = \left(M^{-1}R\right)^{k+1}\boldsymbol{e}_0 \quad k \in \mathbb{N}_{\geq 0}. \tag{5.9}$$

The iteration converges to $\boldsymbol{x} = A^{-1}\boldsymbol{b}$ for nonsingular matrices A and M and for any initial vector $\boldsymbol{x}_0$ if $\lim_{k\to\infty}\left(M^{-1}R\right)^k \to 0$, which is the case if $\rho\left(M^{-1}R\right) < 1$, see [7, *pp. 336, 511*]. Further note that for a matrix $Q^{N\times N}$ (see [20, *p. 13*])

$$\lim_{k\to\infty} Q^k = 0 \quad \text{if and only if} \quad \rho(Q) < 1. \tag{5.10}$$

In this section we will use the fact that several of the matrices under consideration are nonnegative, i.e. none of the entries of these matrices are negative. To this end we have to define the following first ([1, *p. 26*]):

$$\begin{aligned} Q \geq R \quad &\text{if} \quad \forall i,j \quad Q(i,j) \geq R(i,j), \\ Q > R \quad &\text{if} \quad Q \geq R \quad \text{and} \quad Q \neq R, \\ &O(i,j) = 0 \quad \forall i,j. \end{aligned} \tag{5.11}$$

Theorem 5.2.1. *$A = (I - \alpha H)$ is a nonsingular matrix and $A^{-1} > O$.*

Proof. Since H is a column stochastic matrix and $\rho(H) = 1$ (see lemma 3.4.1 and theorem 3.4.3) and since $0 < \alpha < 1$ (see section 3.3.2), $\rho(\alpha H) < 1$ and, consequently, $\lim_{k\to\infty}(\alpha H)^k = 0$. Under this condition is $I - \alpha H$ nonsingular [19, *p. 55*].

The following part of this proof is based on [1, *p. 133*]. Consider the identity

$$(I - \alpha H)\left(I + \alpha H + \ldots + (\alpha H)^r\right) = I - (\alpha H)^{r+1} \quad r \geq 0. \tag{5.12}$$

Now, if we let r approach infinity, the left hand side of (5.12) changes to

$$\lim_{r\to\infty}(1 - \alpha H)\sum_{s=0}^{r}(\alpha H)^s = (I - \alpha H)\sum_{r=0}^{\infty}(\alpha H)^r \tag{5.13}$$

and because $\rho(\alpha H) < 1$, according to (5.10) the right hand side of (5.12) changes to

$$\lim_{r\to\infty} I - (\alpha H)^{r+1} = I. \tag{5.14}$$

Finally, (5.13) and (5.14) yields

$$(I - \alpha H)\sum_{r=0}^{\infty}(\alpha H)^r = I \tag{5.15}$$

and since $H > 0$ (H is column stochastic) and $\alpha > 0$,

$$A^{-1} = (I - \alpha H)^{-1} = \sum_{r=0}^{\infty}(\alpha H)^r > O. \tag{5.16}$$

□

Theorem 5.2.2. *If M^{-1} exists with $M^{-1} \geq 0$ and if $R \geq 0$ and $A^{-1} \geq O$ then iteration (5.8) converges for any initial vector $\boldsymbol{x}_0$.*

Proof. The splitting $A = M - R$ with M nonsingular, $M^{-1} \geq 0$ and $R \geq 0$ is called a *regular splitting* of matrix A [20, *p. 88*]. If $A = M - R$ is a regular splitting and $A^{-1} \geq O$ then $\rho\left(M^{-1}R\right) < 1$ and the iteration (5.8) converges for any initial vector $\boldsymbol{x}_0$ [20, *p. 89*]. We already know that $A^{-1} \geq O$ (c.f. theorem 5.2.1), so in our case it is sufficient to prove that M is nonsingular with $M^{-1} \geq 0$ and $R \geq 0$. □

Another property of matrix A concerns its main diagonal. Define

$$r_i(Q) = \sum_{\substack{j=1 \\ j \neq i}}^{N} |Q(i,j)|, \tag{5.17}$$

the sum of all elements of the i-th row of $N \times N$ matrix Q except for the element that is part of the main diagonal of Q. Then, Q is *strictly diagonally dominant* [7, *p. 120*] if

$$|Q(i,i)| > r_i(Q) \quad \forall i \in \{1, \ldots, N\}. \tag{5.18}$$

Theorem 5.2.3. *Matrix $A^T = I - \alpha H^T$ is strictly diagonally dominant.*

Proof. The following holds for all $i \in \{1, \ldots, N\}$:
We have to prove that $|A(i,i)| = \left|A^T(i,i)\right| > r_i(A^T)$. Because H is column stochastic (see lemma 3.4.1), we know that

$$0 \leq r_i(H^T) \leq 1. \tag{5.19}$$

Notice that

$$r_i(A^T) = r_i(I - \alpha H^T) = \alpha r_i(H^T). \tag{5.20}$$

Now (5.19) and (5.20) yield

$$0 \leq r_i(A^T) \leq \alpha. \tag{5.21}$$

Furthermore,

$$A(i,i) = 1 - \alpha H(i,i) \tag{5.22}$$

and since H is column stochastic

$$H(i,i) = 1 - r_i(H^T). \tag{5.23}$$

Now, substituting (5.23) into (5.22) yields

$$A(i,i) = 1 - \alpha + \alpha r_i(H^T) \tag{5.24}$$

and substituting (5.20) into (5.24) yields

$$A(i,i) = 1 - \alpha + r_i(A^T). \tag{5.25}$$

Since $0 < \alpha < 1$ (see section 3.3.2), (5.25) implies that

$$A(i,i) - r_i(A^T) = 1 - \alpha > 0 \tag{5.26}$$

and we conclude that

$$A(i,i) > r_i(A^T). \tag{5.27}$$

□

5.3 Jacobi iteration

The Jacobi iteration is a splitting of A with M a diagonal matrix with the main diagonal of A: $M_J = \operatorname{diag}(A)$, which means

$$M_J(i,j) = \operatorname{diag}(A)(i,j) \begin{cases} A(i,i) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{5.28}$$

An algorithm with Jacobi iteration for the system (5.4) is given below.

Jacobi method

```
 Input:  $H \in [0,1]^{N\times N}$; $\boldsymbol{\pi}_0 \in \mathbb{R}^N$; $\alpha \in (0,1)$; $\boldsymbol{p} \in (0,1)^N$; tolerance $\varepsilon \in (0,1)$.
Output:  $\boldsymbol{\pi}^*$, approximation of PageRank vector such that $\|\boldsymbol{b} - A\boldsymbol{\pi}^*\| \leq \varepsilon$.
     1:  $A \leftarrow I - \alpha H$
     2:  $M_J \leftarrow \operatorname{diag}(A)$
     3:  $R_J \leftarrow M_J - A$
     4:  $\boldsymbol{\pi}^* \leftarrow \boldsymbol{\pi}_0$
     5:  $\boldsymbol{b} \leftarrow (1-\alpha)\boldsymbol{p}$
     6:  while $\|\boldsymbol{b} - A\boldsymbol{\pi}^*\| > \varepsilon$
     7:      $\boldsymbol{\pi}^* \leftarrow M_J^{-1}(R_J\boldsymbol{\pi}^* + \boldsymbol{b})$
     8:  end while
```

5.3.1 Convergence

In order to show that $\boldsymbol{\pi}^*$ of the given Jacobi method algorithm converges to the PageRank vector $\boldsymbol{\pi} = A^{-1}\boldsymbol{b}$, recall from theorem 5.2.2 that we have to prove that M_J is nonsingular, $M_J^{-1} \geq 0$ and $R_J \geq 0$. Note that since $R_J = \operatorname{diag}(A) - A = \operatorname{diag}(I - \alpha H) - I + \alpha H = \alpha H - \operatorname{diag}(\alpha H)$ and $H > 0$ (H is column stochastic, see lemma 3.4.1 and theorem 3.4.3) it follows that $R_J \geq 0$. The other conditions are proved below.

Theorem 5.3.1. $M_J = \operatorname{diag}(A)$ *is a nonsingular matrix and* $M_J^{-1} \geq O$.

Proof. Since M_J is a diagonal matrix, M_J is nonsingular if its main diagonal is non-zero.[cite] Because H is a column stochastic matrix, all elements of its main diagonal are not larger than 1, and so all elements of $\operatorname{diag}(\alpha H)$ are smaller than 1. But then $M_J = \operatorname{diag}(A) = \operatorname{diag}(I - \alpha H)$ (see 5.4) has a non-zero main diagonal.
Now $M_J^{-1}(i,i) = 1/M_J(i,i) = 1/A(i,i) \quad \forall i \in \{1,\dots,N\}$. Further, (5.21) and (5.27) yield $A(i,i) \geq 0\ \forall i$ and we conclude that $M_J^{-1}(i,i) \geq 0\ \forall i \in \{1,\dots,N\}$. □

5.3.2 Improvements

For the same reason as mentioned in section 4.1, we will make several improvements to the algorithm of section 5.3. To that end, let us take a closer look to the matrices M_J and R_J:

$$M_J = \operatorname{diag}(A) = I - \operatorname{diag}(\alpha S) - \operatorname{diag}(\alpha \boldsymbol{p}\boldsymbol{d}^T), \tag{5.29}$$

see (5.5). For reasons of implementation, it is useful to calculate diag(αS) and diag($\alpha \boldsymbol{p}\boldsymbol{d}^T$) separately. Now, recalling (5.5) and (5.6),

$$\begin{aligned} R_J = M_J - A &= I - \operatorname{diag}(\alpha S) - \operatorname{diag}(\alpha \boldsymbol{p}\boldsymbol{d}^T) - I + \alpha S + \alpha \boldsymbol{p}\boldsymbol{d}^T \\ &= \alpha S - \operatorname{diag}(\alpha S) + \alpha \boldsymbol{p}\boldsymbol{d}^T - \operatorname{diag}(\alpha \boldsymbol{p}\boldsymbol{d}^T) \\ &= R'_J + \alpha \boldsymbol{p}\boldsymbol{d}^T \end{aligned} \tag{5.30}$$

with

$$R'_J = \alpha S - \operatorname{diag}(\alpha S) - \operatorname{diag}(\alpha \boldsymbol{p}\boldsymbol{d}^T). \tag{5.31}$$

Rewriting (5.8) with (5.29) and (5.30) then yields

$$\pi_{k+1} = M_J^{-1}\left(R'_J \boldsymbol{\pi}_k + \alpha \boldsymbol{p}\boldsymbol{d}^T \boldsymbol{\pi}_k + \boldsymbol{b}\right). \tag{5.32}$$

Like in section 4.1, we choose $\boldsymbol{p}$ as initial vector. Finally, recalling (5.5), we rewrite the while loop condition to

$$\|\boldsymbol{b} - A\boldsymbol{\pi}^*\| = \left\|\alpha S\boldsymbol{\pi}^* + \alpha \boldsymbol{p}\boldsymbol{d}^T \boldsymbol{\pi}^* + \boldsymbol{b} - \boldsymbol{\pi}^*\right\|. \tag{5.33}$$

The improved algorithm is given below.

Jacobi method: improved algorithm for the PageRank problem

```
   Input:  $S \in [0,1]^{N\times N}$; $\boldsymbol{p} \in (0,1)^N$; $\boldsymbol{d} \in \{0,1\}^N$; $\alpha \in (0,1)$; tolerance $\varepsilon \in (0,1)$.
  Output:  $\boldsymbol{\pi}^*$, approximation of PageRank vector such that $\|\boldsymbol{b} - A\boldsymbol{\pi}^*\| \le \varepsilon$.
      2:   $M_J \leftarrow I - \operatorname{diag}(\alpha S) - \operatorname{diag}(\alpha \boldsymbol{p}\boldsymbol{d}^T)$
      2:   $R_J \leftarrow \alpha S - \operatorname{diag}(\alpha S) - \operatorname{diag}(\alpha \boldsymbol{p}\boldsymbol{d}^T)$
      4:   $\boldsymbol{\pi}^* \leftarrow \boldsymbol{p}$
      5:   $\boldsymbol{b} \leftarrow (1-\alpha)\boldsymbol{p}$
      6:   while $\left\|\alpha S\boldsymbol{\pi}^* + \alpha \boldsymbol{p}\boldsymbol{d}^T \boldsymbol{\pi}^* + \boldsymbol{b} - \boldsymbol{\pi}^*\right\| > \varepsilon$
      7:       $\boldsymbol{\pi}^* \leftarrow M_J^{-1}\left(R'_J \boldsymbol{\pi}^* + \alpha \boldsymbol{p}\boldsymbol{d}^T \boldsymbol{\pi}^* + \boldsymbol{b}\right)$
      8:   end while
```

5.4 Gauss-Seidel iteration

The Gauss-Seidel iteration is based on a splitting of A in a lower and upper triangular part:

$$M_G = \operatorname{diag}(A) + \operatorname{lower}(A), \tag{5.34}$$

where lower(A) is defined as

$$\operatorname{lower}(A)(i,j) = \begin{cases} A(i,j) & \text{if } i > j \\ 0 & \text{otherwise.} \end{cases} \tag{5.35}$$

upper(A) can be defined analogously:

$$\operatorname{upper}(A)(i,j) = \begin{cases} A(i,j) & \text{if } i < j \\ 0 & \text{otherwise.} \end{cases} \tag{5.36}$$

Now,

$$R_G = M_G - A = \operatorname{diag}(A) + \operatorname{lower}(A) - A = -\operatorname{upper}(A). \tag{5.37}$$

An algorithm with Gauss-Seidel iteration for the system (5.4) is given below.

```
GAUSS-SEIDEL METHOD
 Input:  $H \in [0,1]^{N\times N}$; $\boldsymbol{\pi}_0 \in \mathbb{R}^N$; $\alpha \in (0,1)$; $\boldsymbol{p} \in (0,1)^N$; tolerance $\varepsilon \in (0,1)$.
Output:  $\boldsymbol{\pi}^*$, approximation of PageRank vector such that $\|\boldsymbol{b} - A\boldsymbol{\pi}^*\| \leq \varepsilon$.
     1:  $A \leftarrow I - \alpha H$
     2:  $M_G \leftarrow \text{diag}(A) + \text{lower}(A)$
     3:  $R_G \leftarrow -\text{upper}(A)$
     4:  $\boldsymbol{\pi}^* \leftarrow \boldsymbol{\pi}_0$
     5:  $\boldsymbol{b} \leftarrow (1-\alpha)\boldsymbol{p}$
     6:  while $\|\boldsymbol{b} - A\boldsymbol{\pi}^*\| > \varepsilon$
     7:      $\boldsymbol{\pi}^* \leftarrow M_G^{-1}(R_G\boldsymbol{\pi}^* + \boldsymbol{b})$
     8:  end while
```

5.4.1 Convergence

To show that $\boldsymbol{\pi}^*$ of the given Gauss-Seidel method algorithm converges to the PageRank vector $\boldsymbol{\pi} = A^{-1}\boldsymbol{b}$, we have to prove that M_G is non-singular, $M_G^{-1} \geq 0$ and $R_G \geq 0$ (theorem 5.2.2). Note that since $R_G = -\text{upper}(A) = -\text{upper}(I - \alpha H) = \text{upper}(\alpha H)$ and $H > 0$ (H is column stochastic, see lemma 3.4.1 and theorem 3.4.3) it follows that $R_G \geq 0$. The other conditions are proved below.

Theorem 5.4.1. $M_G = \text{diag}(A) + \text{lower}(A)$ *is a non-singular matrix and* $M_G^{-1} \geq O$.

Proof. Note that $M_G = \text{diag}(I - \alpha H) + \text{lower}(I - \alpha H) = I - \text{diag}(\alpha H) - \text{lower}(\alpha H)$ (see 5.4). Now, according to theorem 5.3.1, all elements of $\text{diag}(\alpha H)$ are smaller than 1, so the main diagonal of $\text{diag}(M_G) = I - \text{diag}(\alpha H)$ consists only of non-zero elements. But since the eigenvalues of a triangular matrix are the entries on its main diagonal [16, *p. 289*], the number of non-zero eigenvalues of $M_G^{N\times N}$ is equal to N. Under this condition is M_G non-singular [16, *p. 290*].
Now, let $M_G = I - \text{diag}(\alpha H) - \text{lower}(\alpha H) = I - \alpha H_{D+L}$ with $H_{D+L} = \text{diag}(H) + \text{lower}(H)$. The eigenvalues of H_{D+L} are the entries on its main diagonal [16, *p. 289*]. Since H is column stochastic (see lemma 3.4.1), all elements of H are smaller than or equal to 1. Consequently all entries on the main diagonal of αH_{D+L} are smaller than 1 (note that $\alpha < 1$) and we conclude that $\rho(\alpha H_{D+L}) < 1$. Then, the proof that $M_G^{-1} \geq O$ is analogous to the second part of the proof of theorem 5.2.1. This can be seen by replacing H by H_{D+L} and A by M_G in equations (5.12) up to (5.16). □

5.4.2 Improvements: modified Gauss-Seidel method

As with the Jacobi iteration, we want to improve the Gauss-Seidel algorithm. This cannot be achieved in the same way as in section 5.3.2, because R_G cannot easily be separated in a matrix R'_G and $\alpha\boldsymbol{p}\boldsymbol{d}^T$ as in (5.30). As a solution to this problem we introduce a modified Gauss-Seidel iteration, in which we exclude the dangling nodes from the lower triangular part:

$$M_{G_M} = \text{diag}(A') + \text{lower}(A') \tag{5.38}$$

with $A' = I - \alpha S$, which we can simplify to

$$M_{G_M} = I - \text{diag}(\alpha S) - \text{lower}(\alpha S). \tag{5.39}$$

Consequently, the upper triangular part will be (recall (5.5) and (5.6))

$$\begin{aligned} R_{G_M} &= M_{G_M} - A = I - \mathrm{diag}(\alpha S) - \mathrm{lower}(\alpha S) - I + \alpha S + \alpha \boldsymbol{p}\boldsymbol{d}^T \\ &= \mathrm{upper}(\alpha S) + \alpha \boldsymbol{p}\boldsymbol{d}^T = R'_{G_M} + \alpha \boldsymbol{p}\boldsymbol{d}^T \end{aligned} \tag{5.40}$$

with $R'_{G_M} = \mathrm{upper}(\alpha S)$. Rewriting (5.8) with (5.39) and (5.40) now yields

$$\pi_{k+1} = M_{G_M}^{-1} \left(R'_{G_M} \boldsymbol{\pi}_k + \alpha \boldsymbol{p}\boldsymbol{d}^T \boldsymbol{\pi}_k + \boldsymbol{b} \right). \tag{5.41}$$

Finally, $\boldsymbol{p}$ is chosen as initial vector again and as while loop condition we use (5.33). The improved algorithm is given below.

```
Modified Gauss-Seidel method
   Input: $S \in [0,1]^{N\times N}$; $\boldsymbol{p} \in (0,1)^N$; $\boldsymbol{d} \in \{0,1\}^N$; $\alpha \in (0,1)$; tolerance $\varepsilon \in (0,1)$.
  Output: $\boldsymbol{\pi}^*$, approximation of PageRank vector such that $\|\boldsymbol{b} - A\boldsymbol{\pi}^*\| \le \varepsilon$.
      2: $M_{G_M} \leftarrow I - \mathrm{diag}(\alpha S) - \mathrm{lower}(\alpha S)$
      2: $R'_{G_M} \leftarrow \mathrm{upper}(\alpha S)$
      4: $\boldsymbol{\pi}^* \leftarrow \boldsymbol{p}$
      5: $\boldsymbol{b} \leftarrow (1-\alpha)\boldsymbol{p}$
      6: while $\left\|\alpha S\boldsymbol{\pi}^* + \alpha \boldsymbol{p}\boldsymbol{d}^T\boldsymbol{\pi}^* + \boldsymbol{b} - \boldsymbol{\pi}^*\right\| > \varepsilon$
      7:     $\boldsymbol{\pi}^* \leftarrow M_{G_M}^{-1}(R'_{G_M}\boldsymbol{\pi}^* + \alpha \boldsymbol{p}\boldsymbol{d}^T\boldsymbol{\pi}^* + \boldsymbol{b})$
      8: end while
```

5.4.3 Convergence of the modified Gauss-Seidel method

The given modified Gauss-Seidel method converges to the PageRank vector $\boldsymbol{\pi} = A^{-1}\boldsymbol{b}$ if M_{G_M} is non-singular, $M_{G_M}^{-1} \ge 0$ and $R_{G_M} \ge 0$ (theorem 5.2.2). Note that since $R_{G_M} = \mathrm{upper}(\alpha S)$ and $S > 0$ (see 3.2) it follows that $R_{G_M} \ge 0$. The other conditions are proved below.

Theorem 5.4.2. $M_{G_M} = I - \mathrm{diag}(\alpha S) - \mathrm{lower}(\alpha S)$ *is a non-singular matrix and* $M_{G_M}^{-1} \ge O$.

Proof. Note that all elements of $\mathrm{diag}(\alpha S)$ are smaller than 1, because all elements of S are smaller than or equal to 1 (see 3.2) and $\alpha < 1$. The main diagonal of $\mathrm{diag}(M_{G_M}) = I - \mathrm{diag}(\alpha S)$ thus consists only of non-zero elements. But since the eigenvalues of a triangular matrix are the entries on its main diagonal [16, *p. 289*], the number of non-zero eigenvalues of $M_{G_M}^{N\times N}$ is equal to N. Under this condition is M_{G_M} non-singular [16, *p. 290*].
Now, let $M_{G_M} = I - \mathrm{diag}(\alpha S) - \mathrm{lower}(\alpha S) = I - \alpha S_{D+L}$ with $S_{D+L} = \mathrm{diag}(S) + \mathrm{lower}(S)$. The eigenvalues of S_{D+L} are the entries on its main diagonal [16, *p. 289*]. All elements of S are smaller than or equal to 1 (see 3.2). Consequently all entries on the main diagonal of αS_{D+L} are smaller than 1 and we conclude that $\rho(\alpha S_{D+L}) < 1$. Then, the proof that $M_{G_M}^{-1} \ge O$ is analogous to the second part of the proof of theorem 5.2.1. This can be seen by replacing S by S_{D+L} and A by M_{G_M} in equations (5.12) up to (5.16).

□

5.5 IDR(s)

Besides basic iterative methods, there are several other methods for solving linear systems. Among them are the Krylov subspace methods, like Bi-CG, Bi-CGSTAB and GMRES, which are popular because of their fast convergence [17, *chapters 6 and 7*]. In this research project we focus on another, new Krylov method for large nonsymmetric linear systems called IDR(s). These method is based on the induced dimension reduction (IDR) theorem. IDR(s) behaves like an iterative method, but computes the true solution using at most $N + N/s$ matrix-vector multiplications, where the parameter s denotes the codimension of a fixed subspace. For a further explanation of this and a prototype for the implementation of the IDR(s) algorithm, see [18].

5.5.1 Modifications and preconditioning

IDR(s) solves the system $A\pi = \boldsymbol{b}$. But as earlier mentioned in this chapter and for the same reason as mentioned in section 4.1, we will split A as follows:

$$A = A' - \alpha \boldsymbol{p}\boldsymbol{d}^T \tag{5.42}$$

with

$$A' = I - \alpha S. \tag{5.43}$$

Now, to obtain a faster implementation, we let IDR(s) solve $A'\boldsymbol{\pi} = b$, but in each iteration we add $\alpha \boldsymbol{p}\boldsymbol{d}^T\boldsymbol{\pi}^*$ to the calculated approximation $\boldsymbol{\pi}^*$.

Recall from section 5.2 the splitting of the linear system $A\boldsymbol{x} = \boldsymbol{b}$ in

$$M\boldsymbol{x} = R\boldsymbol{x} + \boldsymbol{b}. \tag{5.44}$$

We can write (5.44) as $(M - R)\boldsymbol{x} = \boldsymbol{b}$ and use M^{-1} to obtain

$$AM^{-1}\boldsymbol{y} = \boldsymbol{b} \tag{5.45}$$

with

$$\boldsymbol{y} = M\boldsymbol{x}. \tag{5.46}$$

This procedure is called *preconditioning* and here matrix M is called the *right preconditioner* [12, *p. 159*], [17, *p. 252*]. We can use M_J and M_G as preconditioners for the IDR(s) method.

Chapter 6

Practical aspects

6.1 Comparing algorithms

In the previous sections, we have discussed several algorithms to calculate the PageRank vector. We want to investigate which of these has the fastest converge. What 'fastest' means, is not clear immediately. We could say that the algorithm that converges with the smallest number of iterations is the fastest one. But then we ignore the fact that a single iteration step of different algorithms could take different times. On the other hand, one could compare the times that the execution of the algorithms takes. But these times depend largely on how well the algorithms are implemented. In performing experiments we should consider both the number of iterations and the execution time. We assume that an algorithm is faster than another one if it completes in less iterations and if there is a reasonable match between the difference in execution times and the difference between the numbers of iteration.
Furthermore, a suitable investigation of different algorithms requires that we use the same starting vector, the same value of α, the same tolerance and the same norm. Finally, in each iteration step should the number of matrix-vector multiplications be equal. These aspects will be discussed in the next sections, considering that we will compare the following algorithms:

1. The improved Power method algorithm of page 12.
2. The improved Jacobi method algorithm of page 19.
3. The modified Gauss-Seidel method algorithm of page 21.
4. The modified IDR(s) algorithm, see page 22.
5. The modified IDR(s) algorithm with Jacobi preconditioning.
6. The modified IDR(s) algorithm with Gauss-Seidel preconditioning.

6.1.1 Norms

The improved Power method uses as stopping criterion

$$\left\|\boldsymbol{\pi}_c^* - \boldsymbol{\pi}_p^*\right\| = \left\|\boldsymbol{\pi}^* - G\boldsymbol{\pi}^*\right\|, \tag{6.1}$$

see page page 12. The other algorithms use

$$\left\|\boldsymbol{b} - A\boldsymbol{\pi}^*\right\| = \left\|\alpha S\boldsymbol{\pi}^* + \alpha \boldsymbol{p}\boldsymbol{d}^T\boldsymbol{\pi}^* + \boldsymbol{b} - \boldsymbol{\pi}^*\right\|, \tag{6.2}$$

see (5.33) and section 5.4.2. Now, (6.1) and (3.12) yield

$$\left\|\boldsymbol{\pi}^* - \alpha S\boldsymbol{\pi}^* - \alpha\boldsymbol{p}\boldsymbol{d}^T\boldsymbol{\pi}^* - (1-\alpha)\boldsymbol{p}\mathbf{1}^T\boldsymbol{\pi}^*\right\|. \tag{6.3}$$

Since $\mathbf{1}^T\boldsymbol{\pi}^* = 1$ (see page 12) and $\boldsymbol{b} = (1-\alpha)\boldsymbol{p}$ (see page 18), (6.3) equals

$$\left\|\boldsymbol{\pi}^* - \alpha S\boldsymbol{\pi}^* - \alpha\boldsymbol{p}\boldsymbol{d}^T\boldsymbol{\pi}^* - \boldsymbol{b}\right\| \tag{6.4}$$

and because (6.1) and (6.4) are equal, we conclude that the used norms are the same.
The 2-norm is used for the numerical experiments.

6.1.2 Matrix-vector calculations

The iteration in the improved Power method algorithm of page 12 contains one matrix-vector multiplication $S\boldsymbol{\pi}^*$. The improved Jacobi method algorithm of page 19 has one matrix-vector multiplication with the sparse R'_J and a multiplication with the inverse of diagonal matrix M_J. The modified Gauss-Seidel method algorithm of page 21 contains one matrix-vector multiplication with the sparse triangular matrix R'_G and one multiplication of the inverse of the triangular matrix M with a vector, which can be calculated via back substitution. Consequently, the total costs for the improved Power, Jacobi and the modified Gauss-Seidel algorithms are one matrix-vector operations in each iteration step. The IDR(s) algorithms also requires 1 matrix-vector operation in each step [18, *pp. 1040-1041*].

We can conclude that we are able to make a straight comparison of the mentioned algorithms on basis of the number of iterations and the residuals in each iteration.

6.2 Matlab implementations

6.2.1 Web crawling

For the purpose of executing numerical experiments we need test matrices that contain information about which pages of (parts of) the World Wide Web are linked to each other. To obtain these matrices, we have to investigate the hyperlink structure of the World Wide Web. Such an investigation is done by a web crawler called *surfer*, written by Cleve Moler [14] but modified for our experiments. The crawler starts on a given web page and investigates all web pages that it finds via hyperlinks on that websites. It returns an adjacency matrix B of a size to choose with $B(i,j) = 1$ if page j has a hyperlink to page i. See appendix A for the Matlab code. The function `surfer` requires as input the number of pages that has to be investigated and a root URL at which `surfer` will start its search. Besides matrix B, `surfer` returns a list with the URL's of all investigated pages.

6.2.2 Calculating S and $\boldsymbol{d}$

The algorithms mentioned in section 6.1 require five parameters: matrix S, vectors $\boldsymbol{p}$ and $\boldsymbol{d}$, damping factor α and tolerance ε. We have already made a choice for the values of $\boldsymbol{p}$ and α (see for example sections 4.1 and 4.2). However, S and $\boldsymbol{d}$ have to be calculated. In order to create matrix S (c.f. 3.2) given a test matrix B, we have to multiply it with a diagonal matrix containing the sum of each column of B:

$$S = BD \tag{6.5}$$

with

$$D(i,i) = \begin{cases} \frac{1}{l_i} & \text{if } l_i > 0 \\ 0 & \text{if } l_i = 0 \end{cases} \tag{6.6}$$

where $l_i = \sum_{j=1}^{N} B(i,j)$ denotes the number of outlinks on page i. Now, recall from section 3.3.1 $\boldsymbol{d} \in \{0,1\}^N$ with

$$\boldsymbol{d}(i) = \begin{cases} 1 & \text{if the } i\text{-th column of } S \text{ is a dangling column} \\ 0 & \text{otherwise.} \end{cases} \tag{6.7}$$

In order to calculate $\boldsymbol{d}$ with given B we rewrite this as

$$\boldsymbol{d}(i) = \begin{cases} 1 & \text{if } l_i = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6.8}$$

An implementation of what has been stated above is done in the file `create_d_s.m`, see appendix B. The Matlab function `create_d_s` requires as parameter matrix B as defined in section 6.2.1 and has S, $\boldsymbol{d}$ and n as output variables, where $n \times n$ is the size of matrices B and S.

6.2.3 The Power, Jacobi and modified Gauss-Seidel method algorithms

The improved Power method algorithm of page 12, the improved Jacobi method algorithm of page 5.3.2 and the modified Gauss-Seidel algorithm of page 5.4.2 have been implemented in respectively the files `pr_power.m`, see appendix C, the file `pr_jacobi.m`, see appendix D and the file `pr_mod_gauss_seidel.m`, see appendix E. All three algorithms have the following parameters: adjacency matrix B, damping factor α and tolerance ε. The output is

- $\boldsymbol{\pi}^*$, an approximation of the PageRank vector for the given matrix B,
- a vector with the residual norms at each iteration,
- the number of executed iterations,
- a vector with the total elapsed time (s) after each iteration.

6.2.4 The modified IDR(s) method algorithms with preconditioning

A numerically stable implementation of the modified IDR(s) method (c.f. section 5.5) is given in [4]. This implementation is modified for the PageRank problem in the file `pr_mod_idrs.m`, see appendix F. The algorithm has the following parameters:

- adjacency matrix B,
- damping factor α,
- tolerance ε,
- the parameter s of the IDR(s) method,
- a value which specifies whether preconditioning should be applied: no preconditioning or Jacobi or Gauss-Seidel preconditioning.

The output is

- $\boldsymbol{\pi}^*$, an approximation of the PageRank vector for the given matrix B,

- a vector with the residual norms at each iteration,
- the number of executed iterations,
- a vector with the total elapsed time (s) after each iteration.

6.2.5 Scripts for algorithm comparison

The appendices G and H contain the code of scripts for the comparison of the different algorithms and for creating graphs of the convergence rates. The scripts choose $\alpha = 0.85$ and tolerance $\varepsilon = 10^{-10}$. The rates of convergence c_k in the kth iterate are

$$c_k = \frac{\|e_k\|}{\|e_{k-1}\|}, \tag{6.9}$$

recalling (5.9). Because the used norms in the different methods are equal as explained in section 6.1.1, the rates of convergence can be computed by dividing the residual norm in the kth iterate by the residual norm in the $k-1$st iterate.

Chapter 7

Numerical experiments

This chapter presents the results of numerical experiments that has been performed in order to compare the different methods that compute the PageRank vector as described in the last chapter. The creation of test matrices and all algorithm comparisons are performed on a PC with a 2 Ghz CPU and 1 GB RAM. The results are being discussed in each section particularly.

7.1 Test matrices

The following test matrices are generated by the web crawler *Surfer*:

name	*size*	*root URL*
`B_tu_10k`	10,000 × 10,000	`http://www.tudelft.nl`
`B_sf_10k`	10,000 × 10,000	`http://www.stanford.edu`
`B_nl_10k`	10,000 × 10,000	`http://www.huisdiereninfo.nl`
`B_jp_10k`	10,000 × 10,000	`http://www.kantei.go.jp`
`B_tu_5k`	5,000 × 5,000	`http://www.tudelft.nl`
`B_sf_5k`	5,000 × 5,000	`http://www.stanford.edu`
`B_nl_5k`	5,000 × 5,000	`http://www.huisdiereninfo.nl`
`B_jp_5k`	5,000 × 5,000	`http://www.kantei.go.jp`
`B_tu_1k`	1,000 × 1,000	`http://www.tudelft.nl`
`B_sf_1k`	1,000 × 1,000	`http://www.stanford.edu`
`B_nl_1k`	1,000 × 1,000	`http://www.huisdiereninfo.nl`
`B_jp_1k`	1,000 × 1,000	`http://www.kantei.go.jp`

Table 7.1: Test matrices generated by *Surfer*

The sparsity patterns of two of these matrices are given in figure 7.1 and 7.2. There is a notable difference in the patterns of the matrices `B_sf_10k` and `B_jp_10k`. The average number of hyperlinks on pages in `B_sf_10k` turns out te be much larger than the average number of hyperlinks on pages in `B_sf_10k`.
The crawler *surfer* (see section 6.2.1) searches in the pages it visits for parts of the text containing the string 'http:', expecting a hyperlink at that certain position in the text. Then its checks whether these found hyperlinks are existing ULR's. This takes a lot of time. A defect in the crawler is the large number of hyperlinks that are skipped, for example links containing the characters '?' and '!'. The reason for this is that it is hard to find out whether hyperlinks

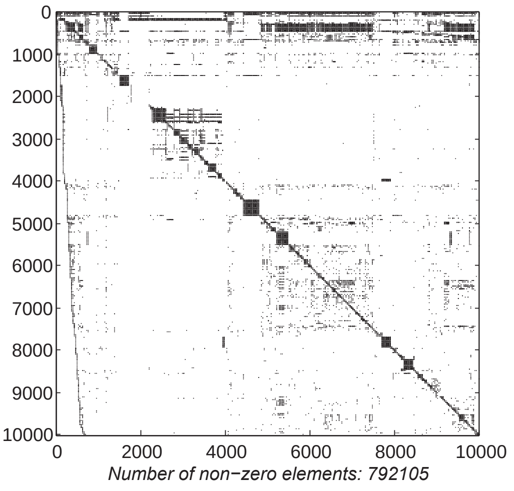


Figure 7.1: Sparsity pattern of `B_sf_10k`

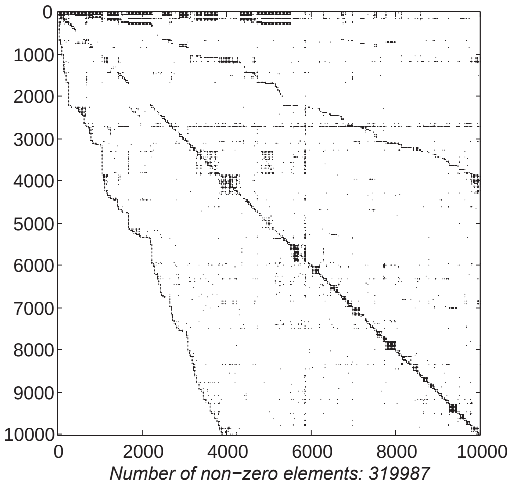


Figure 7.2: Sparsity pattern of `B_nl_10k`

containing these characters point to an existing web page. As a result, hyperlinks containing parameters are being neglected. However, a lot of HTTP or PHP pages are being called with the use of parameters.
Creating the test matrices with size 10,000 × 10,000 took about 7 hours per matrix. As a result, no test matrices larger than these were created. But investigating 10,000 gives only an indication of the hyperlink structure of the World Wide Web, considering the large amount of pages in it.

7.2 General results

The comparison of the different algorithms is done with the four largest test matrices. The smaller matrices are used to examine the influence of the matrix size on the convergence speed. The main results are listed below.

test matrix	`B_tu_10k`	`B_sf_10k`	`B_nl_10k`	`B_jp_10k`
	iterations, time	*iterations, time*	*iterations, time*	*iterations, time*
Power	98, 164 ms.	100, 231 ms.	96, 149 ms.	99, 152 ms.
Jacobi	96, 247 ms.	85, 330 ms.	87, 223 ms.	84, 210 ms.
Gauss-Seidel	61, 198 ms.	56, 269 ms.	59, 190 ms.	64, 189 ms.
IDR(1)	40, 136 ms.	38, 174 ms.	38, 163 ms.	40, 138 ms.
IDR(2)	39, 139 ms.	38, 173 ms.	38, 126 ms.	36, 152 ms.
IDR(4)	38, 145 ms.	35, 179 ms.	38, 134 ms.	38, 136 ms.
IDR(1), Jacobi	44, 142 ms.	36, 171 ms.	38, 126 ms.	40, 143 ms.
IDR(2), Jacobi	36, 138 ms.	38, 176 ms.	36, 127 ms.	41, 135 ms.
IDR(4), Jacobi	39, 155 ms.	39, 189 ms.	37, 136 ms.	40, 142 ms.
IDR(1), Gauss-Seidel	24, 138 ms.	28, 177 ms.	22, 124 ms.	24, 126 ms.
IDR(2), Gauss-Seidel	24, 135 ms.	24, 171 ms.	23, 136 ms.	24, 128 ms.
IDR(4), Gauss-Seidel	24, 139 ms.	24, 179 ms.	24, 130 ms.	23, 132 ms.

Table 7.2: Main results of algorithm comparison

The table contains the number of iterations together with the average execution time of the different algorithms. Each algorithm has been executed ten times to reduce the influence of changes in available memory and CPU speed on the computer that performed the tests. The execution of the Jacobi and Gauss-Seidel methods takes much more time than the execution of the Power method, while both finish in less iterations. This finds its reason in the way these methods compute the residual norms. The power method calculates $\left\|\boldsymbol{\pi}_c^* - \boldsymbol{\pi}_p^*\right\|$ (see section 4.1) and both Jacobi and Gauss-Seidel calculate $\left\|\alpha S\boldsymbol{\pi}^* + \alpha \boldsymbol{p}\boldsymbol{d}^T\boldsymbol{\pi}^* + \boldsymbol{b} - \boldsymbol{\pi}^*\right\|$ (see sections 5.3.2 and 5.4.2). We see that the Jacobi and Gauss-Seidel methods calculate one more matrix-vector calculation than the Power method in each iteration. Considering this, we can conclude that the execution times of the algorithms do not differ very much. This means that we can focus on the number of iterations and the residual norms.

7.3 Comparison of residual norms

We will compare the residual norms generated by the algorithms for the four largest test matrices. See the figures below.

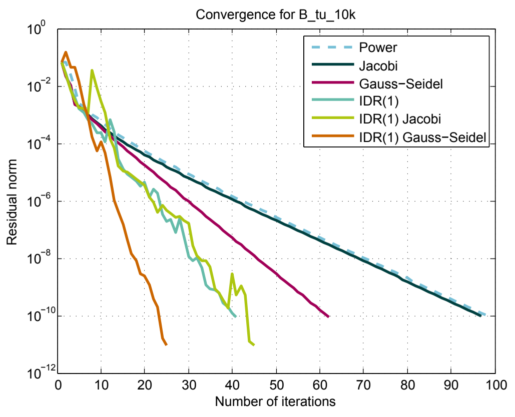


Figure 7.3: Convergence of several methods for `B_tu_10k`

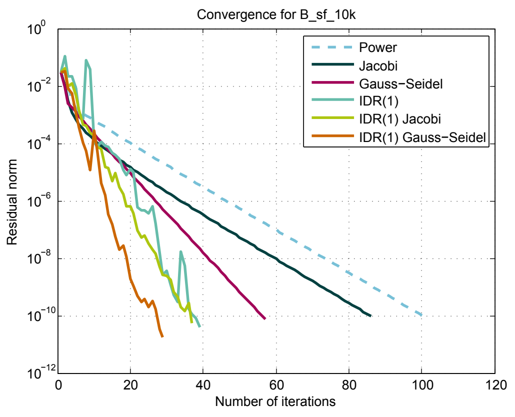


Figure 7.4: Convergence of several methods for `B_sf_10k`

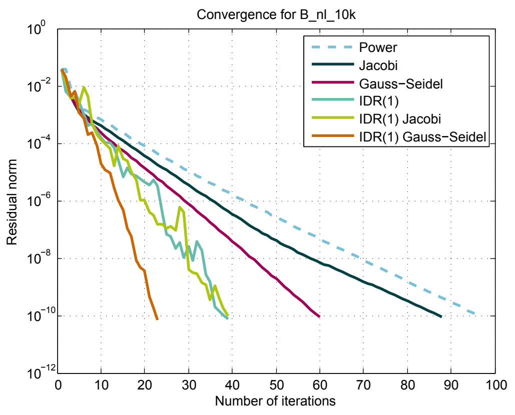


Figure 7.5: Convergence of several methods for `B_nl_10k`

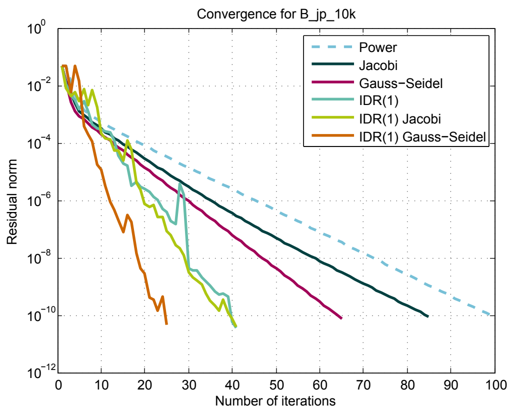


Figure 7.6: Convergence of several methods for `B_jp_10k`

The behaviour of the methods does not differ much for the four test matrices. IDR(1) with Gauss-Seidel preconditioning clearly performs the best. IDR(1) without preconditioning and IDR(1) with Jacobi preconditioning behave almost the same. Further note that the performance of the Gauss-Seidel method is much better than the Jacobi method. All methods outperform the Power method. Now, let us investigate the differences in performance of IDR(1), IDR(2) and IDR(4). See the figures below.

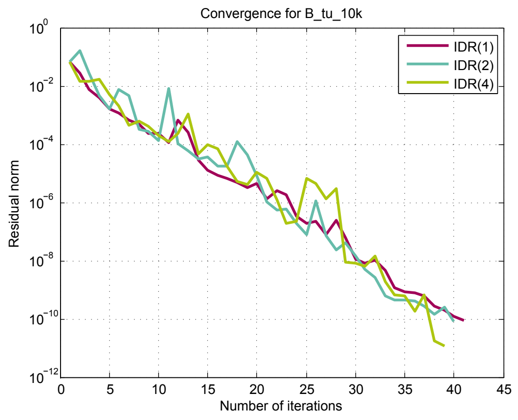


Figure 7.7: Convergence of IDR(s) for `B_tu_10k`

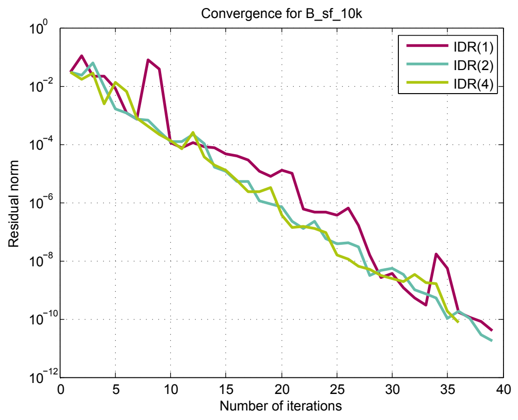


Figure 7.8: Convergence of IDR(s) for `B_sf_10k`

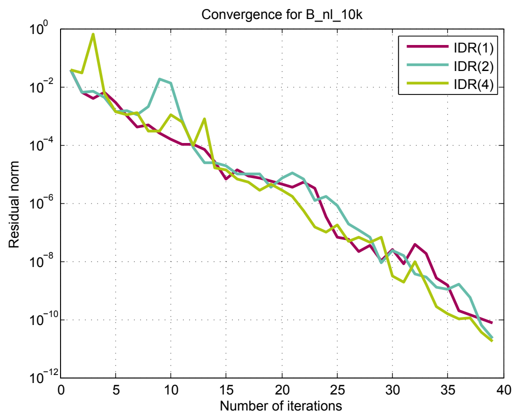


Figure 7.9: Convergence of IDR(s) for `B_nl_10k`

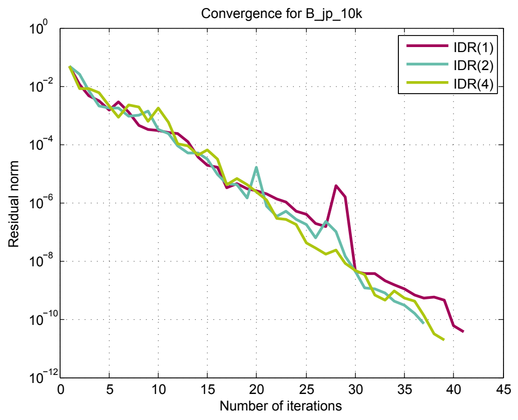


Figure 7.10: Convergence of IDR(s) for `B_jp_10k`

The three methodss IDR(1), IDR(2) and IDR(4) have almost the same convergence rate. None of them is the fastest one for all four test matrices. It is clear that preconditioning makes a difference, where the choice of s does not.

7.4 Comparison of matrix sizes

We will investigate whether choosing another size of the matrices changes the convergence. See the figures on the next page.

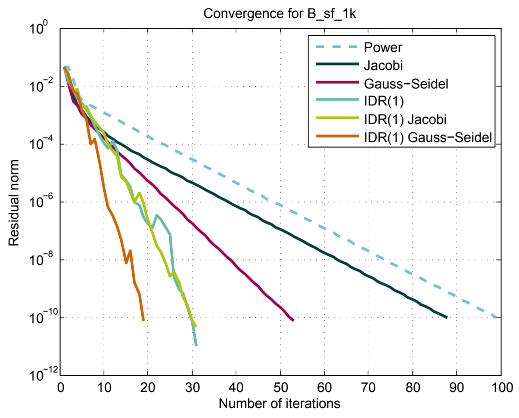


Figure 7.11: Convergence of several methods for `B_sf_1k`

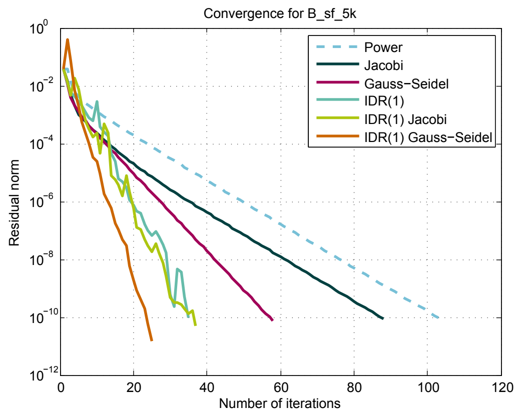


Figure 7.12: Convergence of several methods for `B_sf_5k`

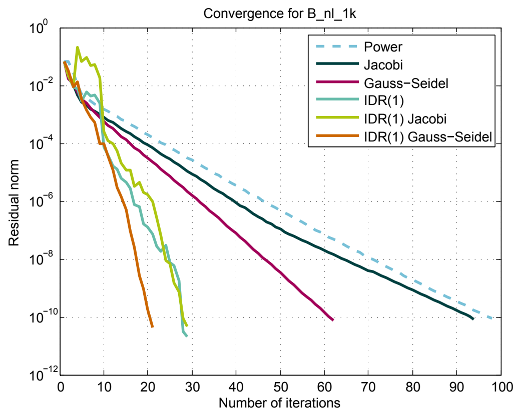


Figure 7.13: Convergence of several methods for `B_nl_1k`

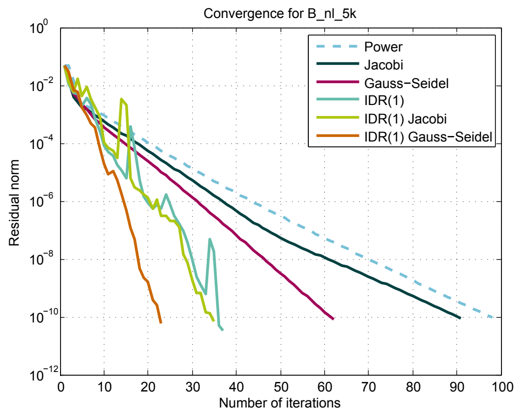


Figure 7.14: Convergence of several methods for `B_nl_5k`

Comparing the figures above with figure 7.4 and 7.5, it is clear that the size of the matrices hardly matters. If we look at the patterns of the matrices, figures 7.1 and 7.2, we see that the hyperlink structure of the first 1,000 web pages, of the first 5,000 web pages and of the whole matrix does not differ much. We cannot conclude in general, however, that the matrix size does not change the convergence. A very large test matrix, although having the same root URL, may contain other parts of the World Wide Web that have another hyperlink structure.

7.5 Rates of convergence

Finally, we will look at the convergence rates of the Power method and the basic iterative methods.

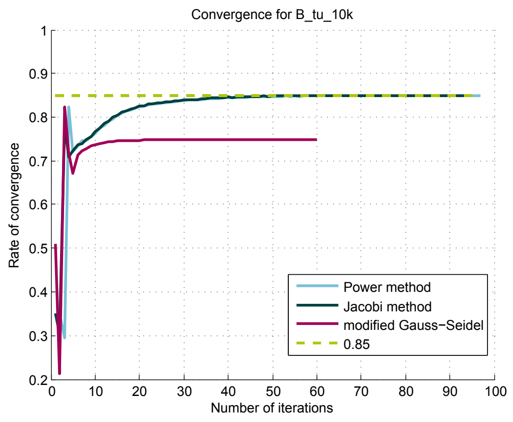


Figure 7.15: Convergence rates for `B_tu_10k`

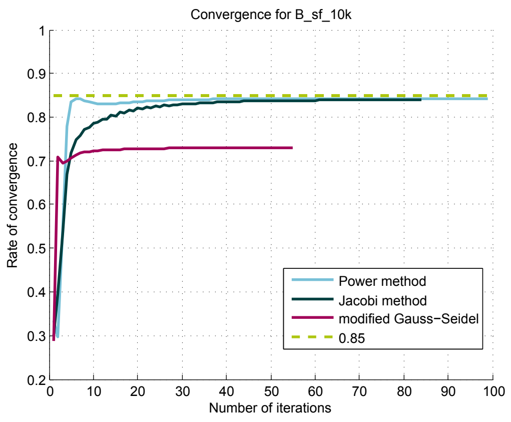


Figure 7.16: Convergence rates for `B_sf_10k`

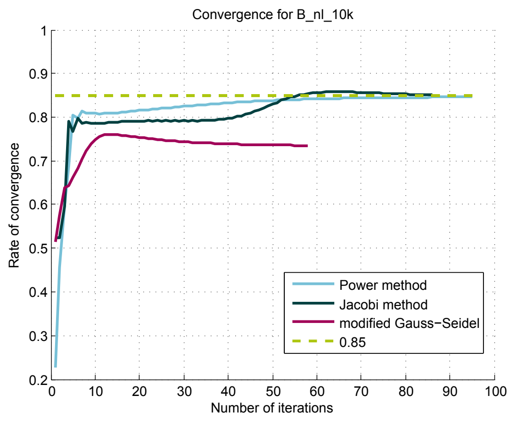


Figure 7.17: Convergence rates for `B_nl_10k`

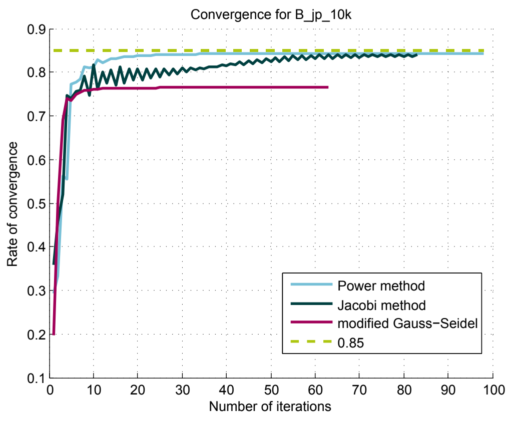


Figure 7.18: Convergence rates for `B_jp_10k`

After a small number of iterations, the convergence rates of all methods stabilises. The theoretical value of the convergence rate of the Power method, $\alpha = 0.85$ (see section 4.2), matches the convergence rate that we find in these results.

Chapter 8

Conclusions and recommendations

In chapters 2 and 3 we have seen the description of an adequate model for the hyperlink structure of the web, which has been adapted to a model for web surfing. With this model the existence and uniqueness of the PageRank vector, which is the solution to the PageRank problem, has been proved.
Several methods for computing the PageRank vector have been mentioned in chapters 4 and 5. For these methods it has been proved that they converge to the PageRank vector. The theoretical value of the convergence speed of the Power method, α, matches the convergence speed that we found in the numerical experiments.

The results of our numerical experiments show that we are able to compare the described algorithms well by the required number of iterations. The way these methods are implemented does not bias our experiments.
The numerical experiments further show that IDR(s) efficiently computes the PageRank vector and in this respect outperforms the Power method and the basic iterative methods, in numbers of iteration as well as in execution time. For which value of s IDR(s) performs best, depends on the choice of the matrix. For the tested matrices, IDR(s) with Gauss-Seidel preconditioning gives the best performance.

Further research to the subject of this thesis is useful, for two reasons:

1. even more interesting than comparing the Power method and the basic iterative methods to IDR(s), may be comparing IDR(s) with other Krylov subspace methods such as Bi-CG, Bi-CGSTAB and GMRES.

2. there is a need for better test matrices: the used web crawler has some important defects, as mentioned in section 7.1. But above that, the used matrices are too small. Numerical experiments should be executed with much larger and more realistic test matrices.

Bibliography

[1] Abraham Berman and Robert J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, London, 1979.

[2] Sergey Brin and Lawrence Page, *The anatomy of a large-scale hypertextual web search engine*, in: Seventh International World-Wide Web Conference (WWW 1998), April 14-18, 1998, Brisbane, Australia.

[3] Gianna M. Del Corso, Antonio Gullí, and Francesco Romani, *Fast PageRank computation via a sparse linear system*, Internet Mathematics Vol. 2, No. 3, 2005, pp. 251 - 273.

[4] Martin B. van Gijzen and Peter Sonneveld, *An elegant IDR(s) variant that efficiently exploits bi-orthogonality properties*, Delft University of Technology, Reports of the Department of Applied Mathematical Analysis, Report 08-21, 2008.

[5] David F. Gleich, Leonid Zhukov and Pavel Berkhin, *Fast parallel PageRank: A linear system approach*, Yahoo! Technical Report, 2004.

[6] David F. Gleich, Andrew P. Gray, Chen Greif and Tracy Lau, *An inner-outer iteration for computing PageRank*, SIAM J. Sci. Comput. Vol. 32, No. 1, 2010, pp. 349 - 371.

[7] Gene H. Golub and Charles F. Van Loan, *Matrix Computations, Third edition*, The Johns Hopkins University Press, Baltimore (Maryland), 1996.

[8] Google Inc., *Corporate Information - Technology overview*, 2010, available at *http://www.google.com/corporate/tech.html.*

[9] Richard T. Griffiths, *History of the Internet*, Universiteit Leiden, 2001, available at *http://www.leidenuniv.nl/letteren/internethistory/index.htm.*

[10] Taher H. Haveliwala and Sepandar D. Kamvar, *The Second Eigenvalue of the Google Matrix*, Technical report 2003-20, Stanford University, 2003.

[11] Ilse C.F. Ipsen and Rebecca S. Wills, *Mathematical properties and analysis of Googles PageRank*, Bol. Soc. Esp. Mat. Apl., vol. 34, 2006, pp. 191 - 196.

[12] Jos van Kan, Guus Segal and Fred Vermolen, *Numerical methods in scientific computing*, VSSD, Delft, 2005.

[13] Chris P. Lee, Gene H. Golub and Stefanos A. Zenios, *A two-stage algorithm for computing PageRank and multistage generalizations*, Internet Mathematics Vol. 4, No. 4, 2007, pp. 299 - 327.

[14] Cleve Moler, *Numerical Computing with MATLAB*, electronic edition, The MathWorks Inc., 2004, available at *http://www.mathworks.nl/moler/ncmfilelist.html.*

[15] Lawrence Page, Sergey Brin, Rajeev Motwani and Terry Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*, Technical Report, Stanford InfoLab, 1999.

[16] David Poole, *Linear Algebra, A Modern Introduction*, Brooks/Cole, Pacific Grove, 2003.

[17] Yousef Saad, *Iterative Methods for Sparse Linear Systems, Second Edition*, SIAM, Philadelphia, 2003.

[18] Peter Sonneveld and Martin B. van Gijzen, *IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems*, SIAM J. Sci. Comput. Vol. 31, No. 2, 2008, pp. 1035 - 1062.

[19] Gilbert Stewart, *Matrix Algorithms, Volume 1: Basic decompositions*, SIAM, Philadelphia, 1998.

[20] Richard S. Varga, *Matrix iterative analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1962.

[21] James H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, 1965.

[22] Rebecca S. Wills and Ilse C. F. Ipsen, *Ordinal ranking for Google's PageRank*, SIAM J. Matrix Anal. Appl., Vol. 30, No. 4, 2009, pp. 1677 - 1696.

Appendix A

Code of the web crawler `surfer`

```
function [U,B] = surfer(root,n)

% SURFER  Create the adjacency graph of a portion of the Web.
%    [U,B] = surfer(root,n) starts at the URL root and follows
%    hyperlinks until it forms an adjacency graph with n nodes.
%    U = a cell array of n strings, the URLs of the nodes.
%    B = a sparse square matrix denoting the webgraph
%        with B(i,j) = 1 if page j is linked to page i.
%
%    Example:  [U,B] = surfer('http://www.harvard.edu',500);
%
%    This function currently has two defects.  (1) The algorithm for
%    finding links is naive.  We just look for the string 'http:'.
%    (2) An attempt to read from a URL that is accessible, but very slow,
%    might take an unacceptably long time to complete.  In some cases,
%    it may be necessary to have the operating system terminate MATLAB.
%
%    With size n = 10000, about 700 MB of free memory is required.
%    Key words from such URLs can be added to the skip list in surfer.m.
%
%    Written by Cleve Moler.
%    Modified by Rien den Besten, June - August 2010

% Initialize

disp(['Surfer will crawl the web starting at <a href = "',...
     deblank(root), '">', deblank(root), '</a> and investigate ', ...
     num2str(n), ' web pages.']); fprintf('\n')

tic;
U = cell(n,1);
Pages = cell(n,1);
hash = zeros(n,1);
B = logical(sparse(n,n));
m = 1;
U{m} = root;
hash(m) = hashfun(root);

try
    Pages{m} = urlread(root);
catch
    error(['Unable to read root page ', root]);
end
```

```

j = 1;
while j < n

   % Follow the links from the open page.
   page = Pages{j};
   for f = findstr('http:',page);

      % A link starts with 'http:' and ends with the next quote.

      e = min([findstr('"',page(f:end)) findstr('''',page(f:end))]);
      if isempty(e), continue, end
      url = deblank(page(f:f+e-2));
      url(url<' ') = '!';    % Nonprintable characters
      if url(end) == '/', url(end) = []; end

      % Look for links that should be skipped.

      skips = {'.gif','.jpg','.jpeg','.css', '.js','.mwc','.ram','.cgi',...
               'lmscadsi','cybernet','w3.org','yahoo','.png','google',...
               'scripts','netscape','shockwave','webex','doubleclick'};
      skip = any(url=='!') | any(url=='?');

      k = 0;
      while ¬skip & (k < length(skips))
         k = k+1;
         skip = ¬isempty(findstr(url,skips{k}));
      end
      if skip
         continue
      end

      % Check if page is already in url list.

      hashcurrent = hashfun(url);
      i = 0;
      for k = find(hash(1:m) == hashcurrent)';
         if isequal(U{k},url)
            i = k;
            break
         end
      end

      % Add a new url to the graph there if are fewer than n and the url
      % can be read

      if (i == 0) && (m < n)
         try
            [newpage, status] = urlread(url);
         catch
            continue
         end
         if status == 0
             continue
         end
         m = m+1;
         U{m} = url;
         Pages{m} = newpage;
         hash(m) = hashcurrent;
         i = m;
      end

      % Add a new link.

      if i > 0
         B(i,j) = 1;
      end
   end

   Pages{j} = [];

   j = j+1;

   if mod(j,10) == 0
      minutes = floor(toc / 60);
      seconds = toc - minutes * 60;
      disp(['Currently investigated: ', num2str(j), ' pages of ', ...
        num2str(n), ' in ', num2str(minutes), ' minutes and ', ...
        num2str(round(seconds)), ' seconds; ', num2str(m), ...
        ' pages found.']);
   end
end
%-------------------------

function h = hashfun(url)
% Almost unique numeric hash code for pages already visited.
h = length(url) + 1024*sum(url);
```

Appendix B

Script for creating S and d

```
function [S, d, n] = create_d_s(B)

%    CREATE_D_S is a function that creates a probability matrix for a
%    given web graph matrix and lists the dangling nodes in a vector.
%
%
%    PARAMETERS:
%       B        a sparse square adjacency matrix denoting the web graph
%                with B(i,j) = 1 if page j is linked to page i.
%
%
%    RETURNS:
%       S        probability matrix
%
%       d        vector denoting the positino of dangling nodes
%
%       n        size of matrices B and S
%
%
%    Rien den Besten
%    Copyright (c) August 2010

[n,n] = size(B);

d = zeros(1,n);
D = sparse(n,n);

for j = 1:n
   %Find indices of non-zero elements in column j of B:
   B_column_indices{j} = find(B(:,j));

   %Count number of non-zero elements in column j of B:
   LO_column_indices(j) = length(B_column_indices{j});

   % Fill d and D:
   if LO_column_indices(j) > 0  % column j is no dangling column
       D(j,j) = 1 / LO_column_indices(j);
   else % column j is a dangling column
       d(j) = 1;
   end
end
S = B * D;
```

Appendix C

Implementation of the Power method

```
function [pi, resvec, iter, tmvec] = pr_power(B, alpha, tol)

%    PR_POWER is an implementation of the Power method
%    for the PageRank problem.
%
%
%    PARAMETERS:
%       B        a sparse square matrix denoting the web graph
%                with B(i,j) = 1 if page j is linked to page i.
%
%       alpha    the damping factor.
%
%       tol      the tolerance of the method.
%
%
%    RETURNS:
%       pi       approximation of the PageRank vector for the given matrix B.
%
%       resvec   vector of the residual norms at each iteration.
%
%       cnt      the number of iterations.
%
%       tmvec    vector of the total elapsed time (s) after each iteration.
%
%
%    Rien den Besten
%    Copyright (c) August 2010

tic;

[S, d, n] = create_d_s(B);
alphaS = alpha * S;
p = ones(n,1)/n;

% Starting values
pi_c = p;
iter = 0;
r = p - alphaS*p - (alpha*dot(d,p) + 1 - alpha)*p;
normr = norm(r);
resvec = [normr];
```

```
tmvec = [toc];

% Power iteration
while normr > tol
    pi_p = pi_c;

    pi_c = alphaS*pi_p + (alpha*dot(d,pi_p) + 1 - alpha)*p;

    r = pi_c - pi_p;
    normr = norm(r);

    resvec = [resvec; normr];
    tmvec = [tmvec; toc];
    iter = iter + 1;
end
pi = pi_c;
```

Appendix D

Implementation of the Jacobi method

```
function [pi, resvec, iter, tmvec] = pr_jacobi(B, alpha, tol)

%    PR_JACOBI is an implementation of the Jacobi iteration method
%    for the PageRank problem.
%
%
%    PARAMETERS:
%       B        a sparse square adjacency matrix denoting the web graph
%                with B(i,j) = 1 if page j is linked to page i.
%
%       alpha    the damping factor.
%
%       tol      the tolerance of the method.
%
%
%    RETURNS:
%       pi       approximation of the PageRank vector for the given matrix B.
%
%       resvec   vector of the residual norms at each iteration.
%
%       cnt      the number of iterations.
%
%       tmvec    vector of the total elapsed time (s) after each iteration.
%
%
%    Rien den Besten
%    Copyright (c) August 2010

tic;

[S, d, n] = create_d_s(B);
alphaS = alpha * S;
p = ones(n,1)/n;
b = (1 - alpha) * p;
Diag_alpha_p_d = diag(sparse(alpha / n * d));

% Create M and Rt
M_J = speye(n) - diag(diag(alphaS)) - Diag_alpha_p_d;
Rt_J = alphaS - diag(diag(alphaS)) - Diag_alpha_p_d;

```

```
% Starting values
pi = p;
iter = 0;
r = b - pi + alphaS*pi + alpha*p*dot(d,pi);
normr = norm(r);
resvec = [normr];
tmvec = [toc];

% Jacobi iteration
while normr > tol

    pi = M_J \ (Rt_J * pi + alpha*p*dot(d,pi) + b);

    r = b - pi + alphaS*pi + alpha*p*dot(d,pi);
    normr = norm(r);

    resvec = [resvec; normr];
    tmvec = [tmvec; toc];
    iter = iter + 1;
end
```

Appendix E

Implementation of the modified Gauss-Seidel method

```
function [pi, resvec, iter, tmvec] = pr_mod_gauss_seidel(B, alpha, tol)

%    PR_MOD_GAUSS_SEIDEL is an implementation of the modified Gauss-Seidel
%    iteration method for the PageRank problem.
%
%
%    PARAMETERS:
%       B        a sparse square adjacency matrix denoting the web graph
%                with B(i,j) = 1 if page j is linked to page i.
%
%       alpha    the damping factor.
%
%       tol      the tolerance of the method.
%
%
%    RETURNS:
%       pi       approximation of the PageRank vector for the given matrix B.
%
%       resvec   vector of the residual norms at each iteration.
%
%       iter     the number of iterations.
%
%       tmvec    vector of the total elapsed time (s) after each iteration.
%
%
%    Rien den Besten
%    Copyright (c) August 2010

tic;

[S, d, n] = create_d_s(B);
alphaS = alpha * S;
p = ones(n,1)/n;
b = (1 - alpha) * p;

% Create M and Rt
M_Gm = speye(n) - tril(alphaS,0);
Rt_Gm = triu(alphaS,1);

% Starting values
```

```
pi = p;
iter = 0;
r = b - pi + alphaS*pi + alpha*p*dot(d,pi);
normr = norm(r);
resvec = [normr];
tmvec = [toc];

% Modified Gauss-Seidel iteration
while normr > tol

    pi = M_Gm \ (Rt_Gm * pi + alpha*p*dot(d,pi) + b);

    r = b - pi + alphaS*pi + alpha*p*dot(d,pi);
    normr = norm(r);

    resvec = [resvec; normr];
    tmvec = [tmvec; toc];
    iter = iter + 1;
end
```

Appendix F

Implementation of the modified IDR(s) method

```
function [pi, resvec, iter, tmvec] = pr_mod_idrs(B, alpha, tol, s, prec)

%    PR_MOD_IDRS is an implementation of the Induced Dimension Reduction
%    method for the PageRank problem.
%
%
%    PARAMETERS:
%       B        a sparse square adjacency matrix denoting the web graph
%                with B(i,j) = 1 if page j is linked to page i.
%
%       alpha    the damping factor.
%
%       tol      the tolerance of the method.
%
%       s        specifies the dimension of the 'shadow space'. Normally,
%                a higher s gives faster convergence, but also makes
%                the method more expensive.
%
%       prec     specifies which preconditioner should be used: 0 for no
%                preconditioning, 1 for Jacobi preconditioning and
%                2 for Gauss-Seidel preconditioning.
%
%    RETURNS:
%       pi       the PageRank vector for the given matrix B.
%
%       resvec   vector of the residual norms at each iteration.
%
%       iter     the number of iterations.
%
%       tmvec    vector of the total elapsed time (s) after each iteration.
%
%   Written by Martin van Gijzen and Peter Sonneveld
%   Copyright (c) December 2008
%
%   Original code in idrs.m, availabe at
%   http://ta.twi.tudelft.nl/nw/users/gijzen/IDR.html
%
%   Modified for the PageRank problem by Martin van Gijzen en Rien den Besten
%   June - August 2010

```

```
tic;

% Create A and b, set parameters
[S, d, n] = create_d_s(B);
A = speye(n) - alpha*S;
p = ones(n,1)/n;
b = (1 - alpha) * p;
maxit = min(2*n,1000);
rand('state', 0);
Q = rand(n,s);
Q = orth(Q);
angle = 0.7;

% Set preconditioning
if( prec == 1)          % Jacobi
    P = diag(diag(A));
    prec = 1;
elseif( prec == 2)      % Gauss-Seidel
    P = tril(A,0);
    prec = 1;
else                    % No preconditioning
    P = [];
    prec = 0;
end

% Check for zero rhs:
if (norm(b) == 0)                % Solution is nulvector
   iter = 0;
   resvec = 0;
   info = 0;
   err = 0;
   return
end

% Compute initial residual:
pi = p;

r = b + alpha*p*dot(d,pi) - A*pi;
normr = norm(r);
resvec=[normr];

if (normr ≤ tol)                % Initial guess is a good enough solution
   iter = 0;
   info = 0;
   err = 0;
   return
end

G = zeros(n,s); U = zeros(n,s); M = eye(s,s);
om = 1;

% Main iteration loop, build G-spaces:
iter = 0;
tmvec = [toc];

while ( normr > tol && iter < maxit )

% New righ-hand size for small system:
   f = (r'*Q)';
   for k = 1:s

```

```
% Solve small system and make v orthogonal to Q:
      c = M(k:s,k:s)\f(k:s);
      v = r - G(:,k:s)*c;
      if ( prec )
         v = P\v;
      end

      U(:,k) = U(:,k:s)*c + om*v;
% Compute G(:,k) = A U(:,k)
      G(:,k) = A*U(:,k) - alpha*p*dot(d,U(:,k));
%
% Bi-Orthogonalise the new basis vectors:
      for i = 1:k-1
         beta =  ( Q(:,i)'*G(:,k) )/M(i,i);
         G(:,k) = G(:,k) - beta*G(:,i);
         U(:,k) = U(:,k) - beta*U(:,i);
      end
% New column of M = Q'*G  (first k-1 entries are zero)
      M(k:s,k) = (G(:,k)'*Q(:,k:s))';
%
%  Make r orthogonal to q_i, i = 1..k
      gamma = f(k)/M(k,k);
      r = r - gamma*G(:,k);
      pi = pi + gamma*U(:,k);

      iter = iter + 1;
      normr = norm(r);
      resvec = [resvec;normr];
      tmvec = [tmvec; toc];
      if ( normr < tol || iter == maxit )
         break
      end

% New f = Q'*r (first k  components are zero)
      if ( k <s )
         f(k+1:s)   = f(k+1:s) - gamma*M(k+1:s,k);
      end
   end

% Now we have sufficient vectors in G_j to compute residual in G_j+1
% Note: r is already perpendicular to Q so v = r
   v = r;
   if ( prec )
      v = P\v;
   end
   t = A*v - alpha*p*dot(d,v);
   om = omega( t, r, angle );
%
   r = r - om*t;
   pi = pi + om*v;
   normr = norm(r);
   resvec = [resvec;normr];
   tmvec = [tmvec; toc];
   iter = iter + 1;

end; %while

err = norm( b - A*pi + alpha*p*dot(d,pi));
if ( err < tol )
   info = 0;
elseif ( iter == maxit )
```

```
   info = 1;
else
   info = 2;
end

return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function om = omega( t, s, angle )

ns = norm(s);
nt = norm(t);
ts = dot(t,s);
rho = abs(ts/(nt*ns));
om=ts/(nt*nt);
if ( abs(rho ) < angle )
   om = om*angle/abs(rho);
end

return
```

Appendix G

Script for creating residual graphs

```
%   Script to compare iterative solvers for the PageRank problem
%
%   Default parameters are:
%
%       - Tolerance: 1e-10
%       - Alpha: 0.85
%
%   Written by Rien den Besten, August 2010.
%
%   With thanks to Martin van Gijzen: the code is based on it_solve.m
%   written by Martin van Gijzen, which is availabe at
%   http://ta.twi.tudelft.nl/nw/users/gijzen/IDR.html
%

clear all; close all;

tol = 1e-10;
alpha = 0.85;

[filename, pathname] = uigetfile('*.mat','Choose a matrix');
S = load([pathname filename], '*');
fn = fieldnames(S);

[dum, name] = fileparts(filename);
choice = 1;

c = [119 192 215; 0 64 64; 161 0 88; 102 188 170; 173 198 16; 204 102 0]/255;
k = 0;
method = char('Power','Jacobi','Gauss-Seidel','IDR(1)','IDR(1) Jacobi', ...
'IDR(1) Gauss-Seidel', 'IDR(2)','IDR(2) Jacobi', ...
'IDR(2) Gauss-Seidel', 'IDR(4)','IDR(4) Jacobi', 'IDR(4) Gauss-Seidel');
methods = [];
number_of_items = 13;

while ( choice ≤ number_of_items )

    choice = menu('Choose a solver: ', 'Power','Jacobi','Gauss-Seidel',...
        'IDR(1)','IDR(1) Jacobi', 'IDR(1) Gauss-Seidel', ...
        'IDR(2)','IDR(2) Jacobi', 'IDR(2) Gauss-Seidel', ...
        'IDR(4)','IDR(4) Jacobi', 'IDR(4) Gauss-Seidel', 'STOP' );

    if ( choice == number_of_items )
```

```
        close all;
        return
    end

   k = k + 1;
   if k > 6
      k = 1;
   end

   solver = method(choice,:);
   disp(['Iterative solution with ',solver,'...'])

   tic;
   if ( choice == 1 )
       [x, resvec, iter] = pr_power(S.(fn{1}), alpha, tol);
   elseif ( choice == 2 )
       [pi, resvec, iter] = pr_jacobi(S.(fn{1}), alpha, tol);
   elseif ( choice == 3 )
       [pi, resvec, iter] = pr_mod_gauss_seidel(S.(fn{1}), alpha, tol);
   elseif ( choice == 4 )
       [pi, resvec, iter] = pr_mod_idrs(S.(fn{1}), alpha, tol, 1, 0);
   elseif ( choice == 5 )
       [pi, resvec, iter] = pr_mod_idrs(S.(fn{1}), alpha, tol, 1, 1);
   elseif ( choice == 6 )
       [pi, resvec, iter] = pr_mod_idrs(S.(fn{1}), alpha, tol, 1, 2);
   elseif ( choice == 7 )
       [pi, resvec, iter] = pr_mod_idrs(S.(fn{1}), alpha, tol, 2, 0);
   elseif ( choice == 8 )
       [pi, resvec, iter] = pr_mod_idrs(S.(fn{1}), alpha, tol, 2, 1);
   elseif ( choice == 9 )
       [pi, resvec, iter] = pr_mod_idrs(S.(fn{1}), alpha, tol, 2, 2);
   elseif ( choice == 10 )
       [pi, resvec, iter] = pr_mod_idrs(S.(fn{1}), alpha, tol, 4, 0);
   elseif ( choice == 11 )
       [pi, resvec, iter] = pr_mod_idrs(S.(fn{1}), alpha, tol, 4, 1);
   elseif ( choice == 12 )
       [pi, resvec, iter] = pr_mod_idrs(S.(fn{1}), alpha, tol, 4, 2);
   end

   disp(['Iterations: ', num2str(iter)]);
   disp(['Elapsed time: ', num2str(toc), ' s.']);
   disp(['Resvec(1): ', num2str(resvec(1))]);
   disp(' ');

   xas = [1:1:length(resvec)];
   if ( choice == 1 )
       semilogy(xas, resvec,'--','Color', c(1,:), 'LineWidth', 2);
   else
       semilogy(xas, resvec,'Color', c(k,:), 'LineWidth', 2);
   end

   xlabel('Number of iterations');
   ylabel('Residual norm');
   title(['Convergence for ' name], 'interpreter', 'none');
   methods = [methods; solver];
   legend(methods);
   grid on;
   hold on;

end
```

Appendix H

Script for creating graphs of convergence rates

```
%   Script to compare the rates of convergence of the Power, Jacobi and
%   Gauss-Seidel method in calculating the PageRank vector.
%
%   Default parameters are:
%
%       - Tolerance: 1e-10
%       - Alpha: 0.85
%
%   Written by Rien den Besten, August 2010.
%


tol = 1e-10;
alpha = 0.85;

[filename, pathname] = uigetfile('*.mat','Choose a matrix');
S = load([pathname filename], '*');
fn = fieldnames(S);

[dum, name] = fileparts(filename);
choice = 1;

c = [119 192 215; 0 64 64; 161 0 88; 0 43 96; 173 198 16; 102 188 170]/255;

[pi, resvec, iter] = pr_power(S.(fn{1}), alpha, tol);
[pi, resvec_j, iter] = pr_jacobi(S.(fn{1}), alpha, tol);
[pi, resvec_g, iter] = pr_mod_gauss_seidel(S.(fn{1}), alpha, tol);

xas = [1:1:(length(resvec)-2)];
xas_j = [1:1:(length(resvec_j)-2)];
xas_g = [1:1:(length(resvec_g)-2)];

valpha = ones(length(xas))*alpha;

diff_resvec = xas';
diff_resvec_j = xas_j';
diff_resvec_g = xas_g';

for i = 2:1:(length(resvec)-1)
   diff_resvec(i-1) =  resvec(i+1)/resvec(i);
```

```
end
for i = 2:1:(length(resvec_j)-1)
   diff_resvec_j(i-1) =  resvec_j(i+1)/resvec_j(i);
end
for i = 2:1:(length(resvec_g)-1)
   diff_resvec_g(i-1) =  resvec_g(i+1)/resvec_g(i);
end
hold on;
plot(xas, diff_resvec,'Color', c(1,:), 'LineWidth', 2);
plot(xas_j, diff_resvec_j,'Color', c(2,:), 'LineWidth', 2);
plot(xas_g, diff_resvec_g,'Color', c(3,:), 'LineWidth', 2);
plot(xas, valpha,'--','Color', c(5,:), 'LineWidth', 2);
legend('Power method','Jacobi method','modified Gauss-Seidel', '0.85');

xlabel('Number of iterations');
ylabel('Rate of convergence');
title(['Convergence for ' name], 'interpreter', 'none');
grid on;
```