Arnoldi Type Methods for Eigenvalue Calculation

Smit, P.

Publication date:
1994

Citation for published version (APA):
Smit, P. (1994). *Arnoldi Type Methods for Eigenvalue Calculation: Theory and Experiments*. (CentER Discussion Paper; Vol. 1994-9). CentER, Center for Economic Research.

entER
for
mic Research

Discussion paper

[redacted]

TILBURG UNIVERSITY

[redacted]

Center
for
Economic Research

No. 9409

ARNOLDI TYPE METHODS FOR EIGENVALUE CALCULATION: THEORY AND EXPERIMENTS

R45

by Paul Smit

Matrices
Eigenvalues

January 1994

ISSN 0924-7815

ARNOLDI TYPE METHODS FOR EIGENVALUE CALCULATION: THEORY AND EXPERIMENTS [1]

P. SMIT

Abstract

The Arnoldi Algorithm is a well known method for the calculation of eigenvalues. A short overview is given of some of its variants and some error estimates. Several variants have been tested in a series of numerical experiments of which the results are included. The purpose of these experiments is to show the dependence of the convergence of these algorithms on some characteristics of the matrices involved.

1 Introduction

Calculating eigenvalues of a matrix is an important problem in many areas of research. When small matrices are involved the QR algorithm is the best choice. But as the computers grew more powerful they were able to solve larger problems, so the size of the used matrices has been increasing during the last few decades. For instance matrices of high order arise in the discretization of partial differential equations. Chatelin [2] calls a matrix large when it is far cheaper to calculate only a few eigenvalues than the complete spectrum. For large matrices the QR algorithm is too slow (the time is cubic in the dimension of the matrix) and moreover the method destroys the sparsity of the matrix. So other algorithms were required. In the Krylovspace methods of Lanczos and Arnoldi the matrix occurs only in matrix-vector multiplications.

Historical survey. In the early fifties Lanczos [9] and Arnoldi [1] described iterative methods based on Krylov subspace iteration which produce approximations of eigenvalues of a matrix. In [12] Paige pointed out the interest of the iterative use of the Lanczos Method for the computation of extreme eigenvalues. Parlett's "The Symmetric Eigenvalue Problem" [13] gives an analysis of many algorithms, especially the Krylov space methods. In [8] Kaniel uses Chebychev polynomials to achieve as first errorbounds; these bounds were improved by Saad [14, 15]. The pioneering work of Saad has caused a revival of the Arnoldi Method. Polynomial filtering on base of Chebychev polynomials for nonsymmetric problems are applicable since Manteuffel [10, 11]. Their use for the nonsymmetric eigenproblem is analysed in [16], [6] and in [7]. Golub and van Loan [4] describe the state of the art in numerical linear algebra, inclusive Krylov space techniques, Chatelin [2] more

[1]This paper is the result of the project "The Arnoldi Method: Analysis and Numerical Results" performed under the auspices and support of NCF/Cray Research University Grant, under contract CRG 92.20. The original version of this paper as was send to NCF contained a Fortran program which implements the Iterative Arnoldi Algorithm with Deflation.

specificly the algebraic eigenproblem. Block Arnoldi algorithms have been analysed and developed in [19, 20]. Sorenson [21] combines the Arnoldi Algorithm and the QR method in a filtering technique for the construction of new startvectors to perform the Arnoldi Method in an iterative way. The Implicit Schur Deflation technique of Saad [18], as Sorenson's Algorithm, improves the facility to compute several eigenvalues of large sparse nonsymmetric matrices with Arnoldi type iterations. The Block Arnoldi Method [19, 20] seems to be promising for the same objective. Finally, Saad's monograph [18] gives an overview of the many aspects and problems related with the computation of the eigenvalues of large (sparse) nonsymmetric matrices.

About this paper. In this paper we are concerned with the Arnoldi Method and two of its variants. These methods sequentially reduce the given matrix to a Hessenberg form by a projection on Krylov spaces. In section 2 some basic theory about the Arnoldi Algorithm is presented. Section 3 discusses the Iterative Arnoldi Algorithm with Schur deflation. Section 4 presents Sorenson's Iterative Arnoldi Algorithm in which the new startvectors are obtained by a QR filtering process. The results of numerical experiments can be found in section 5. In the examples the attention is directed to three characteristics of the convergence in the Arnoldi process: separation of eigenvalues, condition of the eigenvector matrix and the choice of the startvector. The appendices discuss two implementational problems: the storage of the matrix and the multiplication of a vector by the matrix.

Notation. The following notation is used in this paper. The matrix of which the eigenvalues are to be calculated is called A. This matrix is real, has dimension n and can be diagonalized: $AU = U\Lambda$, with $U = (u_1, \ldots, u_n)$, $\|u_i\| = 1$ for all i and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$. $K_j(A, v)$ denotes the j-th Krylov space with respect to A and vector v and is defined by $K_j(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{j-1}v\}$. $V^{(j)}$ is a matrix whose columns form an orthonormal basis for $K_j(A, v)$ such that $H^{(j)} = V^{(j)T}AV^{(j)}$ is a Hessenbergmatrix of dimension j. $H^{(j)}y_i = \theta_i y_i$, $\|y_i\| = 1$ for $i = 1, \ldots, j$. When we write "norm" or $\|\cdot\|$ we mean $\|\cdot\|_2$.

2 Arnoldi's Algorithm and variants

2.1 Arnoldi's Algorithm

To obtain approximations of the eigenvalues of A we can restrict this matrix to a subspace of relative low dimension and calculate its eigenvalues. We get an iterative method if we add a dimension to the subspace in each iteration. When the Krylov subspaces $K_j(A, v_1)$ are used, we get the Arnoldi Algorithm. Let us assume that $K_j(A, v_1)$ has dimension j, then $v_1, Av_1, A^2v_1, \ldots, A^{j-1}v_1$ are independent and we can use the Gram-Schmidt orthogonalization method to obtain an orthonormal basis for this subspace: $v_1, v_2, \ldots, v_j$. Let $H^{(j)}$ denote the restriction of A to $K_j(A, v_1)$ with that basis and $V^{(j)}$ the orthogonal matrix whose columns are the basisvectors. So:

$$AV^{(j)} - V^{(j)}H^{(j)} \perp K_j(A, v_1) \tag{1}$$

or:

$$V^{(j)T}AV^{(j)} = H^{(j)} \tag{2}$$

If $i > j+1$ then $v_i \perp K_{j+1}(A, v_1)$ so $H^{(j)}_{i,j} = (v_i, Av_j) = 0$. This shows that $H^{(j)}$ is a Hessenbergmatrix and also that:

$$AV^{(j)} = V^{(j)}H^{(j)} + r^{(j)}e_j^T \tag{3}$$

where $r^{(j)} \perp K_j(A, v_1)$ and e_j is the j-th unit vector of $\mathbb{R}^j$. We call this an *Arnoldi factorization* of A. The j eigenvalues and -vectors of $H^{(j)}$ are given by:

$$H^{(j)}y_i = \theta_i y_i \qquad (i = 1, \ldots, j) \tag{4}$$

These can be computed efficiently because $H^{(j)}$ is a Hessenberg matrix. We can look upon the θ_i's as approximations of eigenvalues of A in some sense (more about that later). Similarly the vectors $V^{(j)}y_i$ can be regarded as approximations of eigenvectors of A. We call θ_i a *Ritzvalue* and $V^{(j)}y_i$ a *Ritzvector* of A with respect to $K_j(A, v_1)$.
All this leads to the following algorithm that generates the matrices $H^{(j)}$ and $V^{(j)}$.

Algorithm 1 (Arnoldi)

```
1. Choose $v_1$ of norm one.
2. do $j = 1, 2, 3, \ldots$ :
       $w = Av_j$
       do $i = 1, \ldots, j$ :
           $H_{i,j} = (w, v_i)$
       enddo
       $w = w - \sum_{i=1}^{j} H_{i,j} v_i$
       $H_{j+1,j} = \|w\|$
       $v_{j+1} = w / H_{j+1,j}$
       Calculate the eigenvalues $\theta_1, \ldots, \theta_j$ of $H$.
       If the "wanted" eigenvalues have "converged" then stop.
   enddo
```

The terms "wanted" and "converged" are still vague, but section 2.3 tries to make them more precise.
For reasons of numerical stability it is better to use the (in exact arithmetic equivalent) modified Gram-Schmidt method, which gives this alternative algorithm.

Algorithm 2 (Arnoldi, using modified Gram-Schmidt)

```
1. Choose $v_1$ of norm one.
2. do $j = 1, 2, 3, \ldots$ :
       $w = Av_j$
       do $i = 1, \ldots, j$ :
            $H_{i,j} = (w, v_i)$
            $w = w - H_{i,j} v_i$
        enddo
        $H_{j+1,j} = \|w\|$
        $v_{j+1} = w / H_{j+1,j}$
        Calculate the eigenvalues $\theta_1, \ldots, \theta_j$ of $H$.
        If the "wanted" eigenvalues have "converged" then stop.
    enddo
```

2.2 On the meaning of some error estimates

Let π_j be the orthogonal projection on $K_j(A, v_1)$, P_i the spectral projection on the eigenspace corresponding with λ_i. A relevant theorem to understand the Arnoldi Algorithm is the following.

Theorem 3 *Assume $P_i v_1 \neq 0$. $C(U) = \|U\| \cdot \|U^{-1}\|$ denotes the condition number of U and $\mathcal{P}_j^x$ is the set of polynomials of degree j which take the value one in x. Then*

$$\|(I - \pi_j)u_i\| \leq \frac{\|(I - P_i)v_1\|}{\|P_i v_1\|} \cdot C(U) \cdot \min_{p \in \mathcal{P}_{j-1}^{\lambda_i}} \max_{k \neq i} |p(\lambda_k)|$$

Proof: See [2, p.181]. ■

With the notation

$$\varepsilon_i^{(j)} = \min_{p \in \mathcal{P}_{j-1}^{\lambda_i}} \max_{k \neq i} |p(\lambda_k)| \tag{5}$$

theorem 3 reads

$$\sin \phi_i^{(j)} = \frac{\|(I - P_i)v_1\|}{\|P_i v_1\|} \cdot C(U) \cdot \varepsilon_i^{(j)} \tag{6}$$

where $\phi_i^{(j)}$ is the angle between eigenvector u_i and Krylovspace $K_j(A, v_1)$. In (6) $\frac{\|(I-P_i)v_1\|}{\|P_i v_1\|}$ indicates the influence of the angle between startvector v_1 and eigenvector u_i on $\phi_i^{(j)}$. The following theorem indicates the influence of the distribution of the eigenvalues on the angle $\phi_i^{(j)}$.

Theorem 4 *There exist j eigenvalues say $\lambda_1, \lambda_2, \ldots, \lambda_j$ each different from λ_i such that*

$$\varepsilon_i^{(j)} = \left(\sum_{k=1}^{j} \prod_{l \neq k} \left|\frac{\lambda_l - \lambda_i}{\lambda_l - \lambda_k}\right|\right)^{-1}$$

Proof: See [2, p.191]. ■

Chatelin [2] proves the existence of a constant c such that

$$|\lambda_i - \theta_i^{(j)}| \leq c \|(I - \pi_j)u_i\| \tag{7}$$

where c depends on $C(U)$. This emphasizes the role of the condition of the eigenvalue problem for the convergence of the Ritzvalues. On account of these error estimates our numerical experiments are concentrated on the effects of

- separation of the eigenvalues,
- condition of the eigenvector matrix,
- choice of the startvector.

The condition of the eigenvector matrix is related to the rate of non-normality of A. $C(U) \geq 1$ with equality if and only if A is normal. The following functions can be used to describe the non-normality of A.

- $\mu_1(A) = \|AA^T - A^TA\|$
- $\mu_2(A) = (\|A\|_F^2 - \sum_{j=1}^{n} |\lambda_j|^2)^{1/2}$

In general the number $C(U)$ can not be computed efficiently, but μ_1 and, in case of triangular matrices, μ_2 can and they give indications concerning the condition of the eigenproblem (see [3]).

2.3 Implementational problems

In the Arnoldi Algorithm some choices have to be made by the user. First a startvector must be supplied. When the eigenvalues of $H^{(j)}$ have been calculated it must be recognized whether a Ritzvalue is interesting or not. And it has to be decided whether a Ritzvalue is a good enough approximation of an eigenvalue. These problems will be discussed below.

2.3.1 Startvector

The Krylov spaces depend on two parameters: the matrix A and the startvector v_1. For a given matrix we have all freedom to choose v_1, but it is obvious that some choices are better than others. If v_1 has a large component in the direction of an eigenvector, then one of the Ritzvalues approximates the corresponding eigenvalue rather good. Perhaps we know some characteristics of the eigensystem of A that can lead to a good choice of v_1, but in general we do not have such information, so the best thing we can do is take a random vector, or a vector with all entries equal, or whatever we want. Notice that after a number of iterations we do have some information about the eigensystem of A (apart from the question how reliable this information is). In the algorithms that are discussed hereafter we have to choose a startvector repeatedly and then it becomes possible to make more sensible choices for the startvector.

2.3.2 Wanted eigenvalues

Another problem is on which Ritzvalues we should focus. We do not intend to compute all eigenvalues, so we must have a sort of description of the eigenvalues we are interested in. This can be something of the form: "the ten eigenvalues with largest real part", or "the four eigenvalues of largest absolute value". In general we can say: "the k eigenvalues that maximize the function f". Selecting the "wanted" Ritzvalues is the same as selecting the k Ritzvalues that maximize the function f.

2.3.3 Stopping criterion

For iterative algorithms we need a stopping criterion. We would like to stop if our approximation has some predescribed relative accuracy, but as a consequence of lack of knowledge of the true solution this accuracy can be measured only with an indirect criterion. It is important to know the relationship between this criterion and the real error in the approximation to understand the reliability of the former.
In our case we have a computed Ritzvalue θ_i and we would like to know $\min_k |\lambda_k - \theta_i|$. A criterion is suggested by the following theorem.

Theorem 5 *Let $\mu \in \mathbb{R}$ and $x \in \mathbb{R}^n$ with $\|x\| = 1$, then $\min_i |\lambda_i - \mu| \leq C(U) \cdot \|Ax - \mu x\|$ where $C(U) = \|U\| \cdot \|U^{-1}\|$ is the condition number of U.*

Proof:

$$\begin{aligned} 1 &= \|x\| = \|(A-\mu I)^{-1}(A-\mu I)x\| \\ &\leq \|(A-\mu I)^{-1}\| \cdot \|(A-\mu I)x\| = \|U(\Lambda-\mu I)^{-1}U^{-1}\| \cdot \|(A-\mu I)x\| \\ &\leq \|U\| \cdot \|(\Lambda-\mu I)^{-1}\| \cdot \|U^{-1}\| \cdot \|(A-\mu I)x\| = \frac{1}{\min_i |\lambda_i - \mu|} \cdot C(U) \cdot \|Ax - \mu x\| \end{aligned}$$

■

If x_i is the Ritzvector corresponding to θ_i then during the execution of the algorithm we can calculate $\|Ax_i - \theta_i x_i\|$, which is called the residual norm and is denoted by ρ_i. If this number is smaller than a specified tolerance then the Ritzvalue is accepted as an approximation of an eigenvalue of A. Unfortunately we know nothing about the condition number $C(U)$, so we do not know the reliability of this criterion. It is possible that in case of a large $C(U)$ the Ritzvalue is accepted while it is still a bad approximation. But so far a better criterion is not known, so we will use it in the algorithms.
It is not necessary to compute x_i explicitly: x_i is a Ritzvector, so $x_i = V^{(j)}y_i$ and in (3) we had $AV^{(j)} = V^{(j)}H^{(j)} + r^{(j)}e_j^T$ which leads to:

$$Ax_i - \theta_i x_i = AV^{(j)}y_i - \theta_i V^{(j)}y_i = V^{(j)}H^{(j)}y_i + r^{(j)}e_j^T y_i - V^{(j)}\theta_i y_i = r^{(j)}e_j^T y_i \qquad (8)$$

Hence:

$$\rho_i = \|Ax_i - \theta_i x_i\| = \|r^{(j)}\| \cdot |e_j^T y_i| = H_{j+1,j}^{(j)} \cdot |e_j^T y_i| \qquad (9)$$

So we only have to calculate y_i and to perform one simple calculation to obtain the residual norm ρ_i.

2.4 The Block Arnoldi Algorithm

A variant of the Arnoldi Algorithm is the Block Arnoldi Algorithm. There one considers a "generalized Krylov space":

$$K_j(A, v_1, \ldots, v_k) = \text{span}\{v_1, \ldots, v_k, Av_1, \ldots, Av_k, \ldots, A^{j-1}v_1, \ldots, A^{j-1}v_k\} \tag{10}$$

So we start with k vectors and in each iteration step the dimension of the Krylov space increases by k. The matrix $H^{(j)}$ has a block Hessenberg structure, which means that $H^{(j)}_{i_1,i_2}$ is a $k \times k$ matrix and $H^{(j)}_{i_1,i_2} = 0$ if $i_1 > i_2 + 1$.

Algorithm 6 (Block Arnoldi)

```
1. Choose an orthogonal $n \times k$ matrix $V_1$.
2. do $j = 1, 2, 3, \ldots$ :
       $W = AV_j$
       do $i = 1, \ldots, j$ :
           $H_{i,j} = W^T V_i$
       enddo
       $W = W - \sum_{i=1}^{j} H_{i,j} V_i$
       Calculate a QR decomposition of $W$: $W = QR$.
       $H_{j+1,j} = R$
       $V_{j+1} = Q$
       Calculate the eigenvalues of $H$.
       If the wanted eigenvalues have converged then stop.
   enddo
```

In the algorithm we take for $H^{(j)}_{j+1,j}$ an upper triangular matrix, so $H^{(j)}$ has k subdiagonals below the main diagonal.

2.5 Iterative Arnoldi Algorithm

A disadvantage of the Arnoldi Algorithm is the storage requirements for the basis of the computed Krylov space. If A is symmetric, $H^{(j)}$ is also symmetric and thus a tridiagonal matrix. Then the algorithm reduces to the Lanczos algorithm and only three basis vectors have to be stored at each moment. In the general case the storage requirements grow in time. Since the number of iterations cannot be predicted, a priori the space requirements are unknown. Generally the user has a limited memory capacity and does not want the algorithm to use more space.
Let us assume that m is the maximum dimension of the Krylov space one allows. Consider the situation in which we are looking for one eigenvalue λ_i. After m iterations of the Arnoldi algorithm we decide if we accept the wanted Ritzvalue θ. If we do not, we repeat this algorithm with a "better" startvector. This means we need a new startvector v_1 such that $\frac{\|(I-P_i)v_1\|}{\|P_i v_1\|}$ is smaller than it was before (see theorem 3). Therefore it seems reasonable

to choose the Ritzvector $V^{(j)}y$ corresponding to θ as the new startvector, hoping it is a better approximation to u_i than the previous choice.

Algorithm 7 (Iterative Arnoldi)

```
1. Choose tolerance tol.
   Choose $v_1$ of norm one.
2. do $j = 1, \ldots, m$ :
       $w = Av_j$
       do $i = 1, \ldots, j$ :
           $H_{i,j} = (w, v_i)$
       enddo
       $w = w - \sum_{i=1}^{j} H_{i,j} v_i$
       $H_{j+1,j} = \|w\|$
       $v_{j+1} = w / H_{j+1,j}$
   enddo
3. Calculate the wanted Ritzvalue $\theta_i$
   and corresponding Ritzvector $y_i$ of $H$.
4. If $\rho_i < tol$ then stop
   else take $v_1 = V y_i$ and go to 2.
```

There is a complication. When the wanted Ritzvalue is not real, the corresponding eigenvector also has an imaginary part. Restarting using this vector means the introduction of complex arithmetic. We avoid this by using Block Arnoldi.
Suppose we have:

$$Au = \lambda u, \lambda = \lambda_r + i\lambda_i, u = u_r + iu_i \tag{11}$$

then:

$$A(u_r, u_i) = (\lambda_r u_r - \lambda_i u_i, \lambda_i u_r + \lambda_r u_i) = (u_r, u_i) \begin{pmatrix} \lambda_r & \lambda_i \\ -\lambda_i & \lambda_r \end{pmatrix} \tag{12}$$

So A restricted to the space spanned by u_r and u_i has the eigenvalues λ and $\bar{\lambda}$. This suggests the following strategy: instead of restarting with a non-real Ritzvector $x_r + ix_i$ we use Block Arnoldi with blocksize two and $V_1 = (x_r, x_i)$. The number of iterations becomes $\lfloor \frac{m}{2} \rfloor$. If the wanted Ritzvector is real we return to blocksize one.

3 Iterative Arnoldi Algorithm with deflation

Algorithm 7 computes only one eigenvalue, so if we want more eigenvalues the method has to be changed. One possibility is constructing the new startvector by taking a linear combination of wanted Ritzvectors instead of just one Ritzvector, but it is difficult to take this combination in a systematic way and achieve convergence for all wanted Ritzvalues. Another possibility is performing the Iterative Arnoldi Algorithm until the first wanted eigenvalue has been accepted, and then applying the technique of deflation to obtain the second one, and so on. Saad [18] has worked this out. When $k-1$ Ritzvalues have been accepted, the corresponding Ritzvectors span a nearly invariant space. We can orthonormalize these vectors and thus obtain $v_1, \ldots, v_{k-1}$ which can be regarded as approximations of Schurvectors. These vectors are put in the first $k-1$ columns of $V^{(m)}$ and are not changed during the rest of the algorithm. This subspace is completed to a subspace of dimension m by generating a Krylov-space of dimension $m-k+1$ in its orthogonal complement. So, we choose a startvector v_k perpendicular to $v_1, \ldots, v_{k-1}$ and we consider the "modified Krylov space":

$$\mathrm{span}\{v_1, \ldots, v_{k-1}, v_k, Av_k, \ldots, A^{m-k}v_k\} \tag{13}$$

Orthonormalizing the above vectors give the columns of the matrix $V^{(m)}$. If the first $k-1$ vectors span an invariant subspace then for $j<k$:

$$Av_j \in \mathrm{span}\{v_1, \ldots, v_j\} \tag{14}$$

which implies that $H^{(m)}_{j+1,j} = 0$ for $j<k$. Since the subspace we get is not exactly invariant we put zeros in $H^{(m)}_{k+1,k}$ explicitly if the k-th eigenvalue has been accepted. So if, for example, two eigenvalues have been accepted and $m=6$, $H^{(m)}$ has the following structure:

$$H^{(m)} = \left(\begin{array}{cc|cccc} * & * & * & * & * & * \\ & * & * & * & * & * \\ \hline & & * & * & * & * \\ & & * & * & * & * \\ & & & * & * & * \\ & & & & * & * \end{array}\right) \tag{15}$$

We get the following algorithm.

Algorithm 8 (Saad)

```
1. Choose number of wanted eigenvalues nev and tolerance tol.
   Choose $v_1$ of norm one.
   $k = 1$
2. do $j = k, \ldots, m$ :
       $w = Av_j$
       do $i = 1, \ldots, j$ :
           $H_{i,j} = (w, v_i)$
       enddo
       $w = w - \sum_{i=1}^{j} H_{i,j} v_i$
       $H_{j+1,j} = \|w\|$
       $v_{j+1} = w / H_{j+1,j}$
   enddo
3. Calculate eigenvalues of $H$: $\theta_1, \ldots, \theta_m$.
```

```
4.  Calculate eigenvector $y_k$ corresponding to $\theta_k$ : $Hy_k = \theta_k y_k$.
    $\tilde{v}_k = Vy_k$
    Orthonormalize $\tilde{v}_k$ w.r.t. $v_1, \ldots, v_{k-1}$ and put the result in $v_k$.
5.  Calculate the residual $\rho_k = H_{m+1,m} \cdot |e_m^T y_k|$.
    if $\rho_k <$ tol then
        Accept $\theta_k$ as eigenvalue of $A$.
        if $k =$ nev then
            stop
        else
            do $i = 1, \ldots, k$ :
                $H_{i,k} = (Av_k, v_i)$
            enddo
            $H_{k+1,k} = 0$
            $k = k + 1$
            Calculate eigenvector $y_k$ corresponding to $\theta_k$ : $Hy_k = \theta_k y_k$.
            $\tilde{v}_k = Vy_k$
            Orthonormalize $\tilde{v}_k$ w.r.t. $v_1, \ldots, v_{k-1}$ and put the result in $v_k$.
        endif
    endif
    go to 2.
```

This algorithm can be adapted to the case of complex eigenvalues in the same way as described in section 2.5.

4 Sorenson's Algorithm

Suppose we have performed m Arnoldi steps resulting in the Arnoldi factorization:

$$AV^{(m)} = V^{(m)}H^{(m)} + r^{(m)}e_m^T \tag{16}$$

and we want to restart. Sorenson [21] describes an algorithm for generating a new startvector in an implicit way instead of explicitly. The next startvector is a linear combination of wanted Ritzvectors, constructed by filtering out the unwanted Ritzvectors.
Assume we have m Ritzvalues, k of them are "wanted" and $p = m - k$ are "unwanted". The latter are denoted $\theta_{k+1}, \ldots, \theta_{k+p}$. Then define the filterpolynomial:

$$\Psi(t) = \prod_{j=1}^{p}(t - \theta_{k+j}) \tag{17}$$

and take for the new startvector:

$$v_1' = \Psi(A)v_1/\|\Psi(A)v_1\| \tag{18}$$

This has the desired effect as is shown by the following theorem.

Theorem 9 *If $H^{(m)}y_j = \theta_j y_j$ and $x_j = V^{(m)}y_j$ for $j = 1, \dots, m$ and v_1' defined as in (18) then $v_1' = \sum_{j=1}^{k} \xi_j' x_j$ for certain $\xi_1', \dots, \xi_k'$.*

Proof: Let $v_1 = \sum_{j=1}^{m} \xi_j x_j$ and let Π denote the orthogonal projection on $K_m(A, v_1)$. Now $\Psi(A)v_1 \in K_m(A, v_1)$, so $\Psi(A)v_1 = \Psi(\Pi A)v_1$.

$$\begin{aligned} \|\Psi(A)v_1\| v_1' &= \Psi(\Pi A)\sum_{j=1}^{m} \xi_j x_j = \sum_{j=1}^{m} \xi_j \Psi(\Pi A) V^{(m)} y_j \\ &= \sum_{j=1}^{m} \xi_j V^{(m)} \Psi(H^{(m)}) y_j = \sum_{j=1}^{m} \xi_j V^{(m)} \Psi(\theta_j) y_j = \sum_{j=1}^{k} \xi_j \Psi(\theta_j) x_j \end{aligned}$$

■

Now observe that:

$$\Psi(A)v_1, \dots, A^{k-1}\Psi(A)v_1 \in K_m(A, v_1) \tag{19}$$

So:

$$K_k(A, v_1') \subset K_m(A, v_1) \tag{20}$$

Let V' be the Arnoldi basis of $K_k(A, v_1')$. We construct V' from the existing basis by multiplication with a convenient matrix:

$$V' = V^{(m)} Q' \tag{21}$$

Q' is an orthogonal $m \times k$ matrix. If Q' can be efficiently constructed, it is no longer necessary to restart explicitly with v_1', but we can immediatly update the Arnoldi factorization:

$$AV' = V'H' + r'e_k^T \tag{22}$$

where V' is $n \times k$ and $V'e_1 = v_1'$. Sorenson constructs Q' by using the QR-factorization of $(H^{(m)} - \mu I)$ ($\mu \in \mathbb{R}$). So he obtained from the Arnoldi factorization:

$$\begin{aligned} (A - \mu I)V^{(m)} &= V^{(m)}(H^{(m)} - \mu I) + r^{(m)} e_m^T \\ &= V^{(m)} QR + r^{(m)} e_m^T \\ (A - \mu I)(V^{(m)}Q) &= (V^{(m)}Q)(RQ) + r^{(m)} e_m^T Q \\ A(V^{(m)}Q) &= (V^{(m)}Q)(RQ + \mu I) + r^{(m)} e_m^T Q \\ &= (V^{(m)}Q)(Q^T H^{(m)} Q) + r^{(m)} e_m^T Q \end{aligned} \tag{23}$$

Note that:

- Since Q is a Hessenberg matrix, $Q^T H^{(m)} Q = (RQ + \mu I)$ is also one.
- $(V^{(m)}Q)e_1 = V^{(m)} Q R e_1 / R_{1,1} = (A - \mu I)v_1 / R_{1,1}$

This last property is very attractive when we think of Ψ. We repeat the procedure described above for $\mu = \theta_{k+1}, \ldots, \theta_m$, setting $H_0 = H^{(m)}$ and producing in step j the matrices Q_j, R_j, H_j:

$$\begin{aligned} Q_j R_j &= (H_{j-1} - \theta_{k+j} I) \\ H_j &= Q_j^T H_{j-1} Q_j \end{aligned}$$

Take $Q = Q_1 Q_2 \cdots Q_p$, then we have:

- In each step j Q_j is a Hessenbergmatrix, so H_j is also a Hessenbergmatrix and Q has p subdiagonals below the main diagonal.
- $A(V^{(m)}Q) = (V^{(m)}Q)(Q^T H^{(m)} Q) + r^{(m)} e_m^T Q$
- $(V^{(m)}Q)e_1 = \Psi(A)v_1 / \|\Psi(A)v_1\|$

Decompose:

$$V^{(m)}Q = (V', \tilde{V}) \tag{24}$$

$$Q^T H^{(m)} Q = \begin{pmatrix} H' & M \\ \beta e_1 e_k^T & \tilde{H} \end{pmatrix} \tag{25}$$

with V' and H' having k columns, then:

$$\begin{aligned} AV' &= V'H' + \tilde{V}\beta e_1 e_k^T + r^{(m)} e_m^T Q \begin{pmatrix} I_k \\ 0 \end{pmatrix} \\ &= V'H' + \beta(\tilde{V} e_1) e_k^T + r^{(m)} e_k^T Q_{m,k} \\ &= V'H' + (\beta \tilde{V} e_1 + Q_{m,k} r^{(m)}) e_k^T \end{aligned} \tag{26}$$

And this is the intended Arnoldi factorization mentioned in (22), for V' is orthogonal, H' is Hessenberg, $v_1' = (V^{(m)}Q)e_1 = \Psi(A)v_1/\|\Psi(A)v_1\|$ and $AV' - V'H' \perp K_k(A, v_1')$.
These considerations lead to the following algorithm.

Algorithm 10 (Sorenson)

```
1. Choose $v_1$ with norm 1,
   set $H_{1,1} = (v_1, Av_1)$
   $r = Av_1 - v_1 H$
2. Perform $k-1$ additional Arnoldi steps.
3. do $j = 1, 2, \ldots$ :
       if $\|r\| < tol$ then stop
       Perform $p$ additional Arnoldi steps.
       Calculate $p$ shifts $\mu_1, \ldots, \mu_p$.
       $Q = I_{k+p}$
       do $i = 1, \ldots, p$
```

```
            Calculate $Q_iR = (H - \mu_i I)$
            $H = Q_i^T H Q_i$
            $Q = QQ_i$
        enddo
        $v = (VQ)e_{k+1}$
        $r = H_{k+1,k}v + Q_{k+p,k}r$
        $V = (VQ)(I_k, 0)^T$
        $H = (I_k, 0)H(I_k, 0)^T$
    enddo
```

5 Numerical experiments

This chapter gives some testresults of the Fortran code for the Iterative Arnoldi Algorithm with Deflation. There are two approaches. In section 5.1 the qualitative behaviour is displayed and compared to the behaviour of the Arnoldi Algorithm. The used matrices are not very large and the programs have been run on a SUN workstation.
In section 5.2 the Deflation Algorithm is compared to an implementation of Sorenson's Algorithm on some large sparse matrices. The CRAY-YMP of SARA, Amsterdam, was used for these experiments.
The implementation of the Arnoldi Algorithm is denoted by ARN, that of Saad and Sorenson by DEFL(m) and SOREN(k, p) respectively. m is the maximum dimension of the Krylovspace, k is the number of wanted eigenvalues and p is the number of shifts.

5.1 The behaviour of the algorithms of Saad and Arnoldi

In section 2.2 it was shown that the convergence of the Arnoldi Method depends on several characteristics as:

- separation of the eigenvalues,
- condition of the eigenvector matrix,
- choice of the startvector.

We get an idea of their influence from numerical experiments with sets of matrices in which only one of them varies. This is done in the sections 5.1.2, 5.1.3 and 5.1.4. The dimension of the matrices was 100 and we were looking for the eigenvalue with largest real part or, in case this eigenvalue was not real, the two eigenvalues with largest real part.

5.1.1 Notation

The matrices used in this section are defined below. n always denotes the dimension of the matrix.

CIRC(n,r) A block-diagonal matrix with spectrum $\{e^{i\pi(\frac{2j-1}{n})}|j = 2,\ldots,n-1\} \cup \{re^{\pm\frac{i\pi}{n}}\}$.

COM(n,ε) A matrix with spectrum $\{1 \pm \varepsilon i\}, \cup\{\frac{k}{n}|k = 1,\ldots,n-2\}$.

DIA(n) $= \text{diag}(1, \frac{n-1}{n},\ldots,\frac{1}{n})$.

SEP(n,ε) A diagonal matrix with $n-1$ eigenvalues uniformly spread over the interval $[\frac{1}{10}, 1-\varepsilon]$ and an eigenvalue 1.

TRI(n,α) A tridiagonal matrix with 2 on the main diagonal, α on the upper co-diagonal and α^{-1} on the lower co-diagonal.

U(n,nnz) A strict upper triangle matrix with $(nnz-n)$ elements not equal to zero. The positions of these elements are chosen random and their values are taken from a uniform $[0,1]$ distribution.

UPP(n,nnz,ε) $= \text{diag}(1, \frac{n-1}{n},\ldots,\frac{1}{n}) + \varepsilon \cdot \text{U}(n,nnz)$.

5.1.2 Separation of the eigenvalues

It is known that a bad separation between the wanted and the unwanted eigenvalues makes it difficult to find all the wanted eigenvalues (see theorem 4). In SEP(n,ε) all eigenvalues are real and the gap between the largest and the next eigenvalue is ε. CIRC(n,r) has its eigenvalues uniformly spread over the unit circle in the complex plane except for the two eigenvalues with largest real part which are multiplied by r. If r grows, the separation of these gets better. COM(n,ε) has only two non real eigenvalues with have a distance 2ε to eachother. The effects of the gapsize on the convergence using these matrices are shown in the figures 1 to 6. Especially the figures 3 and 4, for $p < -0.75$, are in conformity with the largeness of $\varepsilon_i^{(j)}$ in case of uniform distribution of the spectrum on the unit circle (see [2]).

5.1.3 Condition of the eigenvector matrix

We used matrices with constant spectrum and a variable condition of the eigenvector matrix. We measure this by the rate of non-normality μ_1 (see section 2.2). The first set of matrices contains the tridiagonal matrices TRI(n,α). This matrix is similar to TRI$(n$,1), which is symmetric with spectrum:

$$\{4\sin^2(\frac{\pi k}{2(n+1)})|k = 1,\ldots,n\}$$

We have

$$\mu_1(\text{TRI}(n,\alpha)) = |\alpha^2 - \alpha^{-2}|$$

But the condition of the eigenvector matrix is exponential in α.
The second set of matrices contains the upper triangle matrices UPP(n,nnz,ε) with fixed diagonal. The largest entry in its strict upper triangle part and the rate of non-normality

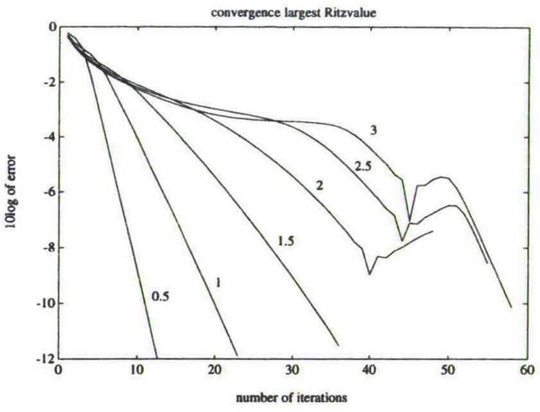

Figure 1: *The* $^{10}\log$ *of the absolute error in the largest Ritzvalue plotted against the number of iterations using* ARN *on the matrix* $\mathrm{SEP}(100, 10^{-p})$ *for the mentioned values of* p.

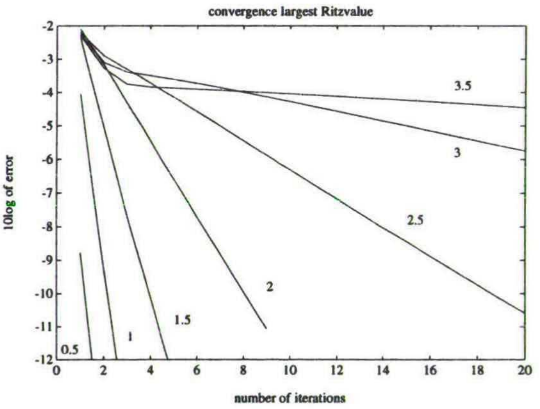

Figure 2: *The* $^{10}\log$ *of the absolute error in the largest Ritzvalue plotted against the number of iterations using* DEFL(10) *on the matrix* $\mathrm{SEP}(100, 10^{-p})$ *for the mentioned values of* p.

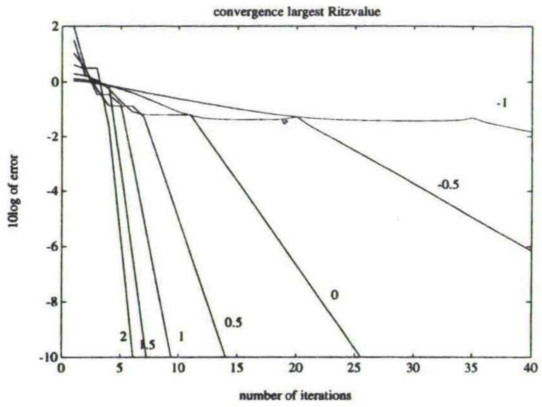

Figure 3: *The* $^{10}\log$ *of the absolute error in the Ritzvalue with largest real part plotted against the number of iterations using* ARN *on the matrix* CIRC$(100, 1 + 10^p)$ *for the mentioned values of* p.

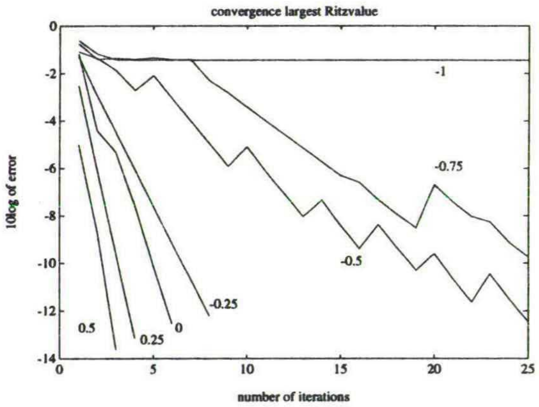

Figure 4: *The* $^{10}\log$ *of the absolute error in the Ritzvalue with largest real part plotted against the number of iterations using* DEFL(10) *on the matrix* CIRC$(100, 1 + 10^p)$ *for the mentioned values of* p.

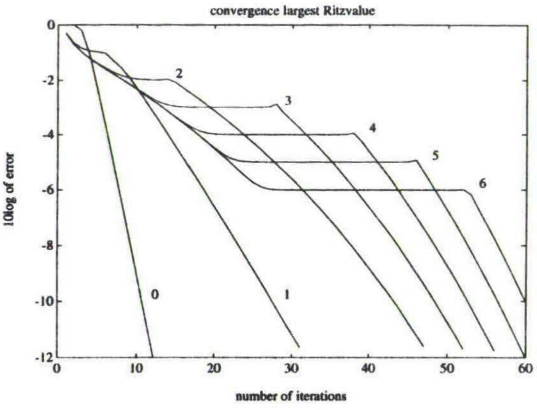

Figure 5: *The* $^{10}\log$ *of the absolute error in the Ritzvalue with largest real part plotted against the number of iterations using* ARN *on the matrix* COM$(100, 10^{-p})$ *for the mentioned values of* p.

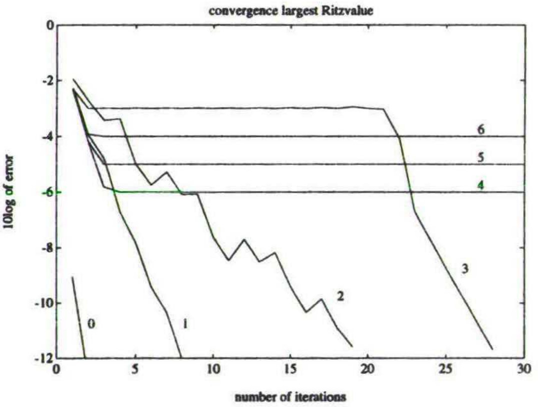

Figure 6: *The* $^{10}\log$ *of the absolute error in the Ritzvalue with largest real part plotted against the number of iterations using* DEFL(10) *on the matrix* COM$(100, 10^{-p})$ *for the mentioned values of* p.

are of order ε. Figures 7 to 10 contain the results using ARN and DEFL. They reflect the fact that the speed of convergence decreases with the increasing non-normality.

5.1.4 Choice of the startvector

Until now the startvector $(1, 1, \dots, 1)/\sqrt{n}$ was used. To study the influence of the startvector we define the vector in $\mathbb{R}^n$ of length one:

$$\mathrm{STVEC}(n,\varepsilon) = \frac{1}{\sqrt{1+\varepsilon^2}} \left[\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \frac{\varepsilon}{\sqrt{n-1}} \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \right]$$

So ε is the tangent of the angle between the first unitvector e_1 and $\mathrm{STVEC}(n,\varepsilon)$. When we use the matrices DIA and UPP the eigenvector corresponding to the largest eigenvalue is e_1. So $\mathrm{STVEC}(n,\varepsilon)$ can be varied from an eigenvector to a vector with almost no component in the direction of the wanted eigenvector. The results of various choices of ε can be found in the figures 11 to 14. It seems that for small ε the logarithm of the error depends linearly on the logarithm of the angle.

5.2 Comparing the algorithms of Saad and Sorenson

The Deflation Algorithm was tested on some large, sparse matrices and compared to the performance of Sorenson's Algorithm on the same matrices. The first set of matrices arises from a partial differential equation [14]. Consider the region $\Omega = (0,1) \times (0,1) \subset \mathbb{R}^2$ and the convection-diffusion equation:

$$-\Delta u + \rho u_x = u \qquad \text{in } \Omega, \qquad u|_{\delta\Omega} = 0 \tag{27}$$

Using a five-point discretization on a rectangular $l \times l$ grid gives rise to the matrix of dimension $n = l^2$:

$$\mathrm{DIF}(l,\rho) = \begin{pmatrix} \mathrm{M}(l,\rho) & -I & & \\ -I & \mathrm{M}(l,\rho) & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & \mathrm{M}(l,\rho) \end{pmatrix}$$

where the $l \times l$ matrix $\mathrm{M}(l,\rho)$ is given by:

$$\mathrm{M}(l,\rho) = \begin{pmatrix} 4 & \alpha & & \\ \beta & 4 & \ddots & \\ & \ddots & \ddots & \alpha \\ & & \beta & 4 \end{pmatrix} \qquad \text{and} \qquad \begin{cases} \alpha = -1 + \frac{\rho}{2(l+1)} \\ \beta = -1 - \frac{\rho}{2(l+1)} \end{cases}$$

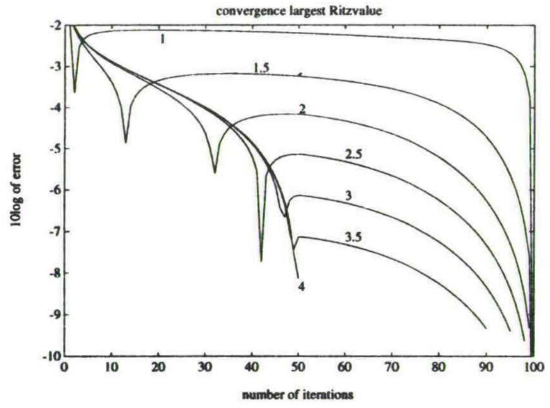

Figure 7: *The* $^{10}\log$ *of the absolute error in the largest Ritzvalue plotted against the number of iterations using* ARN *on the matrix* TRI$(100, 1 + 10^{-p})$ *for the mentioned values of* p.

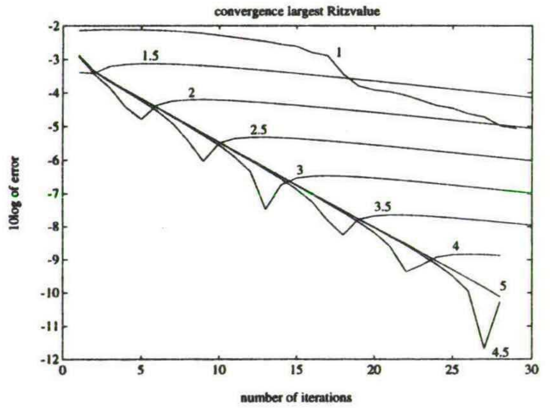

Figure 8: *The* $^{10}\log$ *of the absolute error in the largest Ritzvalue plotted against the number of iterations using* DEFL(10) *on the matrix* TRI$(100, 1 + 10^{-p})$ *for the mentioned values of* p.

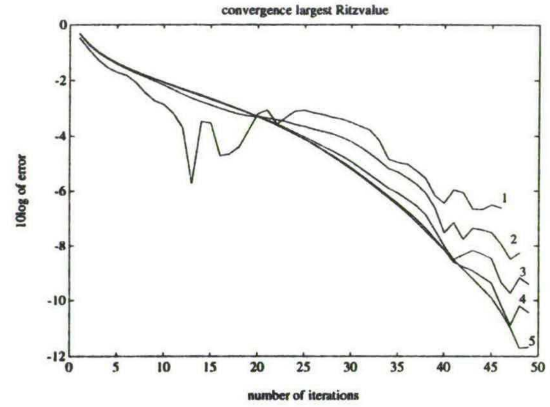

Figure 9: *The* ^{10}log *of the absolute error in the largest Ritzvalue plotted against the number of iterations using* ARN *on the matrix* UPP$(100, 400, 10^{-p})$ *for the mentioned values of* p.

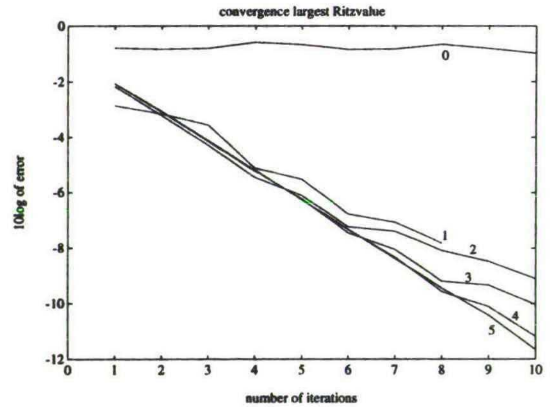

Figure 10: *The* ^{10}log *of the absolute error in the largest Ritzvalue plotted against the number of iterations using* DEFL(10) *on the matrix* UPP$(100, 400, 10^{-p})$ *for the mentioned values of* p.

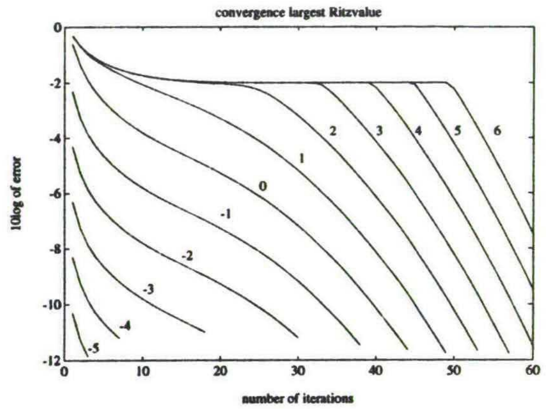

Figure 11: *The* ^{10}log *of the absolute error in the largest Ritzvalue plotted against the number of iterations using* ARN *on the matrix* DIA(100) *and startvector* STVEC(100, 10^p) *for the mentioned values of* p.

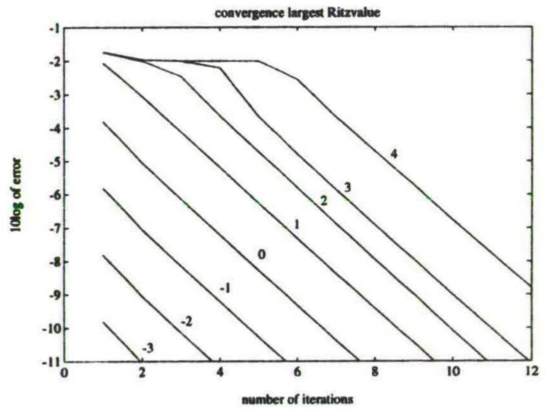

Figure 12: *The* ^{10}log *of the absolute error in the largest Ritzvalue plotted against the number of iterations using* DEFL(10) *on the matrix* DIA(100) *and startvector* STVEC(100, 10^p) *for the mentioned values of* p.

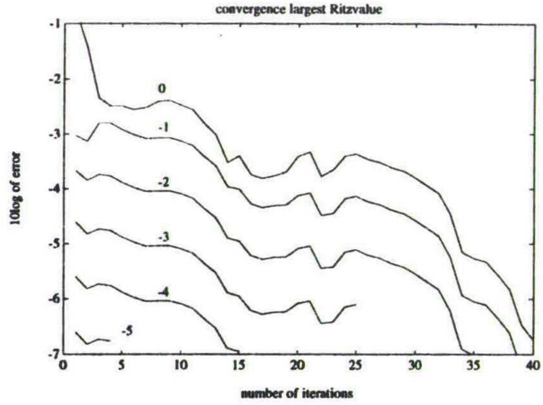

Figure 13: *The* ^{10}log *of the absolute error in the largest Ritzvalue plotted against the number of iterations using* ARN *on the matrix* UPP(100, 400, 0.1) *and startvector* STVEC(100, 10^p) *for the mentioned values of p.*

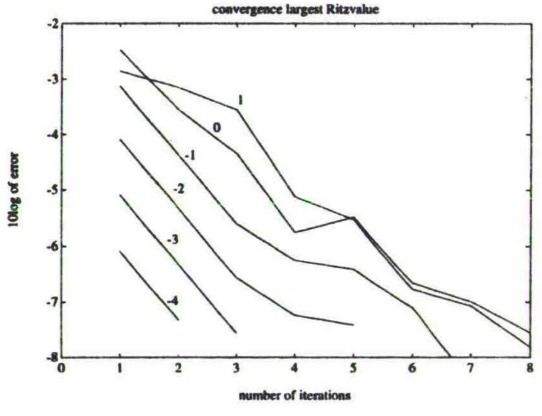

Figure 14: *The* ^{10}log *of the absolute error in the largest Ritzvalue plotted against the number of iterations using* DEFL(10) *on the matrix* UPP(100, 400, 0.1) *and startvector* STVEC(100, 10^p) *for the mentioned values of p.*

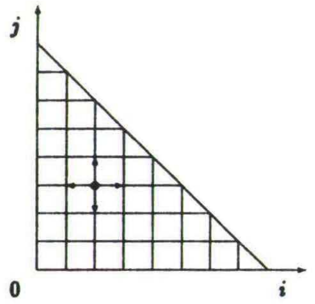

Figure 15: *The grid of the Markov chain problem.*

We have

$$\mu_1(\mathrm{DIF}(l,\rho)) = \frac{2|\rho|}{l+1}$$

The second set of matrices represents the transition matrices of Markov chains [14]. These chains describe random walks on the grid:

$$\{(i,j) \in \mathbb{N}^2 | 0 \le i,j \le l, i+j \le l\}$$

See also figure 15. In a point (i,j) a transition can occur to the points $(i+1,j)$, $(i,j+1)$, $(i-1,j)$, $(i,j-1)$, the first two of them with probability $\frac{n-i-j}{2n}$ and the last two with probability $\frac{i+j}{2n}$. This last probability is doubled if $i = 0$ or $j = 0$. We denote the corresponding transition matrix of dimension $n = \frac{1}{2}(l+1)(l+2)$ by MARK(l).
The tables 1 to 4 contain results of running DEFL and SOREN on the matrices DIF and MARK. "mult" is the number of calls of the matrix-vector multiplication routine and the time is the total running time of the algorithms in seconds. In table 1 DEFL and SOREN are compared using DIF(l,ρ) for several values of ρ. In table 2 DEFL and SOREN are compared using MARK(l) for several values of l. In table 3 DEFL(m) is compared for $m = 10, 20, 30, 40$ using DIF(l,ρ) for several values of ρ. In table 4 DEFL(m) is compared for $m = 10, 20, 30, 40$ using MARK(l) for several values of l.

6 Conclusions

In section 2.2 we considered three characteristics in the error estimates for the computed eigenvalue approximations. The meaning of their role was checked experimentally and is described in chapter 5. Our experiments affirm in a qualitive sense the dependence of the convergence on these characteristics.

		DEFL(10)		DEFL(20)		SOREN(1,9)		SOREN(1,19)	
l	ρ	mult	time	mult	time	mult	time	mult	time
55	0.	431	3.760	281	2.908	838	5.397	457	3.814
55	0.0001	431	3.810	281	2.965	838	5.449	457	3.905
55	0.001	511	4.302	341	3.362	838	5.451	457	3.886
55	0.01	621	4.979	381	3.635	838	5.450	457	3.911
55	0.1	741	5.717	441	4.022	838	5.451	457	3.914
55	1.	851	6.394	481	4.273	838	5.466	457	3.881
55	10.	701	5.473	421	3.887	532	3.874	418	3.677

Table 1: *The performance of* DEFL *and* SOREN *on* DIF(l, ρ). *The eigenvalue with largest real part is wanted and the tolerance is* 10^{-9}.

	DEFL(10)		DEFL(20)		SOREN(1,9)		SOREN(1,19)	
l	mult	time	mult	time	mult	time	mult	time
45	111	0.586	41	0.452	1037	2.540	342	1.466
50	11	0.474	21	0.503	3194	7.779	855	3.192
55	4341	13.216	41	0.667	6374	17.251	457	2.283
60	21	0.716	21	0.719	4032	12.729	550	2.919
65	91	1.114	41	0.932	3338	12.023	586	3.359
70	31	1.013	41	1.070	9285	35.842	1023	5.809
75	21	1.106	21	1.122	35261	149.000	626	4.323

Table 2: *The performance of* DEFL *and* SOREN *on* MARK(l). *The eigenvalue with largest real part is wanted and the tolerance is* 10^{-9}.

		DEFL(10)		DEFL(20)		DEFL(30)		DEFL(40)	
l	ρ	mult	time	mult	time	mult	time	mult	time
100	0.	791	19.692	461	13.333	361	11.537	361	11.817
100	0.0001	6201	128.730	3021	66.155	1951	45.913	1201	30.649
100	0.001	6601	136.785	3481	75.765	2251	52.400	601	17.262
100	0.01	7761	160.219	3361	73.516	2341	54.166	1481	36.887
100	0.1	7641	157.924	3641	79.247	871	22.602	761	20.833
100	1.	7741	159.952	3601	78.568	931	23.968	721	19.998
100	10.	2051	45.298	901	22.569	691	18.886	1161	29.895
100	100.	2721	60.480	2821	62.792	4111	93.982	5201	121.636

Table 3: *The performance of* DEFL *on* DIF(l, ρ). *The eigenvalue with largest real part is wanted and the tolerance is* 10^{-9}.

	DEFL(10)		DEFL(20)		DEFL(30)		DEFL(40)	
l	mult	time	mult	time	mult	time	mult	time
100	11	1.886	21	1.970	31	2.075	41	2.192
105	131	3.279	681	9.005	691	9.434	121	3.320
110	51	2.721	41	2.602	61	2.858	41	2.646
115	41	2.841	41	2.842	31	2.733	41	2.889
120	31	2.968	41	3.090	31	2.973	41	3.134
125	11	2.914	21	3.060	31	3.225	41	3.401
130	81	4.216	101	4.561	61	3.975	81	4.361

Table 4: *The performance of* DEFL *on* MARK(l). *The eigenvalue with largest real part is wanted and the tolerance is* 10^{-9}.

Compared to the symmetric eigenvalue problem the nonsymmetric case introduces two additional sources of delay in the convergence.
Firstly the appearance of nonreal eigenvalues. In the case of the matrix CIRC with almost uniform distribution of its eigenvalues on the unit circle we see slow convergence despite the seperation being $\frac{2\pi}{100}$ and the normality of that matrix. In the case of a symmetric matrix with a uniform distribution on the real axis and comparable seperation the Arnoldi (Lanczos) Method gives a much better performance.
Secondly the occurrence of nonnormality of the matrix. In case of the testmatrix TRI, the numerical results demonstrate the large delay in convergence due to their departure from normality. This already happens with values of α very close to one. Recall that the condition number of the eigenvector matrix is exponential in α and thus is very sensitive to changes in this parameter.
Saad's Deflation Algorithm was compared with that of Sorenson in section 5.2. The number of experiments is too small to get a definitive answer on the question which of these algorithms is superior. In most of our experiments the Deflation Algorithm needs a little less time than Sorenson's Method, but further experiments are necessary. Moreover we lack the theoretical insight in the essential properties of these algorithms.

A Sparse matrix storage

An implementation of one of the algorithms mentioned in this paper requires an efficient storage of the matrix A. Let nnz be the number of nonzero entries of A. There are several storage possibilities, but we have chosen the compressed sparse row format. This means that the nonzeros of A are stored rowwise in a vector a of length nnz. Usually the diagonal element of a row is the first number stored of that row. The vector ja of the same length contains the corresponding indices to the columns of A. The indices to the heads of the rows are stored in the first n positions of the vector ia of length $n+1$ and we define $ia(n+1) = nnz+1$. So $a(ia(i))$ to $a(ia(i+1)-1)$ contain the nonzeros of the ith row of A.
If for example the matrix is:

$$A = \begin{pmatrix} 11 & 12 & 0 & 14 & 0 \\ 21 & 22 & 23 & 0 & 0 \\ 0 & 32 & 33 & 0 & 0 \\ 0 & 0 & 0 & 44 & 0 \\ 51 & 52 & 0 & 0 & 55 \end{pmatrix}$$

then its compressed sparse row format is:

$$\begin{aligned} a &= (11, 12, 14, 22, 21, 23, 33, 32, 44, 55, 51, 52) \\ ia &= (1, 4, 7, 9, 10, 13) \\ ja &= (1, 2, 4, 2, 1, 3, 3, 2, 4, 5, 1, 2) \end{aligned}$$

B Implementation of the matrix-vector multiplication

One of the most important implementational problems concerns the calculation of $y = Ax$ for a given vector x. Several Fortran routines have been written and tested on the CRAY-YMP. Four versions are discussed here and their performance times are given. They are denoted by AMUX and a index number. The CRAY vectorizes loops if possible and in the fragments below it is indicated which loops are vectorized (V) and which ones remain scalar (S).

AMUX1

```
S----------------------<        do 100 i = 1,n
S                                   t = 0.0
S V--------------------<            do 99 k=ia(i), ia(i+1)-1
S V                                     t = t + a(k)*x(ja(k))
S V--------------------> 99         continue
S                                   y(i) = t
S----------------------> 100    continue
```

A is stored in compressed sparse row format, so the first thought was to have a double loop in which the outer one indicates the row i and the inner one walks along that row from $ia(i)$ to $ia(i+1)-1$. The code above shows that the inner loop has been vectorized. But the inner loop is very small (the number of nonzeros on the row) so the speed-up of the vectorization is rather small.

AMUX2

```
                                nnz = ia(n+1)-1
                                t   = a(1)*x(ja(1))
                                i   = 2
S----------------------<        do 100 k = 2,nnz
S                                   if (k.eq.ia(i)) then
S                                       y(i-1) = t
S                                       t      = 0.0
S                                       i      = i+1
S                                   endif
S                                   t = t + a(k)*x(ja(k))
S----------------------> 100    continue
                                y(n) = t
```

The second version has only one loop, with k running over the array ia. At each passage to a new row, detected by an if statement, there is an interruption for the necessary adjustments. AMUX2 shows that this if-statement was an obstacle for the vectorization of the loop.

AMUX3

```
                                     nnz = ia(n+1)-1
V---------------------<              do 10 k=1,nnz
V                                        temp(k) = a(k)*x(ja(k))
V---------------------> 10           continue
S---------------------<              do 100 i = 1,n
S                                        t = 0.0
S V-------------------<                  do 99 k=ia(i), ia(i+1)-1
S V                                          t = t + temp(k)
S V-------------------> 99               continue
S                                        y(i) = t
S---------------------> 100          continue
```

We saw that the vectorization of AMUX1 was not very efficient. In AMUX3 the idea was to relief the inner loop by doing the multiplications in advance in a large vectorizable loop. In the other loops only additions occur. Therefore an extra array *temp* is introduced which contains the products of $a(k)$ and $x(ja(k))$.

AMUX4

```
                                     nnz = ia(n+1)-1
V---------------------<              do 10 k=1,nnz
V                                        temp(k) = a(k)*x(ja(k))
V---------------------> 10           continue
                                     t   = temp(1)
                                     i   = 2
S---------------------<              do 100 k = 2,nnz
S                                        if (k.eq.ia(i)) then
S                                            y(i-1) = t
S                                            t      = 0.0
S                                            i      = i+1
S                                        endif
S                                        t = t + temp(k)
S---------------------> 100          continue
                                     y(n) = t
```

The same idea is applied to AMUX2 and gives the version AMUX4.

TIMES. To compare the performing times of these four routines they have been called a hundred times each with $A = \text{UPP}(1000, 4000, 0.1)$. The flowtrace facility of the CRAY delivered the results in the table below.

```
                    Flowtrace Statistics Report
          Showing Routines Sorted by CPU Time (Descending)
                  (CPU Times are Shown in Seconds)
```

```
Routine Name       Tot Time  # Calls Avg Time
----------------- -------- -------- --------
AMUX1              1.85E-01      100 1.85E-03  ****
AMUX3              1.78E-01      100 1.78E-03  ***
AMUX2              1.62E-01      100 1.62E-03  ***
AMUX4              9.77E-02      100 9.77E-04  **
============================================
```

The performance of AMUX4 is the best, and since similar results were obtained using other matrices, this became the final version of the routine.

Acknowledgements

The results were achieved under supervision of prof.dr. M.H.C. Paardekooper, Tilburg University. I also like to thank prof.dr. H.A. van der Vorst, University of Utrecht, for supplying me with a Fortran program of Sorenson's algorithm.

References

[1] W.E. ARNOLDI, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quart. Appl. Math., 9:17–29, 1951.

[2] F. CHATELIN, *Valeurs propres de matrices*, Masson, Paris, 1988.

[3] L. ELSNER AND M.H.C. PAARDEKOOPER, *On measures of nonnormality of matrices*, Lin. Alg. Appl., 92:107–124, 1987.

[4] G.H. GOLUB AND C.F. VAN LOAN, *Matrix computations*, The Johns Hopkins University Press, Baltimore, 1989.

[5] D. HO, *Tchebychev iteration and its optimal ellipse for nonsymmetric matrices*, Report, 1987.

[6] D. HO, *Tchebychev acceleration technique for large scale nonsymmetric matrices*, Num. Math., 56:721–734, 1990.

[7] D. HO, F. CHATELIN AND M. BENNANI, *Arnoldi–Tchebychev procedure for large scale nonsymmetric matrices*, Math. Modell. Num. Anal., 24(1):53–65, 1990.

[8] S. KANIEL, *Estimates for some computational techniques in linear algebra*, Math. Comp., 20(95):369–378, 1966.

[9] C. LANCZOS, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Standards, 45(4):255–282, 1950.

[10] T.A. MANTEUFFEL, *The Tchebychev iteration for nonsymmetric linear systems*, Num. Math., 28:307–327, 1977.

[11] T.A. MANTEUFFEL, *Adaptive procedure for estimating parameters for the nonsymmetric Tchebychev iteration*, Num. Math., 31:183–208, 1978.

[12] C.C. PAIGE, *Computational variants of the Lanczos method for the eigenproblem*, J. Inst. Math. Appl., 10:373–381, 1972.

[13] B.N. PARLETT, *The symmetric eigenvalue problem*, Prentice-Hall, Englewood Cliffs, N.J., 1980.

[14] Y. SAAD, *Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices*, Lin. Alg. Appl., 34:269–295, 1980.

[15] Y. SAAD, *On the rates of convergence of the Lanczos and the block Lanczos methods*, SIAM J. Num. Anal., 17(5):687–706, 1980.

[16] Y. SAAD, *Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems*, Math. Comp., 42(166):567–588, 1984.

[17] Y. SAAD, *Numerical solution of large nonsymmetric eigenvalue problems*, Comp. Phys. Comm., 53:71–90, 1989.

[18] Y. SAAD, *Numerical methods for large eigenvalue problems*, Manchester University Press, 1992.

[19] M. SADKANE, *A block Arnoldi–Chebychev method for computing the leading eigenpairs of large sparse unsymmetric matrices*, Num. Math., 64:181–193, 1993.

[20] M. SADKANE, *Block–Arnoldi and Davidson methods for unsymmetric large eigenvalue problems*, Num. Math., 64:195–211, 1993.

[21] D.C. SORENSON, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J. Matrix Anal. Appl., 13(1):357–385, 1992.

[22] J.H. WILKINSON, *The algebraic eigenvalue problem*, Clarendon Press, Oxford, 1965.

No.	Author(s)	Title
9321	A.L. Bovenberg and S. Smulders	Environmental Quality and Pollution-saving Technological Change in a Two-sector Endogenous Growth Model
9322	K.-E. Wärneryd	The Will to Save Money: an Essay on Economic Psychology
9323	D. Talman, Y. Yamamoto and Z. Yang	The $(2^{n+m+1} - 2)$-Ray Algorithm: A New Variable Dimension Simplicial Algorithm For Computing Economic Equilibria on $S^n \times R^m_+$
9324	H. Huizinga	The Financing and Taxation of U.S. Direct Investment Abroad
9325	S.C.W. Eijffinger and E. Schaling	Central Bank Independence: Theory and Evidence
9326	T.C. To	Infant Industry Protection with Learning-by-Doing
9327	J.P.J.F. Scheepens	Bankruptcy Litigation and Optimal Debt Contracts
9328	T.C. To	Tariffs, Rent Extraction and Manipulation of Competition
9329	F. de Jong, T. Nijman and A. Röell	A Comparison of the Cost of Trading French Shares on the Paris Bourse and on SEAQ International
9330	H. Huizinga	The Welfare Effects of Individual Retirement Accounts
9331	H. Huizinga	Time Preference and International Tax Competition
9332	V. Feltkamp, A. Koster, A. van den Nouweland, P. Borm and S. Tijs	Linear Production with Transport of Products, Resources and Technology
9333	B. Lauterbach and U. Ben-Zion	Panic Behavior and the Performance of Circuit Breakers: Empirical Evidence
9334	B. Melenberg and A. van Soest	Semi-parametric Estimation of the Sample Selection Model
9335	A.L. Bovenberg and F. van der Ploeg	Green Policies and Public Finance in a Small Open Economy
9336	E. Schaling	On the Economic Independence of the Central Bank and the Persistence of Inflation
9337	G.-J. Otten	Characterizations of a Game Theoretical Cost Allocation Method
9338	M. Gradstein	Provision of Public Goods With Incomplete Information: Decentralization vs. Central Planning
9339	W. Güth and H. Kliemt	Competition or Co-operation
9340	T.C. To	Export Subsidies and Oligopoly with Switching Costs