Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

An Arnoldi-Extrapolation algorithm for computing PageRank

Gang Wu [a], Yimin Wei [b,c,*]

[a] School of Mathematical Sciences, Xuzhou Normal University, Xuzhou, 221116, Jiangsu, PR China
[b] School of Mathematical Sciences, Fudan University, Shanghai, 200433, PR China
[c] Shanghai Key Laboratory of Contemporary Applied Mathematics, PR China

ARTICLE INFO

Article history:
Received 7 February 2008

MSC:
65F15
65F10
65C40
60J22

Keywords:
PageRank
GeneRank
Krylov subspace
Arnoldi method
Ritz value

ABSTRACT

The Arnoldi-type algorithm proposed by Golub and Greif [G. Golub, C. Greif, An Arnoldi-type algorithm for computing PageRank, BIT 46 (2006) 759–771] is a restarted Krylov subspace method for computing PageRank. However, this algorithm may not be efficient when the damping factor is high and the dimension of the search subspace is small. In this paper, we first develop an extrapolation method based on Ritz values. We then consider how to periodically knit this extrapolation method together with the Arnoldi-type algorithm. The resulting algorithm is the Arnoldi-Extrapolation algorithm. The convergence of the new algorithm is analyzed. Numerical experiments demonstrate the numerical behavior of this algorithm.

1. Introduction

The PageRank algorithm is one of the most commonly used techniques that determine the global importance of Web pages [1]. The core of this algorithm involves using the classical power method to compute the principal eigenvector of the Google matrix. However, it is well known that the power method may perform poorly when the second largest eigenvalue is close to the dominant one [2]. This happens when the damping factor is close to 1 [3,4]. Therefore, it is necessary to develop more efficient algorithms for accelerating the computation of PageRank.

Recently, a very interesting research track exploits efficient numerical linear algebra methods to speed up the computation of PageRank [5–8]. Kamvar et al. devised adaptive methods [9] to speed up the computation of PageRank, in which the PageRank of pages that have converged, are not recomputed at each iteration after convergence. The quadratic extrapolation method [10] 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. A reordering method [11] reduces the computation of PageRank to solve a much smaller system, and then uses forward substitution to get a full solution vector. The BlockRank algorithm [12] exploits the block structure of the Web link graph to speed up the PageRank computation via a three step algorithm. The Arnoldi-type algorithm [13] is an explicitly restarted Krylov subspace method, which is a combination of the Arnoldi process and small singular value decomposition (SVD) that relies on the knowledge of the largest eigenvalue. The Power-Arnoldi algorithm [14] proposed recently is a hybrid algorithm that is based on a periodic combination of the power method with the thick restarted Arnoldi algorithm [15]. We refer the reader to [16–29], as well as the references given therein for recent developments on the PageRank problem.

[*] Corresponding author at: School of Mathematical Sciences, Fudan University, Shanghai, 200433, PR China.
E-mail addresses: gangwu76@yahoo.com.cn, wugangzy@gmail.com (G. Wu), ymwei@fudan.edu.cn, yimin.wei@gmail.com (Y. Wei).

0377-0427/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2010.02.009

Let us briefly introduce how Google turns the hyperlink structure of the Web into a stochastic and irreducible matrix, a detailed description of this can be found in [5–8]. Consider the hyperlink structure of the Web as a directed Web graph. If there are n pages in the Web, then the Markov model represents this graph with an n by n matrix P whose entries p_{ij} are the probability of moving from page i to page j in one click of a mouse. However, there are two potential problems. The first one is the existence of *dangling nodes* [23,24]. This means some rows of P may contain all zeros, so that P is not stochastic. A cure is to replace all zero rows in P with e^T/n, where $e = [1, 1, \dots, 1]^T$ is the row of all ones, so we have

$$\tilde{P} = P + fe^T/n,$$

where

$$f_i = \begin{cases} 1, & \text{if page } i \text{ is dangling,} \\ 0, & \text{otherwise.} \end{cases}$$

The second problem is the existence of *cyclic paths*, i.e., some pages may form a closed loop. This is mathematically equivalent to P being reducible. So as to deal with this drawback, another adjustment is required, which yields the Google matrix

$$A = [\alpha\tilde{P} + (1-\alpha)E]^T, \quad 0 < \alpha < 1, \tag{1.1}$$

where $E = \frac{1}{n}ee^T$ is a rank-one matrix,[1] and $0 < \alpha < 1$ is the *damping factor* which models the possibility that a Web surfer jumps from one page to the next without following a link. The convex combination of the stochastic matrix $\tilde{P}$ and a stochastic perturbation matrix E insures that A is both stochastic and irreducible [30,31]. The irreducibility adjustment also insures A is primitive, which implies that the power method will converge to the PageRank vector if the starting vector is a distribution.

It was reported that the damping factor is originally set to 0.85 [1]. The question which value of α is the correct value and what gives a meaningful ranking, is subject to ongoing investigation, one can see [32,22,26,27] for recent study. On the one hand, 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 number of 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. However, if the damping factor is close to 1, the power method will perform poorly, and new approaches for accelerating the PageRank computation will be required. Although the usual value for α is 0.85, the computation of many PageRank vectors with different values of α seems promising for the design of anti-spam mechanism [33].

This paper is organized as follows. In Section 2, we briefly introduce the mechanism of the Arnoldi-type algorithm proposed by Golub and Greif for computing PageRank [13]. In Section 3, we first develop an extrapolation method based on Ritz values, which are available from the Arnoldi iteration. We then consider how to periodically knit this extrapolation procedure together with the Arnoldi-type algorithm. The resulting algorithm is the Arnoldi-Extrapolation algorithm. The convergence of the new algorithm is considered. Numerical experiments given in Section 4 to illustrate the numerical behavior of this algorithm. Some concluding remarks are presented in Section 5.

Matlab notation is used throughout this paper. Let $\mathbb{P}_{m-1}$ be the set of monic polynomials whose degrees are at most $m-1$. Denote by $\|\cdot\|_1$, $\|\cdot\|_2$ the vector 1-norm and 2-norm, as well as the induced matrix norms, respectively, and by $\sigma_{\min}(\cdot)$ the smallest singular value of a given matrix.

2. The Arnoldi-type algorithm for computing PageRank

The Arnoldi method is a well-known Krylov subspace method that can be utilized to find a few eigenpairs of a large matrix [34,35]. Given a unit norm vector v_1, if computations are performed in exact arithmetic, then the Arnoldi process will generate successively an orthonormal basis for the Krylov subspace $\mathcal{K}_m(A, v_1) = span\{v_1, Av_1, \dots, A^{m-1}v_1\}$. In this subspace, the restriction of A is represented by an $m \times m$ upper Hessenberg matrix H_m with the entries h_{ij}. Furthermore, the following relation holds

$$AV_m = V_mH_m + h_{m+1,m}v_{m+1}e_m^T = V_{m+1}\tilde{H}_m, \tag{2.1}$$

where V_m is an orthonormal basis of the Krylov subspace, e_m is the mth coordinate 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 only nonzero entry is $h_{m+1,m}$. The eigenvalues of H_m, $\tilde{\lambda}_i$, $(i = 1, 2, \dots, m)$ called Ritz values of A in $\mathcal{K}_m(A, v_1)$, can be used to approximate some eigenvalues of A. We provide the essential details of the Arnoldi process, for more details, refer to [34,36,35].

Algorithm 1 (*The m-Step Arnoldi Process*).
function $[V_{m+1}, \tilde{H}_m, matvec] = Arnoldi(A, v_1, m, matvec)$

[1] Google eventually replaced the uniform vector e^T/n with a more general probability vector v^T (so that $E = ev^T$) to allow the flexibility to make adjustments to PageRanks as well as to personalize them.

```
for $j = 1, 2, \ldots, m$
  $q = Av_j$;
  matvec=matvec+1; % the number of matrix-vector products
  for $i = 1, 2, \ldots, j$
    $h_{i,j} = v_i^T q$;
    $q = q - h_{i,j} v_i$;
  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
```

Remark 1. For the sake of clarity, we denote by Av_j the "matrix-vector products", and by $P^T v_j$ the "**sparse** matrix-vector products" in this paper. Note that the number of the two products are the same. Given a vector v_j, the matrix-vector products Av_j can be implemented as follows [37,10]:

$$Av_j = \alpha P^T v_j + [e^T v_j - \alpha e^T (P^T v_j)]e^T/n, \quad 1 \le j \le m, \tag{2.2}$$

where e is the vector of all ones. So the computation of Av_j requires one sparse matrix-vector product ($P^T v_j$) and two inner products ($e^T v_j$ and $e^T(P^T v_j)$).

As the size of the Google matrix is very large, we stress that the inner products involved in the Arnoldi process, add a substantial amount of work. If we denote by N_z the nonzero elements of the matrix P, then the m-step Arnoldi process requires $2mN_z$ operations for the sparse matrix-vector products $P^T v_j, j = 1, 2, \ldots, m$; and $2m + \frac{m(m+1)}{2}$ inner products for the computation of $e^T v_j$, $e^T(P^T v_j)$ and h_{ij}, which requires $2n \cdot (2m + \frac{m(m+1)}{2})$ operations. Therefore, the m-step Arnoldi process needs approximately $m + (2m + \frac{m(m+1)}{2})n/N_z$ sparse matrix-vector products per iteration. For instance, when $m = 3$ and the Web graph has, say, 10 nonzero entries on average per row, the Arnoldi process costs practically 1.4 sparse matrix-vector products per step.

The Arnoldi-type algorithm for computing PageRank [13] is a variant of the explicitly restarted refined Arnoldi method [38], which relies on the fact that the largest eigenvalue of the Google matrix is 1. For the known eigenvalue, the Arnoldi-type algorithm seeks a unit norm vector x^{AT} satisfying the following optimal property

$$\|(A - I)x^{AT}\|_2 = \min_{\substack{u \in \mathcal{K}_m(A, v_1), \\ \|u\|_2 = 1}} \|(A - I)u\|_2, \tag{2.3}$$

and exploits x^{AT} as an approximation to the PageRank vector. A sketch of the Arnoldi-type algorithm appears as follows, for more details, refer to [13].

Algorithm 2 (*The Arnoldi-Type Algorithm for Computing PageRank*).
1. *Start: Given an initial guess x of unit norm, the Arnoldi steps number m, and a prescribed tolerance tol; Set the number of matrix-vector products matvec=0;*
2. *Iterate*

```
do
  $[V_{m+1}, \tilde{H}_m, matvec] = Arnoldi(A, x, m, matvec)$;
  Compute small SVD: $\tilde{H}_m - [I; \mathbf{O}] = U \Sigma W^T$;
  Compute the approximate eigenvector: $x^{AT} = V_m W(:, m)$;
  $r = \sigma_{\min}(\tilde{H}_m - [I; \mathbf{O}])$; % residual 2-norm
while $r > tol$
```

The original Arnoldi-type algorithm uses the residual 2-norm as a cheap convergence criterion, which is a by-product of the small SVD [13]. However, as PageRank is the stationary distribution of a Markov chain, it is often normalized so that its 1-norm is one. Thus it is preferable to use the 1-norm as the stopping criterion [10]. To do this, one option is to evaluate the residual 1-norm directly: $r = \|Ax^{AT} - x^{AT}\|_1/\|x^{AT}\|_1$, however, we have to pay a superfluous matrix-vector product. Indeed, the matrix-vector product can be realized *implicitly* [39]: From (2.1), we have

$$\|Ax^{AT} - x^{AT}\|_1/\|x^{AT}\|_1 = \|V_{m+1}[\tilde{H}_m W(:, m)] - V_m W(:, m)\|_1/\|V_m W(:, m)\|_1. \tag{2.4}$$

Therefore, we can use residual 1-norm as stopping criterion in the Arnoldi-type algorithm, without additional matrix-vector products [39].

3. An Arnoldi-Extrapolation algorithm for PageRank

It is well known that the power method may perform poorly when the second largest eigenvalue is close to the largest one [2]. This implies that the power method suffers from slow convergence when the damping factor is close to 1 [3,4], and an alternative is the Arnoldi-type algorithm [13]. Unfortunately, the Arnoldi-type algorithm may not be efficient when m, the dimension of the Krylov subspace is low, and when α, the damping factor is high [13,14].

In this section, we focus on improving the convergence of the Arnoldi-type algorithm. In the new algorithm, the dimension of the Krylov subspace can be very moderate, so that the memorization cost is kept reasonable. Other advantages of the proposed technique are the potential use on parallel architectures and its applicability for a wide range of the parameter α which can be set close to 1.

The following theorem due to Bellalij et al. depicts the convergence of the Krylov subspace methods, which sheds light on how to accelerate the convergence of the Arnoldi-type algorithm.

Theorem 1 (*[40]*)**.** *Let λ_1 be a simple eigenvalue of A, and let x_1, y_1 be the right and left eigenvector associated with λ_1, respectively. Assume that $\cos\angle(y_1, v_1) \neq 0$. Let $Q_1 = x_1 x_1^H$ be the orthogonal projector onto the right eigenspace, and let $B_1 = (I - Q_1)A(I - Q_1)$. Define*

$$\varepsilon_m = \min_{\substack{p\in\mathbb{P}_{m-1}\\ p(\lambda_1)=1}} \|p(B_1)(I - Q_1)v_1\|_2. \tag{3.1}$$

Then, we have

$$\|(I - \mathcal{P}_m)x_1\|_2 \leq \frac{\varepsilon_m}{\cos\angle(y_1, v_1)}, \tag{3.2}$$

where $\mathcal{P}_m$ denotes the orthogonal projection onto the subspace $\mathcal{K}_m(A, v_1)$.

Remark 2. Theorem 1 indicates that in a restarted Krylov subspace method, the better the initial vector is, the faster the convergence will be. One way to achieve this is to use the extrapolation methods [17–19,41,10].

3.1. An extrapolation procedure based on Ritz values

In this subsection, we present an extrapolation procedure that is based on the Ritz values. Heuristically, as the convergence proceeds, the Arnoldi-type algorithm will provide better and better approximate eigenvalues to the extrapolation procedure, and on the other hand, the extrapolation procedure will provide better and better initial guess to the Arnoldi iteration.

For convenience, we denote by $x^{(k-1)}$ (i.e., x^{AT}) the approximation from the Arnoldi-type algorithm. As was done in [10], we make the assumption that $x^{(k-1)}$ can be expressed as a linear combination of the first three eigenvectors. That is,

$$x^{(k-1)} = x_1 + \alpha_2 x_2 + \alpha_3 x_3.$$

So we have

$$x^{(k)} = Ax^{(k-1)} = x_1 + \alpha_2\lambda_2 x_2 + \alpha_3\lambda_3 x_3,$$

and

$$x^{(k+1)} = Ax^{(k)} = x_1 + \alpha_2\lambda_2^2 x_2 + \alpha_3\lambda_3^2 x_3.$$

It is easy to verify that

$$x_1 = \frac{x^{(k+1)} - (\lambda_2 + \lambda_3)x^{(k)} + \lambda_2\lambda_3 x^{(k-1)}}{(1 - \lambda_2)(1 - \lambda_3)},$$

and

$$\hat{x}_1 = \frac{x^{(k+1)} - (\lambda_2 + \lambda_3)x^{(k)} + \lambda_2\lambda_3 x^{(k-1)}}{\|x^{(k+1)} - (\lambda_2 + \lambda_3)x^{(k)} + \lambda_2\lambda_3 x^{(k-1)}\|_1}$$

provides a good approximation to the PageRank vector. Unfortunately, λ_2 and λ_3 are not known in advance, and an obvious situation is to replace them with their approximations. Recall from Section 2 that the second and third largest Ritz values $\tilde{\lambda}_2, \tilde{\lambda}_3$ ($|\tilde{\lambda}_2| \geq |\tilde{\lambda}_3|$) are already *available* from the Arnoldi-type algorithm, which are nothing but the second and the third largest eigenvalues of H_m [34,35].

Since only two approximate eigenvalues are required for the extrapolation procedure discussed above, it is enough to run the Arnoldi process with $m = 3$. This is favorable for huge matrices such as the Google matrices. Furthermore, the extrapolation procedure can be easily extended to the case when $x^{(k-1)}$ being expressed as a linear combination of more than three eigenvectors. Now let us consider some practical implementations. Here we make the assumption that the largest Ritz value $\tilde{\lambda}_1$ is real.

Case 1. Both $\tilde{\lambda}_2$ and $\tilde{\lambda}_3$ are real.

We make use of

$$\tilde{x}_1 = \frac{x^{(k+1)} - (\tilde{\lambda}_2 + \tilde{\lambda}_3)x^{(k)} + \tilde{\lambda}_2\tilde{\lambda}_3 x^{(k-1)}}{\|x^{(k+1)} - (\tilde{\lambda}_2 + \tilde{\lambda}_3)x^{(k)} + \tilde{\lambda}_2\tilde{\lambda}_3 x^{(k-1)}\|_1} \tag{3.3}$$

as an approximation to the PageRank vector.

Case 2. $\tilde{\lambda}_2$ and $\tilde{\lambda}_3$ are conjugate.

We make use of

$$\tilde{x}_1 = \frac{x^{(k+1)} - 2\,\mathbf{Re}(\tilde{\lambda}_2)x^{(k)} + |\tilde{\lambda}_2|^2 x^{(k-1)}}{\|x^{(k+1)} - 2\,\mathbf{Re}(\tilde{\lambda}_2)x^{(k)} + |\tilde{\lambda}_2|^2 x^{(k-1)}\|_1} \tag{3.4}$$

as an approximation to the PageRank vector, where $\mathbf{Re}(\tilde{\lambda}_2)$ denotes the real part of $\tilde{\lambda}_2$.

One may argue that in Case 1, $\tilde{\lambda}_2$ should be replaced by α, since the second largest eigenvalue of the Google matrix is α [4]. However, we find experimentally that $\tilde{\lambda}_2$ is still an appropriate choice. The reason is that the extrapolation formula is related to both λ_2 $(=\alpha)$ *and* λ_3 (which is not known in advance), not λ_2 itself.

In practical computations, it is possible to have defective $\tilde{\lambda}_2$ or $\tilde{\lambda}_3$, and the extrapolation procedure may suffer from this. Fortunately, we can deal with this difficulty as follows. Suppose that $x^{(k-1)} = x_1 + \alpha_2 x_2$, and $x^{(k)} = Ax^{(k-1)} = x_1 + \alpha\alpha_2 x_2$. So we can replace (3.3) and (3.4) with

$$\tilde{x}_1 = \frac{x^{(k)} - \alpha x^{(k-1)}}{1-\alpha}. \tag{3.5}$$

An interesting question is how accurate should the Ritz values be in the worst case? Although it is difficult to answer this question theoretically, numerical experiments given in Section 4 to show that the extrapolation procedure still works, even if the Ritz values are not accurate enough. In practical implementations, we explore the extrapolation procedure as soon as the residual norm is below 10^{-2}. Numerical experiments demonstrate that this scheme is effective.

In summary, we present the following extrapolation procedure for the PageRank computation, where mod$(\cdot)$ denotes the MATLAB command for modulus after division.

Algorithm 3 (*An Extrapolation Procedure Based on Ritz Eigenvalues*).
function $[x^{(k)}, r, k, matvec, \eta] = ExtraRitz(A, x^{(k-1)}, \tilde{\lambda}_2, \tilde{\lambda}_3, \ell, matvec, tol)$

```
$x^{(k)} = Ax^{(k-1)}$;
matvec=matvec+1; % the number of matrix-vector products
$r = \|x^{(k)} - x^{(k-1)}\|_1 / \|x^{(k-1)}\|_1$;
if $tol < r < 1e-2$ & mod(matvec, $\ell$) == 0 % Begin the extrapolation procedure
 if $\tilde{\lambda}_2$ and $\tilde{\lambda}_3$ are nondefective
  if both $\tilde{\lambda}_2$ and $\tilde{\lambda}_3$ are real
   $x^{(k+1)} = Ax^{(k)}$;
   matvec=matvec+1;
   $x^{(k)} = x^{(k+1)} - (\tilde{\lambda}_2 + \tilde{\lambda}_3)x^{(k)} + \tilde{\lambda}_2\tilde{\lambda}_3 x^{(k-1)}$;
  end if
  if $\tilde{\lambda}_2$ and $\tilde{\lambda}_3$ are conjugate
   $x^{(k+1)} = Ax^{(k)}$;
   matvec=matvec+1;
   $x^{(k)} = x^{(k+1)} - 2\,\mathbf{Re}(\tilde{\lambda}_2)x^{(k)} + |\tilde{\lambda}_2|^2 x^{(k-1)}$;
  end if
 else % $\tilde{\lambda}_2$ or $\tilde{\lambda}_3$ is defective
  $x^{(k)} = x^{(k)} - \alpha x^{(k-1)}$;
 end if
 $\eta = \|x^{(k)}\|_1$;
 $x^{(k)} = x^{(k)}/\eta$;
 k=k+1;
end if % End the extrapolation procedure
```

At the first glance, it seems that the extrapolation procedure is a straightforward adaption of what was done in [17–19,41,10]. The difference is that we exploit some approximate eigenvalues which are available from the Arnoldi-type algorithm. The overhead of performing the extrapolation procedure comes primarily from one matrix-vector product (i.e., $x^{(k+1)} = Ax^{(k)}$), in addition to some vector-vector manipulations, in $\mathcal{O}(n)$ flops. We would like to remind the reader that the overhead in performing the quadratic extrapolation includes two matrix-vector products, as well as solving an n by 2 least-squares problem [10]. Therefore, our extrapolation procedure is cheaper than the quadratic extrapolation procedure. For storage, we need to store (at most) three long vectors in Algorithm 3, while four long vectors are needed in the quadratic extrapolation method.

As the extrapolation procedure discussed above does require an overhead, when it is combined with the Arnoldi-type algorithm (which yields the Arnoldi-Extrapolation algorithm), a fair comparison with the original Arnoldi-type algorithm should be in terms of the number of matrix-vector products and the CPU time used for convergence.

3.2. The main algorithm and practical implementations

The Arnoldi-type algorithm is a restarted Krylov subspace method for computing PageRank. In the algorithm, the Arnoldi process is restarted every m steps. By varying m, one can change the number of iterations and also the execution time. Numerical experiments show that increasing m decreases the number of iterations needed to converge, while increasing the work and storage required per iteration [13,14].

Since the size of the Google matrix is very large, it is desirable to pick m as small as possible in practice. In this section, we aim to speed up the computation of PageRank while keeping m very moderate, say, $m = 3$. The idea is to construct an improved starting vector using the extrapolation procedure presented in Section 3.1, and compute a new Arnoldi factorization with the new starting vector. This leads us to combine the Arnoldi-type algorithm with the extrapolation procedure periodically, which gives rise to the Arnoldi-Extrapolation algorithm. However, the key issue is how to create a heuristic procedure for determining when to flip flop between the extrapolation procedure and the Arnoldi iteration. Heuristically, we will terminate the extrapolation procedure and switch to the Arnoldi-type algorithm, as soon as η is smaller than a prescribed threshold *tolnorm*, refer to Algorithm 3.

In practical implementations, we set four parameters β, *restart*, *maxit*, and *tolnorm* to monitor the extrapolation method. Denote by $\|r^{(k)}\|_1$ the residual 1-norm of the current iteration, and by $\|r^{(k-1)}\|_1$ that of the previous iteration. We then examine whether $\|r^{(k)}\|_1/\|r^{(k-1)}\|_1$ is larger than the prescribed threshold β, if so, set $restart := restart + 1$ and check whether *restart* is larger than the pre-determined number *maxit*. If so, we flag the extrapolation method as exhausted and trigger the Arnoldi-type algorithm, otherwise, keep on running the extrapolation method. Now we are ready to present the main algorithm of this paper.

Algorithm 4 (*An Arnoldi-Extrapolation Algorithm for Computing PageRank*).

```
(1) Given $v$, the initial guess; tol, a user described tolerance; $\ell$, the number for applying the extrapolation procedure periodically; four parameters $\beta$, maxit, restart and tolnorm for monitoring the extrapolation procedure; Set $m = 3$ (the number of the Arnoldi steps), $k = 0$, $r = 1$, $r_0 = r$, and matvec=0, restart=0;
(2) while $restart < maxit$ & $r > tol$
(3)   $x = Av$; matvec=matvec+1; % the number of matrix-vector products
(4)   $r = \|x - v\|_1/\|v\|_1$;
(5)   $x = x/\|x\|_1$;
(6)   if $r/r_0 > \beta$
(7)     restart=restart+1;
(8)   end if
(9)   $v = x$;
(10)  $k = k + 1$;
(11)  $r_0 = r$;
(12) end while
(13) while $r > tol$ % Begin the outer iteration
(14)   Perform the Arnoldi process with x as the initial guess: $[V_{m+1},\ \tilde{H}_m, matvec] = Arnoldi(A, x, m, matvec)$;
(15)   Compute all the eigenpairs of $H_m = \tilde{H}_m(1:m,:)$, and select the second and third largest Ritz values, i.e., $\tilde{\lambda}_2, \tilde{\lambda}_3$ to approximate $\lambda_2$ and $\lambda_3$, respectively;
(16)   Compute small SVD: $\tilde{H}_m - [I;\ \mathbf{0}] = U\Sigma W^T$;
(17)   $v = V_m W(:,m)$; % $x^{AT}$
(18)   $x = V_{m+1}(\tilde{H}W(:,m))$; % $Ax^{AT}$
(19)   $r = \|x - v\|_1/\|v\|_1$; % residual 1-norm of the Arnoldi-type algorithm
(20)   $x = x/\|x\|_1$; % initial vector for the extrapolation procedure
(21)   restart=0; % Apply the extrapolation procedure with x as the initial guess
(22)   $r_0 = r$;
(23)   $\eta = 1$;
```

```
(24)  while r>tol & restart<maxit & $\eta$ >=tolnorm
(25)    $[x, r, k, macvec, \eta] = Extrapolation(A, x, \tilde{\lambda}_2, \tilde{\lambda}_3, k, \ell, matvec, tol)$;
(26)    if $r/r_0 > \beta$
(27)      restart=restart+1;
(28)    end if
(29)    $r_0 = r$;
(30)  end while  % End the extrapolation procedure
(31) end while  % End the outer iteration
```

We now describe some practical details, referring to the respective phases in Algorithm 4.

(1): This is the initialization phase of the algorithm. If a good initial 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 β is crucial in practice. It is well known that the *asymptotic* convergence rate of the power method is $|\lambda_2|/|\lambda_1|$. For Google matrix, $\lambda_1 = 1$, moreover, if $\tilde{P}$ has at least two irreducible closed subsets, the second eigenvalue is α [4]. Therefore, it is reasonable from a theoretical point of view that β should be smaller than α, and all what need is to satisfy $0 < \beta < \alpha$. In practical computations, we can choose, say, $\beta = \alpha - 0.1$ or $\alpha - 0.2$. Although it is difficult to determining the optimal β theoretically, the numerical experiments show insensitivity of the choice of β (as long as β is sufficiently smaller than α), see Example 4 of Section 4. Moreover, setting β as, say, 0.8α or another multiple of α may actually be easier to deal with theoretically than setting it as $\alpha - 0.2$ and so on.

(2)–(12): First we iterate a few times using the power method to get a rough convergence. Notice that this procedure is performed only once throughout Algorithm 4. Steps (6)–(8) are devised for determining when to flip flop between the power method and the Arnoldi-type algorithm.

(14)–(17): We run the Arnoldi-type algorithm with x as the initial vector. In Step (15) we evaluate all the Ritz pairs, in $\mathcal{O}(m^3)$ flops. In Step (16) we compute the small sized SVD, where $W(:, m)$ denotes the right singular vector corresponding to the smallest singular value.

(18)–(20): We compute the residual 1-norm of x^{AT}, and make use of Ax^{AT} as the initial vector for the extrapolation procedure, see the discussions made in Section 2. Recall that the original Arnoldi-type algorithm exploits x^{AT} as the initial guess for the next Arnoldi iteration.

(21)–(30): We run the extrapolation procedure with x as the initial guess. Note that $\tilde{\lambda}_2, \tilde{\lambda}_3$ are available from Step (15). Steps (26)–(28) are devised to determine when to flip flop between the extrapolation procedure and Arnoldi, and we terminate the extrapolation procedure and run the Arnoldi-type algorithm as soon as $\eta < tolnorm$, where *tolnorm* is a user described threshold.

A remarkable merit of Algorithm 4 is that one can choose very moderate m in practice. It is obvious that we only need to store $m + 1(=4)$ long vectors in the Arnoldi process, and to store three long vectors in the extrapolation method. Therefore, the storage requirements of the Arnoldi-Extrapolation algorithm are (at most) four long vectors, which are the same as the quadratic extrapolation procedure. Secondly, the new algorithm allows for easy parallelization, which is essential due to the large-scale of the matrices in practical use. Finally, we point out that the computation of Ritz pairs, which are the eigenpairs of a 3 by 3 upper Hessenberg matrix, is neglectable compared with the overall work per iteration.

In essence, the Arnoldi-Extrapolation algorithm and the Power-Arnoldi algorithm [14] are both hybrid algorithms in which they combine two algorithms into a much efficient one. Indeed, both the algorithms can improve the convergence rate by effectively increasing the gap ratio, see Theorem 5.3 of [14] and Theorem 3 of Section 3.3. The difference is that the Arnoldi-Extrapolation algorithm knits an extrapolation procedure that based on Ritz values together with the Arnoldi-type algorithm, while the Power-Arnoldi algorithm combines the power method with the thick-restarted Arnoldi algorithm [15].

The storage requirements of the Arnoldi-Extrapolation and the Power-Arnoldi algorithms are the same, which are approximately $m+1$ length-n vectors. However, the cost of the Power-Arnoldi algorithm is cheaper than that of the Arnoldi-Extrapolation algorithm per cycle. First, it is obvious to see that the power method is cheaper than Algorithm 3. Second, the thick restarted Arnoldi algorithm is cheaper than the regular explicitly restarted Arnoldi algorithm. Indeed, a superiority of the thick restarted Arnoldi algorithm over the Arnoldi-type algorithm is that the former needs only $m - p$ matrix-vector products at each iteration after the first [15], while the latter uses m, where p $(p < m)$ is the number of approximate eigenvectors which are retained from one iteration to the next. However, the Arnoldi-Extrapolation is preferable when m is relatively small. A numerical comparison of the two approaches are given in Example 2 of Section 4.

3.3. Convergence analysis

In this subsection, we consider the convergence of the Arnoldi-Extrapolation algorithm, and shed light on why it can perform better than the Arnoldi-type algorithm and the power method. Let $\lambda_1 = 1 > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$ be the eigenvalues of the Google matrix. It was shown by [22, Theorem 8.2(ii)] that the second largest eigenvalue of the Google matrix is semi-simple, and the Google matrix is *partially* diagonalizable. Here we make the assumption that the third largest eigenvalue λ_3 is also semi-simple, so that the initial vector (which is from the *previous* Arnoldi iteration) for the extrapolation procedure can be expressed as

$$x^{AT} = x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \sum_{j=4}^{s} X_j z_j, \tag{3.6}$$

where X_j are the Jordan vectors satisfying $AX_j = X_j J_j$, $(4 \le j \le s \le n)$. It follows from Algorithm 4 that there exists $d \ge 0$ such that

$$x^{(k-1)} = A^d x^{AT}, \qquad x^{(k)} = A^{d+1} x^{AT}, \qquad x^{(k+1)} = A^{d+2} x^{AT}. \tag{3.7}$$

Suppose for the moment that we are so lucky that we know the *exact* eigenvalues λ_2 and λ_3. The following proposition shows that the spectrum of the Google matrix can be deflated effectively after the extrapolation procedure.

Proposition 1. *Under the above assumptions, we have*

$$x^{(k+1)} - (\lambda_2 + \lambda_3) x^{(k)} + \lambda_2 \lambda_3 x^{(k-1)} \in span\{x_1, X_4, \ldots, X_s\}. \tag{3.8}$$

Proof. By (3.6) and (3.7), we have

$$x^{(k-1)} = x_1 + \alpha_2 \lambda_2^d x_2 + \alpha_3 \lambda_3^d x_3 + \sum_{j=4}^{n} X_j J_j^d z_j,$$

$$x^{(k)} = x_1 + \alpha_2 \lambda_2^{d+1} x_2 + \alpha_3 \lambda_3^{d+1} x_3 + \sum_{j=4}^{n} X_j J_j^{d+1} z_j,$$

and

$$x^{(k+1)} = x_1 + \alpha_2 \lambda_2^{d+2} x_2 + \alpha_3 \lambda_3^{d+2} x_3 + \sum_{j=4}^{n} X_j J_j^{d+2} z_j.$$

Therefore,

$$\begin{aligned} x^{(k+1)} - (\lambda_2 + \lambda_3) x^{(k)} + \lambda_2 \lambda_3 x^{(k-1)} &= (1-\lambda_2)(1-\lambda_3) x_1 + \sum_{j=4}^{s} X_j [J_j^{d+2} - (\lambda_2+\lambda_3) J_j^{d+1} + \lambda_2 \lambda_3 J_j^d] z_j \\ &\in span\{x_1, X_4, \ldots, X_n\}. \quad \square \end{aligned}$$

Note that the initial vector $\tilde{x}_1$ (which is obtained from the extrapolation procedure) for the next Arnoldi iteration is

$$\tilde{x}_1 = \frac{x^{(k+1)} - (\lambda_2 + \lambda_3) x^{(k)} + \lambda_2 \lambda_3 x^{(k-1)}}{\|x^{(k+1)} - (\lambda_2 + \lambda_3) x^{(k)} + \lambda_2 \lambda_3 x^{(k-1)}\|_2}. \tag{3.9}$$

It is well known that the set of diagonalizable matrices is dense in $\mathbb{C}^{n\times n}$ [2, p. 318]. Therefore, it is interesting to consider the case when the Google matrix is diagonalizable. The following theorem indicates how good the subspace $\mathcal{K}_m(A, \tilde{x}_1)$ will be under the above assumptions.

Theorem 2. *Let $\tilde{x}_1$ be defined in (3.9). Let $Q_1 = x_1 x_1^H$ be the orthogonal projector onto the right eigenspace, and let $B_1 = (I - Q_1) A (I - Q_1)$. Define*

$$\epsilon_m = \min_{\substack{p \in \mathbb{P}_{m-1} \\ p(\lambda_1)=1}} \|p(B_1)\|_2. \tag{3.10}$$

If A is diagonalizable, and $x^{AT} = x_1 + \sum_{j=2}^n \alpha_j x_j$, where $x_1, x_2, \ldots, x_n$ are eigenvectors corresponding to $1, \lambda_2, \ldots, \lambda_n$, then

$$\|(I - \mathcal{P}_m) x_1\|_2 \le \frac{\epsilon_m}{\rho \cos \angle(\tilde{x}_1, e)} |\lambda_4|^d \cdot \left| \sum_{j=4}^{n} \alpha_j [\lambda_j^2 - (\lambda_2 + \lambda_3)\lambda_j + \lambda_2 \lambda_3] \right| \cdot \sin \angle(x_1, x_j), \tag{3.11}$$

where $\mathcal{P}_m$ denotes the orthogonal projection onto the subspace $\mathcal{K}_m(A, \tilde{x}_1)$, and $\rho = \|x^{(k+1)} - (\lambda_2 + \lambda_3) x^{(k)} + \lambda_2 \lambda_3 x^{(k-1)}\|_2$ is the scaling factor.

Proof. It follows from (3.2) that

$$\|(I - \mathcal{P}_m) x_1\|_2 \le \min_{\substack{p \in \mathbb{P}_{m-1} \\ p(\lambda_1)=1}} \|p(B_1)\|_2 \cdot \frac{\|(I - Q_1)\tilde{x}_1\|_2}{\cos \angle(\tilde{x}_1, e)}, \tag{3.12}$$

here we used the fact that the left eigenvector of A corresponding to 1 is $e = (1, 1, \ldots, 1)^T$. Therefore, if A is diagonalizable and $x^{AT} = x_1 + \sum_{j=2}^n \alpha_j x_j$, then

$$x^{(k-1)} = A^d x^{AT} = x_1 + \alpha_2\lambda_2^d x_2 + \alpha_3\lambda_3^d x_3 + \sum_{j=4}^{n}\alpha_j\lambda_j^d x_j,$$

$$x^{(k)} = A^{d+1} x^{AT} = x_1 + \alpha_2\lambda_2^{d+1} x_2 + \alpha_3\lambda_3^{d+1} x_3 + \sum_{j=4}^{n}\alpha_j\lambda_j^{d+1} x_j,$$

and

$$x^{(k+1)} = A^{d+2} x^{AT} = x_1 + \alpha_2\lambda_2^{d+2} x_2 + \alpha_3\lambda_3^{d+2} x_3 + \sum_{j=4}^{n}\alpha_j\lambda_j^{d+2} x_j.$$

Some elementary algebra manipulation yields

$$\tilde{x}_1 = \rho^{-1}\left\{(1-\lambda_2)(1-\lambda_3)x_1 + \sum_{j=4}^{n}\alpha_j[\lambda_j^{d+2} - (\lambda_2+\lambda_3)\lambda_j^{d+1} + \lambda_2\lambda_3\lambda_j^d]x_j\right\}. \tag{3.13}$$

So we have

$$(I-Q_1)\tilde{x}_1 = \rho^{-1}\sum_{j=4}^{n}\alpha_j[\lambda_j^{d+2} - (\lambda_2+\lambda_3)\lambda_j^{d+1} + \lambda_2\lambda_3\lambda_j^d](I - x_1x_1^T)x_j,$$

and

$$\|(I-Q_1)\tilde{x}_1\|_2 \le \rho^{-1}|\lambda_4|^d \cdot \left|\sum_{j=4}^{n}\alpha_j[\lambda_j^2 - (\lambda_2+\lambda_3)\lambda_j + \lambda_2\lambda_3]\right| \cdot \sin\angle(x_1, x_j), \tag{3.14}$$

where we used $\sin\angle(x_1, x_j) = \|(I - x_1x_1^T)x_j\|_2$. $\square$

Remark 3. A number of papers give a detailed analysis of minimum of $\|p(A)v\|_2$ or $\|p(A)\|_2$ under a normalization assumption of the form $p(0) = 1$. In [42], it is shown that for any polynomials p, there holds

$$\|p(A)\|_2 \le 11.08 \sup_{z\in\mathcal{W}(A)} |p(z)|,$$

where $\mathcal{W}(A)$ denotes the numerical range of A [42]. This bound can be easily applied to Theorem 2 for estimating ϵ_m.

However, it is one thing to know that an approximation exists in a Krylov subspace and another to have it in hand [35]. It is necessary to consider the convergence of the Rayleigh–Ritz procedure itself. Using $\tilde{x}_1$ as the initial guess for the *next* Arnoldi iteration, the Arnoldi process constructs an orthogonal basis for the Krylov subspace $\mathcal{K}_m(A, \tilde{x}_1) = span\{\tilde{x}_1, A\tilde{x}_1, \ldots, A^{m-1}\tilde{x}_1\}$. Therefore, for any $u \in \mathcal{K}_m(A, \tilde{x}_1)$, there is a polynomial $p(\lambda) \in \mathbb{P}_{m-1}$, such that $u = p(A)\tilde{x}_1$. Hence,

$$\begin{aligned}(A-I)u &= (A-I)p(A)\tilde{x}_1\\ &= p(A)(A-I)\tilde{x}_1.\end{aligned} \tag{3.15}$$

Note that

$$\begin{aligned}(A-I)\tilde{x}_1 &= \rho^{-1}\left\{\sum_{j=4}^{n}\alpha_j\lambda_j^{d+1}[\lambda_j^2 - \lambda_j(\lambda_2+\lambda_3) + \lambda_2\lambda_3]x_j - \sum_{j=4}^{n}\alpha_j\lambda_j^{d}[\lambda_j^2 - \lambda_j(\lambda_2+\lambda_3) + \lambda_2\lambda_3]x_j\right\}\\ &= \rho^{-1}\sum_{j=4}^{n}\alpha_j(\lambda_j^{d+1} - \lambda_j^d)[\lambda_j^2 - \lambda_j(\lambda_2+\lambda_3) + \lambda_2\lambda_3]x_j.\end{aligned} \tag{3.16}$$

So we have

$$(A-I)u = \rho^{-1}\sum_{j=4}^{n}\alpha_j(\lambda_j^{d+1} - \lambda_j^d)[\lambda_j^2 - \lambda_j(\lambda_2+\lambda_3) + \lambda_2\lambda_3]\cdot p(\lambda_j)x_j,$$

and

$$\|(A-I)u\|_2 \le \rho^{-1}\max_{4\le j\le n}|p(\lambda_j)| \sum_{j=4}^{n}|\alpha_j|\cdot|\lambda_j^{d+1} - \lambda_j^d|\cdot|\lambda_j^2 - \lambda_j(\lambda_2+\lambda_3) + \lambda_2\lambda_3|. \tag{3.17}$$

On the other hand, we have from (3.13) that

$$\begin{aligned} u &= p(A)\tilde{x}_1 \\ &= \rho^{-1}\left\{(1-\lambda_2-\lambda_3+\lambda_2\lambda_3)p(1)x_1 + \sum_{j=4}^{n}\alpha_j\lambda_j^d[\lambda_j^2-(\lambda_2+\lambda_3)\lambda_j+\lambda_2\lambda_3]p(\lambda_j)x_j\right\} \\ &= \rho^{-1}\begin{bmatrix} x_1 & x_4 & \cdots & x_n\end{bmatrix}\begin{bmatrix} p(1) & & & \\ & p(\lambda_4) & & \\ & & \ddots & \\ & & & p(\lambda_n)\end{bmatrix}\begin{bmatrix} 1-\lambda_2-\lambda_3+\lambda_2\lambda_3 \\ \alpha_4\lambda_4^d[\lambda_4^2-(\lambda_2+\lambda_3)\lambda_4+\lambda_2\lambda_3] \\ \vdots \\ \alpha_n\lambda_n^d[\lambda_n^2-(\lambda_2+\lambda_3)\lambda_n+\lambda_2\lambda_3]\end{bmatrix}. \end{aligned}$$

So we obtain

$$\|u\|_2 \geq \rho^{-1}\sigma_{\min}([x_1, x_4, \dots, x_n])|p(1)|\ |1-(\lambda_2+\lambda_3)+\lambda_2\lambda_3|. \tag{3.18}$$

Note that $1 > |\lambda_2| \geq |\lambda_3|$, and $1-(\lambda_2+\lambda_3)+\lambda_2\lambda_3 \neq 0$, and we have from (3.17) and (3.18) that

$$\frac{\|(A-I)u\|_2}{\|u\|_2} \leq \max_{4\leq j\leq n}|q(\lambda_j)| \cdot \frac{\sum_{j=4}^{n}|\alpha_j|\ |\lambda_j^{d+1}-\lambda_j^d|\ |\lambda_j^2-\lambda_j(\lambda_2+\lambda_3)+\lambda_2\lambda_3|}{\sigma_{\min}([x_1, x_4, \dots, x_n])|1-(\lambda_2+\lambda_3)+\lambda_2\lambda_3|}, \tag{3.19}$$

where $q(\lambda_j) = p(\lambda_j)/p(1) \in \mathbb{P}_{m-1}$ satisfying $q(1) = 1$.

As a conclusion, we obtain the following theorem, whose proof is along the line of Theorem 2 of [38].

Theorem 3. *Under the above assumptions, then for the Arnoldi-Extrapolation method, there holds*

$$\min_{\substack{u\in\mathcal{K}_m(A,\tilde{x}_1),\\ \|u\|_2=1}} \|(A-I)u\|_2 \leq \min_{\substack{q\in\mathbb{P}_{m-1},\\ q(1)=1}} \max_{\lambda\in\mathcal{C}(0,|\lambda_4|)} |q(\lambda)| \cdot \frac{\xi}{\sigma_{\min}([x_1, x_4, \dots, x_n])} \tag{3.20}$$

where $\xi = \sum_{j=4}^{n}\frac{|\alpha_j|\ |\lambda_j^{d+1}-\lambda_j^d|\ |\lambda_j^2-\lambda_j(\lambda_2+\lambda_3)+\lambda_2\lambda_3|}{|1-(\lambda_2+\lambda_3)+\lambda_2\lambda_3|}$, *and* $\mathcal{C}(0, |\lambda_4|)$ *denotes the circle of center the origin and radius* $|\lambda_4|$.

Remark 4. Theorem 3 indicates that the Arnoldi-Extrapolation converges with increasing m. Notice that 1 is not enclosed by $\mathcal{C}(0, |\lambda_4|)$, so it follows from [43, Lemma 4.3] that

$$\min_{\substack{q\in\mathbb{P}_{m-1},\\ q(1)=1}} \max_{\lambda\in\mathcal{C}(0,|\lambda_4|)} |q(\lambda_j)| = |\lambda_4|^{m-1},$$

and $|\lambda_j| \leq \alpha$ for $j = 2, 3, \dots, n$, so that ξ is uniformly bounded from above. However, we should reminder the reader that how rapidly the right hand side of (3.20) tends to zero, also depends on the condition number of $X_1 \equiv [x_1, x_4, \dots, x_n]$. If X_1 were ill-conditioned, the right hand side would tend to zero slowly.

Denote by $\delta = \sigma_{\max}([x_2, x_3, \dots, x_n]) \cdot (1+\alpha)\sqrt{\sum_{j=2}^{n}|\alpha_j|^2}$, using the same trick, we can prove that for the Arnoldi-type algorithm, there holds

$$\min_{\substack{u\in\mathcal{K}_m(A,x^{AT})\\ \|u\|_2=1}} \|(A-I)u\|_2 \leq \min_{\substack{q\in\mathbb{P}_{m-1}\\ q(1)=1}} \max_{\lambda\in\mathcal{C}(0,\alpha)} |q(\lambda_j)| \cdot \frac{\delta}{\sigma_{\min}([x_1, x_2, \dots, x_n])}. \tag{3.21}$$

Compared with (3.21), Theorem 3 indicates that the Arnoldi-Extrapolation algorithm benefits from the deflation of spectrum. This explains in some degree why our new algorithm often performs better than the Arnoldi-type algorithm and the power method for the PageRank problem.

Remark 5. We would like to remind the reader that the Jordan structure of Google matrix is absolutely crucial for understanding sensitivity and stability of the PageRank vector [26,14,29], and it is not sufficient to consider the case of a diagonalizable Google matrix. When the matrix involved is nondiagonalizable, the convergence of the Krylov methods is still an interesting topic hitherto [44,45,43,35].

4. Numerical experiments

In this section we report some numerical experiments to show the numerical behavior of the Arnoldi-Extrapolation algorithm. All the numerical results are obtained with a MATLAB 7.0 implementation on a 1.6 GZ dual core Intel(R) Pentium(R) processor with 1 G main memory. We stress that the performance of the algorithms relies on the computational environment, the initial vector used, as well as the stopping criterion chosen, and so on.

Table 1
Stanford–Berkeley Web matrix, $tol = 10^{-7}$.

	Power	**Qua-Ext**	**Arn-typ**	**Arn-Ext**
$\alpha = 0.85$				
Mat-Vec	78(81)	65(69)	93(96)	60(61)
CPU	27.9(26.5)	26.7(24.5)	48.9(47.0)	25.6(24.0)
Speedup$_{\textbf{Arn-typ}}$	–	–	–	47.6%
$\alpha = 0.90$				
Mat-Vec	119(125)	99(101)	141(144)	86(104)
CPU	42.5(40.9)	40.6(35.9)	74.3(70.1)	37.2(40.4)
Speedup$_{\textbf{Arn-typ}}$	–	–	–	49.9%
$\alpha = 0.95$				
Mat-Vec	241(260)	195(202)	279(282)	167(171)
CPU	86.2(85.0)	80.0(72.0)	147.6(137.6)	71.9(66.2)
Speedup$_{\textbf{Arn-typ}}$	–	–	–	51.3%
$\alpha = 0.99$				
Mat-Vec	1202(1379)	831(881)	1221(1242)	470(549)
CPU	429.7(450.3)	341.2(314.7)	647.4(607.3)	205.6(213.9)
Speedup$_{\textbf{Arn-typ}}$	–	–	–	68.2%

Example 1: Numerical results of the four algorithms on the 683446 × 683446 Stanford–Berkeley Web matrix, where those in parentheses are in terms of residual 2-norms.

For the sake of justice, the same starting vector $x^{(0)} = e/\|e\|_1$ is used for all the algorithms. The stopping criteria are the residual 1-norms

$$\|Ax^{(k)} - x^{(k)}\|_1/\|x^{(k)}\|_1 \leq tol,$$

where $x^{(k)}$ are the approximations obtained by the current iteration of the involved algorithms, and *tol* is a user described tolerance. As was done in [13,41,9,10,14], we choose $\alpha = 0.85, 0.9, 0.95$ and 0.99, respectively, in all the numerical experiments.

For convenience, in all the tables below we have abbreviated the power method, the quadratic extrapolation method [10], the Arnoldi-type algorithm [13], the Power-Arnoldi algorithm [14], and the Arnoldi-Extrapolation algorithm as **Power**, **Qua-Ext**, **Arn-typ**, **Power-Arn** and **Arn-Ext**, respectively. We denote by **Mat-Vec** the number of matrix-vector products, and by **CPU** the CPU time used in seconds.

In this paper, we set the default values $m = 3$, $\ell = 4$, $\beta = \alpha - 0.2$, $maxit = 6$ and $tolnorm = 0.1$ for the Arnoldi-Extrapolation algorithm. For the sake of justice, in all the numerical experiments, we choose $m = 3$ for the Arnoldi-type algorithm, and exploit the quadratic procedure every 6 matrix-vector products in the quadratic extrapolation method, unless otherwise stated.

Example 1. In this example, we compare the power method, the quadratic extrapolation algorithm, the Arnoldi-type algorithm, as well as the Arnoldi-Extrapolation algorithm for the PageRank problem. The test matrix is the 683,446 by 683,446 *Stanford–Berkeley* Web matrix provided by Kamvar (available from http://www.stanford.edu/~sdkamvar/research.html). It contains 683,446 pages and 7.6 million links. All the algorithms will be terminated as soon as the residual norms are below $tol = 10^{-7}$.

In order to describe the efficiency of the new algorithm, we define

$$\text{Speedup}_{\textbf{Arn-typ}} = \frac{\text{CPU}_{Arn\text{-}typ} - \text{CPU}_{Arn\text{-}Ext}}{\text{CPU}_{Arn\text{-}typ}}, \tag{4.1}$$

to be the speedups of the Arnoldi-Extrapolation algorithm with respect to the Arnoldi-type algorithm in terms of CPU time. Table 1 reports the results obtained.

It is easy to see from Table 1 that the Arnoldi-Extrapolation algorithm performs the best in most cases, both in terms of matrix-vector products and CPU time, especially when the damping factor is close to 1. For example, when $\alpha = 0.99$, the speedup relative to the Arnoldi-type algorithm is up to 68.2%. In other words, the new algorithm is about three times faster than the Arnoldi-type algorithm, even if the storage requirements are the same for the two algorithms.

We also report the numerical results obtained in terms of residual 2-norm. One observes that when using residual 2-norm, it costs us a little more matrix-vector products to achieve the same accuracy. To be more precise, when the damping factor is moderate, say 0.85, the matrix-vector products used are comparable, however, when the damping factor is high, say 0.99, the difference becomes more evident. Moreover, it is interesting to see that if one algorithm outperforms the other with residual 1-norm, then it does so by residual 2-norm. The careful reader may also notice that the algorithms with residual 1-norm can be more time consuming than those with residual 2-norm, even when the former require fewer matrix-vector products than the latter. It seems the reason is that the computation of 1-norm of a long column vector is more time consuming than that of 2-norm of the vector with MATLAB.

Table 2
Wikipedia-20051105 Web matrix, $tol = 10^{-6}$.

	Power	Qua-Ext	Power-Arn	Arn-Ext
$\alpha = 0.85$				
Mat-Vec	52	45	40	35
CPU	106.3	97.5	99.4	88.4
$\text{Speedup}_{\textbf{Pow-Arn}}$	–	–	–	11.1%
$\alpha = 0.90$				
Mat-Vec	77	69	63	46
CPU	157.6	149.6	158.1	101.0
$\text{Speedup}_{\textbf{Pow-Arn}}$	–	–	–	36.1%
$\alpha = 0.95$				
Mat-Vec	151	140	95	76
CPU	308.8	303.8	241.0	168.7
$\text{Speedup}_{\textbf{Pow-Arn}}$	–	–	–	30.0%
$\alpha = 0.99$				
Mat-Vec	720	681	330	266
CPU	1474.0	1476.4	850.6	597.4
$\text{Speedup}_{\textbf{Pow-Arn}}$	–	–	–	29.8%

Example 2: Numerical results of the four algorithms on the 1,634,989 × 1,634,989 Wikipedia-20051105 Web matrix, $tol = 10^{-6}$.

Example 2. This example aims to compare the new algorithm with the Power-Arnoldi algorithm proposed recently by the authors [14]. The test matrix is the *Wikipedia-20051105* Web matrix provided by Gleich, which is available from http://www.cise.ufl.edu/research/sparse/matrices/~Gleich/index.html. The matrix is of size 1,634,989 × 1,634,989, containing 19,753,078 nonzero elements.

We run the power method, the quadratic extrapolation algorithm, the Power-Arnoldi algorithm, as well as the Arnoldi-Extrapolation algorithm on this problem. All the algorithms will be stopped as soon as the residual 1-norms are below $tol = 10^{-6}$. Similarly, so as to describe the efficiency of the Arnoldi-Extrapolation algorithm with respect to the Power-Arnoldi algorithm, we define

$$\text{Speedup}_{\textbf{Pow-Arn}} = \frac{\text{CPU}_{\textit{Pow-Arn}} - \text{CPU}_{\textit{Arn-Ext}}}{\text{CPU}_{\textit{Pow-Arn}}}, \tag{4.2}$$

to be the speedups of the new algorithm in terms of CPU time.

Table 2 lists the results. Several comments are in order. First, when the damping factor is moderate, say, $\alpha = 0.85$, the speedup is marginal, however, when the damping factor is close to 1, the speedup is impressive. Second, although the two methods are different in terms of cost per iteration (recall that the storage requirements are the same), we see from Table 2 that the Arnoldi-Extrapolation algorithm is superior to the Power-Arnoldi algorithm. For example, when $\alpha = 0.90$ and 0.95, the speedups are 36.1% and 30%, respectively. This illustrates that the Arnoldi-Extrapolation algorithm works better when m is small, which is favorable for extremely large sparse matrices such as Web matrices. Third, the number of matrix-vector products shows the numerical behavior of the algorithms in some sense. As we have pointed out in Section 2, the inner products add a substantial amount of computational work. Indeed, the speedups in terms of matrix-vector products are lower than those defined by CPU time in (4.2).

Example 3. The Arnoldi-Extrapolation algorithm is parameter-dependent. In this example, we try to show that the performance of the Arnoldi-Extrapolation algorithm is insensitive to the choice of *tolnorm*, refer to Step (24) of Algorithm 4. The test matrix is the *California* Web matrix (available from http://www.cs.cornell.edu/Courses/~cs685/~2002fa/), which is a widely used test problem [6,11,25,14]. It contains 9664 nodes and 16,150 links. We run the power method, the quadratic extrapolation algorithm, the Arnoldi-type algorithm (with $m = 3, 4, 5, 6$), and the new algorithm on this problem. All the algorithms are stopped as soon as the residual norms are below $tol = 10^{-8}$. Tables 3 and 4 give the results.

Two remarks are given. First, one observes from Table 3 that the numerical behavior of the Arnoldi-type algorithm strongly relies on the choice of m, i.e., the dimension of the Krylov subspace. For example, when m is relatively small, say $m = 3$, the Arnoldi-type algorithm is not better than power method. However, as m increases, the improvement is impressive. Second, as we expected, the new algorithm outperform the other algorithms in many cases. Specifically, it is interesting to see that the Arnoldi-Extrapolation algorithm with $m = 3$ even performs better than the Arnoldi-type algorithm with $m = 6$.

In Fig. 1, we plot the matrix-vector products required by $tolnorm = 0, 0.1, 0.2, \ldots, 1$. Recall that choosing $tolnorm = 0$ corresponds to *without* the threshold $\eta \geq tolnorm$, refer to Step (24) of Algorithm 4. On the other hand, a higher *tolnorm* (e.g., $tolnorm = 1$) corresponds to the condition for exploiting extrapolation procedure is stringent. We see from Fig. 1 that the number of matrix-vector products is insensitive to the choice of *tolnorm*, especially when the damping factor is moderate, say 0.85 and 0.90.

Another interesting question is whether the threshold $\eta \geq tolnorm$ is necessary in practice. To see this, for $\alpha = 0.95$, we depict in Fig. 2 the convergence history of the Arnoldi-Extrapolation algorithm *with* and *without* the threshold. It is obvious to see that the new algorithm converges irregularly without the threshold, which implies that the threshold is indispensable.

Table 3
Arnoldi-type algorithm on California matrix.

α	0.85	0.90	0.95	0.99
$m = 3$	102	156	312	1419
$m = 4$	64	88	172	796
$m = 5$	55	80	130	500
$m = 6$	42	60	108	252

Example 3: Matrix-vector products of the Arnoldi-type algorithm (with various m) on the 9664 × 9664 California Web matrix, $tol = 10^{-8}$.

Table 4
Three algorithms on California matrix, $tol = 10^{-8}$.

α	0.85	0.90	0.95	0.99
Power	73	112	228	1104
Qua-Ext	65	99	207	963
Arn-Ext	37	57	55	94

Example 3: Matrix-vector products of the three algorithms on the 9664 × 9664 California Web matrix, *tolnorm* = 0.1 for the Arnoldi-Extrapolation algorithm.

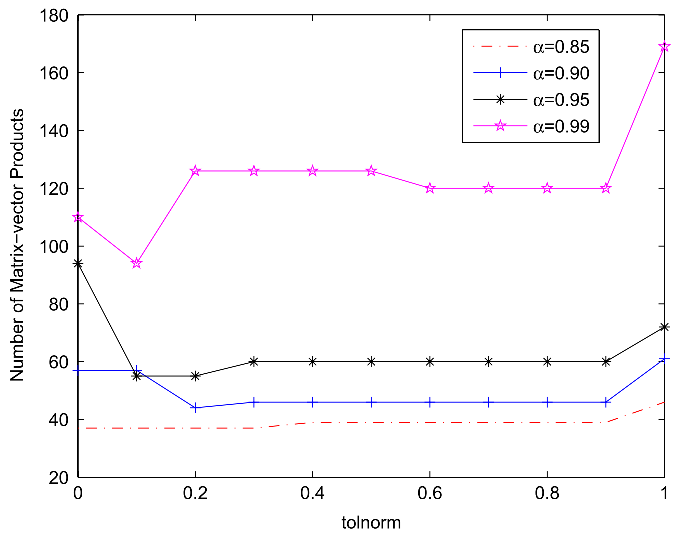

Fig. 1. Example 3: Matrix-vector products for various *tolnorm* in the Arnoldi-Extrapolation algorithm, $tol = 10^{-8}$.

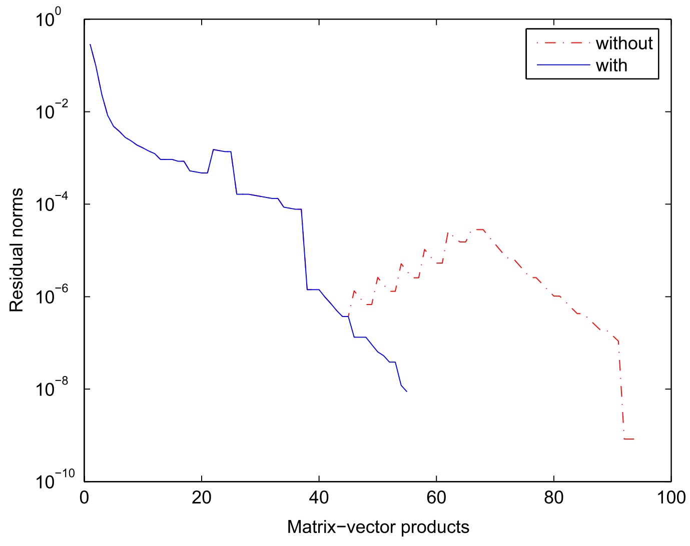

Fig. 2. Example 3: Convergence curves of the Arnoldi-Extrapolation algorithm *with* and *without* the threshold $\eta \geq tolnorm$, $\alpha = 0.95$, $tol = 10^{-8}$.

Table 5
Three algorithms on the CS-Stanford Web matrix.

α	0.85	0.90	0.95	0.99
$\beta = \alpha$	84	123	232	1144
$\beta = \alpha - 0.1$	54	67	90	198
$\beta = \alpha - 0.2$	50	60	90	188
$\beta = \alpha - 0.3$	52	62	91	188
$\beta = \alpha - 0.4$	52	61	94	190
$\beta = \alpha - 0.5$	52	61	90	190
Power	84	123	232	1144
Arn-typ	96	138	255	1158

Example 4: Number of matrix-vector products of the Arnoldi-Extrapolation algorithm, the power method, and the Arnoldi-type algorithm, $tol = 10^{-8}$.

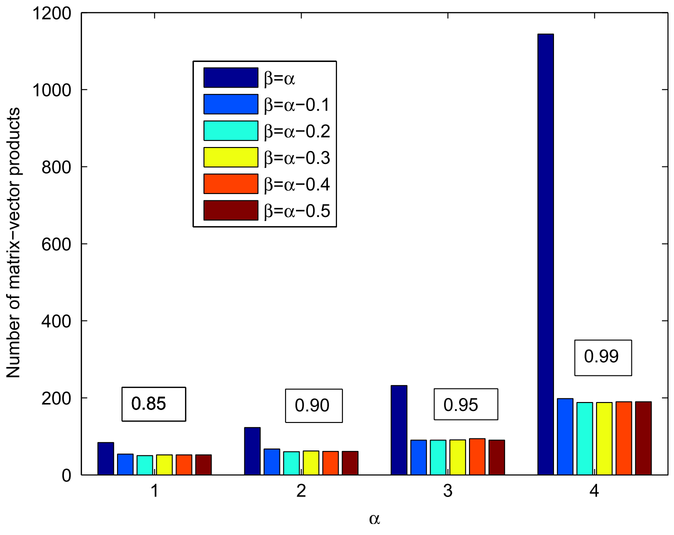

Fig. 3. Example 4: The CS-Stanford Web matrix: Matrix-vector products versus β; $\alpha = 0.85, 0.90, 0.95, 0.99$.

Example 4. The test matrix is the *CS-Stanford* Web matrix, which is available from http://www.cise.ufl.edu/research/sparse/matrices/~Gleich/index.html. It contains 9914 nodes and 35,555 links. We see from Step (26) of Algorithm 4 that the convergence of Algorithm 4 depends on the choice of β. In this example, we try to show that the performance of the new algorithm is also insensitive to the choice of β, and what we should do is to set β to be smaller than α. To do this, we pick $\beta = \alpha, \alpha - 0.1, \alpha - 0.2, \dots, \alpha - 0.5$, respectively.

We run the power method, the Arnoldi-type algorithm, and the Arnoldi-Extrapolation algorithm on this Web matrix. The convergence tolerance is $tol = 10^{-8}$. Table 5 lists our results. In Fig. 3 we depict the bar graph of the number of matrix-vector products versus β, for $\alpha = 0.85, 0.90, 0.95$ and 0.99, respectively. Recall that the *asymptotic* convergence rate of the power method is α [2], this implies that when $\beta = \alpha$, the Arnoldi-Extrapolation algorithm is nothing but the power method, see Steps (2)–(12) of Algorithm 4. We can see from Table 5 and Fig. 3 that the performance of the Arnoldi-Extrapolation algorithm is insensitive to the choice of β, and it is necessary that β be smaller than α.

Example 5. In this example, we aim to show that the number of matrix-vector products required by the Arnoldi-Extrapolation algorithm is (relatively) insensitive to the choice of the convergence tolerance. The test matrix is the *Epa* Web matrix, which is available from http://www.cs.cornell.edu/~Courses/cs685/2002fa. It contains 4772 nodes and 8965 links.

We run the power method, the quadratic extrapolation method, the Arnoldi-type algorithm, and the Arnoldi-Extrapolation algorithm on this example. In order to show that the performance of the Arnoldi-Extrapolation algorithm is (relatively) insensitive to the choice of *tol*, we choose *tol* to be $10^{-6}, 10^{-7}, \dots, 10^{-10}$, respectively, and define

$$\textbf{incr-mv} = \frac{\text{Mat-Vec}_{\text{tol}=1\text{e}-10} - \text{Mat-Vec}_{\text{tol}=1\text{e}-6}}{\text{Mat-Vec}_{\text{tol}=1\text{e}-6}},$$

as the increase of the number of matrix-vector products when $tol = 10^{-10}$ vs. $tol = 10^{-6}$. Table 6 lists the results.

It is seen from Table 6 that the Arnoldi-Extrapolation algorithm is (relatively) insensitive to the choice of *tol*. For instance, when $\alpha = 0.95$, the increase of the power method and the quadratic extrapolation algorithm are 101.7% and 145.6%, respectively, while that of the Arnoldi-Extrapolation algorithm is only 50%. This shows that the new algorithm may be

Table 6
The Epa Web matrix.

	Power	**Qua-Ext**	**Arn-typ**	**Arn-Ext**
$\alpha = 0.85$				
$tol = 10^{-6}$	56	32	51	33
$tol = 10^{-7}$	70	33	60	36
$tol = 10^{-8}$	84	39	69	39
$tol = 10^{-9}$	98	46	78	46
$tol = 10^{-10}$	112	57	84	51
incr-mv	100%	78.1%	64.7%	54.5%
$\alpha = 0.90$				
$tol = 10^{-6}$	86	33	63	38
$tol = 10^{-7}$	107	41	78	43
$tol = 10^{-8}$	129	56	81	50
$tol = 10^{-9}$	150	68	90	56
$tol = 10^{-10}$	172	77	96	62
incr-mv	100%	133.3%	52.4%	63.2%
$\alpha = 0.95$				
$tol = 10^{-6}$	173	57	105	40
$tol = 10^{-7}$	217	75	123	42
$tol = 10^{-8}$	261	99	141	51
$tol = 10^{-9}$	305	118	159	60
$tol = 10^{-10}$	349	140	180	60
incr-mv	101.7%	145.6%	71.4%	50%
$\alpha = 0.99$				
$tol = 10^{-6}$	837	219	366	60
$tol = 10^{-7}$	1062	266	441	82
$tol = 10^{-8}$	1289	291	528	84
$tol = 10^{-9}$	1517	311	609	87
$tol = 10^{-10}$	1746	341	690	97
incr-mv	108.6%	55.7%	88.5%	61.7%

Example 5: Number of matrix-vector products of the four algorithms with various *tol*.

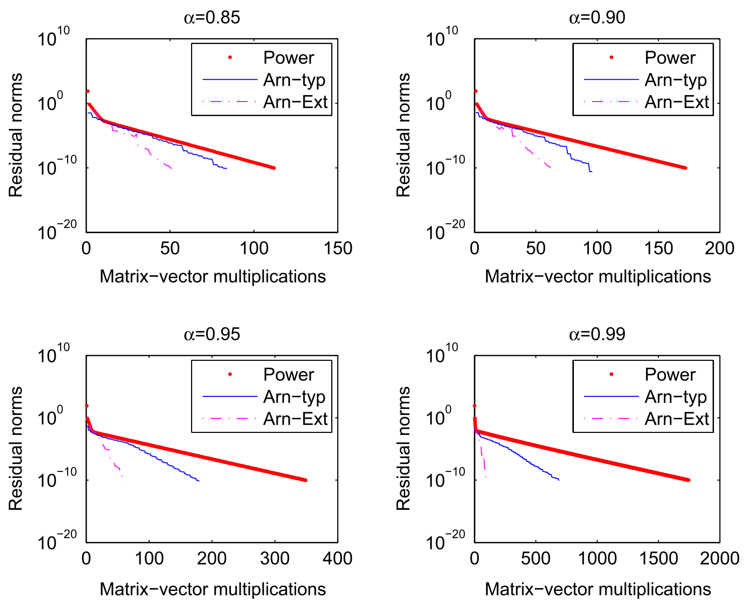

Fig. 4. Example 5: Convergence history of the three algorithms, $tol = 10^{-10}$.

an appropriate choice if high accuracy is required. We also see that the Arnoldi-Extrapolation algorithm and the quadratic extrapolation algorithm is comparable when the damping factor is moderate, say 0.85 and 0.90. However, when the damping factor is high, say 0.95 and 0.99, the Arnoldi-Extrapolation algorithm performs better than the quadratic extrapolation algorithm. Fig. 4 depicts the convergence history of the power method, the Arnoldi-type algorithm, and that of the Arnoldi-Extrapolation algorithm when $tol = 10^{-10}$. It is seen that the Arnoldi-type algorithm converges faster than the power method, while the Arnoldi-Extrapolation algorithm performs the best.

Table 7
The Epa Web matrix, $\alpha = 0.99$, $tol = 10^{-10}$.

Arn-iter	$\tilde{\lambda}_1$	$\tilde{\lambda}_2$	$\tilde{\lambda}_3$
$k = 1$	$0.99425579 + 0.00517722\mathbf{i}$	$0.99425579 - 0.00517722\mathbf{i}$	-0.98952571
$k = 2$	0.99869648	0.98666627	-0.76885465
$k = 3$	1.000362797	$-0.52743885 + 0.38646943\mathbf{i}$	$-0.52743885 - 0.38646943\mathbf{i}$
$k = 4$	1.00004813	-0.87405315	0.80196775
$k = 5$	1.00000033	0.98687139	-0.93940623
$k = 6$	1.00000051	0.68708428	-0.38020555
$k = 7$	0.99999999	-0.96018465	-0.43214249
$k = 8$	1.00000000	0.98736737	-0.47434176

Example 5: Ritz values obtained from the kth Arnoldi-iteration during the Arnoldi-Extrapolation algorithm. The exact eigenvalues are $\lambda_1 = 1$, $\lambda_2 = 0.99$, $\lambda_3 = -0.99$, $\mathbf{i} = \sqrt{-1}$.

The extrapolation procedure developed in Section 3.1 is based on Ritz values. Therefore, an interesting question is how accurate should the Ritz values be in the worst case. In Table 7 we list the three largest Ritz values obtained from the kth Arnoldi iteration during the Arnoldi-Extrapolation iterations. Notice that the extrapolation procedure was not applied when $k = 1$, where the largest Ritz values are complex. We see from Table 7 and Fig. 4 (the right-bottom figure) that the Arnoldi-Extrapolation algorithm still works, even if the Ritz values are not accurate enough.

5. Conclusion and future work

The Arnoldi-type algorithm may not be efficient for the PageRank problem when the damping factor is high and the dimension of search subspace is small. In this paper, we investigate an Arnoldi-Extrapolation algorithm for improving the efficiency of the Arnoldi-type algorithm. We first present an extrapolation procedure based on Ritz values. We then consider how to periodically knit this extrapolation procedure together with the Arnoldi-type algorithm. The resulting algorithm is the Arnoldi-Extrapolation algorithm. In the new algorithm, the dimension of the Krylov subspace can be chosen very moderate, so that the memorization cost is kept reasonable. Other advantages of the proposed technique are the potential use on parallel architectures and its applicability for a wide range of the parameter α which can be set close to 1. Numerical experiments demonstrate that the new algorithm is often more powerful than the Arnoldi-type algorithm and the power method.

However, there is still a lot of work needs to be done. For instance, how to understand the convergence of the Arnoldi-Extrapolation algorithm when the Google matrix is nondiagonalizable? How accurate should the Ritz values be in the worst case? How to pick the *optimal* parameters for this algorithm? The Google matrix has an abundance of structure which is failed to be exploited in the new algorithm. In a realistic Web graph, more than half of the rows in A are the same, due to the large number of Web pages such as *pdf* or *jpeg* files that have no outlinks [23,11,24]. The Google matrix can be reordered according to dangling nodes and non-dangling nodes of the matrix [23,12,7,11,24], which reduces the computation of the PageRank to that of solving a much smaller problem. Indeed, the new algorithm can also be utilized in combination with these reordering schemes. We expect the resulting algorithm is promising. Furthermore, with some modifications, the Arnoldi-Extrapolation algorithm can also be applied to the GeneRank problem [46]. This is under investigation and is definitely a part of our future work.

Acknowledgements

We would like to express our sincere thanks to the Guest Editor Professor S. Serra-Capizzano and the referee for their detailed comments that enabled us to greatly improve the presentation of this paper.

The first author was supported by the National Science Foundation of China under grant 10901132, the Natural Science Foundation for Colleges and Universities in Jiangsu Province under grant 08KJB110012, the Qing-Lan Project of Jiangsu Province under grant QL200612, and the Natural Science Foundation of Xuzhou Normal University under grant 08XLY01. The second author was supported by the National Natural Science Foundation of China under grant 10871051, Shanghai Education Committee (Dawn Project under grant 08SG01), Shanghai Science & Technology Committee under grant 09DZ2272900 and Doctoral Program of the Ministry of Education under grant 20090071110003.

References

[1] L. Page, S. Brin, R. Motwami, T. Winograd, The Pagerank citation ranking: Bring order to the web, Technical report, Computer Science Department, Stanford University, 1998.
[2] G. Golub, C. Van Loan, Matrix Computations, 3rd ed., The Johns Hopkins University Press, Baltimore and London, 1996.
[3] L. Eldén, The eigenvalues of the Google matrix, Technical Report, LiTH-MAT-RC2004-01.
[4] T. Haveliwala, S. Kamvar, The Second eigenvalue of the Google matrix, Stanford University Technical Report, 2003.

[5] P. Berkhin, A survey on PageRank computing, Internet Math. 2 (1) (2005) 73–120.
[6] A. Langville, C. Meyer, Google's PageRank and Beyond: The Science of Search Engine Rankings, Princeton University Press, 2006.
[7] A. Langville, C. Meyer, Deeper inside PageRank, Internet Math. 1 (3) (2005) 335–380.
[8] A. Langville, C. Meyer, A survey of eigenvector methods of web information retrieval, SIAM Rev. 47 (1) (2005) 135–161.
[9] S. Kamvar, T. Haveliwala, G. Golub, Adaptive methods for the computation of PageRank, Linear Algebra Appl. 386 (2004) 51–65.
[10] S. Kamvar, T. Haveliwala, C. Manning, G. Golub, Extrapolation methods for accelerating PageRank computations, in: Twelfth International World Wide Web Conference, 2003.
[11] A. Langville, C. Meyer, A reordering for the PageRank problem, SIAM J. Sci. Comput. 27 (6) (2006) 2112–2120.
[12] S. Kamvar, T. Haveliwala, C. Manning, G. Golub, Exploiting the block structure of the web for computing PageRank, Stanford University Technical Report, SCCM-03-02, 2003.
[13] G. Golub, C. Greif, An Arnoldi-type algorithm for computing PageRank, BIT 46 (2006) 759–771.
[14] G. Wu, Y. Wei, A Power–Arnoldi algorithm for computing PageRank, Numer. Linear Algebra Appl. 14 (2007) 521–546.
[15] R. Morgan, M. Zeng, A harmonic restarted Arnoldi algorithm for calculating eigenvalues and determining multiplicity, Linear Algebra Appl. 45 (2006) 96–113.
[16] K. Avrachenkov, N. Litvak, D. Nemirovsky, N. Osipova, Monte Carlo methods in PageRank computation: When one iteration is sufficient, SIAM J. Numer. Anal. 45 (2) (2007) 890–904.
[17] C. Brezinski, M. Redivo-Zaglia, Rational extrapolation for the PageRank vector, Math. Comput. 77 (2008) 1585–1598.
[18] C. Brezinski, M. Redivo-Zaglia, The PageRank vector: Properties, computation, approximation, and acceleration, SIAM J. Matrix Anal. Appl. 28 (2006) 551–575.
[19] C. Brezinski, M. Redivo-Zaglia, S. Serra-Capizzano, Extrapolation methods for PageRank computations, C. R. Acad. Sci. Pairs, Ser. I 340 (2005) 393–397.
[20] G. Corso, A. Gulli, F. Romani, Fast PageRank via a sparse linear system, Internet Math. 2 (3) (2006) 251–273.
[21] J. Ding, A. Zhou, A spectral theorem for rank-one updated matrices with some applications, Appl. Math. Lett. 20 (2007) 1223–1226.
[22] R. Horn, S. Serra-Capizzano, A general setting for the parametric Google matrix, Internet Math. 3 (4) (2008).
[23] I. Ipsen, T. Selee, PageRank computation, with special attention to dangling nodes, SIAM J. Matrix Anal. Appl. 29 (4) (2007) 1281–1296.
[24] Y. Lin, X. Shi, Y. Wei, On computing PageRank via lumping the Google matrix, J. Comput. Appl. Math. 224 (2009) 702–708.
[25] A. Langville, C. Meyer, Updating the stationary vector of an irreducible Markov chain with an eye on Google's PageRank, SIAM J. Matrix Anal. 27 (4) (2006) 968–987.
[26] S. Serra-Capizzano, Jordan canonical form of the Google matrix: A potential contribution to the PageRank computation, SIAM Matrix Anal. Appl. 27 (2) (2005) 305–312.
[27] S. Serra-Capizzano, Google PageRanking problem: The model and the analysis, in: A. Frommer, M. Mahoney, D. Szyld, Proceedings of the Dagstuhl Conference in Web Retrieval and Numerical Linear Algebra Algorithms, 2007.
[28] G. Wu, Eigenvalues and Jordan canonical form of a successively rank-one updated complex matrix with applications to Google's PageRank problem, J. Comput. Appl. Math. 216 (2008) 364–370.
[29] G. Wu, Y. Wei, Comments on Jordan Canonical form of the Google matrix, SIAM J. Matrix Anal. Appl. 30 (1) (2008) 364–374.
[30] A. Berman, R. Plemmons, Nonnegative Matrices in the Mathematical Sciences, in: Classics in Applied Mathematics, 9, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994, Revised reprint of the 1979 original.
[31] C. Meyer, Matrix Analysis and Applied Linear Algebra, SIAM, Philadelphia, 2000.
[32] P. Boldi, M. Santini, S. Vigna, A deeper investigation of PageRank as a function of the damping factor, in: A. Frommer, M. Mahoney, D. Szyld, Dagstuhl Seminar Proceedings: Web Information Retrieval and Linear Algebra Algorithms, 2007.
[33] H. Zhang, A. Goel, R. Govindan, K. Mason, B. Van Roy, Making eigenvector-based reputation system robust to collusion, 2004, available from www.stanford.edu/group/reputation/WAW-adapt.ps.
[34] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst (Eds.), Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, SIAM, Philadelphia, 2000.
[35] G. Stewart, Matrix Algorithms: Vol. II Eigensystems, SIAM, Philadelphia, PA, 2001.
[36] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd ed., SIAM, Philadelphia, PA, 2003.
[37] D. Gleich, L Zhukov, P. Berkhin, Fast parallel PageRank: A linear system approach, Yahoo! Techniqual Report, 2005.
[38] Z. Jia, Refined iterative algorithms based on Arnoldi's process for large unsymmetric eigenproblems, Linear Algebra Appl. 259 (1997) 1–23.
[39] G. Wu, Y. Zhang, Y. Wei, Accelerated Arnoldi algorithms for computing stationary distribution with application to the PageRank and the GeneRank problems (submitted for publication).
[40] M. Bellalij, Y. Saad, H. Sadok, On the convergence of the Arnoldi process for eigenvalue problems, Report umsi-2007-12, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2007.
[41] T. Haveliwala, S. Kamvar, D. Klein, C. Manning, G.H. Golub, Computing PageRank using power extrapolation, Stanford University Technical Report, 2003.
[42] M. Crouzeix, Numerical range and functional calculus in Hilbert space, J. Funct. Anal. 244 (2007) 668–690.
[43] Y. Saad, Numerical methods for large eigenvalue problems, in: Algorithms and Architectures for Advanced Scientific Computing, Manchester University Press, 1992.
[44] C. Beattie, M. Embree, J. Ross, Convergence of restart Krylov subspaces to invariant subspaces, SIAM Matrix Anal. Appl. 25 (2004) 1074–1109.
[45] C. Beattie, M. Embree, D. Sorensen, Convergence of polynomial restart Krylov methods for eigenvalue computation, SIAM Rev. 47 (2005) 492–515.
[46] J. Morrison, R. Breitling, D. Higham, D. Gilbert, GeneRank: Using search engine for the analysis of microarray experiments, BMC Bioinform. 6 (2005) 233–246.