A Parallel Rational Krylov Algorithm For Eigenvalue Computation

Daniel Skoogh *

November 1998

Abstract

An implementation of a parallel rational Krylov method for the generalised matrix eigenvalue problem is discussed. The implementation has been done on a MIMD computer and a cluster of workstations. The rational Krylov algorithm is an extension of the shift-and-invert Arnoldi method where several shifts are used to compute basis vectors for one subspace. In this parallel implementation, the different shifted matrices are factorised each on one processor and then the iteration vectors are generated in parallel.

Keywords: eigenvalues, eigenvectors, sparse, parallel, rational, Krylov, shift, invert, Arnoldi
AMS subject classification 65F15, 65F50, 65Y05

1 Introduction and Purpose

Arnoldi [2] and Lanczos [4], the currently most successful iterative eigenvalue algorithms, are both based on Krylov subspaces. They start with an arbitrary starting vector and build up a basis one vector at a time, letting the matrix operate on the most recent basis vector and including this new direction into the subspace. Any vector in the Krylov subspace can be expressed as a polynomial of the matrix applied to the starting vector.

The rational Krylov algorithm developed by Ruhe [7, 8, 9, 10] does the same thing by operating with shifted and inverted matrices with different shifts on linear combinations of previous basis vectors. Every vector in the subspace can now be expressed as a rational function of the matrix times the starting vector. In theory it is possible to generate the basis vectors in parallel by operating with shifted and inverted matrices with different shifts, one on each processor. The purpose of this work is to investigate if this can be done in practice [12].

Our algorithm computes one matrix factorisation on each processor. In ARPACK "ARnoldi PACKage" [5] one the other hand, where at most one shift is used, the matrix vector multiplication is parallelised and each basis vector is distributed among the processors; for details see P_ARPACK [6].

*Report No 1998-45 in "Blå serien" preprint series, available at URL: `http://www.math.chalmers.se/Math/Research/Preprints`. Author's address: Department of Mathematics, Chalmers University of Technology and the University of Göteborg, S-41296 Göteborg, Sweden (`skoogh@math.chalmers.se`). This work was partly supported by Swedish National Board for Industrial and Technical Development, grant 8902538-5.

2 Rational Krylov Algorithm

The rational Krylov method is a generalisation of the shift-and-invert Arnoldi method. In the latter method we choose one shift μ in the complex plane where we want the eigenvalues to converge fast, while in the rational Krylov method we choose several shifts $\mu_1, \ldots, \mu_{\bar{i}}$. Let us first consider the eigenvalue problem

$$\boldsymbol{A}\boldsymbol{u} = \lambda \boldsymbol{u}. \tag{1}$$

The space $\mathcal{S}_{\bar{k}+1}$ in the rational Krylov method is

$$\begin{aligned} \mathcal{S}_{\bar{k}+1} \equiv \text{span}\{&\boldsymbol{v}_1, (\boldsymbol{A}-\mu_1\boldsymbol{I})^{-1}\boldsymbol{v}_1, \ldots, (\boldsymbol{A}-\mu_1\boldsymbol{I})^{-\bar{j}_1}\boldsymbol{v}_1, \\ &(\boldsymbol{A}-\mu_2\boldsymbol{I})^{-1}\boldsymbol{v}_1, \ldots, (\boldsymbol{A}-\mu_2\boldsymbol{I})^{-\bar{j}_2}\boldsymbol{v}_1, \\ &\ldots \\ &(\boldsymbol{A}-\mu_{\bar{i}}\,\boldsymbol{I})^{-1}\boldsymbol{v}_1, \ldots, (\boldsymbol{A}-\mu_{\bar{i}}\,\boldsymbol{I})^{-\bar{j}_{\bar{i}}}\boldsymbol{v}_1\}, \end{aligned} \tag{2}$$

where $\bar{k} = \bar{j}_1 + \bar{j}_2 + \ldots + \bar{j}_{\bar{i}}$ and $\text{Dim}(\mathcal{S}_{\bar{k}+1}) \leq \bar{k}+1$. In the rational Krylov method every vector $\boldsymbol{x} \in \mathcal{S}_{\bar{k}}$ can be written as

$$\boldsymbol{x} = r(\boldsymbol{A})\boldsymbol{v}_1$$

where

$$\begin{aligned} r(\lambda) =& \frac{p_k(\lambda)}{(\lambda-\mu_1)^{j_1}(\lambda-\mu_2)^{j_2}\ldots(\lambda-\mu_{\bar{i}}\,)^{j_{\bar{i}}}} \\ =& c_0 + \sum_{k=1}^{j_1}\frac{c_{1,k}}{(\lambda-\mu_1)^k} + \sum_{k=1}^{j_2}\frac{c_{2,k}}{(\lambda-\mu_2)^k} \\ &+ \ldots + \sum_{k=1}^{j_{\bar{i}}}\frac{c_{\bar{i}\,,k}}{(\lambda-\mu_{\bar{i}}\,)^k}, \end{aligned}$$

and $j_1 \leq \bar{j}_1, j_2 \leq \bar{j}_2, \ldots, j_{\bar{i}} \leq \bar{j}_{\bar{i}}$.

Different rational and polynomial matrix functions of the same matrix commute, and thus in creating a basis for $\mathcal{S}_{\bar{k}+1}$ (2) it does not matter in which order the operators $(\boldsymbol{A}-\mu_1\boldsymbol{I})^{-1}, \ldots, (\boldsymbol{A}-\mu_{\bar{i}}\,\boldsymbol{I})^{-1}$ are applied. This is the key to the parallel algorithm.

Let us first describe a sequential rational Krylov method, but apply it to the generalised eigenproblem

$$\boldsymbol{A}\boldsymbol{u} = \lambda \boldsymbol{B}\boldsymbol{u}. \tag{3}$$

If we replace $\boldsymbol{A}$ with $\boldsymbol{B}^{-1}\boldsymbol{A}$ in (2) the operators become $(\boldsymbol{B}^{-1}\boldsymbol{A}-\mu_i\boldsymbol{I})^{-1} = (\boldsymbol{A}-\mu_i\boldsymbol{B})^{-1}\boldsymbol{B}$. All shifts μ_i have to be chosen so that $\boldsymbol{A}-\mu_i\boldsymbol{B}$ is nonsingular. In the algorithm below we use a total of $\bar{i}$ shifts and $\bar{k}$ iterations, $\bar{i} \leq \bar{k}$. The variable i_k stands for the shift μ_i, $1 \leq i \leq \bar{i}$ which is used in the k:th iteration.

Rational Krylov Algorithm 1

Choose start vector $\boldsymbol{v}_1$ of unit length
for $k = 1 : \bar{k}$
  $\boldsymbol{r} = \boldsymbol{V}_k \boldsymbol{t}_k$ Choose starting combination
  $\boldsymbol{r} = (\boldsymbol{A} - \mu_{i_k} \boldsymbol{B})^{-1} \boldsymbol{B} \boldsymbol{r}$
  $\boldsymbol{h}_k = \boldsymbol{V}_k^H \boldsymbol{r}$
  $\boldsymbol{r} = \boldsymbol{r} - \boldsymbol{V}_k \boldsymbol{h}_k$
  $h_{k+1,k} = \| \boldsymbol{r} \|_2$
  $\boldsymbol{v}_{k+1} = \boldsymbol{r} / h_{k+1,k}$
end

By eliminating the intermediate vector $\boldsymbol{r}$ in the rational Krylov algorithm, we get

$$\boldsymbol{v}_{k+1} h_{k+1,k} = (\boldsymbol{A} - \mu_{i_k} \boldsymbol{B})^{-1} \boldsymbol{B} \boldsymbol{V}_k \boldsymbol{t}_k - \boldsymbol{V}_k \boldsymbol{h}_k.$$

Replace $\boldsymbol{h}_k$ with $\left[\begin{smallmatrix} \boldsymbol{h}_k \\ h_{k+1,k} \end{smallmatrix} \right]$

$$\boldsymbol{V}_{k+1} \boldsymbol{h}_k = (\boldsymbol{A} - \mu_{i_k} \boldsymbol{B})^{-1} \boldsymbol{B} \boldsymbol{V}_k \boldsymbol{t}_k,$$

and multiply by $(\boldsymbol{A} - \mu_{i_k} \boldsymbol{B})$ to get

$$(\boldsymbol{A} - \mu_{i_k} \boldsymbol{B}) \boldsymbol{V}_{k+1} \boldsymbol{h}_k = \boldsymbol{B} \boldsymbol{V}_k \boldsymbol{t}_k.$$

Separate terms with $\boldsymbol{A}$ and $\boldsymbol{B}$, and replace $\boldsymbol{t}_k$ with $\left[\begin{smallmatrix} \boldsymbol{t}_k \\ 0 \end{smallmatrix} \right]$ to get the relation at the k:th step,

$$\boldsymbol{A} \boldsymbol{V}_{k+1} \boldsymbol{h}_k = \boldsymbol{B} \boldsymbol{V}_{k+1} (\boldsymbol{h}_k \mu_{i_k} + \boldsymbol{t}_k).$$

Set $\boldsymbol{H}_{\bar{k}+1,\bar{k}} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{\bar{k}}]$ and $\boldsymbol{T}_{\bar{k}+1,\bar{k}} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_{\bar{k}}]$ with appropriate zeros added to the bottom of each $\boldsymbol{h}_k$, $\boldsymbol{t}_k$. Introduce the new matrix

$$\boldsymbol{K}_{\bar{k}+1,\bar{k}} = \boldsymbol{H}_{\bar{k}+1,\bar{k}} \mathrm{diag}(\mu_{i_k}) + \boldsymbol{T}_{\bar{k}+1,\bar{k}}. \tag{4}$$

Note that both $\boldsymbol{H}_{\bar{k}+1,\bar{k}}$ and $\boldsymbol{K}_{\bar{k}+1,\bar{k}}$ are Hessenberg matrices. We finally get the relation

$$\boldsymbol{A} \boldsymbol{V}_{\bar{k}+1} \boldsymbol{H}_{\bar{k}+1,\bar{k}} = \boldsymbol{B} \boldsymbol{V}_{\bar{k}+1} \boldsymbol{K}_{\bar{k}+1,\bar{k}}. \tag{5}$$

From now on we will denote the matrices $\boldsymbol{H}_{\bar{k}+1,\bar{k}}$ and $\boldsymbol{K}_{\bar{k}+1,\bar{k}}$ by $\boldsymbol{H}$ and $\boldsymbol{K}$ respectively.

2.1 Approximate Eigensolution

If $\mathcal{S}_{\bar{k}}$ is invariant under $(\boldsymbol{A} - \mu_i \boldsymbol{B})^{-1} \boldsymbol{B}$ then $h_{\bar{k}+1,\bar{k}} = 0$ and the relation (5) becomes

$$\boldsymbol{A} \boldsymbol{V}_{\bar{k}} \boldsymbol{H}_{\bar{k},\bar{k}} = \boldsymbol{B} \boldsymbol{V}_{\bar{k}} \boldsymbol{K}_{\bar{k},\bar{k}}. \tag{6}$$

If

$$\boldsymbol{K}_{\bar{k},\bar{k}} \boldsymbol{y} = \lambda \boldsymbol{H}_{\bar{k},\bar{k}} \boldsymbol{y} \tag{7}$$

then, inserting it in (6), we get

$$\boldsymbol{A}\boldsymbol{V}_{\bar{k}}\boldsymbol{H}_{\bar{k},\bar{k}}\boldsymbol{y} = \lambda\boldsymbol{B}\boldsymbol{V}_{\bar{k}}\boldsymbol{H}_{\bar{k},\bar{k}}\boldsymbol{y}.$$

Thus if $(\lambda, \boldsymbol{y})$ is an eigenpair to (7) then $(\lambda, \boldsymbol{V}_{\bar{k}}\boldsymbol{H}_{\bar{k},\bar{k}}\boldsymbol{y})$ is an eigenpair to (3).

Add $-\gamma\boldsymbol{B}\boldsymbol{V}_{\bar{k}}\boldsymbol{H}_{\bar{k},\bar{k}}$ to (6):

$$(\boldsymbol{A} - \gamma\boldsymbol{B})\boldsymbol{V}_{\bar{k}}\boldsymbol{H}_{\bar{k},\bar{k}} = \boldsymbol{B}\boldsymbol{V}_{\bar{k}}(\boldsymbol{K}_{\bar{k},\bar{k}} - \gamma\boldsymbol{H}_{\bar{k},\bar{k}}).$$

Multiply with $(\boldsymbol{A} - \gamma\boldsymbol{B})^{-1}$ from the left and $(\boldsymbol{K}_{\bar{k},\bar{k}} - \gamma\boldsymbol{H}_{\bar{k},\bar{k}})^{-1}$ from the right, yielding

$$\boldsymbol{V}_{\bar{k}}\boldsymbol{H}_{\bar{k},\bar{k}}(\boldsymbol{K}_{\bar{k},\bar{k}} - \gamma\boldsymbol{H}_{\bar{k},\bar{k}})^{-1} = (\boldsymbol{A} - \gamma\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{V}_{\bar{k}}.$$

We see that $\mathcal{S}_{\bar{k}}$ is invariant under any $(\boldsymbol{A} - \gamma\boldsymbol{B})^{-1}\boldsymbol{B}$ for which $(\boldsymbol{A} - \gamma\boldsymbol{B})$ is nonsingular.

Now assume that $\mathcal{S}_{\bar{k}}$ is not invariant, which will be the most usual case. Let $(\tilde{\lambda}_j, \tilde{\boldsymbol{y}}_j)$ be an eigenpair of (7) and take

$$\tilde{\boldsymbol{u}}_j = \boldsymbol{V}_{\bar{k}+1}\boldsymbol{H}\tilde{\boldsymbol{y}}_j \tag{8}$$

as the approximate eigenvector and $\tilde{\lambda}_j$ as the approximate eigenvalue to (3). The residual will be

$$\begin{aligned}(\boldsymbol{A} - \tilde{\lambda}_j\boldsymbol{B})\tilde{\boldsymbol{u}}_j &= (\boldsymbol{A} - \tilde{\lambda}_j\boldsymbol{B})\boldsymbol{V}_{\bar{k}+1}\boldsymbol{H}\boldsymbol{y}_j \\ &= \boldsymbol{B}\boldsymbol{V}_{\bar{k}+1}(\boldsymbol{K} - \tilde{\lambda}_j\boldsymbol{H})\boldsymbol{y}_j \\ &= \boldsymbol{B}\boldsymbol{v}_{\bar{k}+1}(k_{\bar{k}+1,\bar{k}} - \tilde{\lambda}_j h_{\bar{k}+1,\bar{k}})\boldsymbol{e}_{\bar{k}}^T\boldsymbol{y}_j \\ &= \boldsymbol{B}\boldsymbol{v}_{\bar{k}+1}(\mu_{i_{\bar{k}}} - \tilde{\lambda}_j)h_{\bar{k}+1,\bar{k}}\boldsymbol{e}_{\bar{k}}^T\boldsymbol{y}_j.\end{aligned} \tag{9}$$

The first equality comes from (8), the second from (5), the third from (7) and the fourth from (4). Note that if $\boldsymbol{B}$ is nonsingular then the residual is orthogonal against $\text{span}\{\boldsymbol{B}^{-H}\boldsymbol{V}_{\bar{k}}\}$.

3 Parallel Rational Krylov Algorithm

3.1 Parallel Rational Krylov Algorithm

In the parallel algorithm which we are now going to describe, the basis vectors are generated in parallel. The matrix operations themselves are not parallelised.

The key to the parallel algorithm is that it does not matter in exact arithmetic in which order the operators $(\boldsymbol{A} - \mu_i\boldsymbol{B})^{-1}\boldsymbol{B}$, $i = 1, \ldots, \bar{i}$ are applied in building a basis for the subspace $\mathcal{S}_{\bar{k}+1}$ (2).

There are several possible parallel rational Krylov algorithms. We have made two different algorithms, a SPMD (Same Program Multiple Data) algorithm and a Master/Slave algorithm. They differ in the way orthogonalisation is done. Both algorithms use $\bar{p}$ different processors to compute $\boldsymbol{r}_p = (\boldsymbol{A} - \mu_p\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{r}_p$. The SPMD algorithm which is described below lets each processor orthogonalise its own vector. The Master/Slave algorithm which is described in [12] uses an additional processor to do the orthogonalisation. In exact arithmetic, given the same conditions, the different algorithms generate identical matrices $\boldsymbol{H}$, $\boldsymbol{K}$ and

$\boldsymbol{V}_{\bar{k}+1}$. We have made two implementations of the SPMD algorithm, which differ in how the communication is done.

The algorithm below uses a total number of $\bar{\imath}$ shifts, $\mu_1, \ldots, \mu_{\bar{\imath}}$. Each shift μ_i, $1 \leq i \leq \bar{\imath}$, is used $\bar{\jmath}$ times in the matrix operator $(\boldsymbol{A} - \mu_i \boldsymbol{B})^{-1}\boldsymbol{B}$. At a given moment at a processor p, $1 \leq p \leq \bar{p}$, the letter k stands for the total number of basis vectors. The vector $\boldsymbol{v}_{k+1}$ is the basis vector being calculated in the current iteration, and $\boldsymbol{v}_{k_p}$ is the vector that the matrix $(\boldsymbol{A} - \mu_i \boldsymbol{B})^{-1}\boldsymbol{B}$ is applied to. When the algorithm has run to completion, the total number of basis vectors is $\bar{k} + 1$, where $\bar{k} = \bar{\imath}\bar{\jmath} = \bar{m}\bar{p}\bar{\jmath}$ and each processor takes $\bar{m}$ shifts.

Parallel Rational Krylov Algorithm

1 Choose $\boldsymbol{v}_1$ such that $\| \boldsymbol{v}_1 \| = 1$ (the same vector on all processors)
2 **start program on** $\bar{p}$ processors, $p = 1 : \bar{p}$
3 **for** $m = 1 : \bar{m}$
4 $\quad i = (m-1)\bar{p} + p$
5 $\quad \boldsymbol{LU} = \boldsymbol{A} - \mu_i \boldsymbol{B}$ (factorise)
6 $\quad$ **for** $j = 1 : \bar{\jmath}$
7 $\quad\quad k = (m-1)\bar{\jmath}\,\bar{p} + (j-1)\bar{p} + p$
8 $\quad\quad k_p = \max(1, k - \bar{p} + 1)$
9 $\quad\quad \boldsymbol{r}_{k+1} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{B}\boldsymbol{v}_{k_p}$ (operate)
10 $\quad\quad \boldsymbol{H}(1 : k_p, k) = \boldsymbol{V}_{k_p}^H \boldsymbol{r}_{k+1}$ (Gram-Schmidt part I)
11 $\quad\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_{k+1} - \boldsymbol{V}_{k_p}\boldsymbol{H}(1 : k_p, k)$
12 $\quad\quad$ **receive** $\boldsymbol{v}_l$ and $\boldsymbol{h}_{l-1}$ from its processor, $l = k_p + 1 : k$
13 $\quad\quad \boldsymbol{H}(k_p + 1 : k, k) = \boldsymbol{V}_{k_p+1:k}^H \boldsymbol{r}_{k+1}$ (Gram-Schmidt part II)
14 $\quad\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_{k+1} - \boldsymbol{V}_{k_p+1:k}\boldsymbol{H}(k_p + 1 : k, k)$
15 $\quad\quad h_{k+1,k} = \| \boldsymbol{r}_{k+1} \|$
16 $\quad\quad \boldsymbol{v}_{k+1} = \boldsymbol{r}_{k+1}/h_{k+1,k}$ (deliver new vector)
17 $\quad\quad$ ($\boldsymbol{v}_{k+1}$ will be the start vector $\boldsymbol{v}_{k_p}$ in the next iteration on line 9)
18 $\quad\quad$ **send** $\boldsymbol{v}_{k+1}$ and $\boldsymbol{h}_k$ to the other processors
19 $\quad$ **end**
20 **end**
21 **receive** $\boldsymbol{v}_l$ and $\boldsymbol{h}_{l-1}$ from its processor, $l = k + 2 : k + (\bar{p} - p) + 1$
(from processor $p + 1, \ldots, \bar{p}$)

Each processor (or node) keeps a copy of all basis vectors that are generated. After the processor p has carried out the operation on line 9, $\boldsymbol{r}_{k+1} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{B}\boldsymbol{v}_{k_p}$, the vector $\boldsymbol{r}_{k+1}$ is first (in Gram-Schmidt part I on line 10) orthogonalised against the basis vectors located at the local processor. The last basis vectors are not yet available because the other processors are working on them. After the first part of the orthogonalisation is finished, the processor is now ready to receive the last basis vectors and (in Gram-Schmidt part II on line 13) orthogonalise $\boldsymbol{r}_{k+1}$ against them, and to send the resulting vector $\boldsymbol{v}_{k+1}$ to the other processors.

3.2 Subspace

The subspace $\mathcal{S}_{\bar{k}+1}$ spanned by the basis vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{\bar{k}+1}$ is a union of Krylov spaces

$$\mathcal{S}_{\bar{k}+1} = \bigcup_{i=1}^{\bar{i}} \mathcal{K}_{\bar{j}+1}((\boldsymbol{A} - \mu_i \boldsymbol{B})^{-1}\boldsymbol{B}, \boldsymbol{v}_1), \tag{10}$$

where $\mathcal{K}_k(\boldsymbol{C}, \boldsymbol{v})$ stands for the Krylov subspace

$$\mathcal{K}_k(\boldsymbol{C}, \boldsymbol{v}) = \text{span}\{\boldsymbol{v}, \boldsymbol{C}\boldsymbol{v}, \boldsymbol{C}^2\boldsymbol{v}, \ldots, \boldsymbol{C}^{k-1}\boldsymbol{v}\}. \tag{11}$$

The particular order in which the subspace is built up is

$$\begin{aligned}
\mathcal{S}_{\bar{k}+1} = \text{span}\{&\boldsymbol{v}_1, (\boldsymbol{A} - \mu_1\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{v}_1, \ldots, (\boldsymbol{A} - \mu_{\bar{p}}\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{v}_1, \\
&((\boldsymbol{A} - \mu_1\boldsymbol{B})^{-1}\boldsymbol{B})^2\boldsymbol{v}_1, \ldots, ((\boldsymbol{A} - \mu_{\bar{p}}\boldsymbol{B})^{-1}\boldsymbol{B})^2\boldsymbol{v}_1, \\
&\cdots \\
&((\boldsymbol{A} - \mu_1\boldsymbol{B})^{-1}\boldsymbol{B})^{\bar{j}}\,\boldsymbol{v}_1, \ldots, ((\boldsymbol{A} - \mu_{\bar{p}}\boldsymbol{B})^{-1}\boldsymbol{B})^{\bar{j}}\,\boldsymbol{v}_1, \\
&\cdots \\
&\cdots \\
&(\boldsymbol{A} - \mu_{\bar{i}-\bar{p}+1}\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{v}_1, \ldots, (\boldsymbol{A} - \mu_{\bar{i}}\,\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{v}_1, \\
&((\boldsymbol{A} - \mu_{\bar{i}-\bar{p}+1}\boldsymbol{B})^{-1}\boldsymbol{B})^2\boldsymbol{v}_1, \ldots, ((\boldsymbol{A} - \mu_{\bar{i}}\,\boldsymbol{B})^{-1}\boldsymbol{B})^2\boldsymbol{v}_1, \\
&\cdots \\
&((\boldsymbol{A} - \mu_{\bar{i}-\bar{p}+1}\boldsymbol{B})^{-1}\boldsymbol{B})^{\bar{j}}\,\boldsymbol{v}_1, \ldots, ((\boldsymbol{A} - \mu_{\bar{i}}\,\boldsymbol{B})^{-1}\boldsymbol{B})^{\bar{j}}\,\boldsymbol{v}_1, \}.
\end{aligned} \tag{12}$$

The first k, $1 \leq k \leq \bar{k}$ basis vectors generated by the parallel rational Krylov algorithm span the same subspace as the first k, $1 \leq k \leq \bar{k}$ basis vectors in (12).

On a particular processor p, $1 \leq p \leq \bar{p}$, let us now express the vectors $\boldsymbol{v}_k$, $\boldsymbol{v}_{k+1}$ and $\boldsymbol{v}_{k_p}$ in terms of these Krylov spaces.

First let $m = 1$ and $j = 1$, then

$$\boldsymbol{v}_{k_p} = \boldsymbol{v}_1, \tag{13}$$

$$\boldsymbol{v}_k \in \mathcal{S}_k = \text{span}\{\boldsymbol{v}_1, (\boldsymbol{A} - \mu_1\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{v}_1, \ldots, (\boldsymbol{A} - \mu_{p-1}\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{v}_1\} \tag{14}$$

and

$$\boldsymbol{v}_{k+1} \in \mathcal{S}_k \bigcup \text{span}\{(\boldsymbol{A} - \mu_p\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{v}_1\}. \tag{15}$$

If either m or j is larger than 1, then

$$\begin{aligned}
\boldsymbol{v}_k \in \mathcal{S}_k = &\bigcup_{i=1}^{(m-1)\bar{p}} \mathcal{K}_{\bar{j}+1}((\boldsymbol{A} - \mu_i\boldsymbol{B})^{-1}\boldsymbol{B}, \boldsymbol{v}_1) \\
&\bigcup_{i=(m-1)\bar{p}+1}^{m\bar{p}} \mathcal{K}_j((\boldsymbol{A} - \mu_i\boldsymbol{B})^{-1}\boldsymbol{B}, \boldsymbol{v}_1) \\
&\bigcup_{i=(m-1)\bar{p}+1}^{(m-1)\bar{p}+p-1} \text{span}\{((\boldsymbol{A} - \mu_i\boldsymbol{B})^{-1}\boldsymbol{B})^j\boldsymbol{v}_1\},
\end{aligned} \tag{16}$$

$$\boldsymbol{v}_{k+1} \in \mathcal{S}_{k+1} = \mathcal{S}_k \bigcup \text{span}\{((\boldsymbol{A} - \mu_{(m-1)\bar{p}+p}\boldsymbol{B})^{-1}\boldsymbol{B})^j \boldsymbol{v}_1\}, \tag{17}$$

and $\boldsymbol{v}_{k+1}$ will be the start vector $\boldsymbol{v}_{k_p}$ in the next iteration on line 9.

4 Structures in the $\boldsymbol{H}$-matrix

4.1 The Nonzero Structure in the $\boldsymbol{H}$-matrix

In this section we will show how the nonzero structure in the $\boldsymbol{H}$-matrix of the parallel rational Krylov algorithm applied to Hermitian matrices is related to different parameters of the algorithm. We will focus on how the total number of processors $\bar{p}$ and the total number of shifts at each processor $\bar{m}$ will influence the nonzero structure.

By its definition $\boldsymbol{H}$ will be a Hessenberg matrix. The sub-diagonal elements are all real and, as long as we do not have an invariant subspace, they will also be all nonzero. Let us concentrate on the elements of the upper triangle, $h_{l,k}, l \leq k$.

Assume that the matrix $\boldsymbol{A}$ is Hermitian, $\boldsymbol{B} = \boldsymbol{I}$ and the shifts μ_i, $i = 1, \ldots, \bar{i}$ are real. On a particular processor p, $1 \leq p \leq \bar{p}$, let us now describe how column k in the matrix $\boldsymbol{H}$ is computed. If we use only one shift ($\bar{i} = 1$) and run on one processor ($\bar{p} = 1$), then we get the shift-and-invert Arnoldi algorithm, and the matrix $\boldsymbol{H}$ is real, symmetric and tridiagonal. If we use more than one processor ($\bar{p} > 1$) and only one shift per processor($\bar{m} = 1$), then

$$\begin{aligned} h_{l,k} &= ((\boldsymbol{A} - \mu_p \boldsymbol{I})^{-1}\boldsymbol{v}_{k_p}, \boldsymbol{v}_l), l = 1, \ldots, k \\ &= (\boldsymbol{v}_{k_p}, (\boldsymbol{A} - \mu_p \boldsymbol{I})^{-H}\boldsymbol{v}_l) \\ &= (\boldsymbol{v}_{k_p}, (\boldsymbol{A} - \mu_p \boldsymbol{I})^{-1}\boldsymbol{v}_l). \end{aligned}$$

Note the relation between the index of the start vector $\boldsymbol{v}_{k_p}$ and the index of the basis vector $\boldsymbol{v}_{k+1}$ being calculated. If we have operated with the matrix $(\boldsymbol{A} - \mu_p \boldsymbol{I})^{-1}$ only once ($j = 1$), then $k + 1 = k_p + p$; otherwise $k + 1 = k_p + \bar{p}$,

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_{k_p+\bar{p}} = (\boldsymbol{A} - \mu_p \boldsymbol{I})^{-1}\boldsymbol{v}_{k_p},\ j > 1.$$

Now assume that we have operated more than two times with the matrix $(\boldsymbol{A} - \mu_p \boldsymbol{I})^{-1}$, ($j > 2$). If $1 \leq l < k_p - \bar{p}$ then

$$(\boldsymbol{A} - \mu_p \boldsymbol{I})^{-1}\boldsymbol{v}_l \in \text{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k_p-\bar{p}}\}.$$

Because we have an orthonormal basis, this means that

$$h_{l,k} = 0,\ l = 1, \ldots, k_p - \bar{p} - 1.$$

If $k_p - \bar{p} \leq l < k_p$ then

$$(\boldsymbol{A} - \mu_p \boldsymbol{I})^{-1}\boldsymbol{v}_l \in \text{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k_p}\}$$

and the element $h_{l,k}$ is nonzero in general. A similar argument for $k_p \leq l \leq k$ shows that the element $h_{l,k}$ is nonzero in general.

The number of nonzero elements in each column will be

$$k + 1 - (k_p - \bar{p} - 1) = k_p + \bar{p} - (k_p - \bar{p} - 1) = 2\bar{p} + 1.$$

If we now look at the first $2\bar{p}$ columns ($j \leq 2$) of the $\boldsymbol{H}$-matrix, we can use a similar argument as above, and conclude that each column $\boldsymbol{H}(1:k+1,k)$, $1 \leq k \leq 2\bar{p}$ is full.

Next assume that each processor uses several shifts ($\bar{m} > 1$). In the first $\bar{j}\,\bar{p}$ columns in the $\boldsymbol{H}$-matrix, the nonzero structure is the same as above. Now assume that we just enter the m loop with the value $m = 2$ (and $j = 1$). In lines 4 and 5 we get a new shift and a new $\boldsymbol{LU}$ factorisation is performed. Note that this is the first time when the shift $\mu_{\bar{p}+p}$ is used in the shifted and inverted matrix $(\boldsymbol{A} - \mu_{\bar{p}+p}\boldsymbol{I})^{-1}$ in the matrix vector product

$$\boldsymbol{r}_{k+1} = (\boldsymbol{A} - \mu_{\bar{p}+p}\boldsymbol{I})^{-1}\boldsymbol{v}_{k_p},$$

and the element $h_{l,k}$ becomes

$$\begin{aligned} h_{l,k} &= ((\boldsymbol{A} - \mu_{\bar{p}+1}\boldsymbol{I})^{-1}\boldsymbol{v}_{k_p}, \boldsymbol{v}_l), l = 1, \ldots, k \\ &= (\boldsymbol{v}_{k_p}, (\boldsymbol{A} - \mu_{\bar{p}+p}\boldsymbol{I})^{-H}\boldsymbol{v}_l) \\ &= (\boldsymbol{v}_{k_p}, (\boldsymbol{A} - \mu_{\bar{p}+p}\boldsymbol{I})^{-1}\boldsymbol{v}_l). \end{aligned}$$

Because the vector $(\boldsymbol{A} - \mu_{\bar{p}+p}\boldsymbol{I})^{-1}\boldsymbol{v}_1$ does not belong to the subspace $\mathcal{S}_k$, the vector $(\boldsymbol{A} - \mu_{\bar{p}+p}\boldsymbol{I})^{-1}\boldsymbol{v}_l$ cannot be expressed as a linear combination of the basis vectors in the subspace $\mathcal{S}_k$ for any $l \leq k$. This means that the element $h_{l,k}$ is nonzero in general for $l \leq k$.

In the next iteration $j = 2$ and $1 \leq l < k_p$ we have

$$(\boldsymbol{A} - \mu_{\bar{p}+p}\boldsymbol{I})^{-1}\boldsymbol{v}_l \in \text{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k_p}\}$$

Because the vector $\boldsymbol{v}_{k_p}$ belongs to this subspace, the element $h_{l,k}$ is nonzero in general. If $k_p \leq l \leq k$, then

$$(\boldsymbol{A} - \mu_{\bar{p}+p}\boldsymbol{I})^{-1}\boldsymbol{v}_l \in \text{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k_p+\bar{p}}\}$$

and note again that $k_p + \bar{p} = k + 1$. Thus the element $h_{l,k}$ is nonzero in general.

First when $j = 3$ and $1 \leq l < k_p - \bar{p}$, we have

$$(\boldsymbol{A} - \mu_{\bar{p}+p}\boldsymbol{I})^{-1}\boldsymbol{v}_l \in \text{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k_p-\bar{p}}\}$$

and thus the element $h_{l,k} = 0$. Using a similar argument as above, one can see that the element $h_{l,k}$ is nonzero for $k_p - \bar{p} \leq l \leq k$.

We can conclude that, every time the shifts are changed ($m := m + 1$), we get twice the number of processors ($2\bar{p}$) of columns with nonzero elements.

4.2 Real Elements in the $\boldsymbol{H}$-matrix

In this section we will show that, if we apply the rational Krylov algorithm to a standard Hermitian eigenvalue problem with real shifts, we obtain a real matrix $\boldsymbol{H}$.

By construction the matrix $\boldsymbol{H}$ is upper Hessenberg, and the elements $h_{k+1,k}$ on the sub-diagonal are real. Let us concentrate on the elements of the upper triangle, $h_{l,k}, l \leq k$.

Let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ be the orthonormal basis vectors generated by the rational Krylov algorithm. Let us express the basis vectors with rational functions,

$$\boldsymbol{v}_l = r_l(\boldsymbol{A})\boldsymbol{v}_1,\ l = 1, \ldots, k$$

and also the continuation vector $\boldsymbol{V}_k\boldsymbol{t}_k$:

$$\boldsymbol{V}_k\boldsymbol{t}_k = q_k(\boldsymbol{A})\boldsymbol{v}_1. \tag{18}$$

In a rational Krylov algorithm the element $h_{l,k}$ in the matrix $\boldsymbol{H}$ is given by

$$\begin{aligned} h_{l,k} &= \boldsymbol{v}_l^H(\boldsymbol{A}-\mu_{i_k}\boldsymbol{I})^{-1}\boldsymbol{V}_k\boldsymbol{t}_k \\ &= \boldsymbol{v}_1^H r_l(\boldsymbol{A})^H(\boldsymbol{A}-\mu_{i_k}\boldsymbol{I})^{-1}q_k(\boldsymbol{A})\boldsymbol{v}_1. \end{aligned}$$

The element $h_{l,k}$ is real if the matrix $r_l(\boldsymbol{A})^H(\boldsymbol{A}-\mu_{i_k}\boldsymbol{I})^{-1}q_k(\boldsymbol{A})$ is Hermitian. Since different rational matrix functions of the same matrix commute, and since the matrix $(\boldsymbol{A}-\mu_i\boldsymbol{I})^{-1}$ is Hermitian, it remains to prove that the matrices $r_l(\boldsymbol{A})$ and $q_k(\boldsymbol{A})$ are Hermitian. The matrix $q_k(\boldsymbol{A})$ is just a linear combination with real coefficients of the matrices $r_1(\boldsymbol{A}),\ldots,r_k(\boldsymbol{A})$ and will also be Hermitian if the matrices $r_1(\boldsymbol{A}),\ldots,r_k(\boldsymbol{A})$ are Hermitian.

Let us now show by induction that the matrices $r_1(\boldsymbol{A}),\ldots,r_{k+1}(\boldsymbol{A})$ are Hermitian and thus that the matrix $\boldsymbol{H}_{k+2,k+1}$ is real. The first two basis vectors and the first column in the matrix $\boldsymbol{H}$ in the rational Krylov algorithm are the same as in the shift-and-invert Arnoldi algorithm for the matrix $(\boldsymbol{A}-\mu_1\boldsymbol{I})^{-1}$. This means that the elements $h_{1,1}$ and $h_{2,1}$ are real and that

$$\boldsymbol{v}_1 = r_1(\boldsymbol{A})\boldsymbol{v}_1 = \boldsymbol{I}\boldsymbol{v}_1.$$

It is clear that $r_1(\boldsymbol{A})$ is Hermitian.

If we now assume that the matrices $r_1(\boldsymbol{A}),\ldots,r_k(\boldsymbol{A})$ are Hermitian and thus the matrix $\boldsymbol{H}_{k+1,k}$ is real, we will prove that the matrix $r_{k+1}(\boldsymbol{A})$ will be Hermitian and thus the matrix $\boldsymbol{H}_{k+2,k+1}$ will be real. The vector $\boldsymbol{v}_{k+1}$ can be expressed as

$$\begin{aligned} \boldsymbol{v}_{k+1} &= ((\boldsymbol{A}-\mu_{i_k}\boldsymbol{I})^{-1}\boldsymbol{V}_k\boldsymbol{t}_k - \sum_{l=1}^{l=k} h_{l,k}\boldsymbol{v}_l)/h_{k+1,k} \\ &= ((\boldsymbol{A}-\mu_{i_k}\boldsymbol{I})^{-1}q_k(\boldsymbol{A})\boldsymbol{v}_1 - \sum_{l=1}^{l=k} h_{l,k}r_l(\boldsymbol{A})\boldsymbol{v}_1)/h_{k+1,k} \\ &= r_{k+1}(\boldsymbol{A})\boldsymbol{v}_1. \end{aligned}$$

Hence the matrix $r_{k+1}(\boldsymbol{A})$ will be Hermitian because the matrices $(\boldsymbol{A}-\mu_i\boldsymbol{I})^{-1}$, $q_k(\boldsymbol{A})$ and $r_1(\boldsymbol{A}),\ldots,r_k(\boldsymbol{A})$ are Hermitian and the elements $h_{l,k}$, $l=1,\ldots,k+1$ are real. As a consequence the matrix $\boldsymbol{H}_{k+2,k+1}$ will be real.

4.3 Related Work

The biorthogonal rational Lanczos algorithm is similar in behaviour, regarding the nonzero structure for general matrices, to the rational Krylov algorithm for Hermitian matrices; see the paper by Gallivan, Grimme and Van Dooren [3]. The nonzero structure discussed here is also related to the case where the DIOM (FOM with incomplete orthogonalisation) generates orthonormal basis vectors; see page 186 in the book by Saad [11].

5 Implementation

5.1 Introduction

The parallel programs described in this section are intended for a multiple instruction multiple data (MIMD) computer with distributed memory or a cluster of workstations connected through a network.

In exact arithmetic, given the same conditions, the different programs discussed below give rational Krylov factorisations (5) that are identical with the factorisation generated by the parallel rational Krylov algorithm discussed in section 3. The programs are implemented in Fortran and use PVM (Parallel Virtual Machine) for communications between the different processors.

5.2 Parallel Program 1

This program has been implemented in a Master/Slave way and uses $\bar{p}+1$ processors. The Master program does all program control and all orthogonalisation. The Slave programs do all operations $\boldsymbol{r}_p := (\boldsymbol{A}-\mu_p\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{r}_p, p = 1,\ldots,\bar{p}$; for details see [12].

The main problem with the Master/Slave implementation is that the slave processors become idle while orthogonalisation is done in the Master program. The time the Master processor spends in each orthogonalisation grows linearly with k. The Master processor is idle while the slaves do the factorisation. We will comment more on this in the tests.

The advantage of this implementation is that the communication grows linearly with the number of processors and the total communication time is smaller than in the second implementation. This will be an advantage on a slow network. If the operations $\boldsymbol{r}_p := (\boldsymbol{A}-\mu_p\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{r}_p, p = 1,\ldots,\bar{p}$ take relatively long time, so that the Master processor has time to orthogonalise while the slaves work, this implementation will serve relatively well.

5.3 Parallel Programs 2A and 2B

The second implementation can be described as same program multiple data (SPMD). The same programs are started on $\bar{p}$ different processors. Processor number p, $1 \le p \le \bar{p}$ operates $\boldsymbol{r}_p := (\boldsymbol{A}-\mu_p\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{r}_p$ and orthogonalises $\boldsymbol{r}_p$ against $\boldsymbol{v}_l, l = 1,\ldots,k$. For this to be possible, the processors exchange the orthogonalised vectors with each other, and each processor keeps a copy of the basis vectors. The main idea is described for the parallel rational Krylov algorithm in section 3. We have implemented two versions, which differ in how the communication is done.

5.3.1 Parallel Program 2A

Parallel program 2A uses blocking `receive` and `send`. A rearrangement of the `receive` and `send` part of the parallel rational Krylov algorithm is needed.

.
.
.

12 **receive** $\boldsymbol{v}_l$ and $\boldsymbol{h}_{l-1}$ from its processor, $l = k-p+2:k$
(from processor $1,\ldots,p-1$)
13 $\boldsymbol{H}(k_p+1:k,k) = \boldsymbol{V}_{k_p+1:k}^H \boldsymbol{r}_{k+1}$ (Gram-Schmidt part II)
14 $\boldsymbol{r}_{k+1} = \boldsymbol{r}_{k+1} - \boldsymbol{V}_{k_p+1:k}\boldsymbol{H}(k_p+1:k,k)$
15 $h_{k+1,k} = \| \boldsymbol{r}_{k+1} \|$
16 $\boldsymbol{v}_{k+1} = \boldsymbol{r}_{k+1}/h_{k+1,k}$ (deliver new vector)
17 ($\boldsymbol{v}_{k+1}$ will be the start vector $\boldsymbol{v}_{k_p}$ in the next iteration on line 9)
18 **send** $\boldsymbol{v}_{k+1}$ and $\boldsymbol{h}_k$ to the other processors
19 **receive** $\boldsymbol{v}_l$ and $\boldsymbol{h}_{l-1}$ from its processor, $l = k+2:k+(\bar{p}-p)+1$
(from processor $p+1,\ldots,\bar{p}$)
.
.
.

For a full program description see [12].

Every processor sends and receives $\bar{p}-1$ vectors of length n. A particular processor waits while the others send and receive. This means that every processor slows down by the system communication time,

$$t_c = C\bar{p}(\bar{p}-1)n + L\bar{p}(\bar{p}-1)$$

where C is a constant which depends on the network and on what precision is used in the program, L is the latency in the network, and n is the dimension of the matrix $\boldsymbol{A}$.

Besides the communication time, every processor has to wait while the others do the second orthogonalisation, implying a total delay

$$t_{ortho} = (1+2+\ldots+\bar{p}-1)D$$

where D is the time it takes to orthogonalise against one vector of length n.

This implementation works well up to around 6 processors on the IBM-SP2; see the test results. The volume of communication grows quadratically with the number of processors $\bar{p}$ and hence this algorithm will never work on a large number of processors.

5.3.2 Parallel Program 2B

The parallel program 2B is a straightforward implementation of the parallel rational Krylov algorithm described in section 3 (no rearrangement is needed). It uses a blocking `receive` and a non-blocking `send`.

The volume of communication for each processor grows linearly with the number of processors $\bar{p}$, but the total communication grows quadratically.

Every processor sends and receives $\bar{p}-1$ vectors of length n. So every processor slows by with the communication time

$$t_c = C(\bar{p}-1)n + L(\bar{p}-1),$$

as long as the system can absorb it. On the other hand, the system has a communication time of

$$t_c = C\bar{p}(\bar{p}-1)n + L\bar{p}(\bar{p}-1).$$

The idea behind this version is to receive the data first when you need it, and to work while the other processors communicate.

The process of sending a message with PVM is composed of (1) initiating a sending buffer, (2) packing data into the buffer and (3) sending the buffer. An arriving message is first stored into a temporary buffer and is later unpacked by the program that receives the message.

That the sending process is blocking means that it returns first when the receiving process has unpacked the data. In order for the parallel program 2B to work properly the `send` operation should be non-blocking, so that the sending process can do useful work and not have to wait until the receiving process unpacks the data. We did not manage to get this version to operate on the IBM-SP2, probably due to a bug in the PVM implementation (PVMe) at that time.

5.4 Implementation Details

5.4.1 Solvers

We have used direct band solvers from LAPACK [1]. Other solvers are possible, such as general sparse solvers. However, there is a problem with using general sparse solvers. The factorisation part of many general sparse solvers can be split into analysing the nonzero structure and numerical factorisation. The analysing phase needs to be done only once for factorisation of different matrices with the same structure, like those we have in this report ($\boldsymbol{A}-\mu_i\boldsymbol{B}$). The analysing phase can take, for example, 3 to 10 times as long as the numerical factorisation. In order for the algorithm to give good speedup results when using direct general sparse solvers, the analysing phase needs to be done in parallel on all processors, and then be distributed to the different processors.

5.4.2 Subroutines in Linear Algebra

It is a nontrivial task to write efficient and accurate subroutines for solving equations and doing vector and matrix operations. These subroutines are needed in order to implement the parallel rational Krylov algorithm. Whenever appropriate, we have used LAPACK and BLAS routines; see [1].

The BLAS routines are a collection of subroutines for basic matrix operations such as matrix-matrix multiplications, matrix-vector multiplications and scalar products. Many computer manufacturers have their own optimised BLAS routines. The speed of these optimised routines can be significantly higher than matrix operations implemented in the naive way, because modern computers use memory with different access time in a hierarchy.

The LAPACK routines are a collection of linear algebra routines such as linear system of equations solvers and eigenvalue solvers. LAPACK uses the BLAS routines. The main work in this implementation is done in the factorisation, solving and multiplication routines; see Table 1 for the specific routines used.

5.4.3 Orthogonalisation

To ensure that the basis is orthonormal to working precision, every vector $\boldsymbol{r}_k$ is orthogonalised twice against the basis.

Subroutines in Linear Algebra			
	double complex	double precision	library
matrix-vector product	zgemv	dgemv	BLAS
scalar product	zdotc	ddot	BLAS
norm	dznrm2		BLAS
scaling	zdscal	dscal	BLAS
general band factorise	zgbtrf	dgbtrf	LAPACK
general band solve	zgbtrs	dgbtrs	LAPACK
generalised eigenvalue solver	zgegv	dgegv	LAPACK

Table 1: The linear algebra subroutines from BLAS and LAPACK used in the test programs.

5.4.4 Program Code

The code is experimental and is only intended for research purposes.

The code for the IBM-SP2 and the cluster of SUN workstations is basically the same. The different versions used by the different computer configurations are made by the preprocessor cpp from a generic program.

On the cluster of SUN workstations, the programs are written in Fortran77 together with the libraries BLAS, LAPACK and PVM. The nonstandard Fortran77 statements `malloc` and `pointer` are used for memory management.

On the IBM-SP2 the xlf Fortran compiler is used. The xlf compiler provides maximum compatibility with existing Fortran77 programs and uses many features in Fortran90. The Fortran90 feature `allocate` is used for memory management in the code. The libraries used are LAPACK, BLAS from the ESSL library and PVMe, the IBM version of PVM.

All programs have a double complex version. Program 1 is the only program with a double precision version.

MATLAB is used to process the data from the tests and to draw the graphs. A semi-parallel code for convergence tests is implemented in MATLAB.

6 Test Results

6.1 Introduction

The main aspects tested are efficiency, speedup and convergence. Efficiency and speedup are related to the speed performance of the algorithm. In section 3.1 of this report we predicted that it should not matter in which order the matrix operators $(\boldsymbol{A} - \mu_k \boldsymbol{B})^{-1}\boldsymbol{B}$, $k = 1, \ldots, \bar{p}$ are applied; we should get the same approximation if we do it in parallel or sequentially, and thus we should get similar convergence behaviour. Is this true on a computer with floating-point arithmetic? We will discuss this issue in subsection 6.8. Most tests and all timing tests in this section are taken from the Licentiate Thesis [12].

6.2 Test Sites

The IBM-SP2 was chosen because it was the best available MIMD computer in Sweden for research purposes. The SUN configuration was chosen because it was available at the Department.

The IBM-SP2 is a distributed memory MIMD computer. The machine used at the Center for Parallel Computers, Royal Institute of Technology, Stockholm, Sweden, had 55 nodes (the configuration is now upgraded). Each node contains a processor and memory. The nodes share data via message passing over a high-performance switch. There are two types of nodes, thin nodes with 128Mbyte memory and wide nodes with 512Mbyte memory. We have only used thin nodes in our tests.

The SUN configuration contains SUN ELC workstations connected through an Ethernet network.

In Table 2 we give some information about the nodes and network used. In Table 2 we give peak performance, and in Table 3 we give measured performance. For the test of the communication bandwidth, we have used PVM on the SUN workstations and PVMe on IBM-SP2 (PVMe is the IBM version of PVM). The data transferred in the tests are double precision vectors of length 10000. The number of flops is measured for a scalar product (ddot in the BLAS).

	IBM-SP2	SUN
Processor	RS/6000	SPARC
Architecture	POWER2	SPARC
Clock frequency	66.7MHz	33MHz
Floating point operations per cycle	4	
Peak performance	266Mflops	10Mflops
Main memory	128Mbyte	24Mbyte
Data cache	64Kbyte	
Instruction cache	32Kbyte	
Bandwidth network	35Mbyte/s	1Mbyte/s
Latency	40μs	2ms

Table 2: Some performance data about the nodes and network used.

	IBM-SP2	SUN
Performance	23Mflops	2.4Mflops
Bandwidth network	13.3Mbyte/s	0.27Mbyte/s

Table 3: measured performance. The number of flops is measured for a scalar product.

6.3 Test Problem

The test matrices consist of a finite difference discretisation of the differential equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{qy}{h}\frac{\partial u}{\partial y} = \lambda u.$$

The test problem is artificial but related to a convection diffusion problem; it is intended for testing convergence and efficiency. This problem has been chosen for the following reasons: it is easy to vary the size of the problem, the eigenvalues are known, and the eigenvalues can be arbitrarily ill or well conditioned. The matrix $\boldsymbol{A}$, generated from the discretisation, is a sparse matrix with 5 diagonals and the matrix $\boldsymbol{B}$ is the identity matrix. The matrix $\boldsymbol{A}$ is real and non-symmetric. Some facts about the test problems for the different configurations are given in Table 4. The eigenvalue distribution and the nonzero structure for the test problem for the IBM-SP2 are given in Figures 1 and 2 respectively. The larger test problem chosen for the IBM-SP2 would not fit on the cluster of SUN workstations due to their small memory. The bandwidth is chosen in such a way that the same time ratio for orthogonalising and solving the linear system of equations is obtained for both test sites.

	IBM-SP2	SUN
Upper and lower bandwidth	100	100
Dimension	10000	1000
Number of nonzero diagonals	5	5
Amount of memory for the LU factor	48M byte	4.8M byte
Amount of memory for the basis (m=150)	24M byte	2.4M byte
Precision	double complex	double complex

Table 4: Facts about the test problems for the different configurations.

6.4 Tests of the Sequential Algorithm

The tests of the sequential algorithm are made in order to obtain data to compare the sequential and parallel algorithms.

6.4.1 Shifts

We use every shift with the same number of iterations in each test; this is because we want to compare these tests with the tests on the parallel algorithms where it is impractical to have different numbers of iterations with different shifts.

6.4.2 Timing

We have kept time on the different parts of the algorithm and the results are given in Table 5. In these tests we have not timed the computation of the approximate solution step. The algorithms can roughly be divided into factorisation $\boldsymbol{LU} = (\boldsymbol{A} - \mu\boldsymbol{B})$, multiplication $\boldsymbol{r} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{B}\boldsymbol{v}$, and orthogonalisation.

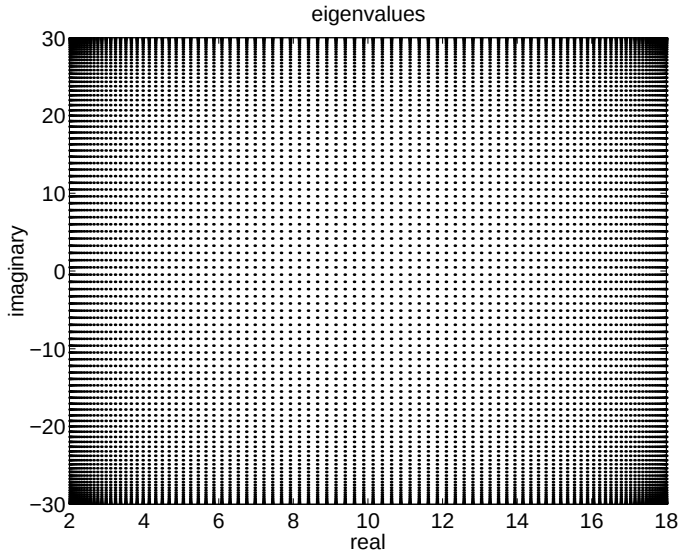


Figure 1: IBM-SP2 exact eigenvalue distribution.

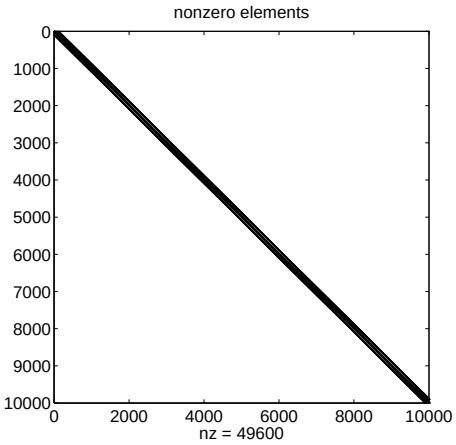


Figure 2: IBM-SP2, the nonzero structure.

The time it takes to orthogonalise a vector against the basis grows linearly with the number of iterations.

Test Results				
Configuration	IBM-SP2	IBM-SP2	SUN	SUN
Test No.	1	2	3	4
Shifts	6	3	6	3
Iterations with each shift	25	30	25	30
Total no. of iterations	150	90	150	90
Timing Results (in seconds)				
One factorisation	16.6	17.0	22.8	24.3
One multiplication	0.483	0.485	0.656	0.648
Factorisation total	99.6	51.0	136.8	73.0
Multiplication total	72.45	43.65	98.4	58.3
Orthogonalisation total	103.5	38.42	113	40.9
Other	0	0	3.6	3.6
Total program	275.6	133.1	351.8	175.8

Table 5: Test results for the sequential algorithm.

6.4.3 Convergence

The convergence is measured by the norm of the residual see (9). In Figure 3 the norms of the residuals are plotted in each iteration with a plus $+$. Lines are drawn between residuals corresponding to the same eigenvalue in different iterations. The plot is for test number 1.

In Figure 4 the exact eigenvalues are plotted with a dot $\cdot$, the approximate eigenvalues with a residual less than 10^{-5} are plotted with circle $\circ$ and the shifts are plotted with a plus $+$. The plot is for test number 1.

6.5 Parallel Program 1

How well this implementation works depends on how much time it spends in orthogonalisation, compared to factorisation and multiplication. The problem with this algorithm is that the processors become idle. The communication time is negligible in the test compared to the idle time.

The advantage of this implementation is that the basis is located on only one processor and the slaves only have to keep the matrices and the factorisation. The communication grows linearly with the number of processors, so the total communication time is short compared to the other parallel algorithms. The test results are given in Table 6.

In Figure 5 we show idle time, working time and communication for the SUN configuration. For this particular test we have used 15 iterations ($3 \cdot 15 = 45$ total iterations), 3 shifts, 1 master and 3 slaves. The white area in the bars corresponds to idle time and the black to working time. The lines between the bars symbolise communication. In the first iteration the master is idle while the slaves do the factorisations; only part of the factorisation time is

Test Results Parallel Program 1		
Configuration	IBM-SP2	SUN
Test No.	5	6
Slaves	3	3
Master	1	1
Shifts	3	3
Iterations with each shift	30	30
Total no. of iterations	90	90
Timing Results On One Slave (in seconds)		
One factorisation	17.0	24.2
One multiplication	0.485	0.66
Factorisation total	17.0	24.2
Multiplication total	14.5	19.8
Communication estimated	0.72	3.6
Idle time	29.18	27.7
Total time	61.4	75.3
Timing Results On Master (in seconds)		
Orthogonalisation total	38.4	39.3
Communication estimated	2.16	10.8
Idle time	20.84	25.2
Total time	61.4	75.3
Timing Results Total (in seconds)		
Parallel Program	61.4	75.3
Sequential Program	133	176
Speedup	2.2	2.3
Efficiency	54%	58%

Table 6: Test results for parallel program 1.

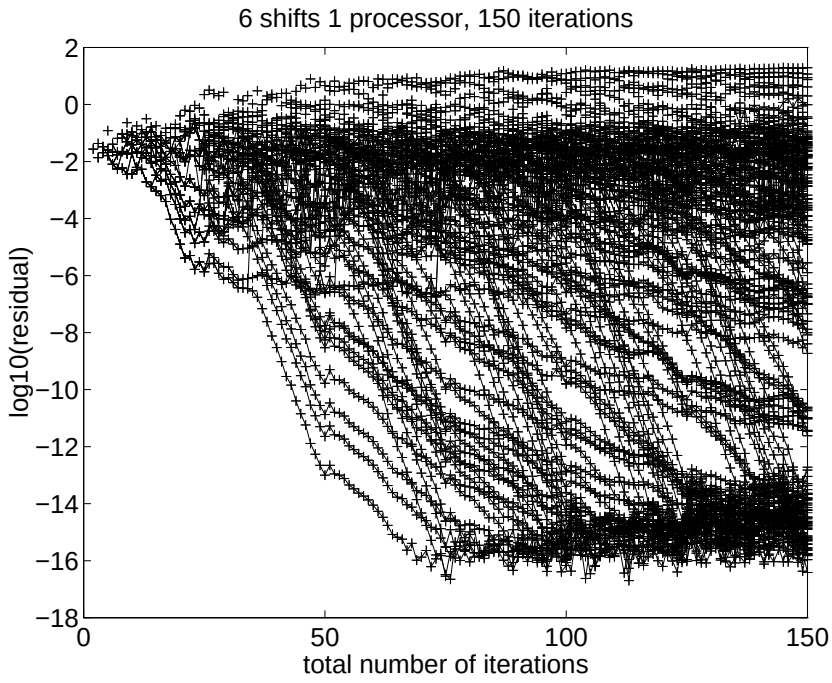


Figure 3: IBM-SP2, Convergence plot for test 1.

shown in the figure. In the second iteration, a particular slave is idle while the master orthogonalises its vector, and the master is idle while it is waiting for the next slave. After a number of iterations the master has no idle time. Orthogonalisation grows linearly with the number of iterations, and so do the slaves' idle time. The speedup and efficiency will depend on the total number of iterations. In Figure 6 we show the average efficiency in each iteration for test 5.

The tests show low efficiency for this algorithm. The algorithm can work rather well if the time spent in factorisation and multiplication is large compared to the orthogonalisation time. If the opposite is true, this algorithm will achieve poor results.

6.6 Parallel Program 2A

This algorithm gives much better performance than the previous one. The algorithm communicates more and the idle time is considerably less. It was the idle time that gave the previous algorithm poor performance. The test results are given in Tables 7 and 8.

In Figure 7 we show speedup from the tests using 6 shifts on the IBM-SP2 configuration. The straight line is the ideal speedup; the pluses + are the speedup from the tests on parallel program 2A.

In Figure 8 we show idle time, working time and communication for the SUN configuration. First every processor factorises; only part of the factorisation time is shown in the figure. Then at each iteration each processor multiplies, orthogonalises its own vector and exchanges the vector with the other processors.

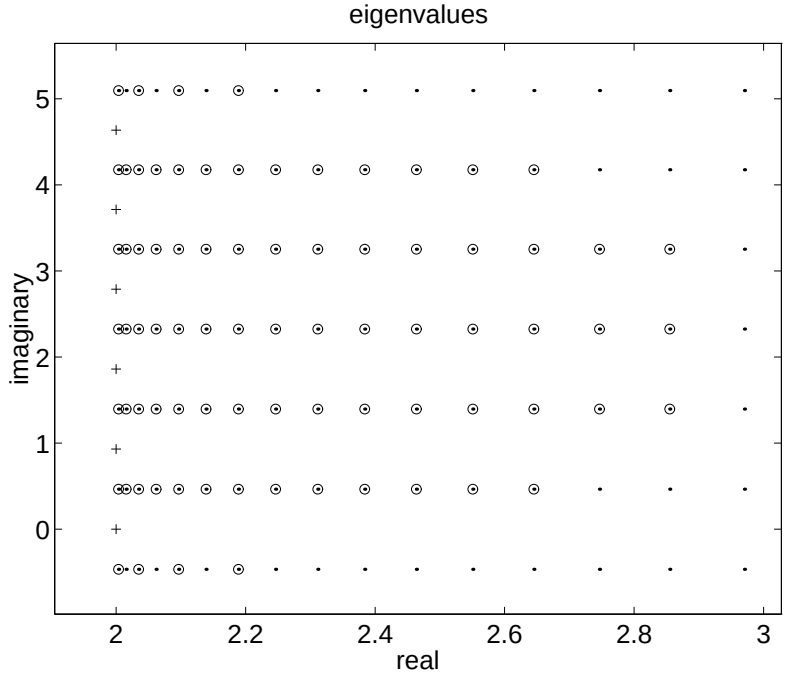
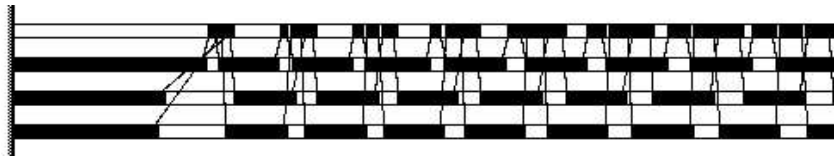


Figure 4: IBM-SP2, Converged eigenvalue distribution for test 1. Exact eigenvalues are plotted with a dot ·, approximate eigenvalues with a circle ∘ and the shifts are plotted with a plus +.

Test Results Parallel Program 2A				
Configuration	IBM	IBM	IBM	IBM
Test No.	7	8	9	10
No. of processors	3	6	3	2
Shifts	3	6	6	6
Iterations with each shift	30	25	25	25
Total no. of iterations	90	150	150	150
Timing Results On One Processor (in seconds)				
One factorisation	18.6	18.65	17.44	17.05
One multiplication	0.4933	0.4908	0.486	0.485
Factorisation total	18.6	18.65	34.88	51.15
Multiplication total	14.8	12.27	24.3	36.37
Orthogonalisation total	12.4	16.8	34.41	51.55
Communication + idle time	3.43	11.07	5.48	3.56
Total time	49.23	58.79	99.07	142.6
Sequential Program	133	275.6	275.6	275.6
Speedup	2.7	4.7	2.8	1.9
Efficiency	90%	78%	93%	97%

Table 7: Test results for parallel program 2A.

The first iterations



The last iterations

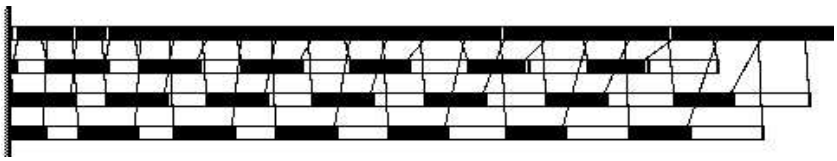


Figure 5: Time graphs on the SUN configuration, 15 parallel iterations. One master processor and three slave processors. The black area in the bars corresponds to working time, and the white area to idle time.

Every processor has to wait while the others communicate.

In Figure 9 we show the average efficiency at each iteration for test 7. In the first iteration most time is spent in factorisation, then in the second iteration the efficiency drops. In the following iterations, the efficiency grows since the orthogonalisation time grows in each iteration. Note that the scaling for the efficiency axis is different from the corresponding graph in the tests for parallel program 1.

The convergence is measured by the norm of the residual see (9). In Figure 10, the norms of the residuals for test 8 are plotted in each iteration with a plus +. Lines are drawn between residuals corresponding to the same eigenvalue in different iterations. Note that the residuals come down together because we use different shifts on different processors at the same time, compared to the sequential algorithm where they come down at each shift. The levels of the residuals are not so good as in the sequential algorithm. The parallel algorithms have some difficulties with the numerical computations that the sequential algorithm does not have. We will discuss the difference in numerical computation between the parallel and sequential algorithms in subsection 6.8.

This algorithm gives much better results than the previous one, even though the total communication time is longer here. The next program obtains even better results.

6.7 Parallel Program 2B

The difference between the parallel program 2A and 2B is how the vectors are exchanged. In 2A the processors waited for each other while the data were exchanged. In 2B an arriving vector is stored in a temporary buffer and the processor unpacks it first when it is needed. The communication measured in Table 9 is the sending and receiving operations. However, there is hidden communication; when a vector arrives at a processor it is stored in a temporary buffer, and this takes processor time. The multiplication and orthogonalisation seem to take longer time as the operations are measured by a real clock. This algorithm did not work on the IBM-SP2, apparently due to a bug in the PVMe,

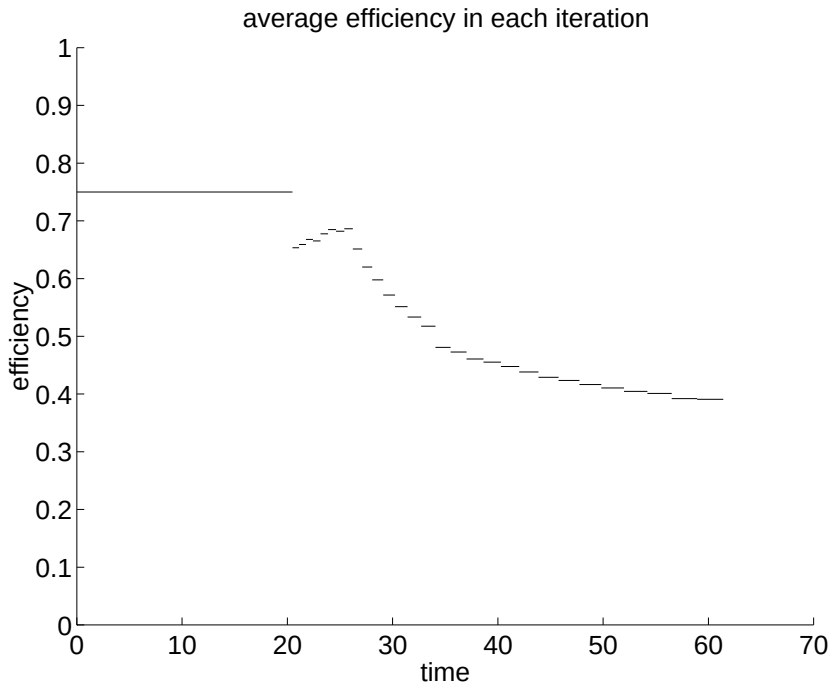


Figure 6: The average efficiency in each iteration for test 5.

Test Results Parallel Program 2A				
Configuration	SUN	SUN	SUN	SUN
Test No.	11	12	13	14
No. of processors	3	6	3	2
Shifts	3	6	6	6
Iterations with each shift	30	25	25	25
Total no. of iterations	90	150	150	150
Timing Results On One Processor (in seconds)				
One factorisation	26.4	24.3	24.2	22.8
One multiplication	0.675	0.658	0.661	0.662
Factorisation total	26.4	24.3	48.4	68.4
Multiplication total	20.3	16.5	33.1	49.7
Orthogonalisation total	13.1	17.8	36.6	54.8
Communication + idle time	10.9	33.2	18.9	16.7
Other	1.0	1.0	1.8	2.6
Total time	71.7	92.8	138.8	192.2
Sequential Program	176	351.8	351.8	351.8
Speedup	2.45	3.79	2.53	1.86
Efficiency	81.8%	63.2%	84.5%	92.8%

Table 8: Test results for parallel program 2A.

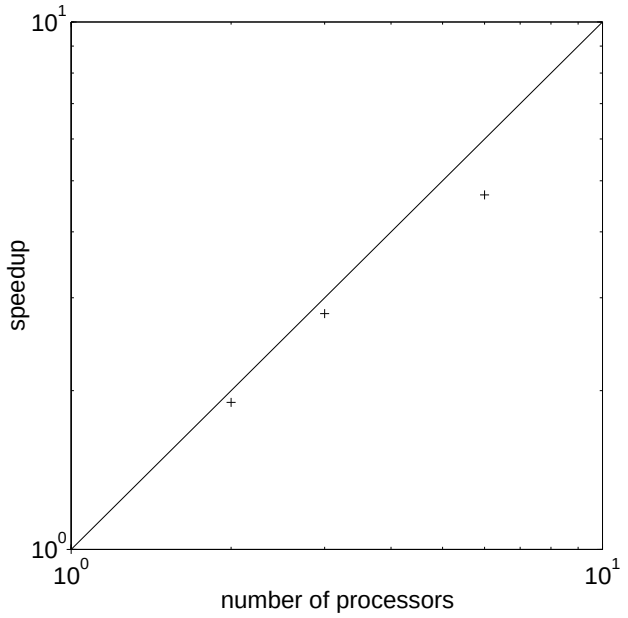


Figure 7: Speedup graph for the IBM-SP2 configuration. The pluses + are the speedup for parallel program 2A.

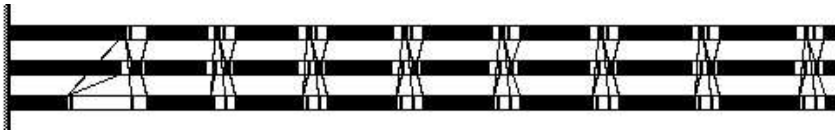


Figure 8: Time graphs on the SUN configuration for parallel program 2A. The black area corresponds to working time, and the white area to idle time.

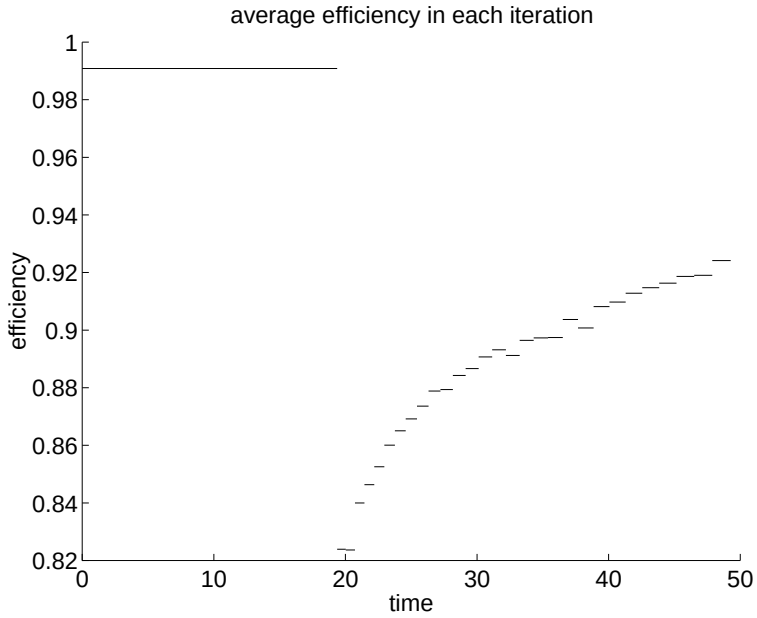


Figure 9: The average efficiency in each iteration for test 7.

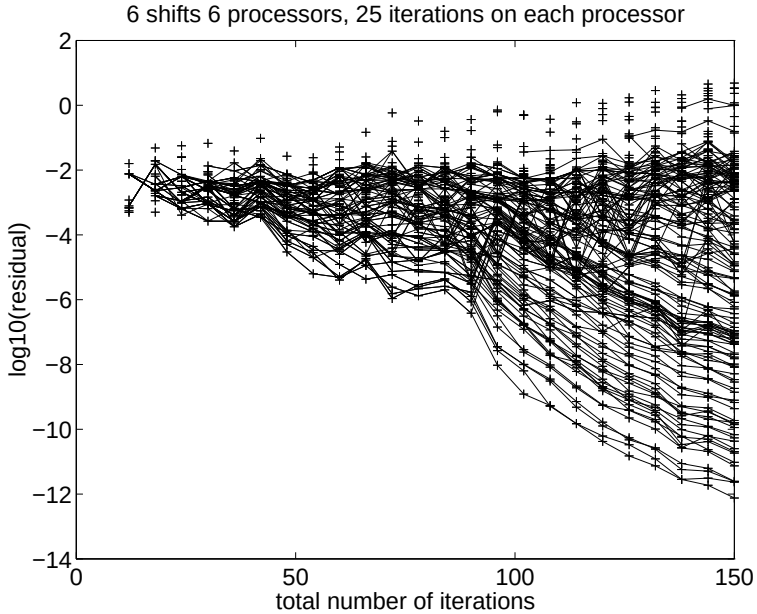


Figure 10: Convergence plot for test 8.

the implementation of PVM on the IBM-SP2. The test results for parallel program 2B are given in Table 9.

Test Results Parallel Program 2B				
Configuration	SUN	SUN	SUN	SUN
Test No.	11	12	13	14
No. of Processors	3	6	3	2
Shifts	3	6	6	6
Iterations with each shift	30	25	25	25
Total no. of iterations	90	150	150	150
Timing Results On One Processor (in seconds)				
One factorisation	23.6	23.8	22.7	23.1
One multiplication	0.73	0.808	0.712	0.691
Factorisation total	23.6	23.8	45.4	69.2
Multiplication total	22.1	20.2	35.6	51.8
Orthogonalisation total	13.1	19.4	37.2	55.2
Communication measured	3.1	5.2	5.6	6.7
Other	1.5	1.6	2.5	3.5
Total time	63.4	70.2	126.3	186.4
Sequential Program	175.8	351.8	351.8	351.8
Speedup	2.77	5.01	2.79	1.89
Efficiency	92.4%	83.5%	92.9%	94.4%

Table 9: Test results for parallel program 2B.

In Figure 11 we show speedup from the tests using 6 shifts on the SUN workstations. The straight line is the ideal speedup, the pluses + are the speedup from the tests on parallel program 2A, and the circles ∘ are the speedup from the tests on parallel program 2B.

In Figure 12 we show idle time, working time and communication for the SUN configuration. First every processor factorises; only part of the factorisation time is shown in the figure. Then at each iteration each processor multiplies, orthogonalises its own vector and exchanges the vector with the other processors. A particular processor unpacks a vector first when it is needed; the idle time is reduced by this procedure compared to the parallel program 2A.

In comparing the timing results with parallel program 2A, this implementation get considerably better results, due to the different way of exchanging the vectors as discussed before.

6.8 Convergence

In this section we will discuss the convergence of the algorithms. The parallel algorithms differ mainly in how communication is organised. The only numerical difference is in the orthogonalisation process, and these algorithms have almost the same numerical behaviour. The big difference is between the sequential algorithm and the parallel algorithms. Here we will discuss the numerical difference between the parallel algorithms and the sequential algorithm.

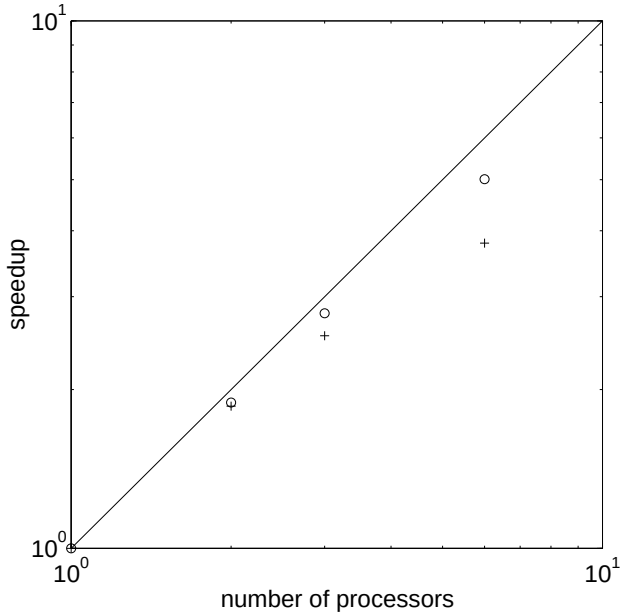


Figure 11: Speedup graph for the SUN configuration. The pluses + are the speedup for the parallel program 2A and the circles ∘ are the speedup for the parallel program 2B.

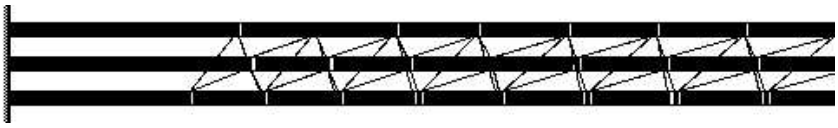


Figure 12: Time graphs on the SUN configuration for the parallel program 2B. Each bar corresponds to a processor. The black area in the bars corresponds to working time and the white area corresponds to idle time.

Tests		
Test No.	15	16
No. of processors	2	1
Shifts	2	2
Iterations with each shift	30	30
Total no. of Iterations	60	60
μ_1	100.5	100.5
μ_2	110.5	110.5

Table 10: Some facts for the convergence tests

6.8.1 Test Matrices

In these tests we have chosen $\boldsymbol{A} = \text{diag}(1:500)$, $\boldsymbol{B} = \boldsymbol{I}$.

6.8.2 Tests

The tests show that the matrices $\boldsymbol{K}$ and $\boldsymbol{H}$ are more unstable numerically in the parallel algorithm than in the sequential algorithm.

As the first two tests we choose 2 shifts, one sequential test and one parallel test. Some facts about the tests are given in Table 10.

In Figure 13 we show the normalised singular values for the matrices $\boldsymbol{H}$ and $\boldsymbol{K}$ as the number of iterations progresses for test 15, the parallel test. The corresponding sequential test, test 16, is given in Figure 15. The pluses + are the singular values of the matrix $\boldsymbol{H}$ and the circles $\circ$ are the singular values of the matrix $\boldsymbol{K}$.

The test shows that the matrices $\boldsymbol{H}$ and $\boldsymbol{K}$ get a null space of dimension one in the parallel test. A singular value decomposition of the matrices shows that they get the same null space in this case.

However, if we take into consideration the null space when the approximate eigenvalues are calculated, we get the same convergence for the parallel and the sequential case; see Figures 14 and 16 for convergence plots for the parallel and sequential tests respectively.

From the convergence graphs for the sequential and parallel algorithms, we can conclude that we get almost the same convergence. The null space in the $\boldsymbol{H}$ and $\boldsymbol{K}$ matrices in the parallel algorithm does not affect the convergence if dealt with properly in this case. However, the null space could be a source of trouble and generally the parallel rational Krylov programs get fewer converged eigenvalues than the corresponding sequential program. In general it seems that we get a null space of dimension $\bar{p}-1$ where $\bar{p}$ is the number of processors used. For more tests see [12].

6.8.3 Distance Between the Shifts

How does the distance between the shifts affect the singular values of the matrices if the number of iterations is kept constant? If the distance is zero, we get the same shifts and linear dependence between the basis vectors.

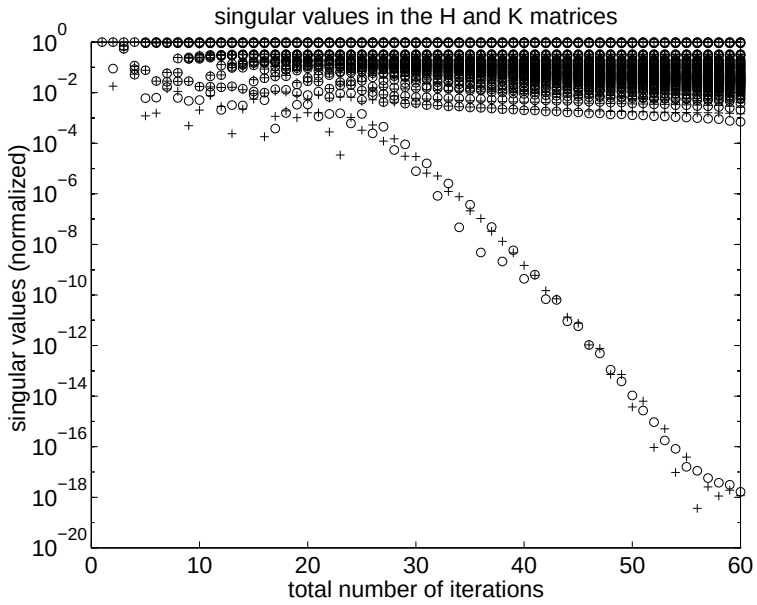


Figure 13: Normalised singular values for the matrices $\boldsymbol{H}$ and $\boldsymbol{K}$ for test 15, the parallel test with 2 shifts and 2 processors. The pluses + are the singular values of the matrix $\boldsymbol{H}$ and the circles ∘ are the singular values of the matrix $\boldsymbol{K}$.

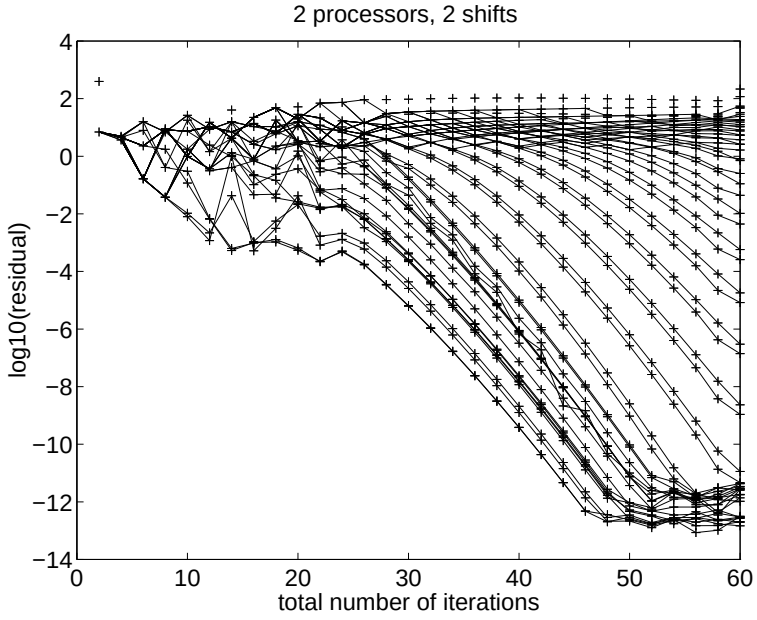


Figure 14: Convergence plot for test 15, the parallel test.

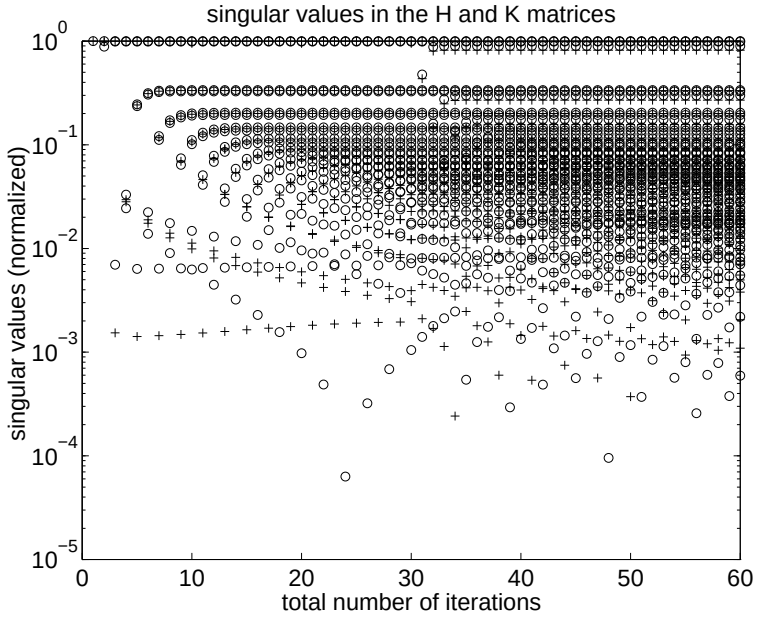


Figure 15: Normalised singular values for the matrices $\boldsymbol{H}$ and $\boldsymbol{K}$ for test 16, the sequential test with 2 shifts. The pluses + are the singular values of the matrix $\boldsymbol{H}$ and the circles ∘ are the singular values of the matrix $\boldsymbol{K}$.

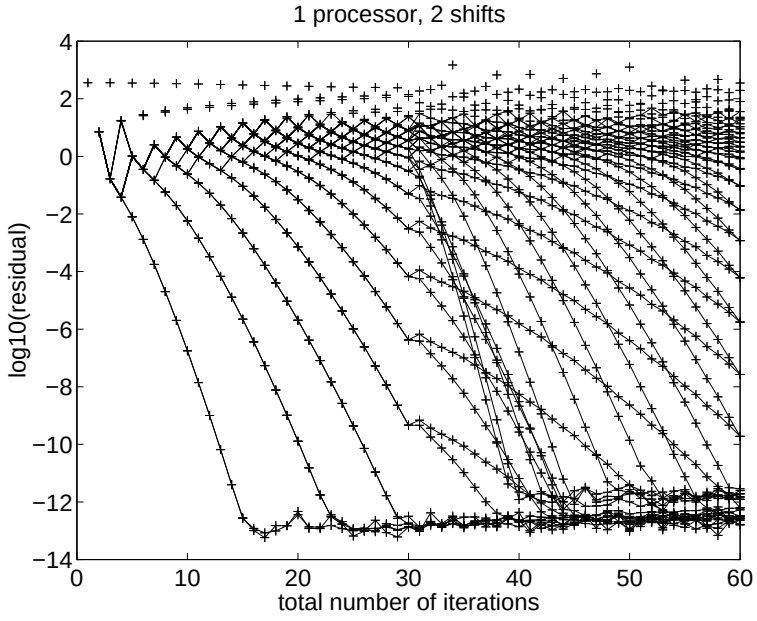


Figure 16: Convergence plot for test 16, the sequential test.

In Figure 17 we have tested how the singular values in the matrix $\boldsymbol{H}$ for the parallel algorithm depends on the distance between the shifts. The shifts are $\mu_1 = 100.5$ and $\mu_2 = 100.5 + 5 * j, j = 1, \ldots, 6$. The total number of iterations is 60, 30 iterations on each processor. As we can see in the figure, if we fix the number of iterations and increase the distance between the shifts, the smallest singular value of $\boldsymbol{H}$ increases in the parallel algorithm.

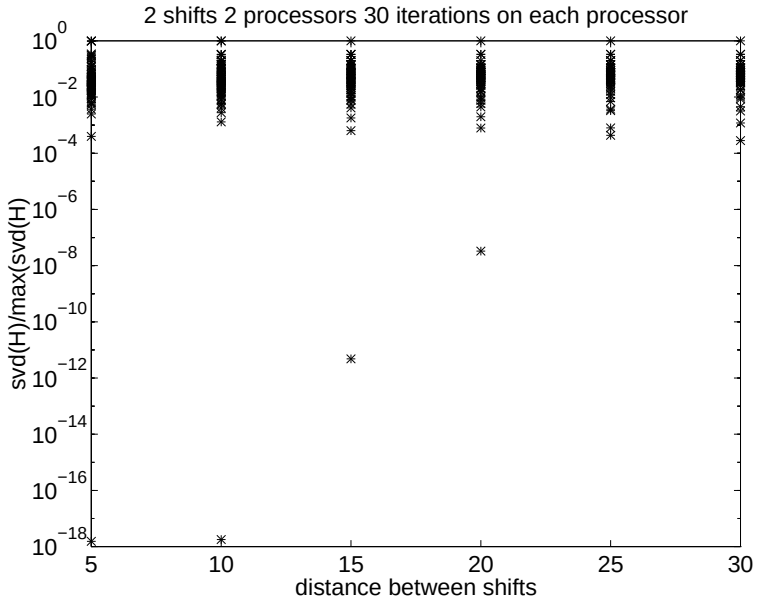


Figure 17: The smallest singular value as function of the distance between the shifts, for the parallel algorithm with 2 processors and 2 shifts.

Below we give a summary and conclusions for the convergence tests. There remains work to do before theorems and proofs can be stated from these conclusions.

1. If $\mu_1, \ldots, \mu_{\bar{p}}$, $\mu_i \neq \mu_j$ shifts are chosen and applied in parallel with the parallel rational Krylov method, with the same number of iterations with each shift, then the matrices $\boldsymbol{H}$ and $\boldsymbol{K}$ get $\bar{p} - 1$ singular values that are significantly smaller than the others. As the number of iterations increases, these singular values can be considered zero and the matrices $\boldsymbol{H}$ and $\boldsymbol{K}$ have the same null space of dimension $\bar{p} - 1$.

2. If $\mu_1, \ldots, \mu_{\bar{p}}$, $\mu_i \neq \mu_j$ shifts are chosen initially and applied in parallel with the parallel rational Krylov method, and the numbers of iterations are fixed and equal on each processor, then the matrices $\boldsymbol{H}$ and $\boldsymbol{K}$ will have $\bar{p} - 1$ singular values that are significantly smaller than the others when the shifts are close enough. A larger distance between the shifts makes the smallest singular value larger.

The null space appears gradually, and the above can only be a rough description of what actually happens with the parallel rational Krylov algorithm.

6.8.4 Some Explanations of the Convergence of the Parallel Algorithm

Why do the parallel and sequential algorithms behave differently? According to the theory, in exact arithmetic they build up the same subspace. Somehow the floating-point arithmetic makes them behave differently.

One explanation is in the way the basis is built up. In the parallel version, different Krylov spaces are intermixed to build up a larger space, while in the sequential version one Krylov space at a time is added to build up a larger space. If the shifts are the same, then the parallel version stops in the orthogonalisation step in the first iteration, because the different processors produce identical vectors. But for the sequential version, if the shifts are the same, the algorithm becomes the shifted and inverted Arnoldi algorithm and does not break down. What happens when the shifts are close but not the same? We have some idea from the tests.

Comparing Figures 13 and 14, we see a clear connection between the convergence of the eigenvalues and the gradual appearance of the null space in the $\boldsymbol{H}$ and $\boldsymbol{K}$ matrices.

Consider a parallel rational Krylov algorithm run on 2 processors and 2 shifts. In the first iterations, the convergence regions of the two shifts are separate and all singular values of the $\boldsymbol{H}$ and $\boldsymbol{K}$ matrices are at normal levels. When the convergence regions begin to grow together, there will be an eigenvector that converges in both regions before the other common eigenvectors. Except for the start vector $\boldsymbol{v}$, the Krylov spaces $\mathcal{K}_j((\boldsymbol{A}-\mu_1\boldsymbol{I})^{-1},\boldsymbol{v})$ and $\mathcal{K}_j((\boldsymbol{A}-\mu_2\boldsymbol{I})^{-1},\boldsymbol{v})$ will have a common vector, the eigenvector that has converged in both spaces. We could generalise this for $\bar{p}$ processors and $\bar{p}$ shifts.

Another explanation lies in the orthogonalisation process. The sequential algorithm builds up the basis one vector at a time, while the parallel version builds up the basis with $\bar{p}$ vectors at a time. The parallel version operates on each processor p, $1 \le p \le \bar{p}$:

$$\boldsymbol{r}_{k+1} = (\boldsymbol{A}-\mu_{i_k}\boldsymbol{B})^{-1}\boldsymbol{B}\boldsymbol{v}_{k-\bar{p}+1}$$

$\boldsymbol{r}_{k+1}$ is orthogonalised against $\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k$. The vectors $\boldsymbol{v}_{k-\bar{p}+2},\ldots,\boldsymbol{v}_k$ cannot be included in the the starting combinations because they are not calculated when the processor needs them. This process could cause numerical instability.

6.9 The Nonzero Structure in the $\boldsymbol{H}$-matrix

In these tests the matrix $\boldsymbol{A} = \mathrm{diag}(1:300)$, and the matrix $\boldsymbol{B}$ is the unit matrix. As long as $\mu_i \neq \mu_l$, $i \neq l$ it is not relevant what the shifts are.

The graph in Figure 18 shows the nonzero structure for the $\boldsymbol{H}$-matrix. We used three processors ($\bar{p}=3$), one shift per processor ($\bar{m}=1$), and each shift was applied ten times ($\bar{j}=10$). The maximum number of nonzero elements in each column (and row) is $2\bar{p}+1 = 2\times 3+1 = 7$ as predicted.

The graph in Figure 19 shows the nonzero structure for the $\boldsymbol{H}$-matrix when we used three processors ($\bar{p}=3$), two shifts per processor ($\bar{m}=2$), and each shift was applied five times ($\bar{j}=5$). Note how the number of nonzero elements in each column changes when the shifts change; the change occurs after column $\bar{p}\bar{j} = 3\times 5 = 15$. First we get $2\bar{p}$ columns with nonzero elements from the

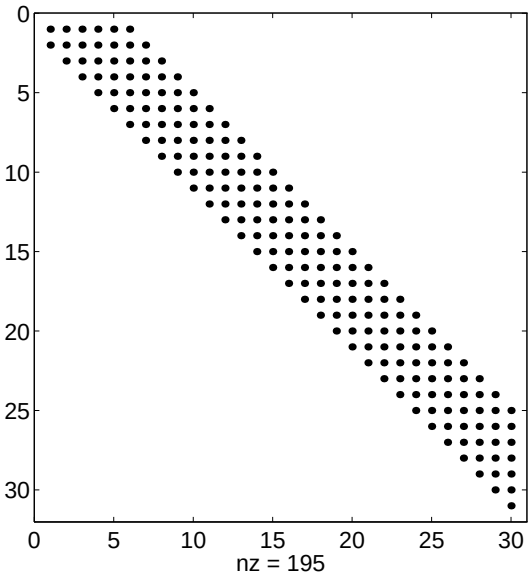


Figure 18: The nonzero structure for the $\boldsymbol{H}$-matrix. We used 3 processors, 1 shift per processor and each shift was applied 10 times.

sub-diagonal down to the bottom; after that, the nonzero structure is as before the change of shifts.

There exist many other ways to implement a parallel rational Krylov algorithm that generates the basis vectors in parallel. The nonzero structure of the $\boldsymbol{H}$-matrix will depend on the shift strategy, the number of processors and how the vector $\boldsymbol{v}_{k_p}$ is chosen.

6.10 The Rational Krylov Method Compared to the Shift-and-Invert Arnoldi Method

In this section we compare the rational Krylov method with the shift-and-invert Arnoldi method. In these tests, we have used a diagonal matrix $\boldsymbol{A}$ and unit $\boldsymbol{B}$ with eigenvalues in the complex integers.

In Figure 20 we show the converged eigenvalues for the parallel rational Krylov algorithm with a total of 160 iterations. Approximate eigenvalues with a relative error between 10^{-5} and 10^{-8} are plotted with a circle $\circ$. Approximate eigenvalues with a relative error between 10^{-8} and 10^{-11} are plotted with a plus $+$. Approximate eigenvalues with a relative error below 10^{-11} are plotted with a cross $\times$. The shifts are plotted with a star $*$. The exact eigenvalues are plotted with a dot $\cdot$.

One advantage with the (parallel) rational Krylov method is that the shifts can be chosen to mark up other regions than circles; see Figure 20. The corresponding convergence test for the 5 shift-and-invert Arnoldi methods is shown in Figure 21. The (parallel) rational Krylov method builds up one subspace of dimension 160, while the 5 shift-and-invert Arnoldi methods build up 5 different subspaces, each of dimension $\frac{160}{5} = 32$.

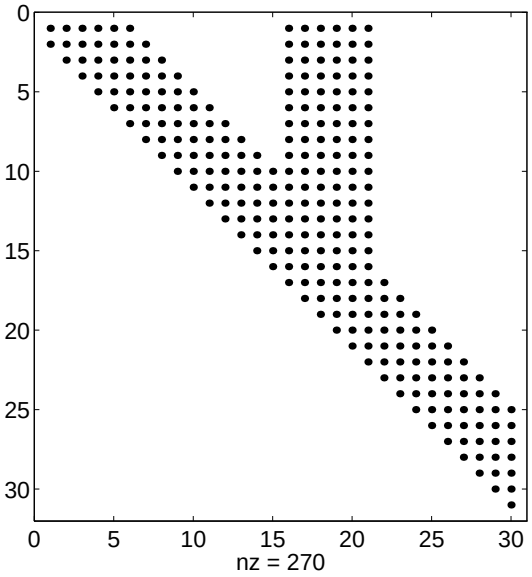


Figure 19: The nonzero structure for the $\boldsymbol{H}$-matrix. We used 3 processors, 2 shifts per processor and each shift was applied 5 times.

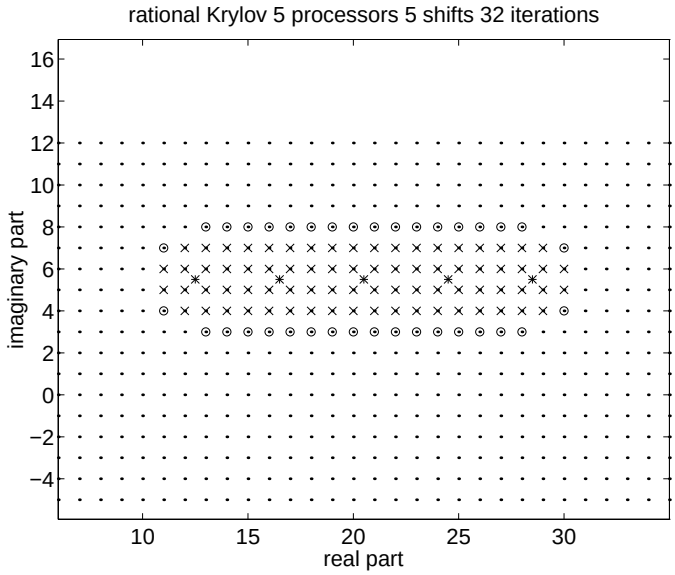


Figure 20: Convergence region for parallel rational Krylov algorithm.

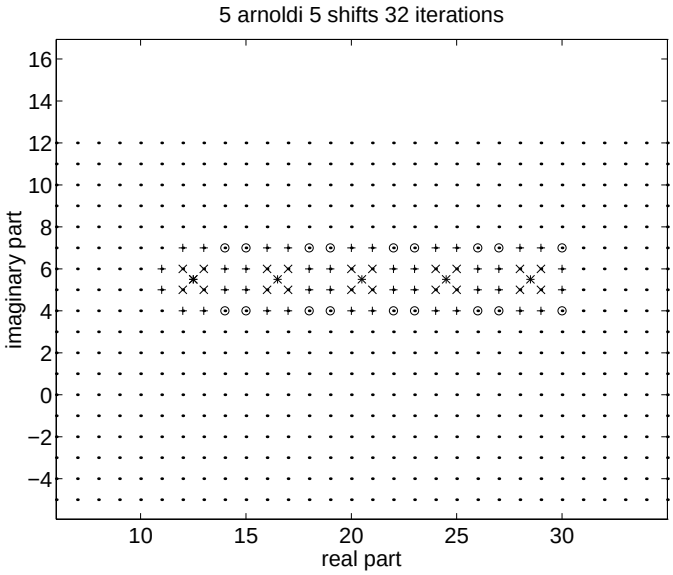


Figure 21: Convergence region for 5 separate Arnoldi methods.

The type of problem where the rational Krylov method is likely to perform better than a standard shift-and-invert Arnoldi is where the desired eigenvalue region differs to a great extent from the typical circular convergence region of the shift-and-invert Arnoldi. For more tests see [12].

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide, Release 2.0.* SIAM, Philadelphia, 1995. 324 pages.

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

[3] K. Gallivan, E. Grimme, and P. Van Dooren. A rational Lanczos algorithm for model reduction. *Numerical Algorithms*, 12:33–64, 1996.

[4] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research, Nat. Bur. of Standards*, 45:255–282, 1950.

[5] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK USERS GUIDE: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods.* SIAM, Phildelphia, PA, 1998.

[6] K. J. Maschhoff and D. C. Sorensen. P_ARPACK: an efficient portable large scale eigenvalue package for distributed memory parallel architectures. In *Applied Parallel Computing in Industrial Problems and Optimization,* J.

Wasniewski, J. Dongarra, K. Madsen, and D. Olesen, eds., Volume 1184 of *Lecture Notes in Computer Science,*. Springer-Verlag, 1996.

[7] Axel Ruhe. Rational Krylov sequence methods for eigenvalue computation. *Lin. Alg. Appl.*, 58:391–405, 1984.

[8] Axel Ruhe. Rational Krylov algorithms for nonsymmetric eigenvalue problems, II: Matrix pairs. *Lin. Alg. Appl.*, 197/198:283–296, 1994.

[9] Axel Ruhe. The Rational Krylov algorithm for nonsymmetric eigenvalue problems. III: Complex shifts for real matrices. *BIT*, 34:165–176, 1994.

[10] Axel Ruhe. Rational Krylov, a practical algorithm for large sparse nonsymmetric matrix pencils. *SIAM J. Sci. Comp.*, 19:1535–1551, 1998.

[11] Y. Saad. *Iterative Methods For Sparse Linear Systems.* PWS Publishing Co., 1996.

[12] Daniel Skoogh. An implementation of a parallel rational Krylov algorithm. Licentiate Thesis, Chalmers University of Technology, Göteborg, Sweden, 1996.