A Power–Arnoldi algorithm for computing PageRank

Gang Wu\textsuperscript{1,\dagger} and Yimin Wei\textsuperscript{2,3,*,†}

\textsuperscript{1}School of Mathematical Sciences, Xuzhou Normal University, Xuzhou, Jiangsu 221116, People’s Republic of China
\textsuperscript{2}School of Mathematical Sciences, Fudan University, Shanghai 200433, People’s Republic of China
\textsuperscript{3}Key Laboratory of Mathematics for Nonlinear Sciences, Fudan University, Ministry of Education, Shanghai 200433, People’s Republic of China

SUMMARY

The PageRank algorithm plays an important role in modern search engine technology. It involves using the classical power method to compute the principle eigenvector of the Google matrix representing the web link graph. However, when the largest eigenvalue is not well separated from the second one, the power method may perform poorly. This happens when the damping factor is sufficiently close to 1. Therefore, it is worth developing new techniques that are more sophisticated than the power method. The approach presented here, called Power–Arnoldi, is based on a periodic combination of the power method with the thick restarted Arnoldi algorithm. The justification for this new approach is presented. Numerical tests illustrate the efficiency and convergence behaviour of the new algorithm. Copyright © 2007 John Wiley & Sons, Ltd.

Received 30 April 2006; Revised 17 December 2006; Accepted 12 January 2007

KEY WORDS: PageRank; Google matrix; thick restarted Arnoldi; Krylov subspace; Power–Arnoldi

Dedicated to Professor Zhihao Cao on the occasion of his 70th birthday

1. INTRODUCTION

Google is one of the most successful search engines in recent years, due to its comprehensive and accurate search results. The key algorithm, called PageRank, plays a major role in Google’s
search engine technology. The PageRank algorithm makes use of the classical power method \[1\] to compute the principle eigenvector (or PageRank vector) of the Google matrix representing the hyperlink structure of the web. In practice, the web graph is so large that the PageRank vector is commonly computed offline, during the pre-processing of the web crawl, before any queries have been issued. However, the power method does not always excite numerical analysts for its slow convergence. Indeed, for large web which contains millions or billions of nodes, computing a PageRank vector often takes several days or even weeks. Therefore, it is necessary to develop techniques that can compute PageRank more efficiently and reliably.

A large number of contributions have been made for this propose, such as the quadratic-extrapolation method \[2\], the adaptive methods \[3\], the BlockRank method \[4\], and the Arnoldi-type algorithm \[5\], see also \[6–11\] and the references therein. Generally speaking, these technologies can be divided into two categories: those which save time by reducing working per iteration and those which do so by reducing the number of iterations. For a general review of these celebrated approaches, refer to \[12–14\].

We are now in a position to summarize the definition of PageRank. A detailed description of how the Google matrix is formed can be found in \[13, 14\]. The mathematical model is based on the underlying assumption that the importance of a webpage is determined by the importance of webpages that link to it. Let us consider the hyperlink structure of the web as a desired graph. The Markov model represents this graph with an \(n \times n\) matrix \(P\) whose element \(p_{ij}\) is the probability of moving from page \(i\) to page \(j\) in one time step. However, if page \(i\) has no outlinks, then the \(i\)th row of \(P\) is zero, such a page is called a dangling node. It can be a pdf file or a page whose links have not yet been crawled. So as to transform \(P\) into a stochastic matrix \(\tilde{P}\) \[15\], we can replace every zero row in \(P\) with a vector \((1/n)e^T\), where \(e^T\) is a positive row vector of all ones. We thus define \(\tilde{P} = P + de^T/n\), so that all zeros rows are eliminated, where the \(n\)-dimensional column vector \(d\) is the dangling page indicator, whose elements \(d_i = 1\) if the \(i\)th row of \(P\) corresponding to a dangling node, and 0 otherwise.

The Google matrix is defined as a convex combination of \(\tilde{P}^T\) and the rank-one matrix \(E^T = ev^T\)

\[
A = \alpha \tilde{P}^T + (1 - \alpha)E^T
\]

\[
= \alpha P^T + \frac{\alpha}{n}de^T + (1 - \alpha)ev^T
\] (1)

where \(v\) is a positive vector whose elements sum to one and \(\alpha (0 < \alpha < 1)\) is the damping factor. The damping factor is originally set to 0.85, which models the possibility that a web surfer jumps from one page to the next without necessarily following a link. After these modifications, the matrix \(A\) is both column-stochastic and irreducible, then it has a positive real simple eigenvalue equal to its spectral radius, associated with a unique right non-negative eigenvector \[15, 16\]. It is the matrix \(A\) called the Google matrix, whose stationary distribution is the PageRank vector.

The damping factor \(\alpha\) is a real positive number that is close to 1. The question of what value of \(\alpha\) is the ‘correct’ value and what gives a meaningful ranking is subject to ongoing investigation, one can see \[17\] for a recent study. It was argued in \[13\] that the PageRank vector derived from larger \(\alpha\) such as 0.99 perhaps gives a ‘truer’ PageRanking than \(\alpha = 0.85\) does. In practice, it may therefore make sense to consider a large range of values \[5\]. The smaller the damping factor is, the easier it is to compute the PageRank vector by simple ways such as the power method (within a
modest iterations). On the other hand, the closer the damping factor is to 1, the closer the Google matrix is to the original web link graph. If the largest eigenvalue is not separated well from the second one, which happens when the damping factor is sufficiently close to 1, then it is necessary to develop more sophisticated approaches than the power method for computing PageRank. In this paper we will focus our attention on high values of $\alpha$.

Large eigenvalue computation is a fruitful area since there exists many excellent algorithms [18], such as the Krylov subspace methods. The Arnoldi-type method proposed by Golub and Grief [5], is an instance of such a method. It is an explicitly restarted Krylov subspace method based on a combination of Arnoldi and the singular value decomposition (SVD) [1] that relies on the knowledge of the dominant eigenvalue. In this paper, we focus on another Arnoldi-based algorithm, that is, the thick restarted Arnoldi algorithm [19, 20]. This method is mathematically equivalent to the famous implicitly restarted Arnoldi advocated by Sorensen [21], but has a simpler implementation. Moreover, this approach is a little bit similar to the Arnoldi-E method [22], except for the approximate eigenvectors are put firstly. Some nice properties of this approach are that the entire subspace being still a Krylov subspace, and it includes small Krylov subspaces with Ritz vectors as starting vectors [20, 22, 23]. Furthermore, $p$ matrix–vector multiplications can be saved before restarting, where $p$ is the number of Ritz vectors from the previous iteration. For more details, we refer to [19, 20, 23] and the references therein.

In this paper, we first present a thick restated Arnoldi algorithm for computing PageRank. However, the algorithm is not satisfactory due to its heavy cost used per iteration. We therefore partially deal with this problem by periodically knitting the cheap power method together with the thick restarted Arnoldi algorithm. This idea stems from the fact that the Arnoldi method can be viewed as an accelerated power method. The resulting algorithm, called Power–Arnoldi, can be understood as a thick restarted Arnoldi algorithm preconditioned with power iteration, or a power method accelerated with thick restarted Arnoldi. The convergence speed of the new method is studied, which quantifies how and why the Power–Arnoldi method can improve over the power method for Google matrices. The sensitivity of the PageRank vector is also discussed.

This plan of this work is as follows. In Section 2, we briefly introduce the power method and the Arnoldi-type algorithm due to Golub and Grief [5]. In Section 3, we describe a thick restarted Arnoldi algorithm for computing PageRank. We give an efficient implementation in detail and discuss performance issues of our Power–Arnoldi algorithm in Section 4. In Section 5, we give insight into the convergence behaviour of our new algorithm, as well as the stability of the PageRank vector. Numerical examples are shown in Section 6 to verify our theoretical results and illustrate the efficiency of the Power–Arnoldi algorithm.

Let us introduce some notation. Throughout this paper, we denote by $I$ the identity matrix with the order clear from context, by $\sigma_{\text{min}}(X)$ the smallest non-zero singular value of a matrix $X$, and by $\Lambda(A)$ the spectrum of the matrix $A$. As PageRank vector is the stationary vector of a Markov chain, it is normalized so that its 1-norm is 1, thus the 1-norm is a reasonable choice for discussing PageRank problem [2]. However, for variants of the Arnoldi algorithm it is natural to use the 2-norm as a measure of convergence. Furthermore, the 2-norm does have a number of nice properties that makes it useful for both theoretical and practical purposes. Specifically, its unitary invariance is favourable for the Krylov subspace methods such as the Arnoldi method, in which an orthonormal basis is built. Therefore, we take the 2-norm $\| \cdot \|_2$ as our choice instead.
2. THE POWER METHOD AND THE ARNOLDI-TYPE ALGORITHM FOR COMPUTING PAGERANK

The power method [1] is one of the oldest methods for computing the dominant eigenvector of a given matrix. There are some good reasons for it being Google’s computational method of choice. Firstly, considering the matrix–vector product: given an \( n \)-dimensional vector \( x \), we have from (1) that

\[
Ax = \alpha P^T x + \left[ \frac{2}{n} (d^T x) + (1 - \alpha) (v^T x) \right] e
\]

Therefore, neither \( \tilde{P} \) nor \( A \) needs to be formed or stored in practice. It becomes clear that a matrix-free method such as the power method applied to \( A \) can be implemented with only matrix–vector multiplications on the large sparse matrix \( P^T \). Secondly, the power method only requires the storage of two vectors per iteration, while the Krylov subspace methods, such as the Arnoldi method [5, 18, 21, 24] or the GMRES method [23, 25], require to store at least several vectors at each iteration, depending on the size of subspace chosen. The basic algorithmic version of the power method is described as follows.

**Algorithm 1** (The power method for computing PageRank)

1. **Start**: Choose a unit positive starting vector \( x^{(0)} \), and a prescribed tolerance \( tol \), set \( k = 1, r = 1; \)
2. **Iterate**: 
   
   while \( r > tol \)
   
   \[
x^{(k)} = Ax^{(k-1)}; \\
r = \|x^{(k)} - x^{(k-1)}\|_2; \\
x^{(k)} = x^{(k)} / \|x^{(k)}\|_2; \\
k = k + 1;
   \]

end

The Arnoldi method is a common technique for finding a few eigenpairs using a Krylov subspace. Given an initial vector \( v_1 \) of norm one, if computations are performed in exact arithmetic, then the Arnoldi’s process generates successively an orthonormal basis \( V_m = [v_1, \ldots, v_m] \) of the Krylov subspace

\[
\mathcal{K}_m(A, v_1) = \text{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}
\]

In this subspace, the restriction of \( A \) to \( \mathcal{K}_m(A, v_1) \) is represented by an \( m \times m \) upper Hessenberg matrix \( H_m \) with the entries \( h_{i,j} \). Furthermore, the following relations hold:

\[
AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T
\]

\[
= V_{m+1} \tilde{H}_m
\]

where \( e_m \) is the \( m \)th co-ordinate vector of dimension \( m \) and \( \tilde{H}_m \) is an \((m+1) \times m\) upper Hessenberg matrix, which is the same as \( H_m \) except for an additional row whose unique non-zero entry is \( h_{m+1,m} \). Now we briefly provide the essential details of the Arnoldi process, for more details, refer to [18, 24].

DOI: 10.1002/nla
Algorithm 2 (The Arnoldi process)
1. Start: Given the initial vector $v_1$ of norm one, and the steps $m$ of the Arnoldi process;
2. Iterate:
   for $j = 1, \ldots, m$
   $q = Av_j$;
   for $i = 1, \ldots, j$
   $h_{i,j} = v_i^T q$;
   $q = q - h_{i,j}v_i$;
   end for
   $h_{j+1,j} = \|q\|_2$;
   if $h_{j+1,j} == 0$
   break;
   end if
   $v_{j+1} = q / h_{j+1,j}$;
end for

Golub and Grief [5] presented a celebrated Arnoldi-type algorithm that is based on a combination of Arnoldi process and the SVD that relies on the knowledge of the largest eigenvalue of the matrix. It is well known that the largest eigenvalue of a Google matrix is 1 [16]. For this known eigenvalue, they seek an unit norm vector $x^R$ satisfying the following optimal property:

$$\|(A - I)x^R\|_2 = \min_{u \in \mathbb{C}^{m}(A,v_1), \|u\|_2 = 1} \|(A - I)u\|_2$$  \hfill (5)

The solution of the above minimization problem resorts to solving the SVD of the $(m + 1) \times m$ matrix $\tilde{H}_m - [I; O]$, where $[I; O]$ is the same as the $m \times m$ identity matrix $I$ except for an additional zero row at the bottom. The residual norm is given by [5]

$$\|Ax^R - x^R\|_2 = \sigma_{\min}(\tilde{H}_m - [I; O])$$  \hfill (6)

where $\sigma_{\min}(\cdot)$ denotes the smallest non-zero singular value of a matrix. In summary, the Arnoldi-type algorithm is outlined as follows, for more details, refer to [5].

Algorithm 3 (The Arnoldi-type algorithm for computing PageRank)
1. Start: Given an initial guess $v$ of norm one, the Arnoldi steps number $m$, as well as a prescribed tolerance $tol$;
2. Iterate:
   do
   (2.1) Run Algorithm 2 for the computation of $V_m$ and $\tilde{H}_m$;
   (2.2) Compute small SVD: $\tilde{H}_m - [I; O] = U \Sigma S^T$;
   (2.3) Compute the approximate eigenvector: $v = V_m \cdot S(:, m)$;
   while $\sigma_{\min}(\tilde{H}_m - [I; O]) > tol$

3. A THICK RESTARTED ARNOLDI ALGORITHM FOR COMPUTING PAGERANK

In this section, we focus on another Arnoldi-based approach. This is the thick restarted Arnoldi method proposed by Wu and Simon [19]. It is mathematically equivalent to the famous implicitly restarted Arnoldi method due to Sorensen [21]. The thick-restarting strategy is less complicated...
than the implicit-restarting strategy, and there is no need for the purging routine that the implicitly restarted Arnoldi used to control some of the roundoff errors [26].

During each iteration after the first, the thick restarted Arnoldi method generates an orthonormal basis \( V_m \) for the subspace

\[
\text{span}\{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_p, v_{m+1}, A v_{m+1}, \ldots, A^{m-p} v_{m+1}\}
\]

where \( \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_p \) are the Ritz vectors and \( v_{m+1} \) is the \((m+1)\)th Arnoldi basis vector, all of them are from the previous iteration. It was proven that this subspace is a Krylov subspace [20], and it is equivalent to

\[
\text{span}\{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_p, A \tilde{x}_i, A^2 \tilde{x}_i, \ldots, A^{m-p} \tilde{x}_i\}, \quad 1 \leq i \leq p
\]

Thus, it contains small Krylov subspaces with each of the desired Ritz vectors as starting vectors. We now give a thick restarted Arnoldi algorithm for computing PageRank, for more details and discussions, refer to [19, 20].

**Algorithm 4** (A thick restarted Arnoldi algorithm for computing PageRank)

1. **Start:** Specify \( m \), the maximum size of the subspace, and \( p \), the number of approximate eigenvectors which are retained from one iteration to the next. Choose an initial vector \( v_1 \) of norm one for the Arnoldi process, and a prescribed tolerance \( tol \);
2. **The first iteration:** Apply Algorithm 2 to form \( V_{m+1} \) as well as \( \tilde{H}_m \). Compute all the eigenpairs \((\tilde{x}_i, y_i)\), \( i = 1, \ldots, m \) of the \( m \times m \) projection matrix \( H_m \), then select \( p \) largest of them and goto Step 6;
3. **Step 6:** Apply the Arnoldi procedure from the current point to form \( V_{m+1} \) as well as the \((m+1) \times m \) matrix \( H_m \), where the current point is \( v_{p+1} \);
4. **Compute all the eigenpairs** \((\tilde{x}_i, y_i)\), \( i = 1, \ldots, m \) of the \( m \times m \) projection matrix \( H_m \), and select \( p \) largest of them;
5. **If the largest eigenpair is accurate enough** (i.e. if \( h_{m+1,p} |y_1| < tol \), see (13) below), then take \( \tilde{x}_1 = v_m y_1 \) as an approximation to the PageRank vector and stop, else continue;
6. **Orthonormalization of \( p \) short vectors.** Orthonormalize the \( y_i \)'s, \( i = 1, \ldots, p \) to form a real \( m \times p \) matrix \( W_p = [w_1, \ldots, w_p] \): First separate \( y_i \) into real part and imaginary part if it is complex, both parts should be included, and adjust \( p \) if necessary (increase or decrease \( p \) by 1, see Remark 2 below);
7. **Extend \( W_p \) to an \((m+1) \times p\) matrix** \( \tilde{W}_p \) by appending a zeros row at the bottom, and set \( W_{p+1} = [\tilde{W}_p, e_{m+1}] \), where \( e_{m+1} \) is the \((m+1)\)th co-ordinate vector. Note that \( W_{p+1} \) is an \((m+1) \times (p+1) \) matrix and is orthonormal;
8. **Use the old \( \tilde{H}_m \) and \( V_{m+1} \) to form portions of the new \( \tilde{H}_m \) and \( V_{m+1} \):** Let \( \tilde{H}_p = W^T_{p+1} \tilde{H}_m W_p \) and \( V_{p+1} = V_{m+1} W_{p+1} \), then set \( \tilde{H}_p = \tilde{H}_p^\text{new} \) and \( V_{p+1} = V_{p+1}^\text{new} \), goto Step 3.

**Remarks**

1. Keep in mind from Step 2 that the first iteration of the thick restarted Arnoldi algorithm is the standard Arnoldi iteration;
2. Note from Step 6 that \( p \) is variable during iterations. In our implementation, we reset \( p \) to be the initial size at restarting if it was changed at the previous iteration;
3. We see from Step 8 that because the first \( p \) vectors of the new \( V_{m+1} \) are all from the previous subspace, so the orthonormalization in Step 6 can be realized with short vectors of length \( m \).

At each iteration, some recurrences similar to the Arnoldi recurrences (3) and (4) are generated by the thick restarted Arnoldi algorithm. This was shown by Mogan and Zeng [20] but without proof, we give a proof here.

**Theorem 3.1**
Under the above notation, the following relation holds for the thick restarted Arnoldi algorithm

\[
AV_m = V_m H_{m,m} + h_{m+1,m} v_{m+1} e_m^T
\]

\[
= V_{m+1} \tilde{H}_m
\]

**Proof**
It follows from (3) and (4) that (7) and (8) hold in the standard Arnoldi iteration. Thus, the relations are established in the first iteration. At each iteration after the first, it is sufficient to prove

\[
AV_{p+1}^{\text{new}} = V_{p+1}^{\text{new}} \tilde{H}_{p+1}^{\text{new}}
\]

in that the other \( m - p \) standard Arnoldi steps are continued with the initial vector \( v_{p+1} \).

By induction, we assume that (7) holds in the previous iteration. So we have

\[
AV_m = V_m H_{m,m} + h_{m+1,m} v_{m+1} e_m^T
\]

and the Rayleigh–Ritz residuals are

\[
AV_m y_i = \tilde{\lambda}_i V_m y_i + h_{m+1,m} (e_m^T y_i) v_{m+1}, \quad i = 1, \ldots, p
\]

From (10), it is easy to see that

\[
AV_m y_i \subset \text{span}\{y_i, v_{m+1}\}, \quad i = 1, \ldots, p
\]

From Steps 6–8 of Algorithm 4, we obtain

\[
AV_m W_p \subset \text{span}\{W_p, v_{m+1}\}
\]

\[
= \text{span}\{V_{m+1} W_{p+1}\}
\]

Therefore, there exists a \((p + 1) \times p\) matrix \( Q_p \) such that

\[
AV_m W_p = A V_{p+1}^{\text{new}}
\]

\[
= (V_{m+1} W_{p+1}) Q_p
\]

\[
= V_{p+1}^{\text{new}} Q_p
\]
The condition \((V_{p+1}^{\text{new}})^T V_{p+1}^{\text{new}} = I\) leads to

\[ Q_p = (V_{p+1}^{\text{new}})^T A V_p^{\text{new}} = \tilde{H}_p^{\text{new}} \tag{12} \]

which completes the proof.

Let \((\tilde{\lambda}_1, y_1)\) be the dominant eigenpair of \(H_m\). Then the thick restarted Arnoldi method uses \(\tilde{x}_1 = V_m y_1\) as an approximation to the PageRank vector. By (7), the residual norm is given by

\[ \| (A - \tilde{\lambda}_1 I) \tilde{x}_1 \|_2 = h_{m+1,m} \cdot |e_m^T y_1| \tag{13} \]

which provides a cheap stopping criterion for the thick restarted Arnoldi algorithm.

We now discuss briefly the time and space complexity of the thick restarted Arnoldi algorithm compared with the power method and the Arnoldi-type algorithm. It is well known that the power method requires only one matrix–vector multiplication and the storage of two length-\(n\) vectors per iteration [1]. The thick restarted Arnoldi algorithm requires about the same storage as the Arnoldi-type algorithm does, both requiring to store \(m + 1\) length-\(n\) vectors, where \(m\) is the dimension of the subspace chosen.

One superiority of the thick restarted Arnoldi algorithm over the Arnoldi-type algorithm is that the former does need only \(m - p\) matrix–vector multiplications at each iteration after the first, while the latter uses \(m\). This is favourable for extremely large sparse matrices such as Google matrices. Therefore, the thick restarted Arnoldi algorithm is cheaper than the regular explicitly restarted Arnoldi algorithm per iteration. Another important reason that the thick restarted Arnoldi algorithm is more promising than the regular version such as the Arnoldi-type algorithm is that it improves the convergence rate by effectively increasing the gap ratio \([20, 22, 23]\).

4. A POWER–ARNOLDI ALGORITHM FOR COMPUTING PAGERANK

The development of techniques for computing PageRank efficiently and reliably for web-scale graphs is crucial for many reasons [2]. For instance, computing PageRank rapidly is necessary to reduce the lag time from when a new crawl is completed to when that crawl can be made available for searching. Moreover, recent approaches to personalized and topic-sensitive PageRank schemes [27] require computing many PageRank vectors, each biased towards certain type of pages. All these require to seek faster algorithms for computing PageRank.

Applying a certain number of steps of the thick restarted Arnoldi algorithm is more computationally intense than applying the same number of iterations of the power method, and we have to keep \(m\) small. Therefore, both the power method and the thick restarted Arnoldi algorithm are unsatisfactory for computing PageRank in some sense. In this section, we consider how to combine the power method with the thick restarted Arnoldi algorithm effectively. The resulting algorithm, called Power–Arnoldi, can be viewed as a thick restarted Arnoldi algorithm preconditioned with power iteration, or a power method accelerated with thick restarted Arnoldi. The hope is that, on the one hand, the resulting algorithm will be much cheaper than thick restarted Arnoldi algorithm per iteration, on the other hand, it will converge within fewer iterations than the power method does in case of high damping factors. The following theorem offers the feasibility of our new algorithm.
The mechanism of the Power–Arnoldi algorithm can be described as follows: given a positive vector $x^{(0)}$, we first run the power method to get a rough convergence. If the convergence is below the prescribed tolerance, then we use the resulting vector as the initial guess, and iterate the thick restarted Arnoldi algorithm a few times (say, 2–3 times) to get another approximation. If this approximation is still unsatisfactory, return to the power method again, using the new approximation as the starting vector. Proceeding the above procedure analogously until the described accuracy is achieved.

The key problem is when and how to terminate the power iteration and trigger the thick restarted Arnoldi in the Power–Arnoldi algorithm. A number of strategies can be coined from which we choose a simple and easily realized one. The strategy is closely related to the (asymptotic) convergence rate of the power method. It is well known that the asymptotic convergence rate of the power method is determined by $|\lambda_2|/|\lambda_1|$, the ratio of the magnitude of the subdominant eigenvalue and that of the dominant eigenvalue of the matrix [1]. The smaller the ratio is, the faster the convergence will be. Since the largest eigenvalue of a Google matrix is 1, the convergence rate of the power method will strictly depend on $|\lambda_2|$. By Theorem 4.1, $|\lambda_2|\leq \alpha$, which implies that the convergence rate of the power method is bounded from above by $\alpha$.

In the Power–Arnoldi algorithm, we use three parameters $\beta$, restart and maxit to flip-flop between the power method and the thick restarted Arnoldi. Denote by $r^{(\text{curr})}$ the residual norm of the current power iteration and by $r^{(\text{prev})}$ that of the previous power iteration. We then check whether $\text{ratio} = r^{(\text{curr})}/r^{(\text{prev})}$ is greater than $\beta$, if so, let restart := restart + 1 and examine whether restart is larger than the pre-determined number maxit. If so, terminate the power iteration and trigger the thick restarted Arnoldi with $x^{(\text{curr})}$ as the initial vector. Otherwise, keep on running the power method. In summary, we have an algorithm that computes PageRank.

**Algorithm 5 (A Power–Arnoldi algorithm for computing PageRank)**

1. **Initialize**: Given $m$, the size of the subspace, and $p$, the number of approximate eigenvectors which are retained from one cycle to the next. Choose a positive vector $x^{(0)}$, a prescribed tolerance tol, two parameters $\beta$ and maxit used to control power iterations, set $k = 1$, restart = 0, and $r = 1$, $r_0 = r$.

2. **Run the power method:**
   
   while restart < maxit & $r > \text{tol}$
   
   $x^{(k)} = Ax^{(k-1)}$;
   
   $r = \|x^{(k)} - x^{(k-1)}\|_2$;
   
   $x^{(k)} = x^{(k)}/\|x^{(k)}\|_2$;
   
   if $r <= \text{tol}$
   
   break;

   end if

   if $r/r_0 > \beta$ & $r^{(\text{curr})}/r^{(\text{prev})}$
   
   restart := restart + 1;

   end if

end if

\[ r_0 := r; \]

\[ k := k + 1; \]

end while

3. Run Algorithm 4 for a few times (say, 2–3 times): Iterate Steps 2–8 of Algorithm 4 for the first run and Steps 3–8 otherwise. If the residual norm satisfies the prescribed tolerance \( \text{tol} \), then stop; else continue.

4. Run the power method again with \( \tilde{x}_1 \) as the initial guess, where \( \tilde{x}_1 = V(:,1) \) is the approximate vector obtained from the thick restarted Arnoldi:

\[ \text{restart} := 0, \]

while \( \text{restart} < \text{maxit} \) \& \( r > \text{tol} \)

(4.1) \[ r_0 := r; \quad \text{ratio} := 0; \quad r_1 := r; \]

(4.2) while \( \text{ratio} < \beta \) \& \( r > \text{tol} \)

(4.3) \[ x^{(k)} = A x^{(k-1)}; \]

(4.4) \[ r = \| x^{(k)} - x^{(k-1)} \|_2; \]

(4.5) \[ \text{ratio} := r/r_0; \quad \% r^{(curr)}/r^{(prev)}; \]

(4.6) \[ r_0 := r; \]

(4.7) \[ x^{(k)} = x^{(k)}/\| x^{(k)} \|_2; \]

(4.8) \[ k := k + 1; \]

(4.9) end while

(4.10) if \( r/r_1 > \beta \) \% \( r^{(curr)}/r^{(prev)} \)

(4.11) \[ \text{restart} := \text{restart} + 1; \]

(4.12) end if

end while

If \( r \leq \text{tol} \), stop, else goto Step 3.

We will now explain the successive main phases of this algorithm.

1. This is the initialization phase of the algorithm. If a good guess for the PageRank vector is available, then we use it. Note that \( r \) stands for the residual norm of the current iteration and \( r_0 \) for that of the previous iteration. The choice of \( \beta \) is crucial in practice. It is well known that the convergence rate of the power method is \( |r_2|/|r_1| \). For Google matrix, \( r_1 = 1 \), moreover, it has been proven that \( |r_2| \leq \alpha \). Therefore, it is reasonable from a theoretical point of view that \( \beta \) should be smaller than \( \alpha \). We can choose, say, \( \beta = \alpha - 0.1 \) or \( \alpha - 0.2 \), and \( \text{maxit} = 6 \), all that is needed is to satisfy \( 0 < \beta < |r_2| \).

2. We first iterate a few times using the power method to get a rough convergence. Notice that this procedure is performed only once throughout the Power–Arnoldi algorithm.

3. If the desired accuracy is not achieved, run the thick restarted Arnoldi algorithm using the approximation \( x^{(k)} \) obtained from the power method as the initial guess.

4. If the convergence is still not satisfactory, run the power method using the Ritz vector \( \tilde{x}_1 \) obtained from thick restarted Arnoldi as the initial vector. Here steps (4.10)–(4.12) are devised to controlling when to stop the power method and trigger the thick restarted Arnoldi. Steps (4.2)–(4.9) are inner iterations for managing power iterations, the trick exploited here is similar to that exploited in Step 2. Heuristically, numerical experiments indicate that these options are beneficial to minimizing the overhead.
Next, we consider the storage requirements and computational cost of the Power–Arnoldi algorithm. Generally speaking, the storage requirements are approximately $m + 1$ length-$n$ vectors in the thick restarted Arnoldi phase and two in the power iteration phase. The main cost of the algorithm consists of the matrix–vector multiplications, which depend on the number of non-zeros of the matrix. In the thick-restarted Arnoldi phase, $m - p$ matrix–vector multiplications are required per iteration (after the first), while in the power iteration phase, only one matrix–vector multiplication is needed. Table I summarizes the main computational cost and storage required by the power method, the Arnoldi-type algorithm, the quadratic-extrapolation method [2] and the Power–Arnoldi algorithm, respectively.

It is seen that applying the Power–Arnoldi algorithm is more computationally intense than applying the same number of iterations of the power method, and more attention should be paid to the trade-off between speedup and storage. We hope that the Power–Arnoldi algorithm can converge within fewer iterations than the power method in case of a high damping factor, while the power method converges slowly.

Compared with the power method, two remarkable merits of our new algorithm are flexibility and fast convergence. Another big advantage of the Power–Arnoldi algorithm over the power method is its parallelizability. The superiority of the Power–Arnoldi algorithm over the Arnoldi-type algorithm is the (relatively) cheap cost used per iteration.

The quadratic-extrapolation method proposed by Kamvar et al. [2], accelerates the convergence of the power method by periodically subtracting off estimates of the non-principal eigenvectors from the current iteration of the power method. At the first glance, it seems that the Power–Arnoldi method and the quadratic-extrapolation method are related ideas of applying an expensive speedup step every $k$ steps. However, our strategy focuses on how to efficiently combine some current algorithms that are at odds with one another into an excellent algorithm. In fact, some acceleration methods such as the extrapolation methods [2, 29] can also be combined with the thick restarted Arnoldi algorithm or the Arnoldi-type algorithm using an analogous technique [30]. We would expect the resulting algorithms to work much better than their original counterparts.

There is a lot of unsolved problems and thus, open questions in the field of research are discussed in this topic. For instance, do the eigenvalues $\lambda_2, \ldots, \lambda_p$ related to cyclicity of the original stochastic matrix? If yes, then how? In a non-favourite case reducing the influence of the eigenvalues mentioned may not lead to acceleration of the procedures, only to elevating the price of computations. This is the case when, say $\lambda_2, \ldots, \lambda_s, s < p$ belong to a (large) irreducible components of the generally reducible original matrix $\tilde{P}$. Another even less favourite case appears if some index of cyclicity $c$ exceeds the chosen value $p$. Since the eigenvalues just mentioned form the part of spectrum of the Google matrix being responsible for the rate of convergence, i.e. the moduli of those eigenvalues equal just $\tilde{z}$, the damping factor of the Google matrix, it would be interesting to know whether their influence is under control. In fact, the sub-dominant eigenvalues cannot be deflated effectively in this situation, and it is preferable from a theoretical point of view.
that the chosen value \( p \) should be no less than the index of cyclicity \( c \). A block Krylov subspace method may be a good choice \([31]\), and much work should be done. However, it seems that our strategy can still accelerate the convergence of the Arnoldi method with a minimal overhead, see the analysis made in Theorem 5.2 and the remark therein.

Finally, we would like to remind the reader that the basic idea of switching among different methods is not novel, for more details about this type of methods, refer to \([32]\).

5. SOME THEORETICAL RESULTS

The contribution of this section is twofold. Firstly, we analyse the speed of convergence of the Power–Arnoldi method, and quantify how and why the Power–Arnoldi method improves on the power method and the Arnoldi method for Google matrices necessarily. Secondly, we give insight into the sensitivity and stability of the PageRank vector in terms of its 2-norm condition number.

5.1. On the convergence speed of the Power–Arnoldi method

The convergence of the power method and that of the thick restarted Arnoldi method can be found in \([1]\) and \([21, 22]\), respectively. Our analysis focuses on the procedure when turning from the power method to the thick restarted Arnoldi. Numerical experiments show that the residual norm often decreases dramatically during this procedure, we will try to explain why this happens.

Without loss of generality, we assume that the eigenvalues of \( A \) are arranged in the decreasing order

\[
1 = |\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|
\]

Let \( \mathscr{P}_{m-1} \) represent the set of polynomials of degree not exceeding \( m - 1 \), then the following theorem due to Saad \([24]\) depicts the convergence speed of the standard Arnoldi method applied to the PageRank problem.

**Theorem 5.1 (Saad \([24]\))**

Assume that \( A \) is diagonalizable and that the initial vector \( v_1 \) in Arnoldi’s method has the expansion

\[
v_1 = \sum_{i=1}^{n} \gamma_i x_i
\]

with respect to the eigenbasis \( \{x_1, x_2, \ldots, x_n\} \) in which \( \|x_i\|_2 = 1, i = 1, \ldots, n \) and \( \gamma_1 \neq 0 \). Then the following inequality holds

\[
\| (I - \mathcal{P}_m) x_1 \|_2 \leq \zeta \varepsilon_m \tag{14}
\]

where \( \mathcal{P}_m \) is the orthogonal projector onto the subspace \( \mathcal{H}_m(A, v_1) \)

\[
\zeta = \sum_{j=2}^{n} \frac{|\gamma_j|}{|\gamma_1|}
\]

and

\[
\varepsilon_m = \min_{p \in \mathcal{P}_{m-1}} \max_{\lambda \in \sigma(A) \setminus \{\lambda_1\}} |p(\lambda)|
\]

Let \( v_1 \) be the initial vector for the power method, which is from the previous thick restarted Arnoldi. With this vector at hand, the power method produced the vector \( v_1^{\text{new}} = \eta A^\ell v_1 \), where
\( \ell \geq \text{maxit} \) and \( \eta = 1/\|A^\ell v_1\|_2 \) is the normalizing factor. In the next cycle of the Power–Arnoldi algorithm, \( v_1^{\text{new}} \) will be used as the initial vector for an \( m \)-step standard Arnoldi process (cf. Step 2 in Algorithm 4, that is, the first iteration of the thick restarted Arnoldi algorithm), so that the new Krylov subspace

\[ \mathcal{K}_m(A, v_1^{\text{new}}) = \text{span}(v_1^{\text{new}}, Av_1^{\text{new}}, \ldots, A^{m-1}v_1^{\text{new}}) \]

will be constructed. The following theorem shows the speed of the convergence of this procedure.

**Theorem 5.2**

Under the assumptions and notation of Theorem 5.1, if we denote by \( \tilde{P}_m \) the orthogonal projector onto the subspace \( \mathcal{K}_m(A, v_1^{\text{new}}) \), then

\[ \| (I - \tilde{P}_m)x_1 \|_2 \leq \alpha^\ell \cdot \zeta_m \]

with \( \ell \geq \text{maxit} \).

**Proof**

We note that for any \( u \in \mathcal{K}_m(A, v_1^{\text{new}}) \), there exists \( q(x) \in \mathcal{Q}_{m-1} \) such that

\[ u = \eta A^\ell \cdot q(A)v_1 \]

\[ = \eta A^\ell \cdot q(A) \left( \gamma_1 x_1 + \sum_{j=2}^{n} \gamma_j x_j \right) \]

\[ = \eta A^\ell \cdot \left( \gamma_1 q(\lambda_1)x_1 + \sum_{j=2}^{n} \gamma_j q(\lambda_j)x_j \right) \]

where \( v_1 = \sum_{i=1}^{n} \gamma_i x_i \) is the expansion of \( v_1 \) within the eigen-basis \([x_1, x_2, \ldots, x_n]\). Hence,

\[ \frac{u}{\eta \gamma_1 q(1)} - x_1 = \sum_{j=2}^{n} \frac{\gamma_j}{\gamma_1} \cdot \frac{q(\lambda_j)}{q(1)} \cdot \lambda_j^\ell x_j \]

where we used the facts \( \lambda_1 = 1 \) and \( Ax_1 = x_1 \). So we have

\[ \left\| \frac{u}{\eta \gamma_1 q(1)} - x_1 \right\|_2 \leq \sum_{j=2}^{n} \left| \frac{\gamma_j}{\gamma_1} \right| \cdot \frac{|q(\lambda_j)|}{|q(1)|} \cdot |\lambda_2|^\ell \]

\[ \leq \alpha^\ell \cdot \sum_{j=2}^{n} \left| \frac{\gamma_j}{\gamma_1} \right| \cdot |p(\lambda_j)| \]

\[ \leq \alpha^\ell \cdot \sum_{j=2}^{n} \left| \frac{\gamma_j}{\gamma_1} \right| \cdot \max_{j \neq 1} |p(\lambda_j)| \]

where we used \( |\lambda_2| \leq \alpha \) and \( p(\lambda) = q(\lambda)/q(1) \) satisfying \( p(1) = 1 \).
Therefore,
\[ \| (I - \tilde{\mathcal{P}}_m) x_1 \|_2 = \min_{u \in \mathcal{N}_m(A, v_{1}^{\text{new}})} \| u - x_1 \|_2 \]
\[ \leq \| \mathcal{P} \| \cdot \min_{p \in \mathcal{P}_{m-1}} \max_{\lambda \in \Lambda(A) \setminus \{ \lambda_1 \}} | p(\lambda) | \]
which completes the proof. \(\square\)

**Remark**
Compared with (14), this theorem shows that the Power–Arnoldi method can increase the speed of convergence by a factor of at least \(\| \mathcal{P} \| \) when turning from the power method to the thick restarted Arnoldi. This quantifies how and why the Power–Arnoldi method improves over the power method and the Arnoldi method for computing PageRank.

Now we consider an ideal case. Suppose for the moment that we know the exact eigenvectors \(x_2, x_3, \ldots, x_p\), so that the subspace constructed by the previous thick restarted Arnoldi is
\[ \text{span}\{\tilde{x}_1, A\tilde{x}_1, \ldots, A^{m-p}\tilde{x}_1, x_2, x_3, \ldots, x_p\} \] (16)
where \(\tilde{x}_1\) is the Ritz vector obtained from the thick restarted Arnoldi and has the expansion
\[ \tilde{x}_1 = \gamma_1 x_1 + \gamma_2 x_2 + \cdots + \gamma_n x_n \]
If we pick
\[ v_1 = v \left( h(A) \tilde{x}_1 + \sum_{j=2}^{p} \mu_j x_j \right) \]
from the subspace (16), where \(h(x) \in \mathcal{P}_{m-p}\) is a polynomial that satisfies \(h(\lambda_1) = 1, \mu_j + \gamma_j h(\lambda_j) = 0, 2 \leq j \leq p, \) and \(|h(\lambda_j)|, p + 1 \leq j \leq n,\) are as small as possible, and \(v\) is the normalizing factor. Using this vector as the initial vector, the power iterations generate the initial guess \(v_{1}^{\text{new}} = \tau A^\ell v_1\) for the thick restarted Arnoldi algorithm, where \(\ell \geq \text{maxit}\) and \(\tau\) is the normalizing factor.

Notice that for any \(u \in \mathcal{N}_m(A, v_{1}^{\text{new}})\), there is a polynomial \(q(x) \in \mathcal{P}_{m-1}\) such that
\[ u = q(A) v_{1}^{\text{new}} \]
\[ = \tau q(A) \cdot A^\ell \cdot \left( h(A) \tilde{x}_1 + \sum_{j=2}^{p} \mu_j x_j \right) \]
\[ = \tau A^\ell \cdot \left( q(A) h(A) \tilde{x}_1 + \sum_{j=2}^{p} \mu_j q(A) x_j \right) \]
It is easy to see that
\[ q(A) h(A) \tilde{x}_1 + \sum_{j=2}^{p} \mu_j q(A) x_j = q(A) h(A) \left( \sum_{j=1}^{n} \gamma_j x_j \right) + \sum_{j=2}^{p} \mu_j q(\lambda_j) x_j \]
here we used \( h(1) = 1 \) and \( \mu_j + \gamma_j h(\lambda_j) = 0 \), \( j = 2, \ldots, p \).

Therefore, if \( \gamma_1 \cdot q(1) \neq 0 \), we obtain

\[
\frac{u}{\tau \cdot \gamma_1 q(1)} - x_1 = \sum_{j=p+1}^{n} \left( \frac{\gamma_j}{\gamma_1} \cdot \frac{q(\lambda_j)}{q(\lambda_1)} \right) h(\lambda_j) \lambda_j^\ell x_j
\]

and

\[
\left\| \frac{u}{\tau \cdot \gamma_1 q(1)} - x_1 \right\|_2 \leq \alpha^\ell \cdot \max_{p+1 \leq j \leq n} |h(\lambda_j)| \cdot \sum_{j=p+1}^{n} \left| \frac{\gamma_j}{\gamma_1} \right| \cdot \max_{p+1 \leq j \leq n} |p(\lambda_j)|
\]

here we used \(|\lambda_j| \leq \alpha\), \(j \geq 2\) and \(p(\lambda) = q(\lambda)/q(\lambda_1)\) satisfying \(p(1) = 1\). If we denote by \( \tilde{P}_m \) the orthogonal projector onto the subspace \( \mathcal{N}_m(A, v_1^{\text{new}}) \), then we obtain the theorem as follows.

**Theorem 5.3**

If \( A \) is diagonalizable, then under the above notation, there holds

\[
\left\| (I - \tilde{P}_m) x_1 \right\|_2 \leq \alpha^\ell \rho \cdot \min_{\substack{p \leq \lambda_{m-1} \in \Lambda(A) \setminus \{\lambda_1, \ldots, \lambda_p\}}} \max_{\ell \geq \text{maxit}} |p(\lambda)|
\]

where \( \rho = \max_{p+1 \leq j \leq n} |h(\lambda_j)| \cdot \sum_{j=p+1}^{n} |\gamma_j/\gamma_1| \) and \( \ell \geq \text{maxit} \).

Theorems 5.2 and 5.3 justify the idea of combining the power method with the thick restarted Arnoldi from a theoretical point of view. It was shown in [22] that adding approximate eigenvectors to the search subspace can be almost beneficial, so that the eigenvalues \( \lambda_2, \lambda_3, \ldots, \lambda_p \) can be effectively eliminated from the spectrum of \( A \), even before they are very accurate. Theorem 5.3 indicates that, in the Power–Arnoldi algorithm, if we can choose a proper initial vector for the power method from the subspace generated by the thick restarted Arnoldi, then the deflation can also occur when turning from the power method to the thick restarted Arnoldi. Although it is purely theoretical and impossibly so lucky in practice, the result has a key impact on guiding us how to select the initial vector from thick restarted Arnoldi efficiently, and much work should be done about it.

### 5.2. On the sensitivity of the PageRank vector

The PageRank problem in its general linear system form is presented in [8, 13, 33]

\[
(I - \alpha \tilde{P}^T)x = qv
\]  

(17)

where \( q \) is a scalar related to \( \alpha \). It is proven in [13, 34] that the 1-norm condition number of the above linear system satisfies

\[
\kappa_1(I - \alpha \tilde{P}^T) = \frac{1 + \alpha}{1 - \alpha}
\]  

(18)
Therefore, as \( z \to 1 \), this linear system will become ill-conditioned, which means that a small change in the coefficient matrix may create a large change in the solution. However, the PageRank vector is an eigenvector for the associated Markov chain matrix. It was pointed out in \([35]\) that the ill-conditioning of the linear system does not imply the corresponding eigenproblem is also ill-conditioned. Therefore, what we should do is to examine the eigenvector sensitivity, not just the linear system sensitivity. In this section, we will give insight into the 2-norm condition number of the PageRank vector with an eye on the sensitivity of it.

Let \( X_\perp \) be an orthonormal basis for the orthogonal complement of the space spanned by the PageRank vector \( x \), so that \( [x, X_\perp] \) is unitary. So we have

\[
\begin{bmatrix}
  x^H \\
  X_\perp^H
\end{bmatrix}
A
\begin{bmatrix}
x \\
X_\perp
\end{bmatrix}
= 
\begin{bmatrix}
  1 & x^H A X_\perp \\
  O & A_2
\end{bmatrix}
\]

(19)

where \( A_2 = X_\perp^H A X_\perp \). Since \( Ax = x \) and \( X_\perp^H x = O \), the (1, 1) and (2, 1) elements of the right-hand side are 1 and \( O \), respectively. Denote by \( \lambda_2, \lambda_3, \ldots, \lambda_n \) the eigenvalues, other than 1, of \( A \). As we observe from (19), \( \lambda_2, \lambda_3, \ldots, \lambda_n \) are also eigenvalues of \( A_2 \). By the Perron–Frobenius theorem \([15, 16]\), 1 is a simple eigenvalue of \( A \), thus we can define

\[
\| \Sigma_\perp \|_2 \equiv \| X_\perp (I - A_2)^{-1} X_\perp^H \|_2 = \| (I - A_2)^{-1} \|_2
\]

(20)

which is the 2-norm condition number for the eigenvector \( x \) \([36]\). This condition number shows how the perturbations to \( A \) are related to perturbations to \( x \) in first order.

By Theorem 4.1, if \( \tilde{P} \) has at least two irreducible closed subsets, which is the case for the web hyperlink matrix, then the second eigenvalue of \( A \) is \( z \). Therefore, the gap

\[
\delta = \text{dist}(1, \Lambda(A) \setminus \{1\})
\]

\[
= \min_{\lambda_i \in \Lambda(A_2)} |1 - \lambda_i|
\]

\[
= 1 - z
\]

(21)

If we denote by \( J \) the Jordan basis for \( A_2 \) and \( l \) the index of the eigenvalue of \( A_2 \) nearest the eigenvalue corresponding to \( x \), then for a general matrix \( A_2 \)

\[
\delta^{-1} \leq \| \Sigma_\perp \|_2 \leq 2 \kappa_2(J) \delta^{-l}
\]

(22)

under the assumption that \( \delta \) is sufficiently small \([36, p. 152]\), that is

\[
2 \delta^2 + \cdots + l \delta^{2(l-1)} \leq 1
\]

Specifically, when \( A_2 \) is diagonalizable, we have

\[
\delta^{-1} \leq \| \Sigma_\perp \|_2 \leq \kappa_2(J) \delta^{-1}
\]

(23)
Table II. Example 1, $\text{tol} = 10^{-8}$, The Power method and the quadratic extrapolation for PageRank.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Power</th>
<th>Quad-extra</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>iter</td>
<td>mv</td>
</tr>
<tr>
<td>0.85</td>
<td>95</td>
<td>95</td>
</tr>
<tr>
<td>0.90</td>
<td>146</td>
<td>146</td>
</tr>
<tr>
<td>0.95</td>
<td>299</td>
<td>299</td>
</tr>
<tr>
<td>0.99</td>
<td>1512</td>
<td>1512</td>
</tr>
</tbody>
</table>

Table III. Example 1, $\text{tol} = 10^{-8}$, The Arnoldi-type algorithm and the Power–Arnoldi algorithm for PageRank.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Arnoldi-type</th>
<th>Power–Arnoldi</th>
<th>Speedup (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>m</td>
<td>iter</td>
<td>mv</td>
</tr>
<tr>
<td>0.85</td>
<td>3</td>
<td>39</td>
<td>117</td>
</tr>
<tr>
<td>0.90</td>
<td>3</td>
<td>59</td>
<td>177</td>
</tr>
<tr>
<td>0.95</td>
<td>3</td>
<td>118</td>
<td>354</td>
</tr>
<tr>
<td>0.99</td>
<td>3</td>
<td>508</td>
<td>1524</td>
</tr>
<tr>
<td>0.85</td>
<td>4</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>0.90</td>
<td>4</td>
<td>34</td>
<td>136</td>
</tr>
<tr>
<td>0.95</td>
<td>4</td>
<td>60</td>
<td>240</td>
</tr>
<tr>
<td>0.99</td>
<td>4</td>
<td>229</td>
<td>916</td>
</tr>
<tr>
<td>0.85</td>
<td>5</td>
<td>18</td>
<td>90</td>
</tr>
<tr>
<td>0.90</td>
<td>5</td>
<td>25</td>
<td>125</td>
</tr>
<tr>
<td>0.95</td>
<td>5</td>
<td>47</td>
<td>235</td>
</tr>
<tr>
<td>0.99</td>
<td>5</td>
<td>173</td>
<td>865</td>
</tr>
</tbody>
</table>

So we obtain trivially from (21)–(23) that

**Theorem 5.4**

Under the above notation, if $\tilde{P}$ has at least two irreducible closed subsets and $\delta$ is sufficiently small, then the 2-norm condition number $\|\Sigma^+\|_2$ of the PageRank vector $x$ satisfies

$$
\frac{1}{1 - \alpha} \leq \|\Sigma^+\|_2 \leq 2\kappa_2(J) \left( \frac{1}{1 - \alpha} \right)^l
$$

Furthermore, if $A_2$ is diagonalizable, then

$$
\frac{1}{1 - \alpha} \leq \|\Sigma^+\|_2 \leq \frac{1}{1 - \alpha} \kappa_2(J)
$$

We can see from the above theorem that $x$ can be ill-conditioned if $\alpha$ is close to 1 or and $\kappa_2(J)$ is large. The value $1/(1 - \alpha)$ plays an important role here, since the left-hand side of (24) indicates that $x$ will be ill-conditioned if $\alpha$ is sufficiently close to 1. This conclusion coincides with the one presented in [13, Section 7]. But we also note that a small $1/(1 - \alpha)$ does not insure insensitivity.
Figure 1. Example 1: convergence history of the Power method, the Arnoldi-type algorithm \((m = 5)\) and that of the Power–Arnoldi algorithm \((m = 5, p = 3)\) on the 281 903 \(\times\) 281 903 Stanford web matrix.

6. NUMERICAL TESTS

On an Intel Pentium IV 2.53 GHz with main memory 512 MB PC, we tested the Power method, the Arnoldi-type algorithm \([5]\), the quadratic-extrapolation method \([2]\), as well as our Power–Arnoldi algorithm using Matlab 7.0 on four real-world problems. For the sake of justification, in all the algorithms we use the same initial vector \(x^{(0)} = e / \|e\|_{2}\), where \(e = (1, 1, \ldots, 1)^{T}\). The stopping criteria are

\[
\|Ax^{(k)} - x^{(k)}\|_{2} \leq \text{tol} \quad (26)
\]

for the power method

\[
\|Ax^{(k)} - x^{(k)}\|_{2} = \sigma_{\text{min}}(\tilde{H}_{m} - [I; O]) \leq \text{tol} \quad (27)
\]

for the Arnoldi-type algorithm \([5]\), and

\[
\|Ax^{(k)} - x^{(k)}\|_{2} = h_{m+1,m} |e_{m,y}^{T}| \leq \text{tol} \quad (28)
\]

for the thick restarted Arnoldi algorithm, respectively, where \(\text{tol}\) is the user described tolerance.
Table IV. Example 2, $\text{tol} = 10^{-8}$. Four algorithms on the $683,446 \times 683,446$ Stanford–Berkeley web matrix.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>iter</th>
<th>mv</th>
<th>time (s)</th>
<th>aver (s)</th>
<th>Speedup (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.85</td>
<td>Power</td>
<td>95</td>
<td>95</td>
<td>43.4</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>Arnoldi-type</td>
<td>19</td>
<td>95</td>
<td>64.7</td>
<td>3.41</td>
</tr>
<tr>
<td></td>
<td>Quad-extr</td>
<td>59</td>
<td>79</td>
<td>38.7</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>Power–Arnoldi</td>
<td>50</td>
<td>80</td>
<td>50.0</td>
<td>1.00</td>
</tr>
<tr>
<td>0.90</td>
<td>Power</td>
<td>147</td>
<td>147</td>
<td>67.6</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>Arnoldi-type</td>
<td>28</td>
<td>140</td>
<td>95.9</td>
<td>3.43</td>
</tr>
<tr>
<td></td>
<td>Quad-extr</td>
<td>88</td>
<td>118</td>
<td>58.1</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>Power–Arnoldi</td>
<td>69</td>
<td>117</td>
<td>73.5</td>
<td>1.07</td>
</tr>
<tr>
<td>0.95</td>
<td>Power</td>
<td>306</td>
<td>306</td>
<td>140.7</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>Arnoldi-type</td>
<td>57</td>
<td>285</td>
<td>196.7</td>
<td>3.45</td>
</tr>
<tr>
<td></td>
<td>Quad-extr</td>
<td>178</td>
<td>238</td>
<td>116.5</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>Power–Arnoldi</td>
<td>113</td>
<td>196</td>
<td>123.8</td>
<td>1.10</td>
</tr>
<tr>
<td>0.99</td>
<td>Power</td>
<td>1610</td>
<td>1610</td>
<td>735.4</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>Arnoldi-type</td>
<td>251</td>
<td>1255</td>
<td>866.8</td>
<td>3.45</td>
</tr>
<tr>
<td></td>
<td>Quad-extr</td>
<td>812</td>
<td>1084</td>
<td>532.3</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>Power–Arnoldi</td>
<td>523</td>
<td>996</td>
<td>631.2</td>
<td>1.21</td>
</tr>
<tr>
<td>0.995</td>
<td>Power</td>
<td>3222</td>
<td>3222</td>
<td>1483.5</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>Arnoldi-type</td>
<td>449</td>
<td>2245</td>
<td>1560.8</td>
<td>3.48</td>
</tr>
<tr>
<td></td>
<td>Quad-extr</td>
<td>1478</td>
<td>1972</td>
<td>975.3</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>Power–Arnoldi</td>
<td>955</td>
<td>1860</td>
<td>1188.2</td>
<td>1.24</td>
</tr>
</tbody>
</table>

In all the tables below, we denote by $\text{iter}$ the number of restarting, by $\text{mv}$ the number of matrix–vector multiplications, by $\text{time}$ the CPU time used in seconds, and by $\text{aver}$ the average CPU time used per iteration. Define

$$\text{Speedup} = \frac{\text{time}_{\text{power}} - \text{time}_{\text{Power–Arnoldi}}}{\text{time}_{\text{power}}}$$

the wall-clock speedups for the Power–Arnoldi algorithm with respect to the power method. In the Power–Arnoldi algorithm, the parameters chosen to flip-flop between power method and the thick restarted Arnoldi are $\beta = \alpha - 0.1$, $\text{maxit} = 6$, and we run the thick-restarted Arnoldi procedure two times per cycle. In the quadratic-extrapolation method, we run the quadratic-extrapolation procedure [2] every six power iterations.

Example 1
This is the Stanford.mat link generated from a crawl of the Standard.edu domain by the Stanford Web-Base project in September of 2002. The link graph contains roughly 280,000 nodes, with three million links, and requires 12 MB of storage. Tables II and III list the results obtained.

One observes that for this example, the quadratic-extrapolation method works better than the power method, while the Power–Arnoldi algorithm performs the best in most cases. The Power–Arnoldi algorithm is cheaper than the Arnoldi-type algorithm, which is shown from the average time used per iteration. In fact, the Power–Arnoldi algorithm may use fewer matrix–vector multiplications and CPU time compared with the Arnoldi-type algorithm, even if the former uses more iterations than the latter does. Furthermore, the Power–Arnoldi algorithm is capable of retaining an arbitrary number of Ritz vectors from the previous iteration with a minimal restarting cost, so it is more flexible than the Arnoldi-type algorithm.

On the other hand, it is seen from Tables II and III that the average time used per iteration of the Power–Arnoldi algorithm is comparable to that of the quadratic-extrapolation method. Furthermore, the Power–Arnoldi algorithm outperforms the quadratic-extrapolation method when the damping factor is high. It was pointed out in [13] that slight different values for $\alpha$ can yield very different PageRanks, and the PageRank vector derived from $\alpha = 0.99$ perhaps gives a ‘truer’ PageRanking than $\alpha = 0.85$ does. We can see that Power–Arnoldi considerably improves convergence over the classical power method when the damping factor is high.

The last column of Table III summarizes the wall-clock speedups of the Power–Arnoldi algorithm with respect to the power method for different $\alpha$. As the overhead is low, the overall speedup is significant. It is interesting to see that, for this example, the Power–Arnoldi algorithm is superior to the power method even for low damping factors. This suggests that our new algorithm has some
Table V. Example 3, \( \text{tol} = 10^{-8} \). Four algorithms on the 9664 \( \times \) 9664 California web matrix.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>iter</th>
<th>mv</th>
<th>time</th>
<th>Speedup (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.85</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>90</td>
<td>90</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>Arnoldi-type</td>
<td>13</td>
<td>65</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>Qua-ext</td>
<td>54</td>
<td>72</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>Power–Arnoldi</td>
<td>32</td>
<td>45</td>
<td>0.17</td>
<td>22.7</td>
</tr>
<tr>
<td>0.90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>138</td>
<td>138</td>
<td>0.35</td>
<td></td>
</tr>
<tr>
<td>Arnoldi-type</td>
<td>18</td>
<td>90</td>
<td>0.43</td>
<td></td>
</tr>
<tr>
<td>Qua-ext</td>
<td>85</td>
<td>115</td>
<td>0.32</td>
<td></td>
</tr>
<tr>
<td>Power–Arnoldi</td>
<td>34</td>
<td>51</td>
<td>0.21</td>
<td>40.0</td>
</tr>
<tr>
<td>0.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>280</td>
<td>280</td>
<td>0.69</td>
<td></td>
</tr>
<tr>
<td>Arnoldi-type</td>
<td>31</td>
<td>155</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td>Qua-ext</td>
<td>158</td>
<td>212</td>
<td>0.58</td>
<td></td>
</tr>
<tr>
<td>Power–Arnoldi</td>
<td>43</td>
<td>66</td>
<td>0.27</td>
<td>60.9</td>
</tr>
<tr>
<td>0.99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>1319</td>
<td>1319</td>
<td>3.81</td>
<td></td>
</tr>
<tr>
<td>Arnoldi-type</td>
<td>116</td>
<td>580</td>
<td>2.83</td>
<td></td>
</tr>
<tr>
<td>Qua-ext</td>
<td>651</td>
<td>869</td>
<td>2.59</td>
<td></td>
</tr>
<tr>
<td>Power–Arnoldi</td>
<td>73</td>
<td>115</td>
<td>0.52</td>
<td>86.4</td>
</tr>
</tbody>
</table>

potential. For instance, when \( \alpha = 0.85 \) and 0.9, the Power–Arnoldi with \( m = 5, p = 3 \) reduces the CPU time required to reach a residual of \( 10^{-8} \) by 20.7 and 33.6\%, respectively. While \( \alpha = 0.95 \), the Power–Arnoldi saves the time required to reach the desired accuracy by about 46.7\%, 55.4 s \textit{versus} 104 s. When \( \alpha = 0.99 \), the speedup is dramatic, the Power–Arnoldi reduces the CPU time needed to compute the PageRank vector by 74.4\%, 135.2 s \textit{versus} 528 s. In other words, the Power–Arnoldi algorithm is about 3.9 times faster than the standard power method in this case. Figure 1 depicts the convergence curves of the power method, the Arnoldi-type algorithm with \( m = 5 \), and the Power–Arnoldi algorithm with \( m = 5, p = 3 \) for \( \alpha = 0.85, 0.9, 0.95, 0.99 \), respectively. One can observe that the Power–Arnoldi algorithm speeds up convergence considerably.

\textit{Example 2}

This test matrix is the 683 446 \( \times \) 683 446 Stanford-Berkeley web matrix provided by Kamvar (available at http://www.stanford.edu/sdkamvar/research.html). It contains 683 446 pages and 7.6 million links, and needs 32.5 MB zipped storage.

We run the power method, the quadratic-extrapolation method, the Arnoldi-type algorithm (with \( m = 5 \)), as well as our Power–Arnoldi algorithm (with \( m = 5, p = 3 \)) on this extremely huge matrix, Table IV gives the results obtained. For this example, when \( \alpha = 0.85 \), the separation of the second eigenvalue from the first one is sufficiently good, so that the power method performs extremely well. On the other hand, for high values of damping factors our new method yields significant savings, at the price of higher memory requirements. Specifically, the gain by using Power–Arnoldi...
is very remarkable when the damping factor is close to 1. For instance, relative to the power method, the Power–Arnoldi algorithm reduces the CPU time by 14.2 and 19.9% for $\alpha = 0.99$ and 0.995, respectively.

One may notice that for this example, the quadratic-extrapolation method outperforms the Power–Arnoldi algorithm in terms of CPU time, while the latter outperforms the former in terms of the number of matrix–vector multiplications. Obviously, only the matrix–vector multiplications do not tell the whole story, and we would like to remind the reader that the overall computational time depends on the computing environment, such as vectorization, parallelization, cache size, initial guess and blocking [5]. Furthermore, we can see that the Power–Arnoldi algorithm works much better than the Arnoldi-type algorithm, and the former is much cheaper than the latter in terms of the average time used per iteration. Figure 2 plots the convergence history of the power method, the Arnoldi-type algorithm, and that of the Power–Arnoldi algorithm for $\alpha = 0.85, 0.9, 0.95$ and 0.99, respectively. They show that the Power–Arnoldi algorithm outperforms its counterparts in many cases.

**Example 3**

This is a relatively ‘small’ example. The test matrix is California.dat, which is available from http://www.cs.cornell.edu/Courses/cs685/2002fa. It contains 9664 nodes and 16773 links, pertaining to the query topic of California.
We run the power method, the quadratic-extrapolation method, the Arnoldi-type algorithm (with $m = 5$), as well as our Power–Arnoldi algorithm (with $m = 5, p = 3$) on this example. Table V reports the results. Of the algorithms we have discussed for accelerating the PageRank computation, the Power–Arnoldi algorithm works the best. It can be found from the numerical results that both the Power–Arnoldi algorithm and the quadratic-extrapolation method make great improvements on the classical power method, while the Power–Arnoldi algorithm preforms the best. For a medium damping factor such as 0.9, the Power–Arnoldi algorithm with $m = 5, p = 3$ takes 0.21 s while the power method takes 0.35 s to reach a residual of $10^{-8}$. That is, the former is about 1.7 times faster than the latter. Note also that the Arnoldi-type algorithm with $m = 5$ uses 0.43 s to achieve the same accuracy. Therefore, our work provides an alternative to the newborn Arnoldi-type algorithm for computing PageRank. Furthermore, we would also like to express that the Power–Arnoldi algorithm is applicable to any Markov chain, not only to the PageRank problem.

Figure 3 plots the convergence behaviour of the power method, the quadratic-extrapolation method, and our new algorithms for $\alpha = 0.85, 0.9, 0.95$ and 0.99, respectively. One can observe that the Power–Arnoldi algorithm accelerates the convergence dramatically.

**Example 4**
This matrix is *Hollins.dat*, which was generated from a web crawl of www.hollins.edu on 15 January 2004. It is available from http://www.limfinity.com/ir/, which contains 6012 pages and

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>iter</th>
<th>mv</th>
<th>time</th>
<th>Speedup (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.85</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>88</td>
<td>88</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>Arnoldi-type</td>
<td>15</td>
<td>75</td>
<td>0.31</td>
<td></td>
</tr>
<tr>
<td>Qua-ext</td>
<td>46</td>
<td>62</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>Power–Arnoldi</td>
<td>42</td>
<td>58</td>
<td>0.16</td>
<td>5.9</td>
</tr>
<tr>
<td>0.90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>128</td>
<td>128</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>Arnoldi-type</td>
<td>21</td>
<td>105</td>
<td>0.38</td>
<td></td>
</tr>
<tr>
<td>Qua-ext</td>
<td>68</td>
<td>92</td>
<td>0.19</td>
<td></td>
</tr>
<tr>
<td>Power–Arnoldi</td>
<td>57</td>
<td>75</td>
<td>0.21</td>
<td>16.0</td>
</tr>
<tr>
<td>0.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>253</td>
<td>253</td>
<td>0.50</td>
<td></td>
</tr>
<tr>
<td>Arnoldi-type</td>
<td>41</td>
<td>205</td>
<td>0.71</td>
<td></td>
</tr>
<tr>
<td>Qua-ext</td>
<td>130</td>
<td>174</td>
<td>0.36</td>
<td></td>
</tr>
<tr>
<td>Power–Arnoldi</td>
<td>87</td>
<td>117</td>
<td>0.34</td>
<td>32.0</td>
</tr>
<tr>
<td>0.99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>1196</td>
<td>1196</td>
<td>2.39</td>
<td></td>
</tr>
<tr>
<td>Arnoldi-type</td>
<td>148</td>
<td>740</td>
<td>2.47</td>
<td></td>
</tr>
<tr>
<td>Qua-ext</td>
<td>583</td>
<td>779</td>
<td>1.65</td>
<td></td>
</tr>
<tr>
<td>Power–Arnoldi</td>
<td>238</td>
<td>328</td>
<td>0.96</td>
<td>59.8</td>
</tr>
</tbody>
</table>
23,875 links. We run the power method, the quadratic-extrapolation method, the Arnoldi-type algorithm (with \(m = 5\)), as well as the Power–Arnoldi algorithm (with \(m = 5, p = 3\)) on this example. Table VI lists the results.

We observe from Table VI and Figure 4 that the quadratic-extrapolation method and Power–Arnoldi algorithm are more efficient than the other two. Furthermore, the Power–Arnoldi algorithm is preferable when the damping factor is high. As we have expected, the Power–Arnoldi algorithm performs much better than the classical power method when the damping factor is close to 1. For instance, when \(\alpha = 0.99\), the Power–Arnoldi algorithm uses 0.96 s to reach the desired accuracy, while the power method uses 2.39 s. So our new algorithm is about 2.5 times faster than the power method in this case. It is obvious to see that the Power–Arnoldi algorithm is more efficient than the Arnoldi-type algorithm, even if they share the same storage.

All the above numerical results have shown that our Power–Arnoldi algorithm is often superior to its counterparts for computing PageRank. However, the experiments have also demonstrated that the power method and the quadratic-extrapolation method may be better than the Power–Arnoldi algorithm in many cases, especially when the damping factor is low.

Two remarkable merits of the Power–Arnoldi algorithm are flexibility and parallelizability, and it is more attractive when the damping factor is high. On the other hand, we have also noticed that some parameters are difficult to handle in the Power–Arnoldi algorithm. For example, how to determine the optimal \(\beta\) and \(\maxit\) so that the new algorithm can work more efficiently? They are still under investigation and much work should be done.
ACKNOWLEDGEMENTS

Special appreciation goes to two anonymous reviewers for their insightful comments and invaluable suggestions that greatly improve the presentation of this paper. Meanwhile, we would also like to express our sincere thanks to Profs. Gene H. Golub and Chen Grief for providing their preprint, and to Prof. Sepandar D. Kamvar for providing the test matrices.

REFERENCES


