UNIVERSITÀ DEGLI STUDI DI PISA

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di laurea magistrale in Matematica

Tesi di Laurea

Rational Krylov methods for linear and nonlinear eigenvalue problems

Relatore
Prof. Dario A. Bini

Candidato
Giampaolo Mele

Controrelatore
Prof. Luca Gemignani

ANNO ACCADEMICO 2012-2013

To my family

Contents

Chapter 1

Introduction

One of the most active research fields of numerical analysis and scientific computation is concerned with the development and analysis of algorithms for problems of big size. The process that leads to them starts with the discretization of a continuum problem. The more accurate the approximation of the continuum problem, the bigger the size of the discrete problem.

In this thesis modern algorithms to solve big sized eigenproblems are studied. They come from various fields as mathematics, informatics, engineering, scientific computing, etc. Usually not all the solutions are needed but just a well defined subset according to the specific application. The large size of the problem makes it impossible to use general algorithms that do not exploit other features of the involved matrices like sparsity or structure.

For the linear eigenproblem the most popular algorithm is Arnoldi iteration (and its variants). A modern algorithm is Rational Krylov algorithm. This algorithm can be adapted to deal with nonlinear eigenproblem.

The thesis is organized in the following way: In chapter 2 the Arnoldi and rational Krylov algorithms for the linear eigenproblem are presented. In chapter 3 the rational Krylov algorithm is applied/extended to solve the nonlinear eigenproblem. In chapter 4 a few applications, mainly from fluid dynamics, are presented and used to test the algorithms.

1.1 Notation

We denote with lowercase letters both vectors and numbers; the nature of a variable will be clear from the context. Capital letters are used for matrices. The subscripts denote the size, e.g., $H_{j+1,j}$ is a matrix with $j+1$ rows and j columns. In the cases where the number of rows is clear, the subscript denotes the number of columns e.g., V_j is a matrix with j columns. Given a vector v we denote with $v^{[i]}$ a sub–vector which usually forms a part in a block subdivision.

1.2 Basic definitions

In this thesis we use as synonymous the terms *eigenvalue problem* and *eigenproblem.* Consider an application

$$A(\cdot) : \mathbb{C} \to \mathbb{C}^{n\times n}$$

and a set $\Omega \subset \mathbb{C}$. Solving an eigenproblem means to find all pairs $(\lambda, x) \in \mathbb{C} \times \mathbb{C}^{n\times n}$ such that $A(\lambda)x = 0$. The pair (λ, x) is called *eigenpair.* In case $A(\lambda)$ is linear then we call the problem *linear eigenvalue problem* or *generalized eigenvalue problem* (GEP). We can express a linear eigenproblem as

$$Ax = \lambda Bx,$$

where $A, B \in \mathbb{C}^{n\times n}$. In case B is the identity matrix, we have a classic eigenproblem.

If the application $A(\lambda)$ is not linear then we call the problem *nonlinear eigenvalue problem* (NLEP). In case $A(\lambda)$ is polynomial, that is,

$$A(\lambda) = \sum_{i=0}^{N} \lambda^i A_i,$$

where $A_i \in \mathbb{C}^{n\times n}$, we call the problem *polynomial eigenvalue problem* (PEP). A linearization of a NLEP is a linear eigenproblem such that its eigenpairs in Ω are near in norm to the eigenpairs of the original NLEP.

We will often use an improper terminology regarding the convergence. Given a matrix $A \in \mathbb{C}^{n\times n}$, a small number *tol* and a finite sequence $\{\theta_i\}_{i=1}^{n}$, then we say that at step k the sequence "convergence" to an eigenvalue λ of A if $|\theta_k - \lambda| < tol$. This means that numerically θ_k is an eigenvalue of A. This terminology is not correct but is convenient to not have hard notation.

Chapter 2

Linear eigenvalue problem

In this chapter we describe the Arnoldi algorithm for the classical eigenvalue problem, see [15] or [19]. A non-Hermitian version of thick-restart [22] [23] will is developed. The generalized eigenvalue problem (GEP) is solved with Arnoldi algorithm and the shift-and-invert Arnoldi is introduced. Finally we present the Rational Krylov algorithm as formulated in [12].

2.1 Arnoldi algorithm

Let $A \in \mathbb{C}^{n \times n}$ be an $n \times n$ matrix of which we want to compute the eigenvalues. Let us start by introducing the concept of Krylov subspace.

Definition 2.1.1 (Krylov subspace)**.** *Given a vector x and $A \in \mathbb{C}^{n \times n}$ define the* Krylov subspace *as*

$$K_m(A, x) := span\left\{x, Ax, A^2x, \dots, A^{m-1}x\right\}.$$

Any vector that lies in such space can be written as $v = p(A)x$ where p is a polynomial of degree less then m.

The idea of the Arnoldi algorithm is to approximate eigenvectors of the matrix A with vectors of Krylov subspace. In order to do that, a suitable orthonormal basis of such space is constructed.

Let us define $v_1 := x/\|x\|$, inductively we orthogonalize Av_j with respect to the vectors previously computed by using the Gram–Schmidt algorithm and we arrive at the following equations

$$h_{j+1,j}v_{j+1} := Av_j - h_{1,j}v_1 - h_{2,j}v_2 - \cdots - h_{j,j}v_j, \qquad j = 1, \dots, m.$$

In compact way, set $V_k := [v_1|v_2| \dots |v_k]$, and find that

$$AV_m = V_{m+1}H_{m+1,m},$$

or equivalently

$$AV_m = V_m H_{m,m} + h_{m+1,m}v_{m+1}e_m^T.$$

By construction the matrix $H_{j+1,j}$ is in Hessenberg form.

Remark 2.1.1. *The vectors v_i are an orthonormal basis of the Krylov subspace, the matrix $H_{m+1,m}$ is in Hessenberg form, the coefficients $h_{i,j}$ are obtained from the orthogonalization process while $h_{j+1,j}$ from the normalization. Therefore, we may choose $h_{j+1,j}$ positive numbers.*

Definition 2.1.2 (Arnoldi's sequence)**.** *A sequence of normal vectors $v_1, \dots, v_m$ is called Arnoldi's sequence if it exists a Hessenberg matrix $H_{m,m+1}$ with positive elements in the subdiagonal such that*

$$AV_m = V_{m+1} H_{m+1,m}.$$

It is clear how to continue the Arnoldi sequence; if we have $v_1, \dots, v_{m+1}$ then

$$\begin{cases} w = Av_{m+1} - h_{1,m+1}v_1 - h_{2,m+1}v_2 - \cdots - h_{m+1,m+1}v_{m+1} & \text{orthogonalization,} \\ h_{m+2,m+1} = \|w\| \\ v_{m+2} = w/h_{m+2,m+1} & \text{normalization} \end{cases}$$

Now we will show that the Arnoldi sequence is deeply connected with the Krylov subspace.

Proposition 2.1.1. *If the matrices V_{m+1} and W_{m+1} are orthonormal, the first column of both is v_1 and there exist two Hessenberg matrices $H_{m+1,m}$ and $K_{m+1,m}$ with positive subdiagonal entries such that*

$$AV_m = V_{m+1} H_{m+1,m},$$
$$AW_m = W_{m+1} K_{m+1,m}.$$

Then $V_{m+1} = W_{m+1}$ and $H_{m+1,m} = K_{m+1,m}$.

Proof. Let us prove the claim by induction. If $m = 1$ there is nothing to prove, then $v_1 = w_1$. Let us prove the case $m = 2$ in order to understand the idea. We have the two equations

$$h_{2,1}v_2 = Av_1 - h_{1,1}v_1,$$
$$k_{2,1}w_2 = Av_1 - k_{1,1}v_1.$$

By using the orthonormality, multiplying by v_1 (that is equal to w_1) both equations, we find

$$0 = v_1^H Av_1 - h_{1,1},$$
$$0 = v_1^H Av_1 - k_{1,1}.$$

Then we have $h_{1,1} = k_{1,1}$, now coming back at the original equation,

$$h_{2,1}v_2 = Av_1 - h_{1,1}v_1 = Av_1 - k_{1,1}v_1 = k_{2,1}w_2.$$

Then we have $h_{2,1}v_2 = k_{2,1}w_2$, but by assumption we have $h_{2,1}, k_{2,1} > 0$ and v_2, w_2 are normal, so it holds that $h_{2,1} = k_{2,1}$ and $v_2 = w_2$. Let us suppose the thesis true for m, so $V_m = W_m$ and $H_{m,m-1} = K_{m,m-1}$. Then by assumption we have the last equations

$$\begin{aligned} h_{m+1,m}v_{m+1} &= Av_m - h_{1,m}v_1 - \cdots - h_{m,m}v_m, \\ k_{m+1,m}w_{m+1} &= Aw_m - k_{1,m}w_1 - \cdots - k_{m,m}w_m. \end{aligned}$$

Since $v_i = w_i$ for $i \leq m$, we can rewrite the equations as

$$\begin{aligned} h_{m+1,m}v_{m+1} &= Av_m - h_{1,m}v_1 - \cdots - h_{m,m}v_m, \\ k_{m+1,m}w_{m+1} &= Av_m - k_{1,m}v_1 - \cdots - k_{m,m}v_m. \end{aligned}$$

By using the orthonormality, multiplying both equations by v_i for $i \leq m$ (that is equal to w_i) we find that

$$\begin{aligned} 0 &= v_i^H Av_m - h_{i,m}v_i, \\ 0 &= v_i^H Av_m - k_{i,m}v_i. \end{aligned}$$

Then we have $h_{i,m} = k_{i,m}$. Replacing the latter equation in the former we have $h_{m+1.m}v_{m+1} = k_{m+1,m}w_{m+1}$. Since $h_{m+1.m}$, $k_{m+1.m} > 0$ and v_{m+1}, w_{m+1} are unitary we find that $h_{m+1.m} = k_{m+1,m}$ and $v_{m+1} = w_{m+1}$. □

Remark 2.1.2. *Any Arnoldi sequence $AV_m = V_{m+1}H_{m+1,m}$ is generated in a unique way by the Gram–Schmidt process starting from the first column vector of V_{m+1}. So the columns of V_{m+1} generates the Krylov subspace $K_m(A, v_1)$.*

Observation 2.1.1. If in the construction of the Arnoldi sequence it holds that $h_{j+1,j} = 0$ for some j, then we cannot continue the sequence. However, looking at the equation

$$h_{j+1,j}v_{j+1} := Av_j - h_{1,j}v_1 - h_{2,j}v_2 - \cdots - h_{j,j}v_j,$$

we find that the vectors $v_1, \ldots, v_j$, and then $x, Ax, \ldots, A^{j-1}x$, are linearly dependent.

Definition 2.1.3. *Let $AV_m = V_{m+1}H_{m+1,m}$ be an Arnoldi sequence, if $H_{m,m}y = \theta y$ then θ is called Ritz value and $V_m y$ Ritz vector. The pairs formed by a Ritz value and a Ritz vector are called Ritz pairs.*

Proposition 2.1.2. *Let $AV_m = V_{m+1}H_{m+1,m}$ be an Arnoldi sequence and $(\theta, V_m y)$ a Ritz pair, then it holds that*

$$AV_m y - \theta V_m y = h_{m+1,m}e_m^H y \; v_{m+1}.$$

Proof. It is possible to write the Arnoldi sequence as

$$AV_m = V_m H_{m,m} + h_{m+1,m} v_{m+1} e_m^H.$$

By using this formula we find that

$$\begin{aligned} AV_m y &= (V_m H_{m,m} + h_{m+1,m} v_{m+1} e_m^H) y, \\ &= \theta V_m y + h_{m+1,m} (e_m^H y) v_{m+1}. \end{aligned}$$

This completes the proof. □

In general we can define $\omega_m(y) := h_{m+1} |e_m^H y|$ and say that if this number is small enough then $V_m y$ is a good approximation of an eigenvector of A. It would be desirable that also θ be a good approximation of an eigenvalue of A. To this regard it is useful to recall a well know theorem.

Theorem 2.1.1 (Bauer–Fike). *Let $A \in \mathbb{C}^{n \times n}$ be a diagonalizable matrix and $V \in \mathbb{C}^{n \times n}$ the eigenvectors matrix. If $(\widetilde{\lambda}, \widetilde{v})$ is an approximate eigenpair, let us define the residual $r := A\widetilde{v} - \widetilde{\lambda}\widetilde{v}$, then it exists an eigenpair (λ, v) of A such that*

$$\begin{aligned} |\lambda - \widetilde{\lambda}| &\le K_p(V) \frac{\|r\|_p}{\|\widetilde{v}\|_p}, \\ \frac{\|v - \widetilde{v}\|_p}{\|\widetilde{v}\|_p} &\le \|r\|_p \|(A - \widetilde{\lambda} I)^+\|_p [1 + K_p(V)], \end{aligned}$$

where $K_p(V) = \|V\|_p \|V^{-1}\|_p$ is the condition number.

Applying this theorem, if $(\theta, V_m y)$ is a Ritz pair, then it exists an eigenpair (λ, v) of A such that

$$\begin{aligned} &|\lambda - \theta| \le K_2(V) \omega_m(y), \\ &\|v - V_m y\|_2 \le \|(A - \widetilde{\lambda} I)^+\|_2 [1 + K_2(V)] \omega_m(y). \end{aligned}$$

Remark 2.1.3. *If $(\theta, V_m y)$ is a Ritz pair and $z = V_m y$, then it exists an eigenpair (θ, v) such that the following inequalities hold*

$$\begin{aligned} &\|Az - \theta z\|_2 \le \omega_m(y), \\ &|\lambda - \theta| \le K_2(V) \omega_m(y), \\ &\|z - v\|_2 \le \|(A - \widetilde{\lambda} I)^+\|_2 [1 + K_2(V)] \omega_m(y). \end{aligned}$$

Note that if A is a Hermitian matrix then $K_2(V) = 1$. Moreover from these inequalities we have that if the last component of y is zero then $(\theta, V_m y)$ is an eigenpair of A. If $h_{m+1,m}$ is zero then all the eigenvalues of $H_{m,m}$ are eigenvalues of A and all the Ritz vectors are eigenvectors of A.

Now we can summarize the Arnoldi process in algorithm 1.

In this algorithm we compute the Arnoldi sequence for m steps and whenever the residual is small enough we store the approximation to the eigenpair; moreover,

Algorithm 1 Arnoldi's algorithm

```
1:  Chose a vector $x$.
2:  Normalize $v_1 := x/\|x\|$.
3:  for $i = 1, \dots, m$ do
4:     $r = Av_i$.
5:     $h_i = V_i^H r$.
6:     $r = r - V_i h_i$.
7:     $h_{i+1,i} = \|r\|$.
8:     Compute the eigenpairs $(\theta_k, y_k)$ of $H_{i,i}$, for $k = 1, \dots, i$.
9:     if $h_{i+1,i} = 0$ then
10:       Store $(\theta_k, V_m y_k)$ as eigenpair of $A$, where $k = 1, \dots, i$.
11:       break
12:    else
13:       $v_{i+1} = r/h_{i+1,i}$.
14:    end if
15:    for $k = 1, \dots, i$ do
16:       if $\omega_i(y_k) <$ tol then
17:          Store $(\theta_k, V_m y_k)$ as approximation of an eigenpair of $A$.
18:       end if
19:    end for
20: end for
```

if $h_{i+1,i} = 0$ for some i, then i exact eigenpairs are delivered. We want to point out that if $h_{i,i+1} \neq 0$ for all the values of i up to n (size of the matrix A), then $h_{n+1,n}$ is zero. This way, exact eigenpairs are computed. Indeed, the advantage of this method relies on the fact that for small values of m a few eigenpair can be computed. In particular, if at some step v_i is an exact eigenvector of A then $h_{i+i,i}$ is zero and the algorithm breaks down. More specifically, if we choose x as eigenvector the algorithm stops right after one step.

If the matrix A is Hermitian then it is possible to show that $H_{m,m}$ is tridiagonal so we can use appropriate strategies for this case. For instance, we can use suitable and more effective algorithm to compute the eigenvalues of a tridiagonal matrix. In this case we call this process Lanczos algorithm [15].

We want to give an idea about convergence properties. Convergence analysis will be better developed in the next section.

The convergence of the algorithm depends on the magnitude of $h_{m+1,m}$ and this is related to the linear dependence of the vectors $v_1, \dots, v_m$ or equivalently of $x, Ax, \dots, A^{m-1}x$. If these vectors are linearly dependent then $h_{m+1,m} = 0$, if they are near to be independent (or are numerically dependent) then $|h_{m,m+1}|$ is small. But how many iteration do we need to perform in order that $|h_{m,m+1}|$ is small enough? Let us suppose A diagonalizable with $(\lambda_1, v_1), \dots, (\lambda_n, v_n)$ eigenpairs and let us suppose $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$. If $x = c_1 v_1 + \dots c_n v_n$ then

$$A^m x = \lambda_1^m c_1 v_1 + \lambda_2^m c_2 v_2 + \dots + \lambda_n^m c_n v_n,$$

$$A^m x = c_1 \lambda_1^m \left[v_1 + \frac{c_2}{c_1} \left(\frac{\lambda_2}{\lambda_1} \right)^m + \dots \frac{c_m}{c_1} \left(\frac{\lambda_m}{\lambda_1} \right)^m v_m \right].$$

We deduce that the sequence $A^m x$ converges to the dominant eigenvector with a rate $|\lambda_2/\lambda_1|$. Intuitively, if the value $|\lambda_2/\lambda_1|^m$ is close to zero then the vectors $A^m x$ and $A^{m+1}x$ are close to be linearly dependent and also $|h_{m+1,m}|$ is close to zero. Numerical convergence can occur even before, in fact, this computation shows the worst case.

We can have the idea that the number of iterations needed to approximate eigenvalues depends on the location of the eigenvalues and on the choice of the first vector x, in fact if we choose $c_1 = 0$ then $A^m x$ will converge to the second dominant eigenvector.

A practical issue is related to the orthogonality. In fact if m is not small, orthogonality will be numerically lost. In this case it can be needed to add a reorthogonalization process. To do that we have to add in algorithm 1 after the row 7 the rows of algorithm 2.

Algorithm 2 Reorthogonalization

```
  $c_i = V_i^H r.$
  $r = r - V_i c_i.$
  $c_{i+i,i} = 0.$
  $h_i = h_i + c_i.$
```

This will simply perform another time the Gram–Schmidt orthogonalization and in exact arithmetic the matrix C would be zero and nothing will change. It is important to point out that also with a reorthogonalization process when m is too big, orthogonality is numerically lost in any case. Moreover it is possible to prove that perform more than one reorthogonalization do not give advantages. The only way to solve this problem is to use a restart strategy as explained in the next sections.

In the next section we will answer to two important questions:

- How long have to be the Arnoldi sequence to have a good approximation of a few eigenvalues of A?
- Which eigenvalues of A will be estimated first?

2.2 Convergence of the Arnoldi algorithm

In this section we will explain the convergence behavior of the Arnoldi process. Till now we proved that, given an Arnoldi sequence, some Ritz values are near some eigenvalues for a certain m. The following theorem shows that, if the distance between the Krylov subspace and an eigenvector is small, then there is a Ritz vector near such eigenvector. To prove that, we need to recall some results.

Lemma 2.2.1. *Given u and v unitary vectors and $\theta(u,v)$ the angle between them, then it holds*

$$\|u - v\|^2 = 2\left[1 - \sqrt{1 - \sin^2(\theta(u,v))}\right].$$

This lemma tells us that if the angle between two unitary vectors becomes small then also the distance becomes small and conversely.

Lemma 2.2.2. *Let V be a finite $\mathbb{C}$-vector space and W a subspace. Set $P : V \to W$ be the orthogonal projection, u a vector and $\theta(u, W)$ the angle between u and W. Then*

$$\theta(u, W) = \theta(u, Pu).$$

Theorem 2.2.1 (Saad [15])**.** *Let P_m be the orthogonal projection into $K_m(A, x)$, $\gamma = \|P_m A(I - P_m)\|_2$, (λ, u) an eigenpair of A and $(\widetilde{\lambda}, \widetilde{u})$ the Ritz pair where $\widetilde{\lambda}$ is the Ritz value closer to λ, δ the distance between λ and the set of Ritz values other than $\widetilde{\lambda}$,. Then*

$$\sin(\theta(u, \widetilde{u})) \leq \sqrt{1 + \frac{\gamma^2}{\delta^2}} \sin(\theta(u, K_m(A, x))).$$

From theorem 2.2.1 and by using Lemma 2.2.2 we find that

$$\sin(\theta(u, \widetilde{u})) \leq \sqrt{1 + \frac{\gamma^2}{\delta^2}} \sin(\theta(u, P_m(u))).$$

It means that if the distance between u and Pu is small then the distance between u and $\widetilde{u}$ is small as well.

Notice that $\|u - P_m(u)\|$ is the distance between u and the Krylov subspace as the following theorem states.

Theorem 2.2.2 (Projection theorem)**.** *Let V be a finite $\mathbb{C}$-vector space and W a subspace, $P : V \to W$ the orthogonal projection. Then we have the following formula to compute the distance $d(u, W)$ between a vector $u \in V$ and the subspace W*

$$d(u, W) := \inf_{w \in W} \|u - w\| = \|u - Pu\|.$$

Remark 2.2.1. *It is important to point out that the Ritz vectors are not the best approximations in K_m to the corresponding eigenvectors. However, they are close to such best approximations. Moreover we show that if the distance between a particular eigenvector u and K_m is small then it exists a Ritz vector $\widetilde{u}$ near u.*

At this point we just need to examine $\|(I - P_m)u\|$ to show the convergence of the algorithm.

Theorem 2.2.3 (Saad [15])**.** *Let $A \in \mathbb{C}^{n \times n}$ be a diagonalizable matrix, $(\lambda_1, u_1), \ldots,$ (λ_n, u_n) the eigenpairs with u_i unitary, and $K_m(A, x)$ the Krylov subspace where to approximate eigenvectors. If $AV_m = V_{m+1} H_{m+1,m}$ is the Arnoldi sequence and*

$$v_1 = \sum_{k=1}^{n} \alpha_k u_k,$$

then

$$\|(I - P)u_1\|_2 \leq \xi_1 \epsilon^{(m)},$$

where

$$\xi_1 = \sum_{\substack{k=1\\k\neq i}}^{n} \frac{|\alpha_k|}{|\alpha_1|}, \qquad \epsilon_1^{(m)} = \left(\sum_{j=2}^{m+1} \prod_{\substack{k=2\\k\neq j}}^{m+1} \frac{|\lambda_k - \lambda_i|}{|\lambda_k - \lambda_j|} \right)^{-1}.$$

Looking at $\epsilon_1^{(m)}$ it can be seen that if λ_1 is in the outermost part of the spectrum, that is, $|\lambda_k - \lambda_1| > |\lambda_k - \lambda_j|$ for $j \neq 1$, then $\epsilon_1^{(m)}$ is small. We can conclude that generally eigenvalues in the outermost part of the spectrum are likely to be well approximated. Moreover the distance between u_1 and $K_m(A, x)$ depends also on the first vector of the Arnoldi sequence v_1. That is, if v_1 is already a good approximation of u_1 then less steps are needed to reach convergence.

Example 1. We can have experience of the previous theorem by using a simple program in matlab. Let us consider the diagonal matrix $D \in \mathbb{C}^{n\times n}$ with $D = \mathrm{diag}(n, n-1, \ldots, 1)$. Then the outermost eigenvalues are 1 and n, and $Ae_1 = ne_1$. Let us consider a unitary matrix Q –in matlab we can generate it as $[Q\ R] = \mathrm{qr}(\mathrm{rand}(n))$– and construct $A := Q^H D Q$. Then the dominant eigenvector of A is $Q^H e_1$. The goal of this example is to show that the closer x (the vector that generates the Krylov subspace) to $Q^H e_1$, the faster the convergence. So we set $n = 100$ and $x = Q^H e_1 + \varepsilon w$ where w is a unitary random vector. In the table in Figure 2.1 the value of $\omega_m(y)$ is reported where y is the dominant eigenvector of the matrix $H_{m,m}$.

m	$\varepsilon = 1$	$\varepsilon = 10^{-1}$	$\varepsilon = 10^{-2}$	$\varepsilon = 10^{-3}$	$\varepsilon = 10^{-4}$
1	28.4188	23.0878	3.0643	0.3087	0.0305
5	4.0535	0.9543	0.1251	0.0097	$9.0556e-04$
10	0.8929	0.1384	0.0179	0.0016	$1.5411e-04$
15	0.4577	0.0314	0.0061	$9.9966e-04$	$5.9527e-05$
20	0.1876	0.0174	0.0020	$3.3721e-04$	$1.9557e-05$
25	0.0436	0.0070	0.0010	$1.2662e-04$	$9.6067e-06$
30	0.0065	0.0019	$2.8252e-04$	$3.1552e-05$	$3.7785e-06$

Figure 2.1: Convergence to dominant eigenvalue with starting vector close to it

Example 2. Let us consider the matrix

$$A = \begin{pmatrix} 0 & & & 1 \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{pmatrix}$$

The eigenvalues of this matrix are the roots of the unity and all the eigenvalues are outermost eigenvalues, so that there are no eigenvalues that will be well approximated for m small. In fact, the Arnoldi algorithm is very slow in this case and the speed of convergence depends mainly on x. For a complete analysis of this case see

[15]. This one is also a pathological example. In fact, if n is the size of A and we start the Arnoldi sequence with e_1 then for every $m < n$ the Ritz values are just zero. We need exactly n steps to compute eigenvalues, so with a naive implementation of the algorithms we have an inefficient program.

Remark 2.2.2. *At a glance we have:*

- *For a moderate m the Ritz values approximate the outermost eigenvalues,*
- *If the eigenvalues are not clustered the algorithm is fast,*
- *The closer x to the eigenvector u_1 the faster the convergence to u_1 of the Ritz vector.*

Keeping in mind these three points, it is easier to understand the ideas behind the restart strategy that are presented in the next section.

2.3 Restart

Restarted Arnoldi

In practical applications it can happen that m becomes too big and the algorithm still does not provide a good approximation of eigenvalues.

There are several problems related to the fact that m becomes big, first of all the increase of the computational cost, then the loss of orthogonality and, because of errors, the computation of some possible spurious eigenvalues.

In order to solve this problem we can use the results of the previous section. We know that in general the Ritz pairs provide approximations to the outermost eigenpairs. Assume that our goal is to approximate the dominant eigenpair (eigenvalue with the largest norm and its eigenvector) and that after m steps there is still no convergence. This means that if $(\theta, V_m y)$ is the dominant Ritz pair, $\omega_m(y)$ is not small enough. We know that, even though $V_m y$ is not good enough, it is anyway an estimation of the dominant eigenvector. Therefore we can restart the algorithm with $x = V_m y$. In fact, we know that the convergence of the algorithm depends on the distance between x and the sought eigenvector and indeed $V_m y$ is closer than x to the dominant eigenvector.

We can summarize the Restarted Arnoldi in the following algorithm. 3.

Algorithm 3 Restarted Arnoldi's algorithm

```
1: Chose a vector $x$.
2: while $\omega(y) > tol$ (the dominant Ritz value did not converge) do
3:     Arnoldi($x, m$) performs $m$ steps of Arnoldi's sequence (Algorithm 1 ).
4:     $x = V_m y$ where $y$ is the dominant Ritz vector.
5:     $x = x/\|x\|$.
6: end while
```

If we want to approximate the first k dominant eigenvectors then it is possible to modify the previous algorithm. The idea is to do m steps of Arnoldi algorithm, compute the first k dominant Ritz vectors and put, for instance, $x = \sum_{i=1}^{k} V_m y_i$ or choose some other linear combination. We can summarize this in the algorithm 4, for a deeper discussion see [15].

Algorithm 4 Generalized restarted Arnoldi's algorithm

```
1: Chose a vector $x$
2: while $\omega(y) > tol$ (the dominant Ritz value did not converge) do
3:     Arnoldi($x, m$) performs $m$ steps of Arnoldi's sequence (algorithm 1 ).
4:     Choose $\alpha_1, \dots, \alpha_k$.
5:     $x = \sum_{i=1}^{k} \alpha_i V_m y_i$ where $y_i$ are the first $k$ dominant Ritz vectors.
6:     $x = x/\|x\|$.
7: end while
```

Note that we can decide to compute also other kind of eigenpairs, e.g., the eigenvalues with biggest real part, then repeat the same argument as above by replacing the words "dominant Ritz pairs" with " Ritz pairs with biggest real part". In fact we have shown in the previous section that, also if the outermost eigenvalues are well approximated for a moderate value of m, the convergence depends also from the distance of x from the desired eigenvector.

There are also other restart strategies. One of the most popular is the implicit restart. Here, we prefer to explore instead the thick restart. The idea is similar to the generalized restarted Arnoldi strategy, but relies on a deeper theoretical background and is well suited for the Rational Krylov algorithm.

Thick restart

In [22] [23] Kesheng and Horst presented the thick restart for the Lanczos algorithm and in [12] Ruhe suggested to use a non–Hermitian version of this restart also for the Arnoldi algorithm. To present this algorithm we need a few technical lemmas.

Preliminary results

Lemma 2.3.1 (Householder matrix). *Let $x \in \mathbb{C}^n$ then there exists a unitary matrix $P \in \mathbb{C}^{n \times n}$ such that $Px = \alpha e_1$, the matrix P is called Householder matrix.*

Proof. Let us define

$$\alpha = -e^{i \arg x(1)} \|x\|, \quad u = x - \alpha e_1, \quad v = \frac{u}{\|u\|}, \quad \beta = x^H v / v^H x.$$

Then let us set

$$P = I - (1+\beta) v v^H$$

Whit a direct computation we can check that this is the sought matrix. □

Lemma 2.3.2 (Hessenberg form). *Given a matrix $L \in \mathbb{C}^{(n+1) \times n}$ there exists a unitary matrix $P \in \mathbb{C}^{n \times n}$ such that*

$$L = \begin{pmatrix} 1 & 0 \\ 0 & P \end{pmatrix} H P^H.$$

where H is a Hessenberg matrix.

Proof. Let P_1 be the Householder matrix such that $P_1 L(2:n+1,1) = \alpha e_1$ where $L(2:n+1,1)$ is the first column of L from the second element to the last. Then it holds that

$$\begin{pmatrix} 1 & 0 \\ 0 & P_1 \end{pmatrix} L P_1^H = \begin{pmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ 0 & & & & & \\ \vdots & & & \hat{L} & & \\ 0 & & & & & \end{pmatrix}.$$

Again we can find a matrix P_2 such that $P_2 L(3:n+1,2) = \hat{\alpha} e_1$ and then

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & P_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & P_1 \end{pmatrix} L P_1^H \begin{pmatrix} 1 & 0 \\ 0 & P_2 \end{pmatrix}^H = \begin{pmatrix} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ 0 & * & * & * & * & * & * \\ 0 & 0 & & & & & \\ \vdots & \vdots & & \hat{\hat{L}} & & & \\ 0 & 0 & & & & & \end{pmatrix}.$$

Iterating this computation we have the thesis. □

In our computations it will be better to have a different factorization where, unlike in the previous lemma, 1 is at the bottom line. Later on we will explain why.

Lemma 2.3.3 (Hessenberg down-up form). *Given a matrix $L \in \mathbb{C}^{(n+1)\times n}$ there exists a unitary matrix $P \in \mathbb{C}^{n\times n}$ such that*

$$L = \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} H P^H.$$

Proof. Let us consider the matrix

$$\Pi = \begin{pmatrix} & & 1 \\ & \cdot^{\cdot^{\cdot}} & \\ 1 & & \end{pmatrix}.$$

It is easy to see that $\Pi^2 = I$, then by using lemma 2.3.2 we find that

$$(\Pi_{m+1} L \Pi_m)^H = \begin{pmatrix} 1 & 0 \\ 0 & P \end{pmatrix} H P^H,$$

then we have

$$\Pi_{m+1} L \Pi_m = \begin{pmatrix} 1 & 0 \\ 0 & P \end{pmatrix}^H H^H P.$$

So

$$L = \Pi_{m+1} \begin{pmatrix} 1 & 0 \\ 0 & P \end{pmatrix}^H H^H P \Pi_m$$

$$= \left[\Pi_{m+1} \begin{pmatrix} 1 & 0 \\ 0 & P \end{pmatrix}^H \Pi_{m+1} \right] \left[\Pi_{m+1} H^H \Pi_m \right] \left[\Pi_m P \Pi_m \right]$$
$$= \begin{pmatrix} \Pi_m P \Pi_m & 0 \\ 0 & 1 \end{pmatrix}^H \left[\Pi_{m+1} H^H \Pi_m \right] \left[\Pi_m P \Pi_m \right].$$

Note that $\Pi_m P \Pi_m$ is unitary and $\Pi_{m+1} H^H \Pi_m$ is Hessenberg. Setting

$$\widetilde{P} := (\Pi_m P \Pi_m)^H, \qquad \widetilde{H} := \Pi_{m+1} H^H \Pi_m,$$

we have

$$L = \begin{pmatrix} \widetilde{P} & 0 \\ 0 & 1 \end{pmatrix} \widetilde{H} \widetilde{P}^H$$

. □

Lemma 2.3.4. *Given a Hessenberg matrix $H \in \mathbb{C}^{(n+1) \times n}$ with real numbers in the subdiagonal, there exists a diagonal matrix $S \in \mathbb{R}^{n \times n}$ with elements* 1 *and* -1 *such that*

$$\begin{pmatrix} S & 0 \\ 0 & 1 \end{pmatrix} HS$$

has positive numbers in the subdiagonal.

Proof. Let us consider S_i the diagonal matrix with ones on the diagonal except in the position (i, i) where there is a -1. Then

$$\begin{pmatrix} S_i & 0 \\ 0 & 1 \end{pmatrix} HS_i$$

is a matrix equal to H where we have multiplied for -1 the i-th row and the i-th row. Then the idea is to change the sign of the subdiagonal of H starting from the bottom. If the last one element of the subdiagonal of H is negative then we perform the transformation with S_n, otherwise we check the previous element on the subdiagonal. Notice that if we perform the transformation with S_n we change the sign also to the element $h_{n,n-1}$. We can summarize the proof in Algorithm 5.

Algorithm 5 Hessenberg matrix with positive subdiagonal entries (real case)

```
1: for $i = n, \dots, 1$ do
2:     if $h_{i+1,i} < 0$ then
3:         $H := \begin{pmatrix} S_i & 0 \\ 0 & 1 \end{pmatrix} HS_i$.
4:         $S = SS_i$.
5:     end if
6: end for
```

□

Lemma 2.3.5. *Given a Hessenberg matrix $H \in \mathbb{C}^{(n+1)\times n}$, there exists a diagonal matrix $S \in \mathbb{C}^{n\times n}$ with unitary elements, called phase matrix, such that*

$$\begin{pmatrix} S & 0 \\ 0 & 1 \end{pmatrix} HS.$$

has positive numbers in the subdiagonal.

Proof. The idea is similar to the one used in Lemma 2.3.4. We first recall that given a complex number z there exists $\theta \in [0\ 2\pi]$ such that $e^{i\theta}z = |z|$. Now consider the diagonal matrix $S_i(\theta)$ with ones on the diagonal except in the position (i,i) where there is the element $e^{i\theta}$. Then, as in Lemma 2.3.4, starting from the bottom, we perform the transformation with $S_n(\theta)$

$$H := \begin{pmatrix} S_n(\theta) & 0 \\ 0 & 1 \end{pmatrix} HS_n(\theta).$$

After this operation $h_{n+1,n}$ is positive, then we repeat with the previous element in the subdiagonal. We can summarize the procedure in Algorithm 6. □

Algorithm 6 Hessenberg matrix with positive subdiagonal entries

```
1: for $i = n, \dots, 1$ do
2:     $\theta = -\arg(h_{i+i,i})$.
3:     $H := \begin{pmatrix} S_i(\theta) & 0 \\ 0 & 1 \end{pmatrix} HS_i(\theta)$.
4:     $S = SS_i(\theta)$.
5: end for
```

Remark 2.3.1. *Given a Hessenberg matrix $H \in \mathbb{C}^{n\times n}$ there exists a phase matrix S such that*

$$\begin{pmatrix} S & 0 \\ 0 & 1 \end{pmatrix} HS,$$

has positive elements in the subdiagonal.

By using the results of this section we arrive at the following result.

Theorem 2.3.1. *Given a matrix $L \in \mathbb{C}^{(n+1)\times n}$ there exists a unitary matrix $P \in \mathbb{C}^{n\times n}$ such that*

$$L = \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} HP^H,$$

and H is a Hessenberg matrix with positive elements on the subdiagonal.

Theorem 2.3.1 provides a matrix H with positive elements on the subdiagonal. This is a very important property since it enables us to interpret these elements as norms of vectors providing the connection with the Arnoldi sequence. This theorem concludes the part of preliminary results.

The thick restart algorithm

Let us consider the Arnoldi sequence $AV_m = V_{m+1}H_{m+1,m}$, suppose that m is too big and the Ritz values still do not reach convergence, so that a restart is needed. Assume we are interested in a few dominant eigenvalues (we can be interested also in some other kind of eigenvalues, say, eigenvalues with positive real part). Then we can choose k Ritz pairs to preserve and restart the algorithm with them. Let us consider the first k dominant Ritz pairs. The number k is called thickness.

Lemma 2.3.6. *Let $H_{m+1,m} \in \mathbb{C}^{(m+1)\times m}$ and $(\theta_1, y_1), \dots, (\theta_k, y_k)$ be a few eigenpairs of $H_{m,m}$ where y_i are normal. Then there exists a unitary matrix $P \in \mathbb{C}^{n\times n}$ such that*

$$H_{m+1,m} = \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \theta_1 & * & * & * & * & \dots & * \\ & \theta_2 & * & * & * & \dots & * \\ & & \ddots & * & * & \dots & * \\ & & & \theta_k & * & \dots & * \\ & & & & * & \dots & * \\ & & & & * & \dots & * \\ * & * & \dots & * & * & \dots & * \end{pmatrix} P^H.$$

Proof. Let P_1 be the unitary matrix such that $P_1 e_1 = y_1$, (it is simply the conjugate of the Householder matrix Q such that $Qy_1 = e_1$). Then, since the first row of P_1 is y_1 we have

$$\begin{pmatrix} P_1^H & 0 \\ 0 & 1 \end{pmatrix} H_{m+1,m} P_1 = \left(\begin{array}{c|cccccc} \theta_1 & * & * & * & * & * & * \\ 0 & & & & & & \\ \vdots & & & H_{m,m-1} & & & \\ 0 & & & & & & \\ * & & & & & & \end{array}\right).$$

Now we want to repeat the same computation with $H_{m,m-1}$. Note that we already know its eigenvalues, they are $(P_1 y_i)(2 : m+1)$ that is, the components from 2 to $m+1$ of the vector $P_1 y_i$. To simplify the notation let us call them $\hat{y}_i$. Let us consider the unitary matrix P_2 such that $P_2 e_1 = \hat{y}_2$. Then we have

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & P_2^H & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \theta_1 & * & * & * & * & * & * \\ 0 & & & & & & \\ \vdots & & & H_{m,m-1} & & & \\ 0 & & & & & & \\ * & & & & & & \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & P_2 \end{pmatrix} = \begin{pmatrix} \theta_1 & * & * & * & * & * & * & * \\ 0 & \theta_2 & & & & & & \\ \vdots & 0 & & & & & & \\ \vdots & \vdots & & & H_{m-1,m-2} & & & \\ 0 & 0 & & & & & & \\ * & * & & & & & & \end{pmatrix}.$$

We can iterate these computations an arrive at the thesis. $\square$

Let $AV_m = V_{m+1}H_{m+1,m}$ be the Arnoldi sequence and $(\theta_1, V_m y_1), \dots, (\theta_k, V_m y_k)$ the Ritz pairs that we want to preserve in order to improve the estimated eigenpair.

Then using lemma 2.3.6 we have

$$H_{m+1,m} = \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \theta_1 & * & * & * & * & \dots & * \\ & \theta_2 & * & * & * & \dots & * \\ & & \ddots & * & * & \dots & * \\ & & & \theta_k & * & \dots & * \\ & & & & * & \dots & * \\ & & & & * & \dots & * \\ * & * & \dots & * & * & \dots & * \end{pmatrix} P^H.$$

From the Arnoldi sequence we have

$$AV_mP = V_{m+1} \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \theta_1 & * & * & * & * & \dots & * \\ & \theta_2 & * & * & * & \dots & * \\ & & \ddots & * & * & \dots & * \\ & & & \theta_k & * & \dots & * \\ & & & & * & \dots & * \\ & & & & * & \dots & * \\ * & * & \dots & * & * & \dots & * \end{pmatrix}.$$

Setting

$$W_{m+1} := \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} V_{m+1},$$

we have

$$AW_m = W_{m+1} \begin{pmatrix} \theta_1 & * & * & * & * & \dots & * \\ & \theta_2 & * & * & * & \dots & * \\ & & \ddots & * & * & \dots & * \\ & & & \theta_k & * & \dots & * \\ & & & & * & \dots & * \\ & & & & * & \dots & * \\ * & * & \dots & * & * & \dots & * \end{pmatrix}.$$

Now let $\widetilde{W}_{k+1} := [w_1|\dots|w_k|w_{m+1}]$, it holds

$$A\widetilde{W}_k = \widetilde{W}_{k+1} \begin{pmatrix} \theta_1 & * & * & * \\ & \theta_2 & * & * \\ & & \ddots & * \\ & & & \theta_k \\ * & * & \dots & * \end{pmatrix}.$$

Now we use the theorem 2.3.1 then we have

$$A\widetilde{W}_kP = \widetilde{W}_{k+1} \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} \widetilde{H}_{k+1,k},$$

then setting

$$\widetilde{V}_{k+1} := \widetilde{W}_{k+1} \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix},$$

we have $A\widetilde{V}_k = \widetilde{V}_{k+1}\widetilde{H}_{k+1,k}$ and this is an Arnoldi sequence. Notice that the eigenvalues of $\widetilde{H}_{k,k}$ are $\theta_1, \dots, \theta_k$ that in our case are the dominant Ritz values. Then it is like if we start the Arnoldi sequence with the first k vectors near the first k dominant eigenvectors then the convergence will be faster.

We can summarize this process in the algorithm 7.

Remark 2.3.2. *The thick restart is a generalization of the classic restart strategy, such kind of restart is equivalent to start the Arnoldi sequence with the first k vectors near the k desired eigenvectors. Using the results of section 2.2 we have that the convergence after restart is faster than the convergence obtained by the customary restart with a random vector.*

Observation 2.3.1. We can change the thickness every time we restart the algorithm. The idea is that if we want for instance the eigenvalues with positive real part for every restart we can restart taking all the Ritz values near the real axis, then of course this number will not be fixed.

Observation 2.3.2. If we do a restart taking just k converged Ritz values then there are two possibilities: the algorithm stops with such converged Ritz values or the error grows. In fact the problem is that during restart we have to transform the following formula

$$A\widetilde{W}_k = \widetilde{W}_{k+1} \begin{pmatrix} \theta_1 & * & * & * \\ & \theta_2 & * & * \\ & & \ddots & * \\ & & & \theta_k \\ * & * & \dots & * \end{pmatrix}$$

in an Arnoldi sequence, but if the Ritz values are close to convergence then the elements in the last row are small. Then the operation that transforms such matrix in the Hessenberg form is ill–conditioned. In general, we have the same problem if the first Ritz value that we lock has converged. Then if we need to restart with k Ritz values, maybe some of them converged, then the idea is to sort such values so that θ_1 is far from convergence.

One can wonder why we need to restart with a converged Ritz value, the idea is to avoid that this Ritz value converges again after restart, this operation is called "lock" and it will be explained better later on.

Algorithm 7 Thick restarted Arnoldi algorithm

```
1: Chose a vector $x$ and set $k = 1$.
2: while $\omega(y_i) > tol$   (the desired Ritz value has not converged) do
3:     Starting from Arnoldi's sequence $AV_k = V_{k+1}H_{k+1,k}$ continue the sequence till
       $m$ (Algorithm 1 ).
4:     Choose $k \geq 1$ pairs $(\theta_1, V_m y_1), \dots, (\theta_k, V_m y_k)$ (Ritz pairs to preserve).
5:     Compute the factorization
```

$$H_{m+1,m} = \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \theta_1 & * & * & * & * & \dots & * \\ & \theta_2 & * & * & * & \dots & * \\ & & \ddots & * & * & \dots & * \\ & & & \theta_k & * & \dots & * \\ & & & & * & \dots & * \\ & & & & * & \dots & * \\ * & * & \dots & * & * & \dots & * \end{pmatrix} P^H.$$

```
6:     Set
```

$$\widetilde{W}_{k+1} := [Pv_1|\dots|Pv_k|v_{m+1}],$$
$$\widetilde{H}_{k+1,k} := \begin{pmatrix} \theta_1 & * & * & * \\ & \theta_2 & * & * \\ & & \ddots & * \\ & & & \theta_k \\ * & * & \dots & * \end{pmatrix}.$$

```
7:     Compute the Hessenberg form with positive subdiagonal entries
```

$$\widetilde{H}_{k+1,k} = \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} H_{k+1,k} P^H.$$

```
8:     Set
```

$$V_{k+1} = \begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix} \widetilde{W}_{k+1}.$$

```
9: end while
```

A connection between Arnoldi and the Power Method

A very well know method to estimate the dominant eigenvalue of a matrix A is the Power Method. Starting from a normal vector v_1 the Power Method generates a sequence $(h_{m+1,m}, v_m)$ where

$$h_{m+1,m} = \|Av_m\|, \tag{2.1}$$
$$v_{m+1} = Av_m / h_{m+1,m}. \tag{2.2}$$

It holds that $(h_{m+1,m}, v_m)$ converges to the dominant eigenpair of A under mild assumptions.

From (2.2) we have $Av_m = h_{m+1,m}$ that in vectorial form becomes

$$AV_m = V_{m+1}H_{m+1,m}.$$

This is not an Arnoldi sequence because V_m is not orthonormal. We can orthonormalize this matrix with a QR factorization, then

$$\begin{aligned} AV_m &= V_{m+1}H_{m+1,m}, \\ AQ_mR_{m,m} &= Q_{m+1}R_{m+1,m+1}H_{m+1,m}, \\ AQ_m &= Q_{m+1}R_{m+1,m+1}H_{m+1,m}R_{m,m}^{-1}. \end{aligned}$$

Now we use Theorem 2.3.1 with the matrix $R_{m+1,m+1}H_{m+1,m}R_{m,m}^{-1}$, so we have

$$AQ_mP_{m,m} = Q_{m+1}\begin{pmatrix} P_{m,m} & 0 \\ 0 & 1 \end{pmatrix}\widetilde{H}_{m+1,m},$$

setting

$$V_{m+1} := Q_{m+1}\begin{pmatrix} P_{m,m} & 0 \\ 0 & 1 \end{pmatrix}.$$

That is we obtain the Arnoldi sequence $AV_m = V_{m+1}\widetilde{H}_{m+1,m}$.

Therefore, if we are using the Power Method and we store all the sequence, then it is possible to compute not just the dominant eigenpair but a few of them. Unluckily this is not a practical algorithm since generally, the closer v_m to convergence the larger the condition number of $R_{m,m}$.

2.4 Generalized eigenvalue problem

Let us consider the generalized eigenvalue problem

$$Ax = \lambda Bx,$$

where $A, B \in \mathbb{C}^{n\times n}$ and B is nonsingular.

One simple way to solve this problem is to compute B^{-1} and then the eigenvalues of $B^{-1}A$. But since we are interested in large–scale and sparse problems we have to avoid the computation of B^{-1} and we rather prefer to solve linear systems with the matrix B.

We can use the Arnoldi algorithm, the only wariness to use is that during the computation of the next vector of the sequence, we have to split the computation of $r = B^{-1}Av_i$ into two steps. Firstly we compute $\hat{r} = Av_i$ and then we solve $Br = \hat{r}$.

The Arnoldi sequence for the GEP is $B^{-1}AV_m = V_{m+1}H_{m+1,m}$ that can be written also as $AV_m = BV_{m+1}H_{m+1,m}$. We can summarize this process in Algorithm 8.

Note that the reorthogonalization process and the restart can be used also in this case without any change, in fact we are simply applying the Arnoldi algorithm to the matrix $B^{-1}A$ and the only practical difference in the algorithm is the way to compute the Arnoldi sequence. For every step we have to solve a linear system with the matrix B, since in many applications B is sparse, we have two main strategy: it is possible to use some iterative method or compute a sparse LU factorization of B, then the initialization will be slow but the subsequent steps faster.

Another possibility, that we will not investigate, is when the matrix B is positive definite, in this case we can compute a B-orthogonal Arnoldi's sequence. Moreover if A is also Hermitian then the matrix $H_{m,m}$ will be tridiagonal with all the consequent computational advantages. For details [9].

Algorithm 8 Arnoldi's algorithm for GEP

```
 1: Chose a vector $x$.
 2: Normalize $v_1 := x/\|x\|$.
 3: for $i = 1, \dots, m$ do
 4:    $\hat{r} = Av_i$.
 5:    Solve the linear system $Br = \hat{r}$.
 6:    $h_i = V_i^H r$.
 7:    $r = r - V_i h_i$.
 8:    $h_{i+1,i} = \|r\|$.
 9:    Compute eigenpair $(\theta_k, y_k)$ of $H_{i,i}$, where $k = 1, \dots, i$.
10:    if $h_{i+1,i} = 0$ then
11:       Store $(\theta_k, V_m y_k)$ as eigenpair of the pencil $(A, B)$, where $k = 1, \dots, i$.
12:       break
13:    else
14:       $v_{i+1} = r/h_{i+1,i}$.
15:    end if
16:    for $k = 1, \dots, i$ do
17:       if $\omega_i(y_k) < \text{tol}$ then
18:          Store $(\theta_k, V_m y_k)$ as approximation of eigenpair of the pencil $(A, B)$.
19:       end if
20:    end for
21: end for
```

2.5 Shift–and–invert Arnoldi

In the previous sections we showed that the Arnoldi method is suitable to compute eigenvalues in the outermost part of the spectrum, in the practical applications often it is needed to compute eigenvalues near some points or in a bounded (or unbounded) region of the complex plane. The shift–and–invert Arnoldi allows one to compute eigenvalues of a generalized eigenvalue problem $Ax = \lambda Bx$ near a point $\sigma \in \mathbb{C}$ where σ is not an eigenvalue.

Lemma 2.5.1. *If (θ, x) is an eigenpair of $(A - \sigma B)^{-1}B$ then $(\sigma + 1/\theta, x)$ is an eigenpair of the pencil (A, B).*

Proof. It is a direct computation

$$(A - \sigma B)^{-1}Bx = \theta x,$$

$$\begin{aligned} Bx &= \theta(A-\sigma B)x, \\ \frac{1}{\theta}Bx &= Ax - \sigma Bx, \\ Ax &= \left(\sigma + \frac{1}{\theta}\right) Bx. \end{aligned}$$

□

If $\theta_1, \dots, \theta_n$ are eigenvectors of $(A-\sigma B)^{-1}B$ then $\sigma + 1/\theta_1, \dots, \sigma + 1/\theta_n$ are eigenvalues of the pencil (A, B). Moreover if $|\theta_1| \geq \cdots \geq |\theta_n|$ then $|\sigma + 1/\theta_1| \leq \cdots \leq |\sigma + 1/\theta_n|$ in particular, if θ_1 is the outermost eigenvalue of $(A-\sigma B)^{-1}B$ then $\sigma + 1/\theta_1$ is the eigenvalue of the pencil (A, B) nearest σ.

Remark 2.5.1. *From the outermost eigenvalues of $(A-\sigma B)^{-1}B$ we can compute the eigenvalues of the pencil (A, B) nearest σ. Then we can use the Arnoldi algorithm 1 if needed with some restarting strategy like in the Algorithms 3, 4 of better 7 to compute outermost eigenvalues of $(A-\sigma B)^{-1}B$ and Lemma 2.5.1 to transform into the eigenvalues of the pencil (A, B) nearest to σ.*

We are now in the same situation of Section 2.4, in fact we have to use the Arnoldi algorithm with the matrix $(A-\sigma B)^{-1}B$ but we want to avoid to compute $(A-\sigma B)^{-1}$. In the applications of interest, A and B are sparse or structured, so that is more convenient to solve linear systems rather than inverting matrices. The process can be summarized in Algorithm 9.

Algorithm 9 Shift–and–invert Arnoldi's algorithm for GEP

```
1: Chose a vector $x$.
2: Normalize $v_1 := x/\|x\|$.
3: for $i = 1, \dots, m$ do
4:     $\hat{r} = Bv_i$.
5:     Solve the linear system $(A - \sigma B)r = \hat{r}$.
6:     $h_i = V_i^H r$.
7:     $r = r - V_i h_i$.
8:     $h_{i+1,i} = \|r\|$.
9:     Compute eigenpair $(\theta_k, y_k)$ of $H_{i,i}$, where $k = 1, \dots, i$.
10:    if $h_{i+1,i} = 0$ then
11:        Store $(\sigma + 1/\theta_k, V_m y_k)$ as eigenpair of $(A, B)$, where $k = 1, \dots, i$.
12:        break
13:    else
14:        $v_{i+1} = r/h_{i+1,i}$.
15:    end if
16:    for $k = 1, \dots, i$ do
17:        if $\omega_i(y_k) < \text{tol}$ then
18:            Store $(\sigma + 1/\theta_k, V_m y_k)$ as approximation of eigenpair of $(A, B)$.
19:        end if
20:    end for
21: end for
```

A practical issue is how to solve for every step the linear system with the matrix $(A - \sigma B)$. An idea can be to use an iterative method, otherwise if the matrices are sparse, one can compute a sparse LU factorization. Before to perform the sparse LU factorization can be needed to reduce the bandwidth of the matrix. This step is usually needed in these kind of iterative algorithms.

2.6 Rational Krylov

As explained in Section 2.5, in many application it is needed to compute a few eigenvalues in a bounded (or unbounded) region of interest in the complex plane. Most popular examples are eigenvalues with positive real parts, with zero imaginary parts, near zero, inside a given rectangle, etc.

The easiest way to solve the problem can be: start with a point σ_1 inside the region of interest by using Shift–and–Invert Arnoldi (Algorithm 9) compute eigenvalues near such point, after that choose another point σ_2 inside the region of interest (for example we can choose a Ritz value that did not converge before) and use again Shift–and–Invert Arnoldi. This algorithm looks slow. The main problem is that in order to compute eigenvalues near σ_1 it is needed to have an enough long Arnoldi sequence say m_1 (possibly doing a few restart). When we want to compute eigenvalues near σ_2 we have to discard everything and start again from the beginning to have an enough long Arnoldi sequence say m_2. The rational Krylov method as exposed in [12] solves exactly this problem.

The idea of the algorithm is to start from an Arnoldi sequence $(A-\sigma_1 B)^{-1} B V_m = V_{m+1} H_{m+1,m}$ and with a few algebraic manipulations to get another Arnoldi sequence $(A - \sigma_2 B)^{-1} B W_m = W_{m+1} \widetilde{H}_{m+1,m}$.

Let us start with the Arnoldi sequence pointed in σ_1

$$\begin{aligned}(A - \sigma_1 B)^{-1} B V_m &= V_{m+1} H_{m+1,m},\\ B V_m &= (A - \sigma_1 B) V_{m+1} H_{m+1,m}.\end{aligned}$$

adding $(\sigma_1 - \sigma_2) B V_{m+1} H_{m+1,m}$ in both sides

$$\begin{aligned}(\sigma_1 - \sigma_2) B V_{m+1} H_{m+1,m} + B V_m &= (A - \sigma_2 B) V_{m+1} H_{m+1,m},\\ B V_{m+1} [(\sigma_1 - \sigma_2) H_{m+1,m} + I_{m+1,m}] &= (A - \sigma_2 B) V_{m+1} H_{m+1,m}.\end{aligned}$$

Let us define $K_{m+1,m} := (\sigma_1 - \sigma_2) H_{m+1,m} + I_{m+1,m}$, so we have

$$(A - \sigma_2 B)^{-1} B V_{m+1} K_{m+1,m} = V_{m+1} H_{m+1,m}.$$

Now with a QR factorization of $K_{m+1,m}$

$$\begin{aligned}(A - \sigma_2 B)^{-1} B V_{m+1} Q_{m+1,m+1} \begin{pmatrix} R_{m,m} \\ 0 \end{pmatrix} &= V_{m+1} H_{m+1,m},\\ (A - \sigma_2 B)^{-1} B V_{m+1} Q_{m+1,m} &= V_{m+1} H_{m+1,m} R_{m,m}^{-1},\\ (A - \sigma_2 B)^{-1} B V_{m+1} Q_{m+1,m} &= V_{m+1} Q_{j+1,j+1} (Q_{j+1,j+1}^H H_{m+1,m} R_{m,m}^{-1}).\end{aligned}$$

Let $L_{j+1,j} := Q^H_{j+1,j+1} H_{m+1,m} R^{-1}_{m,m}$ then

$$(A - \sigma_2 B)^{-1} B V_{m+1} Q_{m+1,m} = V_{m+1} Q_{m+1,m+1} L_{m+1,m}.$$

In view of Theorem 2.3.1 we have

$$L_{m+1,m} = \begin{pmatrix} P_{m,m} & 0 \\ 0 & 1 \end{pmatrix} \widetilde{H}_{m+1,m} P^H_{m,m}$$

Where $\widetilde{H}_{m+1,m}$ is a Hessenberg matrix with positive elements in the subdiagonal. Replacing we get

$$(A - \sigma_2 B)^{-1} B V_{m+1} Q_{m+1,m} P_{m,m} = V_{m+1} Q_{m+1,m+1} \begin{pmatrix} P_{m,m} & 0 \\ 0 & 1 \end{pmatrix} \widetilde{H}_{m+1,m}.$$

Let us define

$$W_{m+1} := V_{m+1} Q_{m+1,m+1} \begin{pmatrix} P_{m,m} & 0 \\ 0 & 1 \end{pmatrix}.$$

Then finally we get

$$(A - \sigma_2 B)^{-1} B W_m = W_{m+1} \widetilde{H}_{m+1,m},$$

and this is an Arnoldi sequence. We can summarize this process in Algorithm 10.

Observation 2.6.1. Note that the space generated by the columns of V_{m+1} is the same generated by the columns of W_{m+1}. In fact, this matrix is obtained as linear combination of the columns of V_{m+1}. These algebraic manipulations were used to transform the Arnoldi sequence $(A - \sigma_1 B)^{-1} B V_m = V_{m+1} H_{m+1,m}$ into another sequence $(A - \sigma_2 B)^{-1} B W_m = W_{m+1} \widetilde{H}_{m+1,m}$.

As already explained in Section 2.5 it is not convenient to compute $(A - \sigma_i B)^{-1} B$, instead it is preferable to solve systems with the matrix $A - \sigma_i B$. This way, as usual, we have two possibilities: using iterative methods or sparse LU factorization.

The numbers σ_i are called shifts or poles (the latter term will be clarified later on). A practical issue is that if we choose a shift σ_i too close to the same eigenvalue of the pencil (A, B) then the matrix $A - \sigma B$ is close to be singular and numerical errors will occur. Special strategies are needed in order to choose new shifts.

Algorithm 10 Rational Krylov's algorithm for GEP

```
1: Chose a vector $x$ and a first shift $\sigma_1$.
2: Normalize $v_1 := x/\|x\|$.
3: while $\omega(y_k) > tol$ (the desired Ritz value did not converge) do
4:    Expand the Arnoldi sequence $(A - \sigma_i B)^{-1} B V_m = V_{m+1} H_{m+1,m}$ checking if
      Ritz values converge and doing restart when needed .
5:    Determine new shift $\sigma_{i+1}$ in the interesting region.
6:    Factorize $I_{m+1,m} + (\sigma_i - \sigma_{i+1}) H_{m+1,m} = Q_{m+1,m+1} R_{m+1,m}$.
7:    Get the Hessenberg form
      $$H_{m+1,m} := \begin{pmatrix} P_{m,m}^H & 0 \\ 0 & 1 \end{pmatrix} Q_{m+1,m+1}^H H_{m+1,m} R_{m,m}^{-1} P_{m,m}.$$
8:    $V_{m+1} := V_{m+1} Q_{m+1,m+1} \begin{pmatrix} P_{m,m} & 0 \\ 0 & 1 \end{pmatrix}$.
9: end while
```

Why rational?

In the classic Arnoldi algorithm we approximate the eigenvectors of A in the Krylov subspace

$$K_m(A, x) = \operatorname{span}\left(x, Ax, \dots, A^{m-1}x\right),$$

so the Ritz vectors can be written as $p(A)x$ where $p(t)$ is a polynomial of degree less then m. The original idea of A. Ruhe [10] was to use rational functions $r(t)$ instead than polynomials, so to build the Rational Krylov subspace

$$R_m(A, x) = \operatorname{span}\left(\phi_1(A)x, \dots, \phi_m(A)x\right),$$

where the most natural choise for these functions is

$$\begin{cases} \phi_1(t) = 1 \\ \phi_{i+1}(t) = \dfrac{\phi_i(t)}{t - \sigma_i} \end{cases},$$

and the sequence of vectors is $x_{i+1} = (A - \sigma_i I)^{-1} x_k$. Then we compute the sequence untill the vectors became independet

$$\begin{aligned} 0 &= c_1 x_1 + \dots + c_m x_m \\ &= c_1 \phi_1(A)x + \dots + c_m \phi_m(A) \\ &= \left(c_1 \phi_1(t) + \dots + c_m \phi_m(t)\right)(A)x. \end{aligned}$$

Then let us define

$$\begin{aligned} \phi(t) &:= c_1 \phi_1(t) + \dots + c_m \phi_m(t) \\ &= c_1 + \frac{c_2}{t - \sigma_1} + \dots + \frac{c_m}{(t - \sigma_1) \dots (t - \sigma_{m-1})}. \end{aligned}$$

then with an easy computation it is possible to show that the zeros of this function are eigenalues of A and they are near the poles σ_i. In general we can seek an approximate linear dependece by using the singular value decomposition $X_m = U_m \Sigma_m V_m^H$ and choose $c = v_m$. The problem of this approach is that there is no theory to explain the convergence.

In later articles like [11], A. Ruhe proposed a different algorithm with a deeper theoretical background preserving the main feature: the possibility to change shift. The problem related with this second version is that the Ritz values are computed by solving a Hessenberg GEP. Whereas in the last version [12] that we presented, the Ritz values are computed as eigenvalues of a Hessenberg matrix. Moreover the theory of convergence can be taken from the classical Arnoldi algorithm.

2.7 Practical issues

In this section we report some heuristics and recipes concerning the implementation of the Rational Krylov algorithm as suggested in [11].

In the Rational Krylov algorithm 10 there are three important points:

- solve the linear system in the step 4,
- choice of the shifts,
- when to do a restart.

Assume we are interested in computing the eigenvalues of the pencil (A, B) in a region of interest $\Omega \subset \mathbb{C}$, where Ω can be bounded or not. If this region is bounded and we have no information about the distribution of eigenvalues then the first shift σ_1 has to be in the middle of Ω, or at least not near the border set.

If Ω is not bounded, e.g., we are interested in eigenvalues along real axis and $B = I$, then we start with $\sigma_1 = \pm\|A\|$. Similar strategies can be used for the GEP once we located the outermost eigenvalues (e.g. with Arnoldi algorithm).

When we are in the step 4 of the algorithm 10, for every iteration we need to solve a linear system with the matrix $A - \sigma_i B$, then we can do a sparse LU factorization (once reducing the bandwidth if needed) and keep the same shift for a few steps.

It is suggested to keep the same shift until enough Ritz values, say *cstep*, converged. Usually *cstep* is a small number depending on the problem, in our test we chose *cstep* = 2 and *cstep* = 4. Anyway if after a too larger number of steps, say *mstep*, the Ritz values did not converge then it is better to change shift. After that we will choose the next shift. The main strategy can be to choose the next shift as the average of the *cstep* not converged Ritz values closest to the current shift. If the Ritz values are clustered, then it is better to chose the next shift as the average of the Ritz values in the bigger cluster.

From the numerical experimentation it seems that after a few steps where the algorithm gathers information (building the Kylov subspace), the Ritz values start to converge linearly to some eigenvalues. Then, if we did j steps of the Arnoldi sequence, let us consider j_1 the number of Ritz values we are interested (inside the region of interest and not too far from the shift), then $j - j_1$ are purgeable Ritz values. The computational cost to do other k steps with $(j - j_1)$ purgeable Ritz vectors is $4n(j - j_1)$ where reorthogonalization is done for every step. Instead if we do a restart, the computational cost is $j \times j_1 \times n$ (we neglected the cost of operations with k as the transformation in Hessenberg form after restart because $k << n$). If we purge the $j - j_1$ unwanted Ritz vectors, the number of vectors will grow again. If nothing special happens, when after restart we do $j - j_1$ steps we are in te same decision situation, then we can say that if doing a restart is cheaper than doing other $j - j_1$ steps with $j - j_1$ unwanted Ritz vector, then it is better to do a restart. So with $k = j - j_1$ we will restart when

$$4n(j - j_1)^2 > njj_1,$$

that is when

$$j > 1.6j_1.$$

Anyway it is important to point out that this is a heuristic that works well for a few cases. In our numerical experimentation we used an easier restarting strategy, we restart when the length of the sequence is more that *jmax* and we kept *jkept* Ritz values. It is worth recalling that a restart is performed to reduce the length of the Arnoldi sequence and this is needed mainly for two reasons. The first is that with a long sequence, convergence is faster (less steps are needed) but the size of the reduced problem is big and every step is slow. The second reason to have a not long sequence is to avoid the loss of orthogonality, then in a restart strategy we have to consider also this point. Unfortunately, in many real problems, having a long sequence is needed to arrive at convergence.

In the rational Krylov algorithm, restarting strategy is also connected to the idea of lock and purge. If we are computing eigenvalues inside a region of interest Ω then during restart we can purge Ritz values outside this region, it means that we will not store them for the restart. It is worth pointing out that if we purge Ritz values near the shift already converged, after restart they will converge again and this would be a waste of resources. Then the idea is to take also them for the restarting, this operation is called lock. Moreover, the comments reported in Section 2.3 still hold true, we also need to take for the restart the sought Ritz values of which we want to improve convergence.

Remark 2.7.1. *At a glance we have:*

- *a good strategy is to keep the same shift σ_i for a few steps, e.g., untill cstep Ritz pair converged, the number cstep depends on the problem, in general is small;*
- *in order to solve the linear systems we perform a sparse LU factorization of $A - \sigma_i B$;*
- *under the assumption that the Ritz value converges lineary, we can restart when $j > 1.6j_1$ where j_1 are the wanted Ritz values, for instance inside the region of interest and not far from the shift;*
- *during restart we can purge unwanted Ritz values and lock the ones already converged near the shift. Moreover the sought Ritz values near convergence will be also used for the restart.*

If the goal is computing eigenvalues in a bounded set $\Omega \subset \mathbb{C}$ we need to know how many eigenalues are inside this region.

In [7] a strategy is proposed based on residue theorem, the idea is to find a method to compute

$$N_\Omega = \frac{1}{2\pi i}\int_{\partial\Omega}\frac{f'(z)}{f(z)}dz,$$

where $f(z) = \det(A - zB)$, and N_Ω is the number of eigenvalues inside Ω.

In Rational Krylov algorithm, as stop criterium we choose $\omega_m(y) \leq \ tol$ where $\omega_m(y) = h_{m+1,m}|e_m^H y|$, because we proved that $\|(A - \sigma B)^{-1}V_m y - \theta V_m y\| = \omega_m(y)$ but in the algorithm there are a few sources of error, e.g., for every step we have to solve a linear system, during the change of the shift we need to invert a matrix, loss of orthogonality etc. If $(\lambda_1, z_1), \dots (\lambda_m, z_m)$ are the converged Ritz values,in particular for the last ones, it holds

$$\|Az_i - \lambda_i B z_i\| \geq tol.$$

Anyway the computed Ritz pairs are a good estimation of eigenpairs, then we need to refine. There are few strategies to refine, if we have a classic problem we can use Rayleigh quotient iteration, in the general case instead we can use Shift–and–Invert Arnoldi pointed in the Ritz value that we want to refine and starting with the corresponding Ritz vector. We can also use the Shifted Inverse Power method. In general after few steps numerical converge is reached. The problem is that during the refine process ill–conditioned linear systems must be solved. For example in the Rayleigh quotient for every step we need to solve a system with the matrix $A - \mu_k I$ where after every step μ_k is closer to some eigenvalue of A, the same happens with the Shifted Inverse Power Method. The idea suggested by Wilkinson in [21] is to do a few steps with such methods that are faster, when the linear systems become too ill–conditioned, take the last approximation to the eigenvector and use it to start an Arnoldi sequence. The classic Arnoldi algorithm 1 converges to the sought eigenvector (see Theorem 2.2.1), the convergence is slower but the algorithm is stable.

Chapter 3

Nonlinear eigenvalue problem

The nonlinear eigenvalue problem (NLEP) can be formulated in the following way:

given $A(\cdot) : \mathbb{C} \to \mathbb{C}^{n\times n}$, compute the pairs (λ, x) such that $A(\lambda)x = 0$.

λ is called eigenvalue, x eigenvector and the pair (λ, x) is called eigenpair.

We can see A in different ways, for istance as a matrix depending on a parameter. In this case, solving the NLEP means to find the values of this parameter which make the matrix singular and then compute the nullspace. We can also see A as a matrix with coefficients that are functions $a_{i,j}(\lambda)$. The first interpretation is useful to understand the meaning of solving a NLEP, the second one is suitable to find numerical algorithms to estimate the solutions.

It is important to point out that it is possible to write

$$A(\lambda) = \sum_{i=1}^{m} f_i(\lambda)B_i. \tag{3.1}$$

For instance, a naive way to write the application $A(\lambda)$ in this form is

$$A(\lambda) = \sum_{i,j=0}^{n} a_{i,j}(\lambda)E_{i,j},$$

where $E_{i,j}$ denote the matrices of the canonical basis of $\mathbb{R}^{n\times n}$, i.e., $E_{i,j}$ has zero entries except for the entry in position (i, j) which is 1. Here, we adopt the formulation of the NLEP given in the form (3.1).

Usually, we are interested in computing the eigenvalues belonging to a given set $\Omega \subset \mathbb{C}$ analogously to the linear case. Moreover we suppose that $f_i(\lambda)$ is regular enough. Note that the generalized (and then the classic) eigenvalue problem can be formulated also as NLEP by setting $A(\lambda) = A - \lambda B$.

If the functions $f_i(\lambda)$ are polynomials the problem is called *polynomial eigenvalue* problem PEP and it is very easy to write an equivalent GEP. The basic idea of the algorithms to solve NLEP is to write a GEP such that the eigenpairs (in Ω) of this easier problem are good approximations to the eigenpairs (in Ω) of the original problem, this operation is called linearization.

The first algorithm that is presented in this thesis is HIRK (Hermite Interpolation Rational Krylov) and the idea is to Hermite–interpolate the function $A(\lambda)$ in a few points and take advantage of this representation in solving the linearized problem with the Rational Krylov algorithm presented in the previous chapter.

Another algorithm is NLRK (NonLinear Rational Krylov). This is an extension of Rational Krylov algorithm for the nonlinear case and consists in estimate $A(\lambda)$ with linear interpolations. This second algorithm is faster but it works well just in a few cases, e.g., if the NLEP is a small perturbation of a linear problem.

3.1 Rational Krylov based on Hermite interpolations

The idea of Meerbergen et al. [18] is to replace the functions $f_i(\lambda)$ with polynomials using Hermite interpolation. After that, it is very easy to write a GEP. The structure of this GEP will be exploited and we will show that a suitable way to solve the linearized problem is using Rational Krylov algorithm.

We will present a slight variant of the algorithm proposed in [18], in particular the original algorithm uses the old version of Rational Krylov algorithm to solve the linearized problem. We have modified the algorithm in order to use the last version of Rational Krylov algorithm as presented the previous chapter.

In the next subsection we will recall basic ideas of Hermite interpolation.

3.1.1 Newton polynomials and Hermite interpolation

Let suppose we want interpolate a function $f(x)$ at the points $\sigma_0, \dots, \sigma_N$. That is, we are looking for a polynomial $p_N(x)$ of degree N such that $p_N(\sigma_i) = f(\sigma_i)$. The classical strategy is to use Lagrange polynomials. Given the interpolation points $\sigma_0, \dots, \sigma_N$, consider the Lagrange polynomials defined as

$$l_i^{(N)}(x) = \prod_{j=0}^{N} \frac{x - \sigma_j}{\sigma_i - \sigma_j}, \qquad i \le N,$$

then we express the interpolation polynomial using the basis of Lagrange polynomials that is

$$p_N(x) = \sum_{i=0}^{N} \alpha_i^{(N)} l_i^{(N)}(x).$$

The coefficients $\alpha_i^{(N)}$ are called *interpolation coefficients.* A drawback is that when we want to add a new interpolation point using Lagrange interpolations, the interpolations coefficients already computed must be updated. That is, if we know the interpolation polynomial p_N at the nodes $\sigma_0, \dots, \sigma_N$ and we want to add another interpolation point σ_{N+1}, then

$$p_{N+1}(x) = \sum_{i=0}^{N+1} \alpha_i^{(N+1)} l_i^{(N+1)}(x).$$

All Lagrange basis polynomials have to be recalculated. Moreover $\alpha_i^{(N)} \neq \alpha_i^{(N+1)}$ for $i \leq N$. Our goal is to build a linearization of a NLEP using the interpolation coefficients, for this reason we prefer that adding an interpolation point does not change already computed coefficients. This is possible using Newton polynomials.

Given the interpolation points $\sigma_0, \dots, \sigma_N$, consider the Newton polynomials defined as

$$\begin{cases} n_0(x) := 1, \\ n_i(x) := \displaystyle\prod_{j=0}^{i-1} (x - \sigma_j). \end{cases}$$

Note that the degree of $n_i(x)$ is i (unlike the Lagrange polynomials which are of the same degree). We want to express the interpolation polynomial p_N as linear combination of Newton polynomials.

$$p_N(x) = \sum_{i=0}^{N} \alpha_i n_i(x).$$

It is possible to prove that the coefficients of this linear combination are the divided differences

$$\alpha_i = f\left[\sigma_0, \dots, \sigma_i\right].$$

The divided differences are defined as

$$\begin{cases} f[\sigma_i] := f(\sigma_i), \\ f[\sigma_\nu, \dots, \sigma_{\nu+j}] := \dfrac{f[\sigma_{\nu+1}, \dots, \sigma_{\nu+j}] - f[\sigma_\nu, \dots, \sigma_{\nu+j-1}]}{\sigma_{\nu+j} - \sigma_\nu}. \end{cases}$$

Now we will give an easy algorithm to compute the coefficients α_i based on the divided differences table.

Consider the following table, called divided differences table

$$\begin{array}{c|cccccc}
\sigma_0 & f[\sigma_0] & & & & & \\
\sigma_1 & f[\sigma_1] & f[\sigma_0,\sigma_1] & & & & \\
\sigma_2 & f[\sigma_2] & f[\sigma_1,\sigma_2] & f[\sigma_0,\sigma_1,\sigma_2] & & & \\
\sigma_3 & f[\sigma_3] & f[\sigma_2,\sigma_3] & f[\sigma_1,\sigma_2,\sigma_3] & f[\sigma_0,\sigma_1,\sigma_2,\sigma_3] & & \\
\sigma_4 & f[\sigma_4] & f[\sigma_3,\sigma_4] & f[\sigma_2,\sigma_3,\sigma_4] & f[\sigma_1,\sigma_2,\sigma_3,\sigma_4] & f[\sigma_0,\sigma_1,\sigma_2,\sigma_3,\sigma_4] & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

It is easy to compute this table. For instance, using the definition of divided differences we can compute

$$f[\sigma_1,\sigma_2,\sigma_3] = \frac{f[\sigma_2,\sigma_3] - f[\sigma_1,\sigma_2]}{\sigma_3 - \sigma_1} = \frac{\frac{f[\sigma_3]-f[\sigma_2]}{\sigma_3-\sigma_2} - \frac{f[\sigma_2]-f[\sigma_1]}{\sigma_2-\sigma_1}}{\sigma_3 - \sigma_1} = \frac{\frac{f(\sigma_3)-f(\sigma_2)}{\sigma_3-\sigma_2} - \frac{f(\sigma_2)-f(\sigma_1)}{\sigma_2-\sigma_1}}{\sigma_3 - \sigma_1}.$$

For the problem of interpolation we are interested in computing the diagonal elements of this table but it is obvious that we need to compute all the table. This table gives us a recipe to build a matrix that we will call DD (divided differences matrix) such that

$$\mathrm{DD}(i,j) = f[\sigma_{j-i}, \dots, \sigma_{i-1}].$$

Then with a direct computation it is easy to see that the divided differences table it is equivalent to

$$\begin{array}{c|cccccc}
\sigma_0 & \mathrm{DD}(1,1) & & & & & \\
\sigma_1 & \mathrm{DD}(2,1) & \mathrm{DD}(2,2) & & & & \\
\sigma_2 & \mathrm{DD}(3,1) & \mathrm{DD}(3,2) & \mathrm{DD}(3,3) & & & \\
\sigma_3 & \mathrm{DD}(4,1) & \mathrm{DD}(4,2) & \mathrm{DD}(4,3) & \mathrm{DD}(4,4) & & \\
\sigma_4 & \mathrm{DD}(5,1) & \mathrm{DD}(5,2) & \mathrm{DD}(5,3) & \mathrm{DD}(5,4) & \mathrm{DD}(5,5) & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

We can build the matrix DD column by column, in fact using the definition of divided differences we get

$$\begin{cases}
\mathrm{DD}(i,1) = f(\sigma_{i-1}), & \\
\\
\mathrm{DD}(i,j) = \dfrac{\mathrm{DD}(i,j-1) - \mathrm{DD}(i-1,j-1)}{\sigma_{i-1} - \sigma_{j-i}}, & \begin{array}{l} i \geq j \\ j \geq 2 \end{array}
\end{cases}.$$

At this point we have an algorithm to perform interpolation using Newton polynomials. If we want to interpolate $f(x)$ in the nodes $\sigma_0, \dots, \sigma_N$ then we need to compute the matrix $\mathrm{DD} \in \mathbb{R}^{(N+1)\times(N+1)}$, then define $\alpha_i = \mathrm{DD}(i+1, i+1)$ and the interpolation polynomial is

$$p_N(x) = \sum_{i=0}^{N} \alpha_i n_i(x).$$

It is important to point out that if we want to add an interpolation point σ_{N+1} we need to compute another row and another column of DD and then $\alpha_{N+1} = \mathrm{DD}(N+2, N+2)$, so we get

$$p_{N+1}(x) = \sum_{i=0}^{N+1} \alpha_i n_i(x).$$

Relying on the same ideas, we describe the Hermite interpolation. In this case we are given nodes $\sigma_0, \dots, \sigma_N$ with the possibility of some repetition. In a certain sense, we are interpolating more then one time on the same node. This means that if the node σ_i appears k times in the sequence then the polynomial interpolates f and the first $k-1$ derivatives in σ_i, that is $f^{(t)}(\sigma_i) = p^{(t)}(\sigma_i)$ for $1 \leq t \leq k-1$. To perform this task we can use the same strategy applied to the Hermite interpolating polynomial.

Define

$$f[\underbrace{\sigma_i, \dots, \sigma_i}_{j+1 \text{ times}}] := \frac{f^{(j)}(\sigma_i)}{j!}.$$

Then the rule to compute the divided differences in this case is

$$\begin{cases} f[\sigma_i] := f(\sigma_i), \\ f[\sigma_\nu, \dots, \sigma_{\nu+j}] := \dfrac{f[\sigma_{\nu+1}, \dots, \sigma_{\nu+j}] - f[\sigma_\nu, \dots, \sigma_{\nu+j-1}]}{\sigma_{\nu+j} - \sigma_\nu} & \sigma_\nu \neq \sigma_{\nu+j}, \\ f[\underbrace{\sigma_i, \dots, \sigma_i}_{j+1 \text{ times}}] := \dfrac{f^{(j)}(\sigma_i)}{j!}. \end{cases}$$

We can construct again the divided differences table. Nothing is changed, we just need to pay attention when we compute $f[\sigma_i, \dots, \sigma_i]$ (the same interpolation point). For instance, like in the previous example, we want to compute $f[\sigma_0, \sigma_1, \sigma_2]$ where $\sigma_1 = \sigma_2$. Then we have

$$f[\sigma_1, \sigma_2, \sigma_3] = f[\sigma_1, \sigma_2, \sigma_2] = \frac{f[\sigma_2, \sigma_2] - f[\sigma_1, \sigma_2]}{\sigma_2 - \sigma_1} = \frac{f'(\sigma_2) - \frac{f[\sigma_2] - f[\sigma_1]}{\sigma_2 - \sigma_1}}{\sigma_2 - \sigma_1} = \frac{f'(\sigma_2) - \frac{f(\sigma_2) - f(\sigma_1)}{\sigma_2 - \sigma_1}}{\sigma_2 - \sigma_1}.$$

With the same computation we can build the divided differences matrix using the following rule

$$\begin{cases} \mathrm{DD}(i,1) = f(\sigma_{i-1}), \\ \\ \mathrm{DD}(i,j) = \dfrac{\mathrm{DD}(i,j-1) - \mathrm{DD}(i-1,j-1)}{\sigma_{i-1} - \sigma_{j-i}} & \begin{array}{l} i \geq j \\ j \geq 2 \end{array}, \\ \\ \mathrm{DD}(i,j) = \dfrac{f^{(j-1)}(\sigma_i)}{(j-1)!} & \begin{array}{l} i \geq j \\ \sigma_{i-1} = \sigma_{i-j} \end{array}. \end{cases}$$

Example 3. Consider the function $f(x) = \cos(2\pi x)$ and perform Hermite interpolation

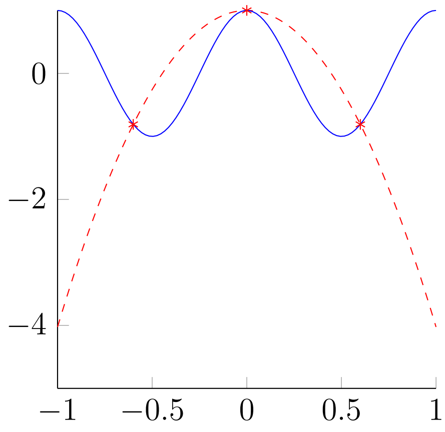
From this example we can understand that doing few interpolations in the same point, like -0.6 or 0.6, can be crucial to have a good approximation. If we interpolate more then one time where we already have a good approximation, like in 0, then it is a waste of resources. Notice that the derivative of the function in 0 and the derivative of $p_4(x)$ was already equal to 0, this means that $p_5(x) = p_4(x)$.

3.1.2 Semianalytical computation of Hermite interpolation

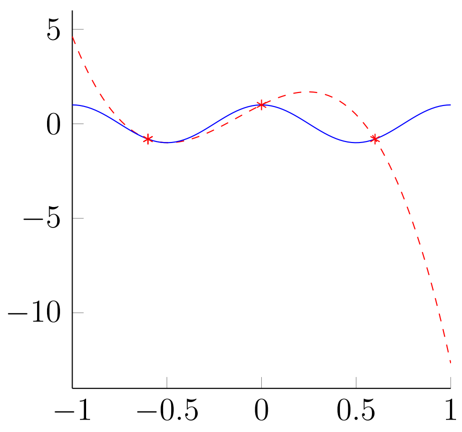
Consider the function f, and the goal is Hermite–interpolating it at the points $\sigma_0, \dots, \sigma_N$ using Newton polynomials. So we have

$$p_N(x) = \sum_{i=0}^{N} \alpha_i n_i(x).$$

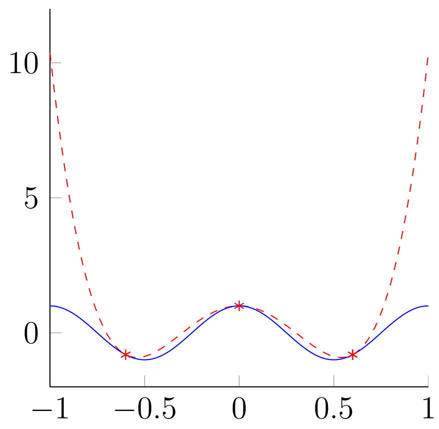
Figure 3.1: The blu line is the function, the red stars the interpolation points, the red dashed line the interpolation polynomial



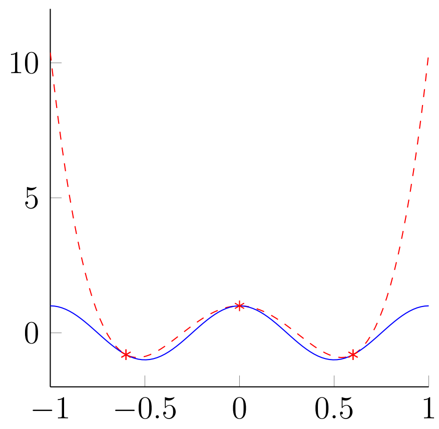
(a) nodes: -0.6 0 0.6



(b) nodes: -0.6 -0.6 0 0.6



(c) nodes: -0.6 -0.6 0 0.6 0.6



(d) nodes: -0.6 -0.6 0 0 0.6 0.6

In the previous subsection we have shown how to do it efficiently by using the divided differences table/matrix. But if we need to make interpolation at a large number of nodes this method is numerical unstable. For this reason sometimes it is better to compute the coefficients semi–analytically. With this expression we mean that in a practical implementation, part of the computation is performed symbolically –say the computation of derivatives– and part is performed numerically, say the evaluation of functions.

Suppose that the interpolation node σ_i is repeated m_i times. Then the first interpolation point σ_0 is repeated m_0 times. Let $t_0(x)$ be the Taylor series truncated ad m_0

$$t_0(x) = f(\sigma_0) + f'(\sigma_0)(x-\sigma_0) + \cdots + \frac{f^{(m_0-1)}(\sigma_0)}{(m_0-1)!}(x-\sigma_0)^{m_0-1}.$$

From this we get the first m_0 coefficients of Hermite interpolation

$$\alpha_i = \frac{f^{(i)}(\sigma_0)}{i!} \qquad i = 0, \dots, m_0 - 1,$$

Now let us define the new function

$$f_1(x) = \frac{f(x) - t_0(x)}{(x-\sigma_0)^{m_0}}.$$

Using this new function again, we can compute the next m_1 coefficients of Hemite interpolation, in fact let $t_1(x)$ the Taylor series of $f_1(x)$ truncated ad m_1 then

$$t_1(x) = f_1(\sigma_1) + f_1'(\sigma_1)(x-\sigma_1) + \dots + \frac{f_1^{(m_1-1)}(\sigma_1)}{(m_1-1)!}(x-\sigma_1)^{m_1-1},$$

then we have

$$\alpha_i = \frac{f_1^{(i)}(\sigma_1)}{i!} \qquad i = m_0, \dots, m_1 - 1.$$

Again we define

$$f_2(x) = \frac{f_1(x) - t_1(x)}{(x-\sigma_1)^{m_1}}$$

and iterate this process. We can summarize this method to compute the coefficients of Hermite–interpolation in the algorithm 11.

Algorithm 11 Semianalytical computation of coefficients of Hermite interpolation

```
1: $f_0(x) := f(x)$
2: for $i = 0, \dots, N$ do
3:     Compute Taylor series truncated to $m_i$
           $$t_i(x) = f_i(\sigma_i) + f_i'(\sigma_i)(x-\sigma_i) + \dots + \frac{f_i^{(m_i-1)}(\sigma_i)}{(m_i-1)!}(x-\sigma_i)^{m_i-1}.$$
4:     Store the coefficients of Hermite interpolation
           $$\alpha_k = \frac{f_i^{(k)}(\sigma_i)}{k!} \qquad k = m_{i-1}, \dots, m_i - 1.$$
5:     Define
           $$f_{i+1}(x) := \frac{f_i(x) - t_i(x)}{(x-\sigma_i)^{m_i}}.$$
6: end for
```

We implemented this algorithm in MATLAB using the symbolic toolbox. In a practical implementation we need to be careful because in every iteration the function $f_i(x)$ becomes more and more complicated. In order to solve this problem in MATLAB, after definition of $f_i(x)$, we can simplifying such expression with the function `simplify` . In the algorithm 11 the use of this function is needed after the step 5. The function `simplify` try (when possible) to simplify the form of $f_i(x)$ for instance performing divisions and multiplications.

Example 4. Consider the same example of [18] concerning the Hermite-interpolation of the function $f(x) = e^{-x}$ at the points $\sigma_i = 0.1, 0.2, 0.3, 0.4, 0.5$ with $m_i = 5$. It is possible to show that the coefficients of this interpolation must decrease but if we use the divided differences matrix to compute such coefficients in floating point arithmetic at a certain point they will start to diverge to infinity. This does not happen by computing the coefficients semi–analytically as shown in figure 3.2.

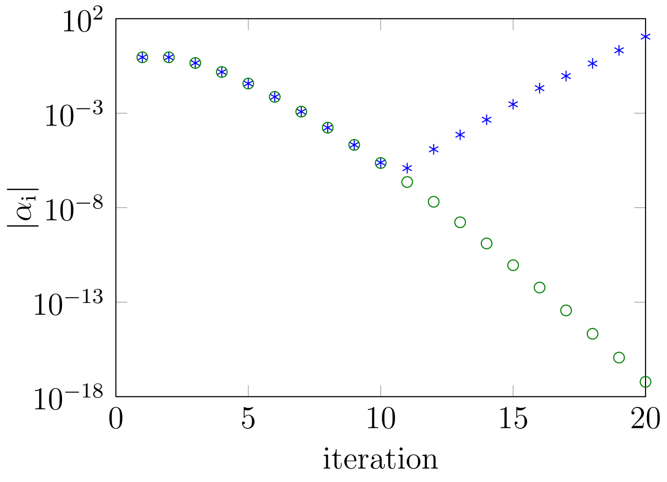


Figure 3.2: Stars indicate the norm of coefficients computed with divided differences matrix, circles the norm of the coefficients computed semi–analytically

We can see the effect of these errors in figure 3.3. Therefore, when the number of interpolation points is large, it is needed to compute the coefficients of Hermite interpolation semi–analytically .

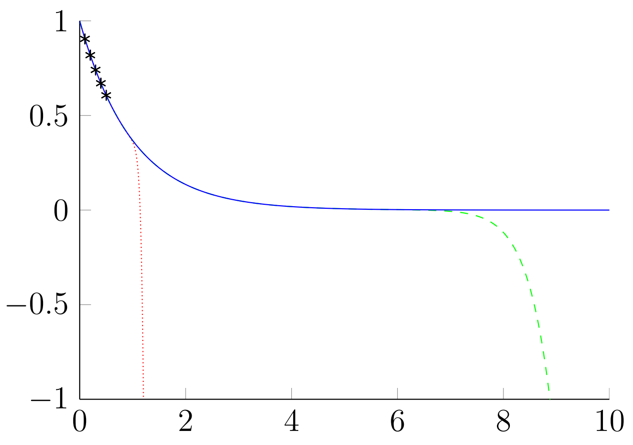


Figure 3.3: Stars denote the interpolation points, red dotted line the interpolation polynomial where we computed coefficients with the divided differences matrix. The green dashed line is the interpolation polynomial where we computed coefficients semi–analytically

3.1.3 Hermite Interpolation Rational Krylov (HIRK)

Now we describe the core of the Hemrite interpolation rational Krylov method (HIRK) presented in [18]. As already mentioned before, we can write $A(\lambda)$ as

$$A(\lambda) = \sum_{i=1}^{m} f_i(\lambda) B_i.$$

Let $\sigma_0, \dots, \sigma_N$ be some points where we want to interpolate the functions $f_i(\lambda)$ (repetitions are allowed), then we can replace the functions $f_i(\lambda)$ with the Hermite interpolations $p_i(\lambda)$. We have

$$P_N(\lambda) = \sum_{j=1}^{m} B_j p_j(\lambda) = \sum_{j=1}^{m} B_j \sum_{i=0}^{N} \alpha_{i,j} n_i(\lambda) = \sum_{i=0}^{N} \left(\sum_{j=1}^{m} \alpha_{i,j} B_j \right) n_i(\lambda) =: \sum_{i=0}^{N} A_i n_i(\lambda).$$

Then $P_n(\lambda)x = 0$ is a PEP. The following theorem provides a linearization of such PEP.

Theorem 3.1.1 (Companion-type linearization)**.** *The pair $(\lambda, x) \neq 0$ is an eigenpair of the PEP if and only if $\mathcal{A}_N y_N = \lambda \mathcal{B}_N y_N$ where*

$$\mathcal{A}_N := \begin{pmatrix} A_0 & A_1 & A_2 & \dots & A_N \\ \sigma_0 I & I & & & \\ & \sigma_1 I & I & & \\ & & \ddots & \ddots & \\ & & & \sigma_{N-1} I & I \end{pmatrix}, \mathcal{B}_N := \begin{pmatrix} 0 & & & & \\ I & 0 & & & \\ & I & 0 & & \\ & & \ddots & \ddots & \\ & & & I & 0 \end{pmatrix}, \quad y_N := \begin{pmatrix} x \\ n_1(\lambda)x \\ n_2(\lambda)x \\ n_3(\lambda)x \\ \vdots \\ n_N(\lambda)x \end{pmatrix}.$$

The proof of this theorem is a direct computation.

Remark 3.1.1. *Starting from the NLEP defined by $A(\lambda)$, Hermite–interpolating at $N+1$ points yields the PEP defined by P_N. Using Theorem 3.1.1 we arrive at a GEP defined by the pencil $(\mathcal{A}_N, \mathcal{B}_N)$. Then we can consider $(\mathcal{A}_N, \mathcal{B}_N)$ as linerization of $A(\lambda)$.*

To understand why this is a linearization, in the sense that we expect that solutions of GEP approximate solution of NLEP, we can restrict to the case of one-dimensional problem where $N = 1$. In this case we are approximating the zeros of a function with the zeros of the Hermite interpolating polynomial. It is important to point out that the linearization can provide spurious eigenvalues, e.g., if we take the classical eigenvalue problem and we consider it as NLEP, then the linearization with $N > 1$ has $n \cdot N$ eigenvalues meanwhile A has just n eigenvalues.

Observation 3.1.1. If we want to include an additional interpolation point, $\mathcal{A}_{N+1}$ and $\mathcal{B}_{N+1}$ are obtained by adding one block column to the right and one block row at the bottom of the matrices $\mathcal{A}_N$ and $\mathcal{B}_N$ already computed.

An important step of the algorithm is the computation of the matrices A_i, then it is important to focus on them. A practical way to build these matrices is to construct the following table: on the columns there are the interpolation coefficients of f_i and therefore on the rows there are the coefficients to construct A_i.

	p_1	p_2	p_3	$\dots$	p_m
A_0	$\alpha_{0,1}$	$\alpha_{0,2}$	$\alpha_{0,3}$	$\dots$	$\alpha_{0,m}$
A_1	$\alpha_{1,1}$	$\alpha_{1,2}$	$\alpha_{1,3}$	$\dots$	$\alpha_{1,m}$
A_2	$\alpha_{2,1}$	$\alpha_{2,2}$	$\alpha_{2,3}$	$\dots$	$\alpha_{2,m}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
A_N	$\alpha_{N,1}$	$\alpha_{N,2}$	$\alpha_{N,3}$	$\dots$	$\alpha_{N,m}$

Then we have

$$A_i = \sum_{j=1}^{m} \alpha_{i,j} B_j.$$

Computing A_{N+1} means adding an interpolation point σ_{N+1} and then to compute another coefficient of interpolation and write another row of this table. Then it is possible to store implicitly the linearization using the interpolation coefficients $\alpha_{i,j}$.

There are four important lemmas that are the key of the algorithm that we will present later on.

Lemma 3.1.1. *Given $\mathcal{A}_N$ and $\mathcal{B}_N$ the linearization matrices and*

$$y_j = vec\left(y_j^{[1]}, y_j^{[2]}, \dots, y_j^{[j+1]}, 0, \dots, 0\right),$$

where $y_j \in \mathbb{C}^{(N+1)n}$ and $y_j^{[i]} \in \mathbb{C}^n$ for $i = 1, \dots, j+1$. Then for all j such that $1 \le j \le N$ the solution x_j of the system

$$(\mathcal{A}_N - \sigma_j \mathcal{B}_N) x_j = y_j,$$

has the following structure

$$x_j = vec\left(x_j^{[1]}, x_j^{[2]}, \dots, x_j^{[j+1]}, 0, \dots, 0\right),$$

where again $x_j \in \mathbb{C}^{(N+1)n}$ and $x_j^{[i]} \in \mathbb{C}^n$ for $i = 1, \dots, j+1$.

Proof. Expand the linear system $(\mathcal{A}_N - \sigma_j \mathcal{B}_N) x_j = y_j$, in order to simplify the notation we define $\mu_i^{(j)} := \sigma_j - \sigma_i$ where $i = 0, \dots, N-1$, then we have

$$\left(\begin{array}{cccc|cccc}
A_0 & A_1 & \dots & A_j & A_{j+1} & A_{j+2} & \dots & A_N \\
-\mu_0^{(j)} I & I & & & & & & \\
 & \ddots & \ddots & & & & & \\
 & & -\mu_{j-1}^{(j)} I & I & & & & \\
\hline
 & & & 0 & I & & & \\
 & & & & -\mu_{j+1}^{(j)} I & I & & \\
 & & & & & \ddots & \ddots & \\
 & & & & & & -\mu_{N-1}^{(j)} I & I
\end{array}\right)
\left(\begin{array}{c}
x_j^{[1]} \\ x_j^{[2]} \\ \vdots \\ x_j^{[j+1]} \\ \hline x_j^{(j+2)} \\ x_j^{(j+3)} \\ \vdots \\ x_j^{N+1}
\end{array}\right)
=
\left(\begin{array}{c}
y_j^{[1]} \\ y_j^{[2]} \\ \vdots \\ y_j^{[j+1]} \\ \hline 0 \\ 0 \\ \vdots \\ 0
\end{array}\right).$$

At this point the variables $x_j^{[i]}$ for $i > j+1$ are the solution of the subsystem

$$\begin{pmatrix} I & & & \\ -\mu_{j+1}^{(j)} I & I & & \\ & \ddots & \ddots & \\ & & -\mu_{N-1}^{(j)} I & I \end{pmatrix} \begin{pmatrix} x_j^{[j+2]} \\ x_j^{[j+3]} \\ \vdots \\ x_j^{[N+1]} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

The matrix that defines this system is not singular so that the only solution is $x_j^i = 0$ for $i > j+1$

□

Now we apply the rational Krylov algorithm 10 for the GEP defined by the linearization matrices $\mathcal{A}_N$ and $\mathcal{B}_N$. To exploit the structure of such problem we choose a particular starting vector as suggested by lemma 3.1.2 and the interpolations point as shifts.

Lemma 3.1.2. *Consider the GEP defined by linearization $(\mathcal{A}_N, \mathcal{B}_N)$. We solve it with rational Krylov algorithm 10. We choose as shifts the interpolation points and as starting vector*

$$v_1 := vec\left(v_1^{[1]}, 0, \ldots, 0\right),$$

where $v_1 \in \mathbb{C}^{(N+1)n}$ and $v_1^{[1]} \in \mathbb{C}^n$. Then it holds that at the j-th step of rational Krylov algorithm 10 the vectors of the Arnoldi sequence have the following structure

$$v_k = vec\left(v_k^{[1]}, v_k^{[2]}, \ldots, v_k^{[j]}, 0, \ldots, 0\right), \quad for \ \ k \leq j,$$

where $v_k^{[i]} \in \mathbb{C}^n$ for $i = 1, \ldots, j$.

Proof. We prove this lemma by induction. For $j = 1$ there is nothing to prove. If $j = 2$ then v_1 is the starting vector, let v_2 be the solution of

$$(\mathcal{A}_N - \sigma_1 \mathcal{B}_N) v_2 = \mathcal{B}_N v_1.$$

We can expand this system, the effect of the matrix $\mathcal{B}_N$ is a block down–shift:

$$\begin{pmatrix} A_0 & A_1 & A_2 & A_3 & \ldots & A_N \\ -\mu_0^{(1)} I & I & & & & \\ & 0 & I & & & \\ & & -\mu_2^{(1)} I & I & & \\ & & & \ddots & \ddots & \\ & & & & -\mu_{N-1}^{(1)} & I \end{pmatrix} v_2 = \begin{pmatrix} 0 \\ v_1^{[1]} \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Using the previous lemma we have that the solution has the form

$$v_2 = \text{vec}\left(v_2^{[1]}, v_2^{[2]}, 0, \ldots, 0\right).$$

So that the only first two block of v_2 are nonzero. At this point v_2 is not the next vector of the Arnoldi sequence. We need to apply normalization and after that we modify v_1 that takes the form

$$v_1 = \text{vec}\ \left(v_1^{[1]}, v_1^{[2]}, 0, \dots, 0\right).$$

In fact, in our version of rational Krylov algorithm, we change basis every time we change shift. This concludes the proof for the case $j = 2$. Let now suppose that the lemma holds for j, then we have that the vectors v_k take the form

$$v_k = \text{vec}\ \left(v_k^{[1]}, v_k^{[2]}, \dots, v_k^{[j]}, 0, \dots, 0\right), \quad \text{for} \quad k \leq j.$$

Consider the solution of the system

$$(\mathcal{A}_N - \sigma_j \mathcal{B}_N) v_{j+1} = \mathcal{B}_N v_j,$$

and expand it so that it holds

$$\begin{pmatrix}
A_0 & A_1 & \dots & A_j & A_{j+1} & A_{j+2} & \dots & A_N \\
-\mu_0^j I & I & & & & & & \\
& \ddots & \ddots & & & & & \\
& & -\mu_{j-1}^{(j)} I & I & & & & \\
& & & 0 & I & & & \\
& & & & -\mu_{j+1}^{(j)} I & I & & \\
& & & & & \ddots & \ddots & \\
& & & & & & \mu_{N-1}^{(j)} I & I
\end{pmatrix} v_{j+1} = \begin{pmatrix} 0 \\ v_j^{[1]} \\ \vdots \\ v_j^{[j]} \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Using the previous lemma we have that the solution of such system has the form

$$v_{j+1} = \text{vec}\ (v_{j+1}^{[1]}, \dots, v_{j+1}^{[j+1]}, 0, 0, \dots, 0).$$

This is not the next vector of the Arnoldi sequence. We need to apply normalization and after that to change basis. This leads to a linear combination of the v_k for $k \leq j + 1$. Then the new vectors of the basis have the first $(k + 1)$–blocks generally nonzero while the remaining blocks are zero. □

Lemma 3.1.3. *At each iteration j of the rational Krylov algorithm 10, only the top-left parts of the matrices $\mathcal{A}_N - \sigma_j \mathcal{B}_N$ are used to compute the nonzero top parts $\tilde{v}_{j+1}$ of the vectors v_{j+1}, i.e.,*

$$(\mathcal{A}_j - \sigma_j \mathcal{B}_j) \tilde{v}_{j+1} = \mathcal{B}_j \tilde{v}_j,$$

where

$$\tilde{v}_{j+1} = vec\ \left(v_{j+1}^{[1]}, v_{j+1}^{[2]}, \dots, v_{j+1}^{[j+1]}\right),$$

and

$$\tilde{v}_j = vec\ \left(v_j^{[1]}, v_j^{[2]}, \dots, v_j^{[j]}, 0\right),$$

The proof of this lemma is a direct consequence of the previous lemma.

Lemma 3.1.4. *The linear system $(\mathcal{A}_j - \sigma_j \mathcal{B}_j)\tilde{v}_{j+1} = \mathcal{B}_j \tilde{v}_j$ can be efficiently solved by using the following equations*

$$A(\sigma_j) v_{j+1}^{[1]} = y_0^{(j)},$$

where

$$y_0^{(j)} = -\sum_{i=1}^{j} A_j \left(v_j^{[i]} + \sum_{k=1}^{i-1} \left(\prod_{l=k}^{i-1} \mu_l^{(j)} \right) v_j^{[k]} \right),$$

and

$$\begin{aligned} v_{j+1}^{[2]} &= v_j^{[1]} + \mu_0^{(j)} v_{j+1}^{[1]}, \\ v_{j+1}^{[3]} &= v_j^{[2]} + \mu_1^{(j)} v_{j+1}^{[2]}, \\ &\vdots, \\ v_{j+1}^{[j+1]} &= v_j^{[j]} + \mu_{j-1}^{(j)} v_{j+1}^{[j]}. \end{aligned}$$

Proof. Expand the system $(\mathcal{A}_j - \sigma_j \mathcal{B}_j)\tilde{v}_{j+1} = \mathcal{B}_j \tilde{v}_j$, recall that $\mathcal{B}_j$, perform a block down–shift and get

$$\begin{pmatrix} A_0 & A_1 & A_2 & \dots & A_j \\ -\mu_0^{(j)} I & I & & & \\ & -\mu_1^{(j)} I & I & & \\ & & \ddots & \ddots & \\ & & & -\mu_{j-1}^{(j)} & I \end{pmatrix} \begin{pmatrix} v_{j+1}^{[1]} \\ v_{j+1}^{[2]} \\ v_{j+1}^{[3]} \\ \vdots \\ v_{j+1}^{[j+1]} \end{pmatrix} = \begin{pmatrix} 0 \\ v_j^{[1]} \\ v_j^{[2]} \\ \vdots \\ v_j^{[j]} \end{pmatrix}.$$

Now the block equations from the second to the last yield

$$\begin{aligned} v_{j+1}^{[2]} &= v_j^{[1]} + \mu_0^{(j)} v_{j+1}^{[1]}, \\ v_{j+1}^{[3]} &= v_j^{[2]} + \mu_1^{(j)} v_{j+1}^{[2]}, \\ &\vdots, \\ v_{j+1}^{[j+1]} &= v_j^{[j]} + \mu_{j-1}^{(j)} v_{j+1}^{[j]}, \end{aligned}$$

that is a part of the thesis. Consider the first equation

$$\sum_{i=0}^{j} A_i v_{j+1}^{[i]} = 0.$$

Using the previous equations yields

$$\begin{aligned}
&A_1 v_{j+1}^{[1]} + A_2 v_{j+1}^{[2]} + A_3 v_{j+1}^{[3]} + \dots + A_j v_{j+1}^{[j]} = 0, \\
&A_1 v_{j+1}^{[1]} + A_2 (v_j^{[1]} + \mu_0^{(j)} v_{j+1}^{[1]}) + A_3 (v_j^{[2]} + \mu_1^{(j)} v_{j+1}^{[2]}) + \dots + A_j (v_j^{[j]} + \mu_{j-1}^{(j)} v_{j+1}^{[j]}) = 0.
\end{aligned}$$

Recursively replacing $v_{j+1}^{[i]}$ in the previous formula we get

$$\begin{aligned}
&\left(A_0 + \mu_0^{(j)} A_1 + \mu_0^{(j)} \mu_1^{(j)} A_2 + \mu_0^{(j)} \mu_1^{(j)} \mu_2^{(j)} A_3 + \dots + \mu_0^{(j)} \mu_1^{(j)} \dots \mu_{j-1}^{(j)} A_j\right) v_{j+1}^{[1]} = \\
&- A_1 v_j^{[1]} - A_2 \left(v_j^{[2]} + \mu_1^{[j]} v_j^{[1]}\right) - A_3 \left(v_j^{[3]} + \mu_2^{(j)} v_j^{[2]} + \mu_1^{(j)} \mu_2^{(j)} v_j^{[1]}\right) - \dots \\
&- A_j \left(v_j^{[j]} + \mu_{j-1}^{(j)} v_j^{[j-1]} + \mu_{j-2}^{(j)} \mu_{j-1}^{(j)} v_j^{[j-2]} + \dots + \mu_1^{(j)} \mu_2^{[j]} \dots \mu_{j-1}^{(j)} v_j^{[1]}\right).
\end{aligned}$$

Note that the left-hand side is equal to $P_N(\sigma_j)$, that is the evaluation of the polynomials that interpolate $A(\lambda)$, then it holds that $P_N(\sigma_j) = A(\sigma_j)$. The right part is what we wanted. $\square$

At this point we have an algorithm for the NLEP, we start by linearising $A(\lambda)$ in σ_0, after that we choose a starting vector $v_1 \in \mathbb{C}^n$ and a shift σ_1 and start the Arnoldi sequence.

At each step it holds

$$(\mathcal{A}_j - \sigma_j \mathcal{B}_j)^{-1} \mathcal{B}_j \tilde{V}_j = \tilde{H}_{j,j-1} V_{j-1},$$

and we need to expand the vectors v_i by adding zeros block so that at the j-th step it holds that $v_i \in \mathbb{C}^{jn}$. This process is summarized in the algorithm 12. At step 11 we use lemma 3.1.4 to effectively compute the next vector of the sequence. Moreover, if the same shift is repeated more times, then it is convenient to compute an LU factorization of $A(\sigma_i)$. Of course, we do not store the matrices $\mathcal{A}_j$ and $\mathcal{B}_j$ but we just need to store the divided difference matrices A_j. If the problem has the form 3.1 then we just need to store the interpolation coefficients $\alpha_{i,j}$.

In algorithm 12 we wanted to emphasize that the process can be understood as expansion and rational Krylov step. It is easy to see that this is the Rational Krylov algorithm 10 applied to a matrix that for every step becomes bigger and bigger. But the new elements that we add do not affect the previous elements of the basis V_j. We can look at the NLEP as to a linear eigenvalue problem with matrices of the pencil that have infinite elements. Because we have an (infinite) linear problem we can use also tipical strategy of linear algorithms as thick restart 2.3.

Algorithm 12 HIRK (Hermite Interpolation Rational Krylov Method)

```
1: Choose the shift $\sigma_0$ and the starting vector $v_1$.
2: for $j = 1, \dots, m$ do
3:    EXPANSION PHASE.
4:    Choose the shift $\sigma_j$.
5:    Compute the next divided difference: $A_j$.
6:    Expand $\mathcal{A}_j$, $\mathcal{B}_j$ and $V_j$.
7:    RATIONAL KRYLOV STEP
8:    if $\sigma_{j-1} \neq \sigma_j$ then
9:       Change the basis $V_j \to \tilde{V}_j$ and the matrix $H_{j,j-1} \to \tilde{H}_{j,j-1}$
         (according to the algorithm of Rational Krylov 10)
         such that the Arnoldi sequence is
             $(\mathcal{A}_j - \sigma_j \mathcal{B}_j)^{-1} \mathcal{B}_j \tilde{V}_j = \tilde{H}_{j,j-1} V_{j-1}.$
10:   end if
11:   Compute next vector of the sequence:
         $r = (\mathcal{A}_j - \sigma_j \mathcal{B}_j)^{-1} \mathcal{B}_j v_j,$
         $r = v - V_j h_j,$            where $h_j = V_j^H r$          orthogonalization,
         $v_{j+1} = r / h_{j+1,j},$    where $h_{j+1,j} = \|r\|$       normalization.
12:   Compute the eigenpairs $(\theta_i, y_i)$ for $i = 1, \dots, j$ of $H_{j,j-1}$ and then the Ritz
      pairs $(\theta_i, V_j y_i)$.
13:   Test the convergence for the NLEP.
14: end for
```

3.1.4 Exploiting low rank structure of coefficient matrices

Consider the NLEP defined by

$$A(\lambda) = \sum_{i=1}^{m} f_i(\lambda) B_i.$$

In many applications, the matrices B_i have a low rank structure, the goal of this section is to exploit this property. In general, we can write $A(\lambda)$ by splitting the polynomial part and nonpolynomial part as

$$A(\lambda) = \underbrace{\sum_{j=0}^{p} \lambda^j B_j}_{\text{polynomial}} + \underbrace{\sum_{j=1}^{m} f_j(\lambda) C_j}_{\text{nonpolynomial}}.$$

We will assume that matrices C_i have low rank revealing decompositions

$$C_j = L_j U_j^H \quad \text{with} \quad L_j \in \mathbb{C}^{n \times r_j} \ , \ U_j^H \in \mathbb{C}^{r_j \times n},$$

where r_j is the rank of C_j. Now let us interpolate the NLEP at the points $\sigma_0, \sigma_1, \dots, \sigma_N$. Let p_j be the polynomial that interpolates λ^j (note that the degree is less then p) and q_j the polynomial that interpolate f_j. Then we have

$$\begin{aligned} P_N(\lambda) &= \sum_{j=0}^{p} B_j p_j(\lambda) + \sum_{j=1}^{m} C_j q_j(\lambda) \\ &= \sum_{j=0}^{p} B_j \sum_{i=0}^{p} \beta_{i,j} n_i(\lambda) + \sum_{j=1}^{m} C_j \sum_{i=0}^{N} \gamma_{i,j} n_i(\lambda) \\ &= \sum_{i=0}^{p} \left(\sum_{j=0}^{p} \beta_{i,j} B_j \right) n_i(\lambda) + \sum_{i=0}^{N} \left(\sum_{j=1}^{m} \gamma_{i,j} C_j \right) n_i(\lambda). \end{aligned}$$

Now define

$$\widetilde{B}_i := \sum_{j=0}^{p} \beta_{i,j} B_j, \qquad \widetilde{C}_i := \sum_{j=1}^{m} \gamma_{i,j} C_j.$$

Then we have

$$P_N(\lambda) = \sum_{i=0}^{p} \widetilde{B}_i n_i(\lambda) + \sum_{i=0}^{N} \widetilde{C}_i n_i(\lambda) = \sum_{i=0}^{p} \left(\widetilde{B}_i + \widetilde{C}_i \right) n_i(\lambda) + \sum_{i=p+1}^{N} \widetilde{C}_i n_i(\lambda).$$

As we did in previous subsection $P_N(\lambda)$ define the PEP which we expect the eigenpairs approximate the eigenpairs of the NLEP. Define the matrices

$$\begin{aligned} \widetilde{L}_i &:= \left(\gamma_{i,1} L_1 | \gamma_{i,2} L_2 | \dots | \gamma_{i,m} L_m \right), \\ \widetilde{U} &:= \left(U_1 | U_2 | \dots | U_m \right), \end{aligned}$$

where $\widetilde{L}_i, \widetilde{U} \in \mathbb{C}^{n \times r}$ with $r = r_1 + \dots r_m$. At this point we are ready to extend the results of the previous subsection.

Theorem 3.1.2 (Low–rank companion-type linearization). *The pair $(\lambda, x) \neq 0$ is an eigenpair of the PEP if and only if $\widetilde{\mathcal{A}}_N \widetilde{y}_N = \lambda \widetilde{\mathcal{B}}_N \widetilde{y}_N$ where*

$$\widetilde{\mathcal{A}}_N = \begin{pmatrix} \widetilde{B}_0 + \widetilde{C}_0 & \widetilde{B}_1 + \widetilde{C}_1 & \dots & \widetilde{B}_p + \widetilde{C}_p & \widetilde{L}_{p+1} & \widetilde{L}_{p+2} & \dots & \widetilde{L}_N \\ \sigma_0 I & I & & & & & & \\ & \ddots & \ddots & & & & & \\ & & & \sigma_{p-1} I & I & & & \\ & & & & \sigma_p \widetilde{U}^H & I & & \\ & & & & & \ddots & \ddots & \\ & & & & & & \sigma_{N-1} I & I \end{pmatrix}$$

$$\widetilde{\mathcal{B}}_N = \begin{pmatrix} I & & & & & \\ & \ddots & & & & \\ & & I & & & \\ & & & \widetilde{U}^H & & \\ & & & & \ddots & \\ & & & & & I \end{pmatrix} \qquad \widetilde{y}_N = \begin{pmatrix} x \\ n_1(\lambda) x \\ \vdots \\ n_p(\lambda) x \\ n_{p+1}(\lambda) \widetilde{U}^H x \\ n_{p+2}(\lambda) \widetilde{U}^H x \\ \vdots \\ n_N(\lambda) \widetilde{U}^H \end{pmatrix}$$

Observation 3.1.2. If $N < p$ then the linearization is equivalent to the companion–tipe linearization done by theorem 3.1.1. If $N > p$ the size of the identities on the block diagonal are different, this point will be emphasized later on.

Lemma 3.1.1 holds with the difference that this time the vectors have different size block structure, in particular $y_j^{[i]}, x_j^{[i]} \in \mathbb{C}^n$ if $i \leq p+1$ and $y_j^{[i]}, x_j^{[i]} \in \mathbb{C}^r$ if $i \geq p+2$. As in lemma 3.1.2 we choose as starting vector

$$v_1 = \text{ vec } \left(v_1^{[1]}, 0, \dots, 0 \right),$$

and again we have that at the j-th step of rational Krylov algorithm 10 the vectors of Arnoldi sequence have the following structure

$$v_k = \text{vec } \left(v_k^{[1]}, v_k^{[2]}, \dots, v_k^{[j]}, 0, \dots, 0 \right), \quad \text{for} \quad k \leq j,$$

where $v_k^{[i]} \in \mathbb{C}^n$ if $i \leq p+1$ and $v_k^{[i]} \in \mathbb{C}^r$ if $i \geq p+2$. Moreover lemma 3.1.3 holds.

These results are easy to prove, the critical part is to compute a formula to extend the Arnoldi sequence as in lemma 3.1.4.

Lemma 3.1.5. *The linear system $(\mathcal{A}_j - \sigma_j \mathcal{B}_j) \tilde{v}_{j+1} = \mathcal{B}_j \tilde{v}_j$ can be efficiently solved by using the following equations*

$$A(\sigma_j) v_{j+1}^{[1]} = y_0^{(j)},$$

where

$$
\begin{aligned}
y_0^{(j)} = & -\sum_{i=1}^{p}\left(\widetilde{B}_i+\widetilde{C}_i\right)\left(v_j^{[i]}+\sum_{k=1}^{i-1}\left(\prod_{l=k}^{i-1}\mu_l^{(j)}\right)v_j^{[k]}\right)+\\
& -\widetilde{C}_{p+1}\left(v_j^{[p+1]}+\sum_{k=1}^{p}\left(\prod_{l=k}^{p}\mu_l^{(j)}\right)v_j^{[k]}\right)+\\
& -\sum_{i=p+2}^{j}\widetilde{C}_i\left(\sum_{k=1}^{p+1}\left(\prod_{l=k}^{i-1}\mu_l^{(j)}\right)v_j^{[k]}\right)+\\
& -\sum_{i=p+2}^{j}\widetilde{L}_i\left(v_j^{[i]}+\sum_{k=p+2}^{i-1}\left(\prod_{l=k}^{i-1}\mu_l^{(j)}\right)v_j^{[k]}\right)
\end{aligned}
$$

and

$$
\begin{aligned}
v_{j+1}^{[k+1]} &= v_j^{[k]}+\mu_{k-1}^{(j)}v_{j+1}^{[k]} && \textit{if } j\neq p+2\\
v_{j+1}^{[p+2]} &= \widetilde{U}^H\left(v_j^{[p+1]}+\mu_p^{(j)}v_{j+1}^{[p+1]}\right)
\end{aligned}
$$

The advantage in exploiting the low rank property is that the vectors of the sequence are shorter and the Gram–Smith process is faster.

3.2 Iterative projection methods

In this section we will present another class of algorithms based on Rational Krylov method for the linear case. These algorithms are fast but less effective and work just on some NLEP. In particular they are based on the approximation of $A(\lambda)$ with a linear interpolation and as we expect they work nice if the NLEP is a small perturbation of a linear problem or if we are interested in eigenvalues in a region where the NLEP is almost linear.

The original idea of Axel Ruhe can be found in [14] and other works of the same author. We prefer to follow the description given by Elias Jarlebring in [6] and [5]. In this class of algorithms it is exended the idea of Arnoldi algorithms for the nonlinear case and, as we will explain, it is possible to change shift at every step solving a projected eigenvalue problem.

In order to understand the most efficient algorithm that we will call *Nonlinear Rational Krylov*, or shortly, NLRK, we will present firstly an inefficient algorithm called *regula falsi*. At the end of the chapter will be showed that NLEP is a particular case of a largest class of algorithms called *iterative projection algorithms*.

3.2.1 Regula falsi

Given the matrix-function $A(\lambda)$ we use the Lagrange interpolation between two points σ (pole) and λ_1 (shift), then we have

$$A(\lambda) = \frac{\lambda - \lambda_1}{\sigma - \lambda_1} A(\sigma) + \frac{\lambda - \sigma}{\lambda_1 - \sigma} A(\lambda_1) + \text{highter order terms.}$$

If we neglect the higher order terms we can approximate the NLEP $A(\lambda)x = 0$ with

$$A(\sigma)^{-1} A(\lambda_1) x = \theta x,$$

where $\theta = (\lambda - \lambda_1)/(\lambda - \sigma)$. In other words we can solve the GEP given by the pencil $(A(\sigma), A(\lambda_1))$ and then if (θ, x) is an eigenpair of such problem we compute

$$\lambda = \lambda_1 + \frac{\theta}{1-\theta}(\lambda_1 - \sigma),$$

and (λ, x) is an approximation of an eigenpair of the NLEP. For large θ we have that the approximation given by λ is near σ and for small θ we have that λ approximate eigenvalues near λ_1. In general the other eigenvalues of the linearization do not provide a good approximation for the NLEP. Then we are interested only in the smallest eigenvalues of the linearized problem. As we know from theorem 2.2.3 the Arnoldi algorithm is suitable for this task. Then we will use the Arnoldi algorithm to compute the outermost eigenvalues of the linearization provided by $(A(\sigma), A(\lambda_1))$, in particular we will choose the smallest λ_2 and we will use it as next shift. This is the idea of the regula falsi, we can summarize this process in the algorithm 13.

Algorithm 13 Regula falsi

```
1: Choose a pole $\sigma$ and the first shift $\lambda_1$.
2: for $j = 1, \dots, m$ do
3:     Compute $\theta_j$ the smallest (in norm) eigenvalue of the linearization provided by
       $(A(\sigma), A(\lambda_{j-1}))$
4:
       $$\lambda_j = \lambda_{j-1} + \frac{\theta}{1-\theta}(\lambda_{j-1} - \sigma),$$
5: end for
```

Then the strategy is to choose σ and θ_1 inside the zone of interest Ω where we want to compute the eigenvalues of the NLEP. After each step, we can change shift in order to improve the approximation given by λ_j. With this algorithm we can approximate one eigenvalue of the NLEP or in general the eigenvalues near λ_j. It turns out that this is not an efficient algorithm, we can use it if the problem is not big or as rafinement method.

3.2.2 Nonlinear Rational Krylov (NLKR)

The idea is to merge the Arnoldi and regula falsi algorithm. We will perform steps of the Arnoldi algorithm by changing shift at every iteration like in regula falsi and we also update the projection matrix (this step is not needed in linear case).

Define the matrix–function $T(\lambda) := A(\sigma)^{-1}A(\lambda)$. As we already said, the shift λ_j is changed at every step. Let us now introduce a new definition

Definition 3.2.1 (Generalized Arnoldi's sequence)**.** *Given the matrix–function $A(\lambda)$, a shift λ_m and a pole σ, then a sequence of vectors $v_1, \dots, v_m$ is called generalized Arnoldi's sequence if it exists a Hessenberg matrix $H_{m+1,m}$ with positive elements in the subdiagonal such that*

$$A(\sigma)^{-1}A(\lambda_m)V_m = V_{m+1}H_{m+1,m}$$

Now we will show how to generate inductively a generalized Arnoldi sequence. Suppose we have a generalized Arnoldi sequence (this can hold approximately) of length $j-1$.

$$T(\lambda_{j-1})V_{j-1} = V_j H_{j,j-1}.$$

Then we want to perform another step of Arnoldi sequence by changing shift. For the nonlinear case we expect that it is needed to upload the projection matrix $H_{j,j-1}$. Then in the next step we require that

$$T(\lambda_j)V_j = V_{j+1}\bar{H}_{j+1,j}. \tag{3.2}$$

Now we propose a method to update the projection matrix. Suppose we can express

$$\bar{H}_{j+1,j} = \begin{pmatrix} \alpha H_{j,j-1} - \beta I_{j,j-1} & k_j \\ 0 & \|r_\perp\| \end{pmatrix}, \tag{3.3}$$

where

$$k_j = V_j^H r_j, \quad r_\perp = r_j - V_j V_j^H r_j, \quad r_j = T(\lambda_j)v_j.$$

Note that in the linear case we choose $\alpha = 1$ and $\beta = 0$. We want to choose α and β in order to approximately fulfill the generalized Arnoldi sequence. If we substitute (3.3) in (3.2) in order to approximately fulfill the generalized Arnoldi's sequence we get

$$\begin{aligned} T(\lambda_j)V_j &= [T(\lambda_j)V_{j-1}, r_j] = \\ V_{j+1}\bar{H}_{j+1,j} &= [V_j(\alpha H_{j,j-1} - \beta I_{j,j-1}), V_j k_j + \|r_\perp\| v_{j+1}] \\ &= [\alpha T(\lambda_{j-1})V_{j-1} - \beta V_{j-1}, r_j] \\ &= [(\alpha T(\lambda_{j-1}) - \beta I)V_{j-1}, r_j]. \end{aligned}$$

If we impose that the relation holds componentwise we get

$$T(\lambda_j)V_{j-1} = (\alpha T(\lambda_{j-1}) - \beta I)V_{j-1}.$$

Then a sufficient condition in order that this relation is satisfied is to impose that

$$T(\lambda_j) = \alpha T(\lambda_{j-1}) - \beta I.$$

At this point, in order to find a reasonable choice of the parameters α and β we have to Lagrange–interpolate $T(\lambda)$ between λ_{j-1} and σ and to evaluate in λ_j. Notice that $T(\sigma) = I$. Then we have

$$T(\lambda_j) = \frac{\lambda_j - \sigma}{\lambda_{j-1} - \sigma} T(\lambda_{j-1}) - \frac{\lambda_j - \lambda_{j-1}}{\lambda_j - \sigma} I.$$

Therefore a reasonable and possible choice of such parameters is

$$\begin{cases} \alpha &= \dfrac{\lambda_j - \sigma}{\lambda_{j-1} - \sigma} \\ \\ \beta &= \dfrac{\lambda_j - \lambda_{j-1}}{\lambda_j - \sigma} \end{cases}$$

Now we design a strategy to choose the shifts. The first one will be chosen in the region of interest. In general we wish that λ_j is an approximation of one eigenvalue of $A(\lambda)$ and we already have shown that with the Lagrange–interpolation when we compute the eigenvalues of the pencil $(A(\sigma), A(\lambda_j))$, the smallest θ is such that

$$\lambda_{j+1} = \lambda_j + \frac{\theta}{1-\theta}(\lambda_j - \sigma)$$

is an estimation of one eigenvalue of the NLEP near λ_j. Then with this algorithm the sequence λ_j will estimate better and better an eigenvalue of the NLEP.

With this choice of the next shift we have a better way to write the parameters

$$\begin{cases} \alpha &= \dfrac{1}{1-\theta} \\ \\ \beta &= \dfrac{\theta}{1-\theta} \end{cases}$$

At this point we can understand why NLRK works well just with NLEP that are a small perturbation of linear eigenvalues problem. In fact the point is that a linear approximation is good if we are approximating a near–linear function. Sometimes it can be difficult to understand when this can be done. A practical way to check it is to choose a tolerance (like $tol = 10^{-5}$ or bigger, depending on the problem) and stop the algorithm when

$$\|A(\lambda_j)V_j - A(\sigma)V_{j+1}H_{j+1,j}\| > tol.$$

Sometimes in order to avoid this problem we can take the same shift for a few steps. We are ready to write a preliminary version of the NLRK see algorithm 14 (this is actually the first version of the algorithm proposed by Ruhe [13]).

The updating process does not work well for the initial steps in the iteration as the approximation varies a lot. A naive way to solve this problem is perform at the start a couple of exact steps without changing shift. That is the reason for the if–statement in step 8, MINIT is selected as small as possible.

Algorithm 14 NLRK (preliminary version)

```
1: Choose a starting vector $v_1$ with $\|v_1\| = 1$, a starting shift $\lambda_1$ and a pole $\sigma$.
2: $r = A(\sigma)^{-1}A(\lambda_1)v_1$
3: for $j = 1, 2, \dots$ until convergence do
4:    Orthogonalize $h_j = V^H r$, $r_\perp = r - Vh_j$
5:    $h_{j+1,j} = \|r_\perp\|$
6:    Compute the smallest (in norm) eigenpair $(\theta, y)$ of $H_{j,j}$
7:    Set $(\theta, V_j y)$ (Ritz pair)
8:    if $j > MINIT$ then
9:
          $$\lambda_{j+1} = \lambda_j + \frac{\theta}{1-\theta}(\lambda_j - \sigma)$$
10:       Update $H$
          $$H_{j+1,j} = \frac{1}{1-\theta}H_{j+1,j} - \frac{\theta}{1-\theta}I_{j+1,j}$$
11:   else
12:       $\lambda_{j+1} = \lambda_j$
13:   end if
14:   $v_{j+1} = r_\perp/\|r_\perp\|$
15:   $r = A(\sigma)^{-1}A(\lambda_{j+1})v_{j+1}$
16: end for
```

In this version of the algorithm at each step the eigenpairs of the projection of the pencil $(A(\sigma), A(\lambda_j))$ are computed, that is a linearization of the NLEP. It means that we compute the eigenpairs of $VA(\sigma)^{-1}A(\lambda_j)V^H$ that is an approximation of the projected NLEP $V^H A(\sigma)^{-1}A(\lambda)V$. The idea to improve the algorithm is to compute instead the eigenpairs of the projected original NLEP. This means that we compute directly the eigenpairs of $V^H A(\sigma)^{-1}A(\lambda)V$. Note that the projected NLEP is small sized.

We have defined the generalized Arnoldi sequence 3.2.1 as

$$T(\lambda_j)V_j = V_{j+1}H_{j+1,j},$$

In the next step we have (by updating the projection H as in algorithm 14 step 10)

$$T(\lambda_{j+1})V_{j+1} = V_{j+2}\bar{H}_{j+2,j+1}.$$

It is not difficult, by computing componentwise, to obtain

$$T(\lambda_{j+1})V_j = V_j\bar{H}_{j,j} + v_{j+1}e_j^H. \tag{3.4}$$

It is worth pointing out that these relations hold approximately and that they became sharp in the linear case.

Consider the linear case and let (θ, s) be an eigenpair of $H_{j,j}$ (before the update, algorithm 14 step 7). After the update, it holds that $\bar{H}_{j,j}s = 0$, and multiplying (3.4) by s yields

$$T(\lambda_{j+1})V_j s = h_{j+1}v_{j+1}e_j^H s = h_{j+1}v_{j+1}s_j. \tag{3.5}$$

Equation (3.5) tells us that $T(\lambda_{j+1})V_j s$ is orthogonal to V_j (because is a multiple of v_{j+1}). This does not hold in the nonlinear case but we can enforce it to hold.

From (3.5) we have that

$$T[V_{j-1}\ V_j s] = V_j[\bar{H}_{j,j-1}\ k_j] + h_{j+1,j}s_j v_{j+1}e_j^H, \tag{3.6}$$

where $k_j = V_j^H T(\lambda_{j+1})V_j s$. We can rewrite (3.6) as

$$TV_j \begin{pmatrix} I_{j-1} & \tilde{s} \\ 0 & s_j \end{pmatrix} = V_j[\bar{H}_{j,j-1}\ k_j] + h_{j+1,j}s_j v_{j+1}e_j^H, \tag{3.7}$$

where $\tilde{s}$ is the leading $j-1$ vector of s. Now we multiply by the inverse of the matrix in brackets from the right and by V_j^H and we get

$$V_j^H T(\lambda_{j+1})V_j = [\bar{H}_{j,j-1}\ k_j] \begin{pmatrix} I_{j-1} & -s_j^{-1}\tilde{s} \\ 0 & s_j^{-1} \end{pmatrix} = [\bar{H}_{j,j-1} \quad -s_j^{-1}\bar{H}_{j,j-1}\tilde{s} + s_j^{-1}k_j].$$

Using that $\bar{H}_{j,j}s = \bar{H}_{j,j-1}\tilde{s} + s_j h_j = 0$ we have

$$V_j^H T(\lambda_{j+1})V_j = [H_{j,j-1}\ \ h_j - s_j^{-1}k_j].$$

Notice that in the linear case $k_j = 0$ and all these computations are useless.

The idea of Ruhe ([14] and [5]) is to choose the next vector of the sequence (algorithm 14 step 15) as $r := T(\lambda_{j+1})V_j s$, $v_{j+1} := r/\|r\|$, (according to (3.5)) $h_{j+1,j} := \|r\|/s_j$ and to modify the last column of $\bar{H}_{j,j}$ by replacing it with $h_j - s_j^{-1}k_j$. After that (λ_{j+1}, s) and $\bar{H}$ will be updated according to the steps 7, 9 and 10 of algorithm 14. These operations will be repeated till $\|k_j\|$ becomes small enough. These iterations are called *inner iterations.* We can summarize this in the final version of the algorithm NLRK 15. In the next subsection we will explain how to change pole.

As we already said, the inner iterations are inside the **while**, instead the rest is the outer iteration.

The strategy to choose the next θ and s (step 20) is to select the Ritz value not converging near to the current pole.

Lemma 3.2.1. *If the* inner iterations *converges, then it converges to a pair* $(\widehat{\lambda}, x)$*, where* $x = Vs$*, and* $(\widetilde{\lambda}, s)$ *is an eigenpair of the projected nonlinear eigenproblem* $V_j^H T(\lambda)V_j$*.*

Proof. At the i–th inner iteration (and j-th outer iteration) it holds

$$\|k_j^{(i)}\| = \|V_j^H T(\lambda^{(i)})V_j s\|.$$

If $\|k_j^{(i)}\| \to 0$ then $V_j^H T(\widehat{\lambda})V_j s = 0$. □

Algorithm 15 NLRK (final version)

```
1: Choose a starting vector $v_1$ with $\|v_1\| = 1$, a starting shift $\lambda_1$ and a pole $\sigma$ and
   set $j = 1$.
2: OUTER ITERATION
3: Set $h_j = 0; s = e_j = (0, \dots, 0, 1)^H \in \mathbb{R}^j; x = v_j$;
4: Compute $r = A(\sigma)^{-1}A(\lambda)x$ and $k_j = V_j^H r$
5: while $\|k_J\| > ResTol$ do
6:    INNER ITERATION
7:    Orthogonalize $r = r - Vk_j$
8:    Set $h_j = h_j + s_j^{-1}k_j$
9:    Compute $(\theta, s)$ smallest eigenpair of $H_{j,j}$
10:   $x = V_j s$
11:   Update $\lambda = \lambda + \frac{\theta}{1-\theta}(\lambda - \theta)$
12:   Update $H_{j,j} = \frac{1}{1-\theta}H_{j,j} - \frac{\theta}{1-\theta}I$
13:   Compute $r = A(\sigma)^{-1}A(\lambda)x$ and $k_j = V_j^H r$
14: end while
15: Compute $h_{j+1,j} = \|r\|/s_j$
16: if $|h_{j+1,j}s_j| > EigTol$ then
17:    $v_{j+1} = r/\|r\|$; $j = j + 1$; GOTO 3
18: end if
19: Store $(\theta, x)$ as eigenpair
20: If more eigenvalues are wanted, choose the next $\theta$ and $s$, and GOTO 10
```

It can happen that the inner iterations do not converge. For this reason, in the implementation it is convenient to choose a maximum number of inner iterations. In case we exceed this number the algorithm will continue with the outer iterations. In certain cases it happens that the inner iterations do not converge at a step j but the algorithm still works. So far there is no proofs/condition to have convergence of the inner iteration. In our numerical test it was sufficient to set the maximum number of inner iterations to 20.

Remark 3.2.1. *In the first version or NLRK (algorithm 14) we choose next shift as the smallest eigenvalue of the projected linearization of the NLEP. In the final version of NLEP (algorithm 15) we choose the next shift as an eigenvalue of the projected NLEP. Also in this last version of the algorithm for every step we have that the generalized Arnoldi sequence 3.2.1 approximately holds.*

Updates of pole

Consider the problem of updating the shift and the pole. If we already performed j steps of the algorithm and the current pole–shift is (σ, λ) the generalized Arnoldi sequence holds approximately

$$A(\sigma)^{-1}A(\lambda)V_j = V_{j+1}H_{j+1,j}.$$

Now we want to change the shift–pole and use the pair $(\bar{\sigma}, \bar{\lambda})$, the idea to do that is very similar to the linear case (chapter 2 section 2.6). Under the hypothesis that the NLEP is a small perturbation of a linear eigevalue problem and that the new pair $(\bar{\sigma}, \bar{\lambda})$ is near the old one (σ, λ), then it holds approximately (Lagrange interpolation) that

$$\begin{aligned} A(\lambda) &= \frac{\lambda - \bar{\sigma}}{\bar{\lambda} - \bar{\sigma}} A(\bar{\lambda}) + \frac{\bar{\lambda} - \lambda}{\bar{\lambda} - \bar{\sigma}} A(\bar{\sigma}) \\ A(\sigma) &= \frac{\sigma - \bar{\sigma}}{\bar{\lambda} - \bar{\sigma}} A(\bar{\lambda}) + \frac{\bar{\lambda} - \sigma}{\bar{\lambda} - \bar{\sigma}} A(\bar{\sigma}) \end{aligned}$$

Substituting these relations in the generalized Arnoldi sequence we have

$$A(\bar{\lambda}) V_{j+1} K_{j+1,j} = A(\bar{\sigma}) V_{j+1} L_{j+1,j}, \tag{3.8}$$

where

$$\begin{aligned} K_{j+1,j} &= \frac{\lambda - \bar{\sigma}}{\bar{\lambda} - \bar{\sigma}} I_{j+1,j} - \frac{\sigma - \bar{\sigma}}{\bar{\lambda} - \bar{\sigma}} H_{j+1,j}, \\ L_{j+1,j} &= \frac{\bar{\lambda} - \sigma}{\bar{\lambda} - \bar{\sigma}} H_{j+1,j} - \frac{\bar{\lambda} - \lambda}{\bar{\lambda} - \bar{\sigma}} I_{j+1,j}. \end{aligned}$$

At this point, starting from (3.8) with a QR factorization of $K_{j+1,j}$ we get

$$A(\bar{\lambda}) V_{j+1} Q_{j+1,j+1} \begin{pmatrix} R_{j,j} \\ 0 \end{pmatrix} = A(\bar{\sigma}) V_{j+1} L_{j+1,j},$$

and then

$$A(\bar{\lambda}) V_{j+1} Q_{j+1,j} = A(\bar{\sigma}) V_{j+1} Q_{j+1,j+1} Q_{j+1,j+1}^H L_{j+1,j} R_{j,j}^{-1}.$$

Using theorem 2.3.1 on the matrix $Q_{j+1,j+1}^H L_{j+1,j} R_{j,j}^{-1}$, we have

$$A(\bar{\lambda}) V_{j+1} Q_{j+1,j} = A(\bar{\sigma}) V_{j+1} \begin{pmatrix} P_{j,j} & 0 \\ 0 & 1 \end{pmatrix} \widetilde{H}_{j+1,j} P_{j,j}^H.$$

Let us define the new basis as

$$W_{j+1} = V_{j+1} Q_{j+1,j+1} \begin{pmatrix} P_{j,j} & 0 \\ 0 & 1 \end{pmatrix}.$$

Then we finally have

$$A(\bar{\lambda}) W_j = A(\sigma) W_{j+1} \widetilde{H}_{j+1,j}.$$

This last relation is a generalized Arnoldi sequence with the new shift–pole $(\bar{\lambda}, \bar{\sigma})$. In practical applications the shift is never changed we just change the pole. The same pole will be fixed for few step till enough Ritz values have converged. Then as before it can be useful to compute a sparse LU factorization (if needed reducing the bandwidth). Usually the strategy to change the pole depends on the problem. A naive idea can be to choose a convex combination between the old pole and the new shift.

Thick restart

During the execution of the algorithm NLRK it can happen that the dimension of the basis V_j becomes too big and a restart is needed. Our aim is to not discard information taken till now. Therefore we want to lock converged Ritz pairs and purge the others. We can repeat exacly the same computations done in the linear case (see subsection 2.3).

Now we will show how to make it in the nonlinear case. For details we refer to the linear case already explained before.

Let us start from the generalized Arnoldi sequence

$$A(\sigma)^{-1}A(\lambda_j)V_j = V_{j+1}H_{j+1,j},$$

with the aim to lock the Ritz values $\theta_1, \ldots, \theta_k$. Then we have (subsection 2.3))

$$A(\sigma)^{-1}A(\lambda_j)V_jP_{j,j} = V_{j+1}\begin{pmatrix} P_{j,j} & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} \theta_1 & * & * & * & * & \ldots & * \\ & \theta_2 & * & * & * & \ldots & * \\ & & \ddots & * & * & \ldots & * \\ & & & \theta_k & * & \ldots & * \\ & & & & * & \ldots & * \\ & & & & * & \ldots & * \\ * & * & \ldots & * & * & \ldots & * \end{pmatrix}.$$

If we define

$$W_{j+1} = \begin{pmatrix} P_{j,j} & 0 \\ 0 & 1 \end{pmatrix} V_{j+1},$$

and $\widetilde{W}_{k+1} = [w_1|\ldots|w_k|w_{j+1}]$ then we have

$$A(\sigma)^{-1}A(\lambda_j)\widetilde{W}_k = \widetilde{W}_{k+1}\begin{pmatrix} \theta_1 & * & * & * \\ & \theta_2 & * & * \\ & & \ddots & * \\ & & & \theta_k \\ * & * & \ldots & * \end{pmatrix}.$$

Using theorem 2.3.1 we have

$$A(\sigma)^{-1}A(\lambda_j)\widetilde{W}_kP_{k,k} = W_{k+1}\begin{pmatrix} P_{k,k} & 0 \\ 0 & 1 \end{pmatrix}\widetilde{H}_{k+1,k}.$$

Let us define

$$\widetilde{V}_{k+1} := \widetilde{W}_{k+1}\begin{pmatrix} P & 0 \\ 0 & 1 \end{pmatrix},$$

Then we have

$$A(\sigma)^{-1}A(\lambda_j)\widetilde{V}_k = \widetilde{V}_{k+1}\widetilde{H}_{k+1,k}.$$

The eigenvalues of $\widetilde{H}_{k,k}$ are the Ritz values that we wanted to lock. Heuristically it seems that a hard purging must be done. It means that every time that a Ritz value converges we lock it together with the other converged Ritz values and purge all the others except the second smallest one (that still did not converged), that hopefully will be the next to converge.

We recall that in the locking process we cannot chose only converged Ritz values otherwise a break down occurs. Moreover if θ_1 is the smallest eigenvalue of $H_{j+1,j}$ and it converged, and if θ_2 is the second smallest eigenvalue, then the idea is to change shift according to step 9 of the algorithm 15 and update H according to step 10. At this point we can complete the algorithm 15. When a Ritz pair converges a hard purge will be performed and after a few steps we can change pole.

3.2.3 Iterative projection method

As explained in previous section, performing inner iterations is a way to solve the projected NLEP. Then the idea of Jarlebring [6] is to use also other algorithms to solve such projected problem, see algorithm 16.

Algorithm 16 Iterative projection method

```
1: Choose a unitary starting vector $v_1$, initial shift $\lambda$ and pole $\sigma$
2: for $j = 1, 2, \dots$ untill convergence do
3:     Solve the projected eigenproblem $V^H A(\sigma)^{-1} A(\lambda) V s = 0$ for $(\lambda, s)$
4:     Compute the Ritz vector $x = Vs$ and the residual $r = A(\sigma)^{-1} A(\lambda) x$
5:     Orthogonalize $r = r - VV^H r$
6:     Expand the search space $V = [V, r/\|r\|]$
7: end for
```

It is clear that NLRK is a particular case of algorithm 16 where the inner iterations are used to solve the projected eigenproblem. Moreover it is worth pointing out that projected eigenproblems are small sized so that we can use more expensive/sharp algorithms. For details see [6].

Chapter 4

Applications

In this chapter, a few application of linear and nonlinear eigenproblems are presented together with some experiments in order to test the algorithms presentend in this thesis. In particular it is pointed out when the Rational Krylov works better than the shifted–and–inverted Arnoldi. In the nonlinear case, it is shown how to use the algorithms presented and which errors can occur. All the implementations of the algorithms are done in a matlab-environment [8] on a quad core amd athlon ii x4 640 processor running Linux. For the last example of the set reported in this chapter, it was used FreeFem++ [4] to discretize the problem.

4.1 Linear eigenvalue problems

A few non-Hermitian eigenvalue problems can be found in the collection [1], we tested the algorithms on few of them and on a classic problem of fluid dynamics.

4.1.1 Tubolar reactor model

Let us start with a problem that arises from computational fluid dynamics [1].

The conservation of reactant and energy in a homogeneous tube of length L in dimensionless form is modeled by the differential equation:

$$\begin{aligned}
\frac{L}{v}\frac{dy}{dt} &= -\frac{1}{Pe_m}\frac{\partial^2 y}{\partial X^2} + \frac{\partial y}{\partial X} + Dye^{\gamma-\gamma T^{-1}},\\
\frac{L}{v}\frac{dT}{dt} &= -\frac{1}{Pe_h}\frac{\partial^2 T}{\partial X^2} + \frac{\partial T}{\partial X} + \beta(T-T_0) - BDye^{\gamma-\gamma T^{-1}},
\end{aligned}$$

where y and T represent concentration and temperature, and $0 \le X \le 1$ denote the spatial coordinate. The boundary conditions are $y'(0) = Pe_m y(0)$, $T'(0) = Pe_h T(0)$, $y'(1) = 0$ and $T'(1) = 0$. The parameters in the differential equation are set to $Pe_m = Pe_h = 5$, $B = 0,5$, $\gamma = 25$; $\beta = 3,5$ and $D = 0,2662$.

The method of lines is used for the numerical solution of this differential equation and central differences are used to semi–discretize it in space.

For $x = (y_1, T1, y_2, T_2, \dots, y_{N/2}, T_{N/2})$ the equations can be written as $\dot{x} = Ax$. With the discretization step $h = 1/50$ we get the Jacobi matrix $A \in \mathbb{R}^{100\times100}$ where A is a banded matrix with bandwidth 5. In order to choose a stable time discretization it is needed to compute the rightmost eigenvalues. In particular it can be needed to compute a big number of eigenvalues in order to choose a time discretization with a large step Δt that is also stable.

We already know that all eigenvalues have negative real part. This small problem is useful because we can use the algorithms without restart and we can understand how they work with a real problem.

We compare three Algorithms: classic Arnoldi, shift–and–invert Arnoldi and Rational Krylov without using restarting strategies. We consider a Ritz pair (θ, z) converged if $\omega_m(y) \leq 10^{-12}$ (see 2.1.2) that hopefully means that $\|Az - \theta z\| \approx 10^{-12}$ (in general it is bigger because of the rounding off errors in the LU factorizations and solution of linear systems).

Consider the Arnoldi algorithm with 20 iterations, starting from a random vector and compute the Ritz values. In figure 4.1 we can see the results. The Ritz values are bad approximation of eigenvalues. To undertand why, it is needed to see theorem 2.2.3. In general the outermost eigenvalues will be well approximated after a few steps under the condition that they are enough separated from the others, this is not the case.

In conclusion with 20 steps Arnoldi's algorithm does not provide approximation to the eigenvalues since no Ritz value converged.

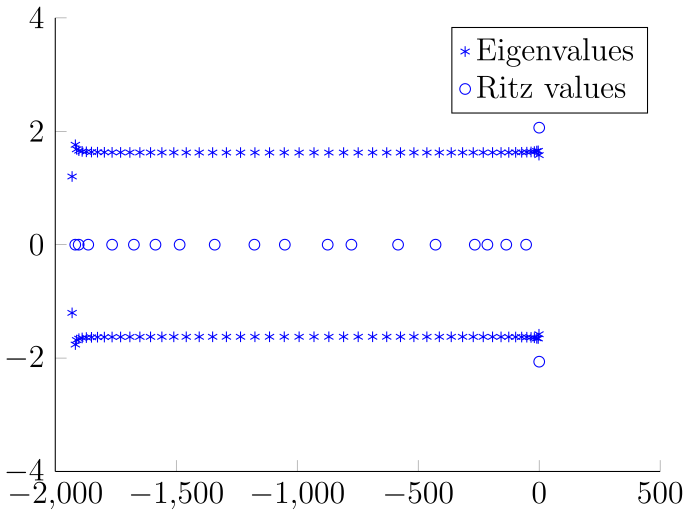


Figure 4.1: Arnoldi with $m = 20$

Consider the shift–and–invert Arnoldi without change of shift, so we are looking for eigenvalues near zero. The results of algorithm are in Figure 4.1.1, 6 Ritz values are marked as converged but for two of them $\|Az - \theta z\| \approx 10^{-11}$. Moreover 2 Ritz values are near convergence with $\|Az - \theta z\| \approx 10^{-6}$.

Let now consider the Shift–and–invert Arnoldi with 20 iterations and changing shifts, we start with the shift zero and change shift every 5 steps taking as new shift the average of unconverged Ritz values. The results are in Figure 4.3. We have no convergence and the closer Ritz value near convergence satisfies $\|Az - \theta z\| \approx 10^{-1}$.

Now we apply Rational Krylov with 20 steps, starting with the shift zero and changing shift every 5 steps taking as new shift the average of unconverged Ritz

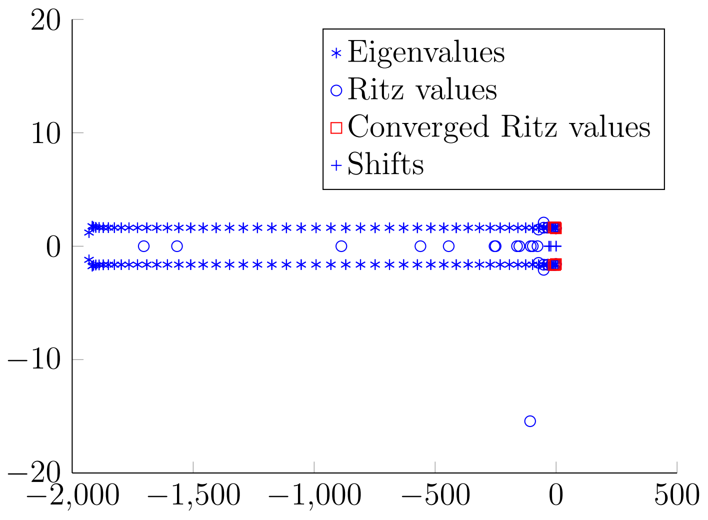


Figure 4.2: Shift–and–invert Arnoldi with $m = 20$

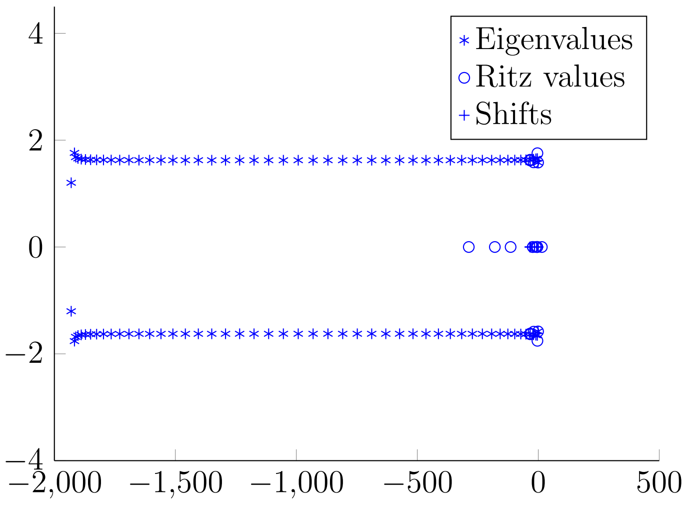


Figure 4.3: Shift–and–invert Arnoldi with $m = 20$ and 5 shifts

values. The results are in Figure 4.4. In this case the situation is much better, in fact 8 Ritz values converge and approximate the 8 rightmost eigenvalues. Concerning accuracy, we notice that for the converged Ritz values it holds $\|Az - \theta z\| \approx 10^{-9}$ that is a good result. Moreover others 2 Ritz values are near convergence with $\|Az - \theta z\| \approx 10^{-6}$.

In conclusion for this example we have that Rational Krylov works better but the eigensolution have to be refined, moreover changing shift with shift–and–invert is not a good idea. There is a light difference between Rational Krylov and shift–and–invert Arnoldi without shift, the reason is because in both cases we start to look from eigenvalues near zero. If we start a little far from origin, Rational Krylov is faster. Anyway in this example we wanted to show that also for an extimation of outermost eigenvalues, Arnoldi algorithm can fail. It is interesting to point out that it was not used the optimal strategy to change shift as explained in the subsection

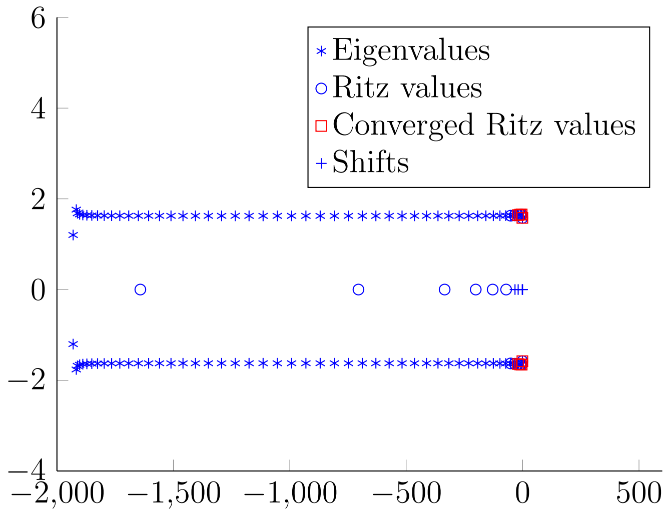


Figure 4.4: Rational Krylov with $m = 20$ and $k = 5$

2.7. We will follow that strategy from now on.

Consider the same problem with a discretization step $h = 1/500$, in that case $A \in \mathbb{R}^{1000\times1000}$. The algorithm of Arnoldi does not work for this problem. We show it for the small case, then we can compare the shift-and-invert Arnoldi, that is the most used algorithm for such problems, and Rational Krylov. We will flag as converged the Ritz pairs such that $\omega_i(z) \leq 10^{-12}$. We have already explained that after this process it is needed a refinement but we will neglect it. The goal is to compute the first k rightmost eigenvalues, so shift-and-invert Arnoldi will be pointed in zero and Rational Krylov will start with the first shift in zero. In Rational Krylov algorithm we set *cstep* $= 2$, this means that we change shift when at least 2 Ritz values converged.

Results are in the Figure 4.5. The number of steps, in this case without restart, coincides with the length of the Arnoldi sequence.

Wanted eigenvalues	Shift–and–inverted (number of steps)	Rational Krylov (number of steps)	Savings percentage (steps)
20	45	38	16 %
40	79	64	19 %
60	112	89	21 %
80	144	113	22 %

Figure 4.5: Convergence of the rightmost eigenvalues with Shift-and-inverted Arnoldi and with Rational Krylov

In Figure 4.6 it is showed how Rational Krylov works and how shifts move.

The saving in computations depends mainly on the problem. In this case Rational Krylov works better than shift–and–invert Arnoldi but one can see that the difference is not so big. We stress that already with this problem that is not so big, when we compute 60 eigenvalues both algorithms became slow and errors grow up, therefore

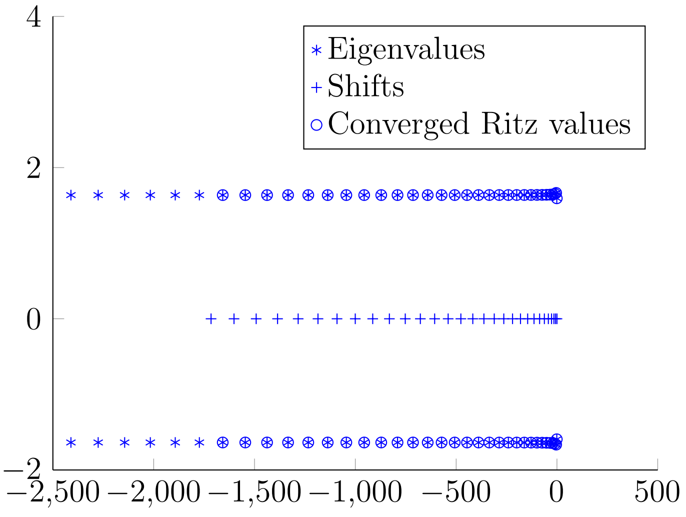


Figure 4.6: Rational Krylov algorithm to compute 60 rightmost eigenvalues

a restart is needed.

The problem of shift–and–invert Arnoldi is that we need to lock all converged Ritz values for every restart otherwise a loop is encountered because after restart Ritz values near zero will converge again.

This means that we need to have an Arnoldi sequence at least of 60 vectors. With Rational Krylov we have not this problem, but we have eigenvalues very near to each other. Then during restart we have to take enough Ritz values otherwise we will restart the process with already converged Ritz values.

This means that the error will grow, see observation 2.3.2. For instance, for computing the first 100 rightmost eigenvalues we can consider the following restart strategy: we restart when the length of the Arnoldi sequence is more than 60 and we lock the first 30 Ritz values. This means that we lock the 30 Ritz values nearest to the current shift. If we do not use this restarting strategy the algorithm will be five times slower. We have this because the most expansive part of the algorithm is the Gram–Schmidt process and if we do not perform the restart the dimension of Krylov subspace is greater than 180. This is a lucky example because at least with a long Arnoldi sequence the orthogonality is approximatly preserved.

In this case there will be error after every restart because the eigenvalues are too close to each other, so we need to apply refinement after the computation. The important point is that this restart strategy is possible with Rational Krylov and impossible with shift–and–invert Arnoldi.

4.1.2 A stability problem in aerodynamic

The Tolosa matrix arises in the stability analysis of a model of an airplane in flight [1]. The interesting modes of this system are described by complex eigenvalues whose imaginary parts lie in a prescribed frequency range. We will consider the Tolosa matrix of size 2000, we are interested in computing the 23 eigenvalues in the rectangle $-750 < Re\lambda < -650$ and $2200 < Im\lambda < 2400$. We will compare Arnoldi, shift–and–invert Arnoldi and Rational Krylov, where the first shift for Rational Krylov is $-750+2390i$, this is also the shift of shifted–and–inverted Arnoldi. In this case the basic Arnoldi does not work well because these eigenvalues are the worst conditioned ones. If we run Arnoldi without restart the algorithm stops after 223 steps, but the errors are big and Ritz values are flagged as converged while they are far from convergence. Figure 4.7 reports the results.

We can solve this problem using a restart strategy but it is clear that in this case Arnoldi it is not suitable. If we run Rational Krylov algorithm we have much better results, see Figure 4.8.

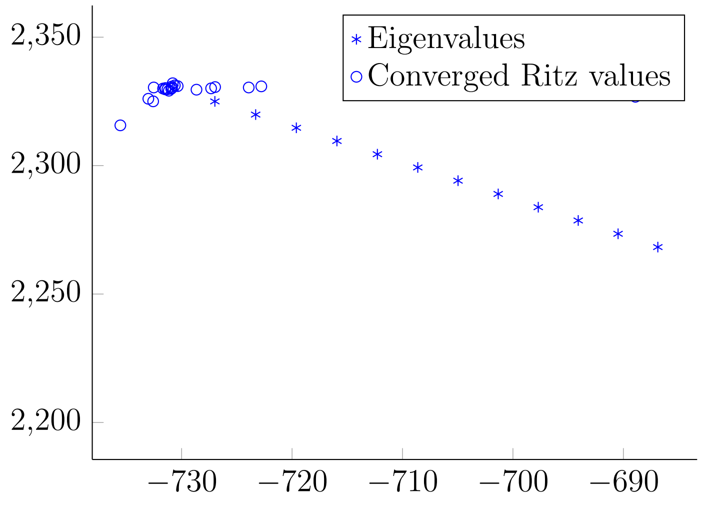


Figure 4.7: Arnoldi algorithm to compute 23 eigenvalues in the rectangle $-750 < Re\lambda < -650$ and $2200 < Im\lambda < 2400$

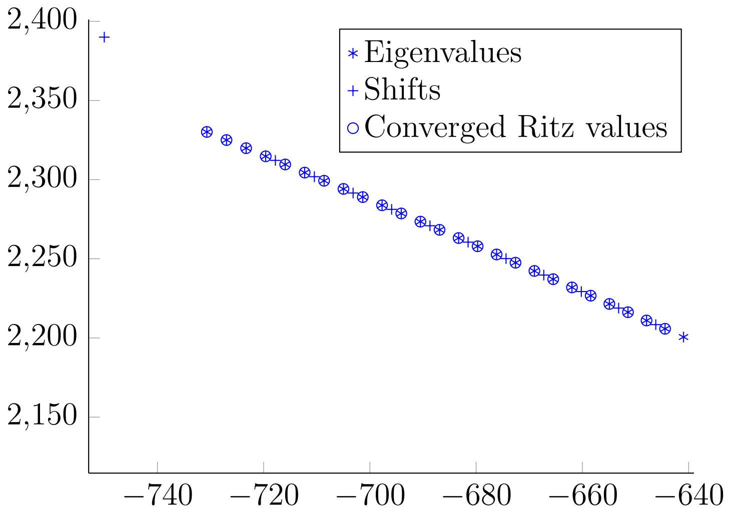


Figure 4.8: Rational Krylov algorithm to compute 23 eigenvalues in the rectangle $-750 < Re\lambda < -650$ and $2200 < Im\lambda < 2400$

If we use the Shift–and–invert Arnoldi algorithm we need 80 steps to reach converge while with Rational Krylov we need 57 steps (Savings percentage: 29 %).

4.1.3 Stability of a flow in a pipe

Let us consider a pipe flow with the goal of studying the stability of the system. The derivation of the problem is very well explained in [16] (Section 3.1.5) and it is technical and long. We can summarize the derivation of the problem in these steps: the Navier–Stokes equation for incompressible flow in circular coordinates in a fixed base flow are linearized. A bidimensional (sinusoidal) perturbation is introduced. After a few manipulations (for details see [16]) the following monodimensional linear eigenproblem is obtained

$$\begin{cases} \{(D^2-\alpha)^2 - i\alpha Re[\mathcal{U}_0(D^2-\alpha^2) - \mathcal{U}_0'']\}\,\tilde{v} = -ic\alpha Re(D^2-\alpha^2)\tilde{v} \\ \tilde{v}(1) = 0 \\ \tilde{v}(-1) = 0 \\ D\tilde{v}(1) = 0 \\ D\tilde{v}(-1) = 0 \end{cases}$$

We used the same notation of [16]. In this case $\mathcal{U}_0(y) = (1-y)(1+y)$, α and Re are fixed numbers. The indipendent variable is y and the problem is defined in $[-1,1]$. D is the derivative, and the powers have to be read as formal powers, in sense that $(D^2-\alpha^2)^2 = D^4 - 2\alpha D^2 + \alpha^4$. The variable $\tilde{v}$ denotes the eigenfunction and c the eigenvalue.

We are interesting in computing the smallest eigenpairs. In particular if $Im(c) > 0$ the disturbance is unstable otherwise is stable. At this point we discretize the problem by means of finite difference method. The domain $[-1,1]$ is uniformly discretized in N nodes. We used the following schems

$$(D^2\tilde{v})_j = \frac{\tilde{v}_{j+1} - 2\tilde{v}_j + \tilde{v}_{j-1}}{\Delta y^2}, \qquad (D^4\tilde{v})_j = \frac{\tilde{v}_{j-2} - 4\tilde{v}_{j-1} + 6\tilde{v}_j - 4\tilde{v}_{j+1} + \tilde{v}_{j+2}}{\Delta y^4}.$$

For the border conditions we used the decentred schemes

$$\begin{aligned} &\tilde{v}_1 = 0, && \tilde{v}_N = 0, \\ &(D\tilde{v})_1 = \frac{-3\tilde{v}_1 + 4\tilde{v}_2 - \tilde{v}_3}{2\Delta y}, && (D\tilde{v})_N = \frac{3\tilde{v}_N - 4\tilde{v}_{N-1} - \tilde{v}_{N-2}}{2\Delta y}. \end{aligned}$$

With a consistency analysis we have that these schemes are accurate at the second order. After discretization we can write the problem as a generalized eigenvalue problem

$$A\underline{\tilde{v}} = cB\underline{\tilde{v}},$$

where $\underline{\tilde{v}} = (\tilde{v}_1, \dots, \tilde{v}_N)^H$ and B is a singular matrix, in particular it has rank $N-4$, this depends on the border conditions. Finally we are ready to perform some numerical test on this problem.

In our test we will consider $\alpha = 1$ and $Re = 10000$. It is possible to prove that the spectrum of the continuum problem has a branch structure, in particular it looks like a Y see picture 4.9.

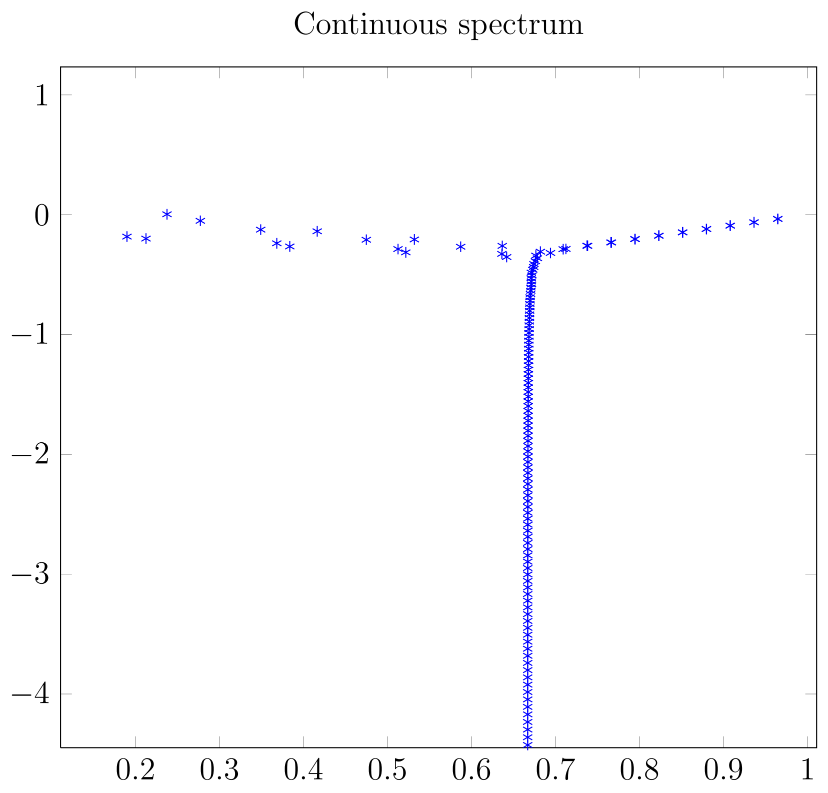


Figure 4.9: Continuous spectrum

We are interested in computing the branch connected to zero. In the discrete case some spurious eigenvalues appear. Let now consider $N = 100$, the goal is to compute first 15 eigenvalues in the branch connected with zero. In this case Rational Krylov is faster (in steps and time of exection) and works correctly see Figure 4.10. Shifted–inverted Arnoldi instead converges to a couple of spurious eigenvalues (spurious for the continuum case) see Figure 4.11. This is because the spurious eigenvalues are near the origin (see theorem 2.2.3). Our goal instead was to compute eigenvalues in the branch. Rational Krylov method is suitable for this goal because in some way this algorithm follows the eigenvalues and go in the direction of Ritz values already converged. This problem does not occur anymore when N gets larger.

In Figure 4.12 it is shown how many iterations the algorithms need to compute the 15 eigenvalues nearest zero. For $N = 10000$ shifted–inverted Arnoldi needed more than the double of time of Rational Krylov. For this example it happened that increasing the discretization, other eigenvalues near zero appear more than for $N = 10000$, but we need less steps.

Also for this application we can use a restarting strategy. For instance we can restart when the dimension of the Krylov subspace is bigger than 40 and lock 20 Ritz values nearest the current shift. With $N = 5000$ with the goal of computing the 15 smallest eigenvalues, applying a restart speeds up the algorithm (23% of time computation saved). In this example we can also avoid the restart since the eigenvalues that we want to compute are few and the length of the Arnoldi sequence is short.

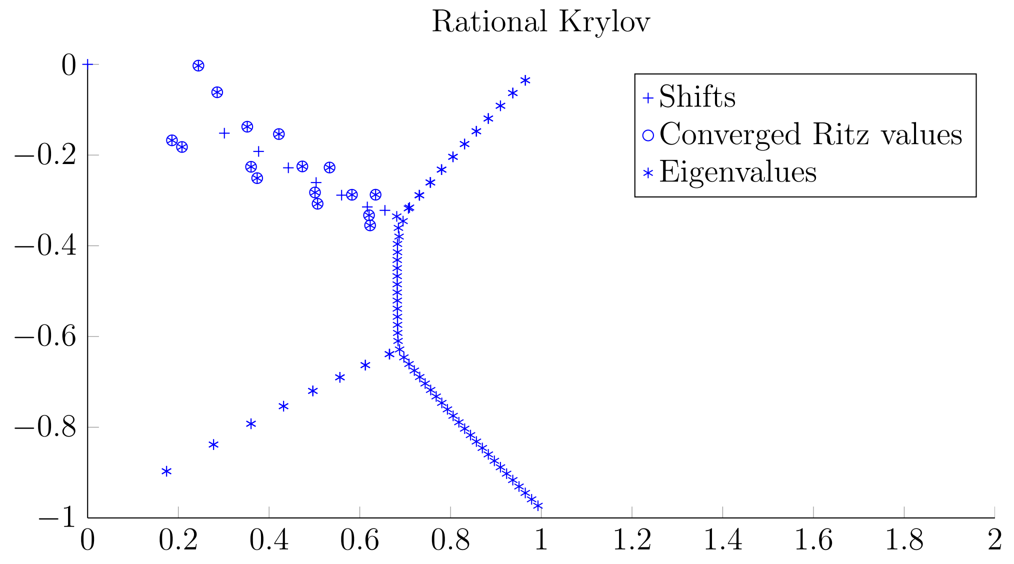


Figure 4.10: Ritz values computed with Rational Krylov, where $N = 100$

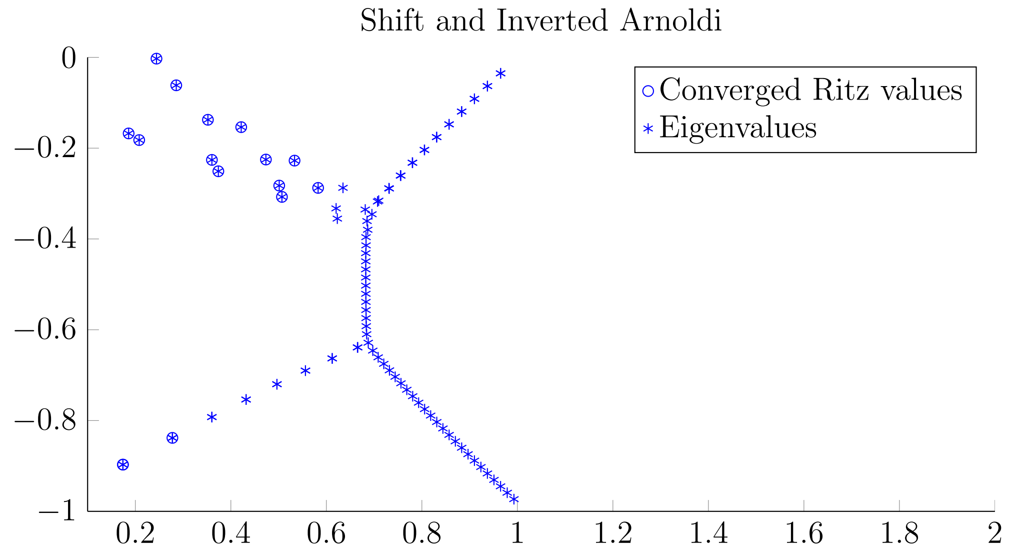


Figure 4.11: Ritz values computed with Shifted–inverted Arnondi, where $N = 100$

N	Shift–and–inverted (number of steps)	Rational Krylov (number of steps)	Savings percentage (steps)
100	79	68	14 %
1000	113	84	26 %
10000	99	78	38 %

Figure 4.12: Convergence of the rightmost eigenvalues with Shift-and-inverted Arnoldi and with Rational Krylov

4.2 Nonlinear eigenvalue problems

A few test problems can be found in the Manchester collection of NLEPs [2], we run the algorithms on the GUN problem. Moreover we tested the algorithms also on the problem of *vibrating string with elastically attached mass* ([17], [3] and [2]) and on a problem of *fluid-solid structure interaction* ([20], [5] and [6]).

4.2.1 Gun problem

We will test our algorithms on the Gun problem from the Manchester collection of NLEPs [2]. This is a large-scale NLEP that models a radio frequency gun cavity and is of the form

$$F(\lambda)x = \left(K - \lambda M + i\sqrt{\lambda - \sigma_1^2}W_1 + i\sqrt{\lambda - \sigma_2^2}W_2\right) = 0$$

Where M, K, W_1 and W_2 are real symmetric matrices of size 9956×9956, K is positive semidefinite, and M is positive definite. We take $\sigma_1 = 0$ and $\sigma_2 = 108.8774$, the notation of the complex square root, $\sqrt{\cdot}$ denotes the principal branch. The domain of interest is

$$\Omega = \{\lambda \in \mathbb{C} \text{ such that } |\lambda - \mu| \leq \gamma \text{ and } Im(\lambda) \geq 0\}$$

where $\gamma = 50000$ and $\mu = 62500$. Before solving this problem we will perform a shift and scale in order to improve convergence, then we will consider the map

$$\begin{array}{rccc} \phi: & \mathbb{C} & \rightarrow & \mathbb{C} \\ & \lambda & \rightarrow & \dfrac{\lambda - \mu}{\gamma} \end{array}$$

It is clear that with this map we transform the domain of interest Ω in the upper part of the unit circle. Then we consider

$$\widehat{\lambda} = \phi(\lambda) = \frac{\lambda - \mu}{\gamma}$$

then we have $\lambda = \gamma\widehat{\lambda} + \mu$, so we have the NLEP in this new coordinates

$$\widehat{F}(\hat{\lambda})x = \left(K - (\gamma\widehat{\lambda} + \mu)M + i\sqrt{\gamma\widehat{\lambda} + \mu - \sigma_1^2}W_1 + i\sqrt{\gamma\widehat{\lambda} + \mu - \sigma_2^2}W_2\right) = 0$$

For measuring the convergence of an approximate eigenpair (λ, x) we use the relative residual norm

$$E(\lambda, x) = \frac{\|F(\lambda)x\|_2}{\left(\|K\|_1 + |\lambda|\|M\|_1 + \sqrt{|\lambda - \sigma_1^2|}\|W\|_1 + \sqrt{|\lambda - \sigma_2^2|}\|W_2\|_1\right)\|x\|_2}$$

To solve this problem we apply HIRKM algorithm. In Figure 4.13 there are the eigenvalues computed with 60 iterations. We got exactly the same results of [18] though we did a light different change in the original algorithm. In Figure 4.14 the

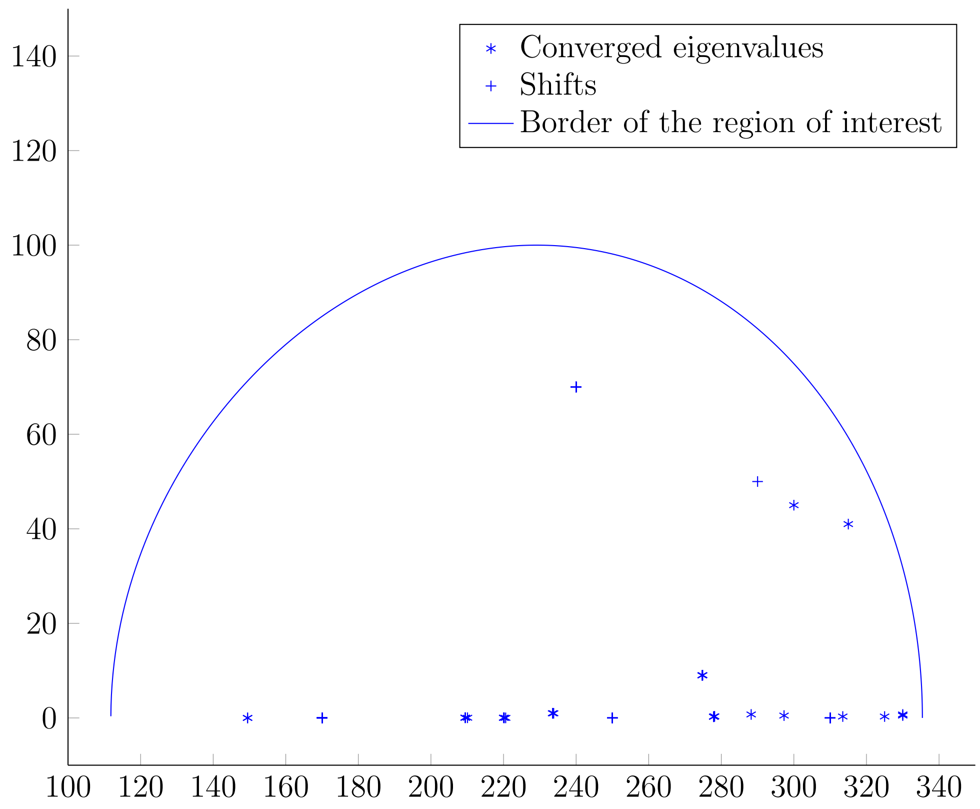
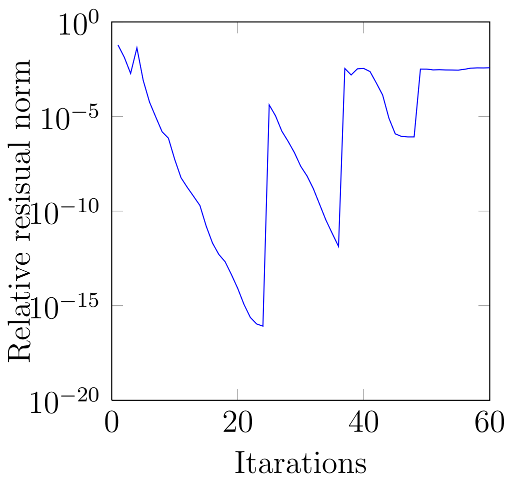


Figure 4.13: Eigenvalues of gun problem computed with 60 itarations

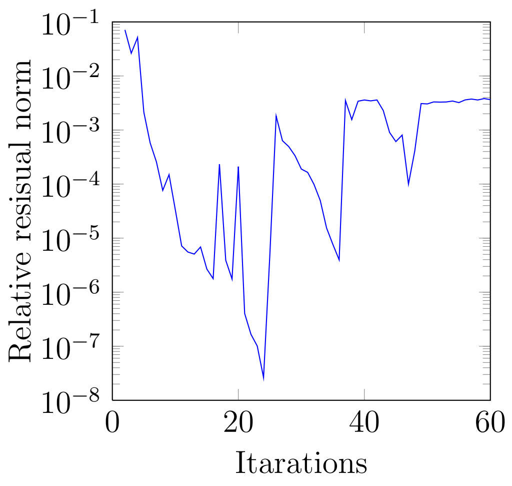
errors of the first four Ritz values are reported. After a few steps the rounding off errors appear, this explain why it seems that at the start the errors decrease and then grow again.

If we try to run the algorithm exploiting the low rank of coefficients we find that $r = 84$ and the algorithm becomes faster because the more expensive part is the Gram–Schmidt process and in this case with shorter vectors we improve this step. For this problem the NLRK algorithm does not work since we are far from linearity.

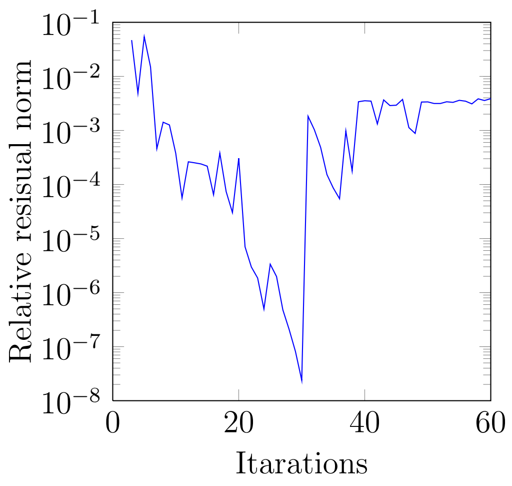
Figure 4.14: Relative Residual norm $E(\lambda, x)$ history for the first four Ritz values



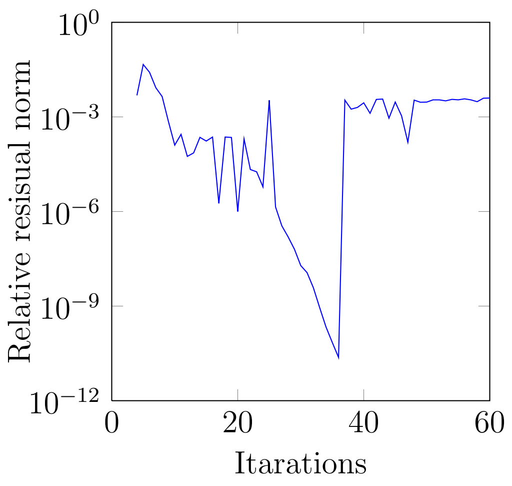
(a) First Ritz value



(b) Second Ritz value



(c) Third Ritz value



(d) Fourth Ritz value

4.2.2 Vibrating string with elastically attached mass

The analysis of eigenvibrations for mechanical structures with elastically attached masses frequently leads to nonlinear eigenvalue problems. As a model problem, we consider here a limp string of unit length, which is clamped at one end. The other end is free but has a mass m attached to it via an elastic spring of stiffness k, see Figure 4.15.

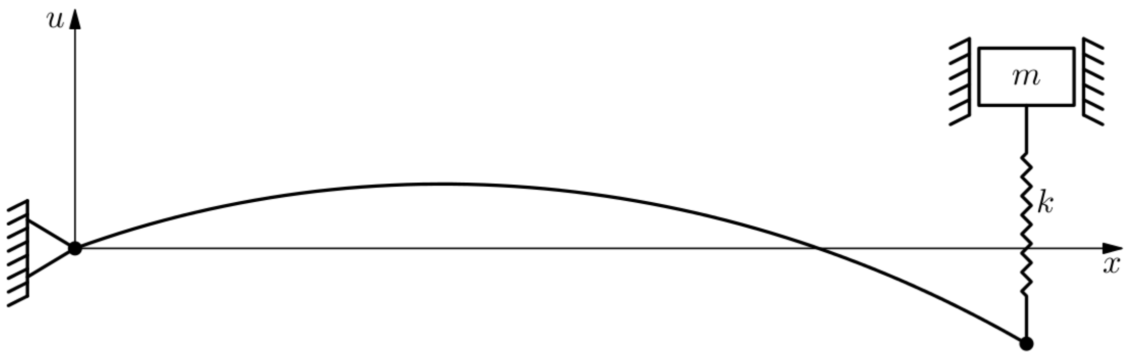


Figure 4.15: Illustration of a vibrating string, which is clamped at the left end and has a vertically moving mass attached to the right end via an elastic spring.

Under these circumstances, the eigenvibrations of the string are governed by the eigenvalue problem ([17], [3] and [2]).

$$\begin{cases} -u''(x) = \lambda u(x) \\ u(0) = 0 \\ u'(1) + k\frac{\lambda}{\lambda - k/m} u(1) = 0 \end{cases}$$

where u denotes the displacement. Furthermore, it is assumed that the clamped end resides at $x = 0$ and the free end with the mass at $x = 1$. This eigenproblem is simple enough to admit a semi–analytic solution. One easily calculates that the differential equation together with the boundary condition at $x = 0$ implies

$$u(x) = C\sin(\sqrt{\lambda}x). \tag{4.1}$$

Inserting this expression into the boundary condition at $x = 1$ and rearranging shows that λ is an eigenvalue if and only if

$$\tan(\sqrt{\lambda}) = \frac{1}{m\lambda} - \frac{\sqrt{\lambda}}{k}. \tag{4.2}$$

Solving the above equation numerically yields approximations to all the eigenfrequencies λ of the string. The corresponding eigenmodes are then given by 4.1. Discretizing the eigenvalue problem with finite elements $P1$ on the uniform grid

$\{x_i = i/N \ : \ i = 0, \dots, N\}$ of size $h = 1/N$ yields a nonlinear matrix eigenvalue problem of the form

$$A - \lambda B + k\frac{\lambda}{\lambda - k/m}C = 0, \tag{4.3}$$

where

$$A = \frac{1}{h}\begin{pmatrix} 2 & -1 & & & \\ -1 & \ddots & \ddots & & \\ & \ddots & & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}, \quad B = \frac{h}{6}\begin{pmatrix} 4 & 1 & & & \\ 1 & \ddots & \ddots & & \\ & \ddots & & 4 & 1 \\ & & & 1 & 2 \end{pmatrix}, \quad C = e_n e_n^H.$$

Observe that if $k = 0$ we have a linear eigenvalue problem. Moreover if k is small enough or m is big we have a little perturbation of a linear eigenvalue problem so that NLRK is a suitable algorithm to solve it.

For this example we set $ResTol = 10^{-6}$ and $EigTol = 10^{-6}$. We are itenterested in computing the second smallest eigenvalue [17].

Let us start with $m = 1$, $k = 0.01$. It is possible to compute the eigenvalues of the continuum problem by solving numerically 4.2. We find that the first two eigenvalues are $\lambda_1 \simeq 0.0461$ and $\lambda_2 \simeq 2.4874$, we are itenterested in computing the second one. We choose the first pole in 1 and the fist shift in 2.

With $N = 100$ the algorithm needed 5 steps to converge to 2.4875 that is a good estimation of the sought eigevalue. In fact the error has the order of 10^{-3} but we have to take into account the error due to the discretization. Moreover very few inner iterations are done, at most 2 inner iterations for every outer iteration.

If we choose $N = 10000$ the algorithm is again fast, again in 5 steps we have numerical convergence and this time the error is of the order of 10^{-5}. In every case this is a lucky example since k is small, this means that near 0 the problem is linear and in fact the residual of Arnoldi sequence is of the order of 10^{-6}.

If we choose $k = 0.1$, the sought eigevalue is 2.6679. The problem is less linear but as before, with $N = 100$ the algorithm needs again 5 steps to converge but it is needed to perform a large number of inner iterations, sometimes 3 inner iterations for one outer iteration. This time NLRK converge to 2.6709 and we get an error of 10^{-2} (due mainly to the discretization error).

Moreover the Arnoldi sequence residual is 10^{-5}. Again if we increase N we can decrease the discretization error. If $k = 1$ the algorithm succeeds to maintain small the residual of the Arnoldi sequence for the first steps, and after that diverges.

This example points out that NLRK works well (mainly) with a nonlinear eigenvalue problem that is a small perturbation of a linear problem.

In conclusion, if we want to do a comparison with HIRKM, this is slower and converge just if we chose the poles near the desired eigenvalues. The advantage of this second algorithm is that we can solve the problem also for $k = 1$ or larger values.

4.2.3 Fluid-solid structure interaction

We consider a mathematical model which describes the problem governing free vibrations of a tube bundle immersed in a slightly compressible fluid (see [20], [5] and

[6]).

We will study the problem under the following simplifying assumptions: the tubes are assumed to be rigid, assembled in parallel inside the fluid, and elastically mounted in such a way that they can vibrate transversally, but they can not move in the direction perpendicular to their sections. The fluid is assumed to be contained in a cavity which is infinitely long, and each tube is supported by an independent system of springs (which simulates the specific elasticity of each tube). Due to these assumptions, three-dimensional effects are neglected, and so the problem can be studied in any transversal section of the cavity. Considering small vibrations of the fluid (and the tubes) around the state of rest, it can also be assumed that the fluid is irrotational.

We can describe the problem in the following way: let $\Omega \subset \mathbb{R}^2$ the section of the cavity and Ω_j for $j = 1, \dots, k$ the sections of the tubes. We assume that $\Omega_j \subset \Omega$ and $\Omega_i \cap \Omega_j = \emptyset$ if $i \neq j$. Define $\Omega_0 = \Omega \setminus \bigcup_{j=1}^{k} \Omega_j$. Let Γ_j the border of Ω_j. Then it is possible to write the problem as: find $\lambda \in \mathbb{R}$ and $u \in H^1(\Omega_0)$ such that for every $v \in H^1(\Omega_0)$

$$c^2 \int_{\Omega_0} \nabla u \cdot \nabla v dx = \lambda \int_{\Omega_0} uv dx + \sum_{j=1^k} \frac{\lambda \rho_0}{k_j - \lambda m_j} \int_{\Gamma_j} un ds \cdot \int_{\Gamma_j} vn ds$$

Here u is the potential of the velocity of the fluid, c denotes the speed of sound in the fluid, ρ_0 is the specific density of the fluid, k_j represents the stiffness constant of the spring system supporting the tube j, m_j is the mass per unit length of the tube j, and n is the outward unit normal to the boundary of Ω_0.

We consider the rational eigenvalue problem where Ω is the ellipse with center $(0, 0)$ and length of semiaxes 8 and 4, and Ω_j , $j = 1, \dots, 9$ are circles with radius 0.3 and centers $(-4, -2)$, $(0, -2)$, $(4, -2)$, $(-5, 0)$, $(0, 0)$, $(5, 0)$, $(-4, 2)$, $(0, 2)$ and $(4, 2)$. All constants in the problem are set equal to 1.

We discretized the problem by finite elements with FreeFem++ [4] using P1 triangular elements getting the nonlinear eigenvalue problem

$$A(\lambda)x = -Ax + \lambda Bx + \frac{\lambda}{1 - \lambda} Cx = 0$$

where C collects the contributions of all tubes, A, B, and C are symmetric matrices, A and C are positive semidefinite, and B is positive definite. See Figure 4.16 to have idea of the discretization performed.

There are 28 eigenvalues in $[0, 1)$, we tried to execute the NLRK (algorithm 15) to compute the first 10 eigenvalues. We set $ResTol = 10^{-6}$, $EigTol = 10^{-6}$ and the maximum number of inner iterations at 20 (as suggested in [5]). We tried different discretization sizes. In FreeFem++ the border of domain is discretized with a piecewise–linear curve. It is possible to choose how many points to use to describe the border of domain before to perform a linear interpolation. We choose n points for the ellipse (outer border) and m points for the circles (inner border). We denote with N the size of the matrices obtained with the discretization. We performed a change of pole every 10 iterations. In Figure 4.17 there is the convergence history.

We also tried to run the algorithm on the original matrices of [6], in this case we do not now how discretization was performed, convergence history is in Figure 4.18.

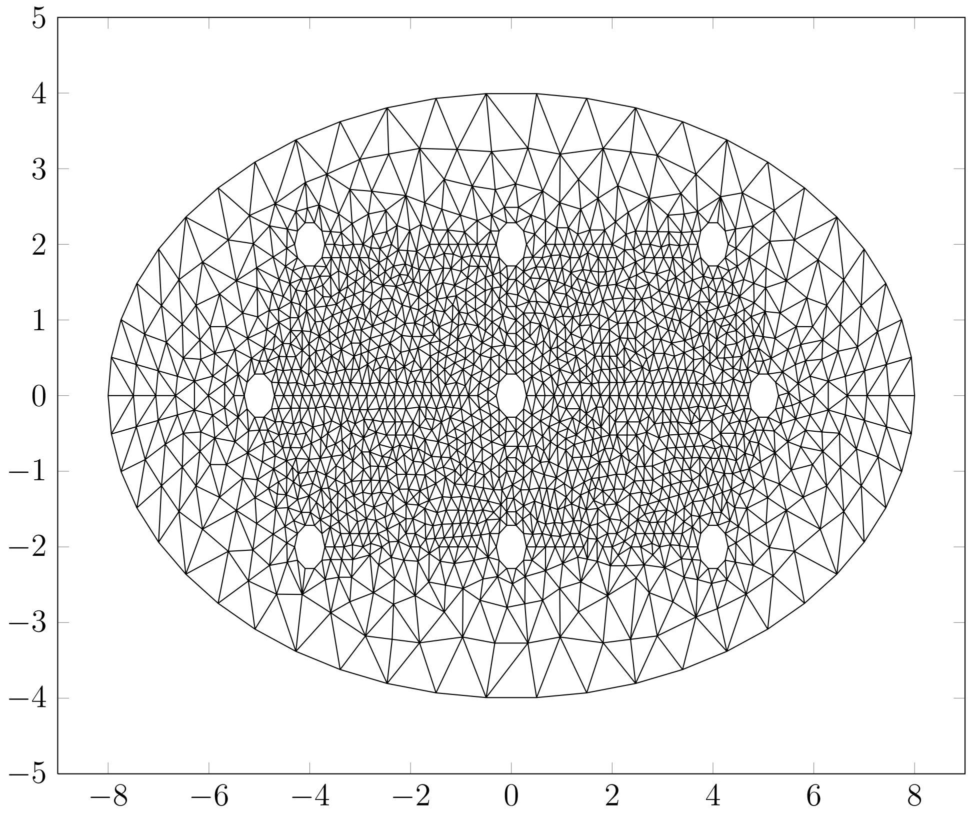
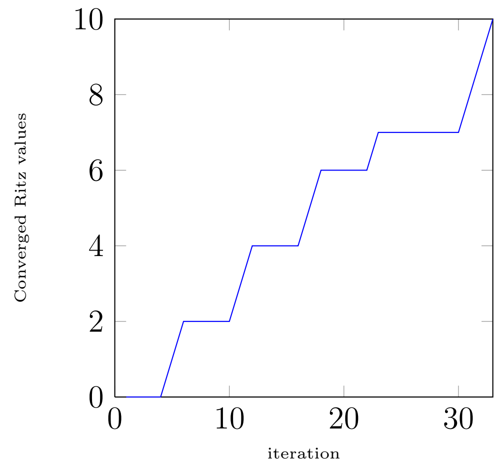


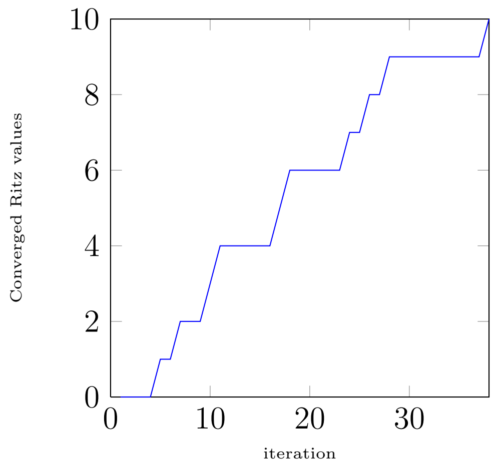
Figure 4.16: Example of discretization of domain with FreeFem++ where $m = 10$ and $n = 50$

The reason why NLRK works well for this problem is not know. In fact this problem is not a little perturbation of one linear eigenproblem. One possible explanation can be found in [5]. In the linear case we have that the eigenvectors of a symmetric real matrix are orthogonal and the eigenvalues real. This nonlinear problem has a similar propriety, the eigenvalues in $[0, 1)$ are real and the eigenvectors are nearly-orthogonal. Therefore one can think that this is a connection with the linear problem. In order to deeply understand this point other test must be performed. As for the previous example, HIRK succeeds to compute eigenvalues but is very slow and unusable when the discretization parameter N is big.

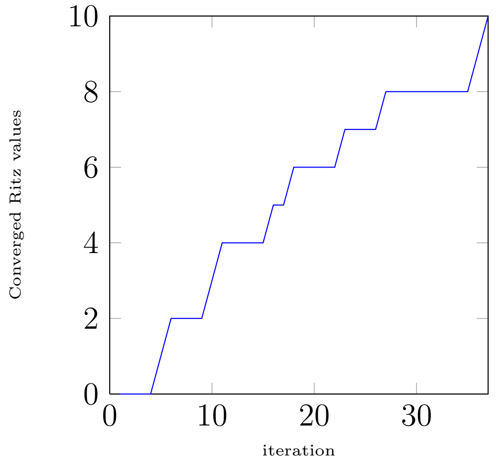
Figure 4.17: Convergence history of Ritz values computed with the discretization of FreeFem++



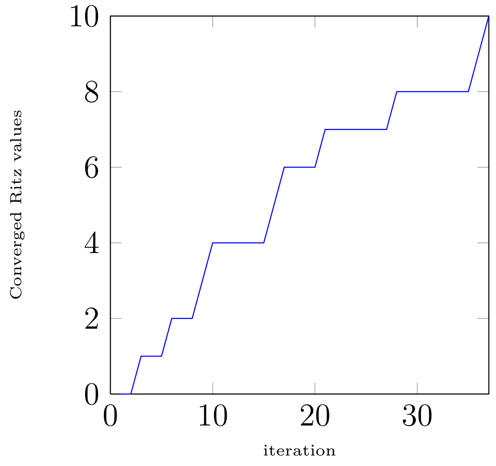
(a) $n = 50$, $m = 10$, $N = 1636$



(b) $n = 100$, $m = 10$, $N = 2156$

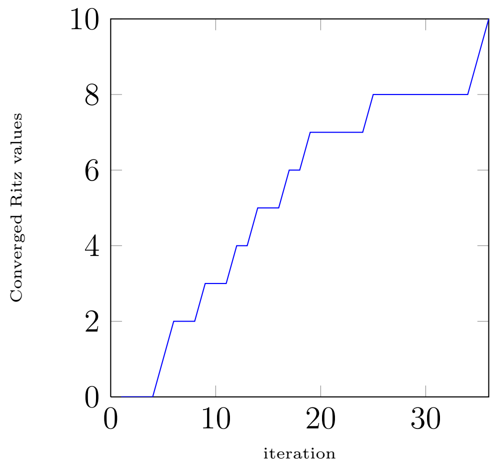


(c) $n = 200$, $m = 10$, $N = 3277$

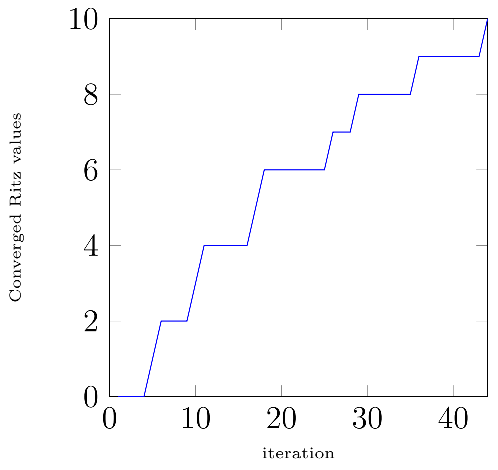


(d) $n = 400$, $m = 10$, $N = 5604$

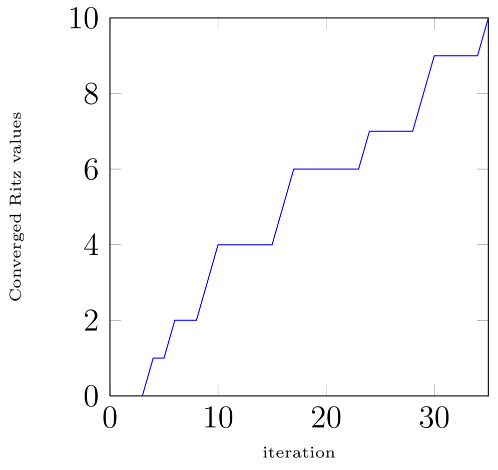
Figure 4.18: Convergence history of Ritz values computed on the matrices provided by Voss



(a) $N = 2350$



(b) $N = 9144$



(c) $N = 3640$

Chapter 5

Conclusions and future developments

In this thesis we showed how is possible to solve big sized eigenproblem by means of Rational Krylov algorithm. Regarding the linear case the biggest problem is the loss of orthogonality, this problem occur when the Arnoldi sequence is too long. To avoid this problem we presented the thick restart. Sometimes it happened that this is not sufficient and after a few restart the residual of the Arnoldi sequence became big, that is the algorithm diverge. An idea to solve this problem can be to modify the orthogonalization process. It is possible in fact to use Householder transformations to orthogonalize. This approach is numerically stable but inefficient. An idea can be to perform the orthogonalization with Gramm–Schmit and a selective reorthogonalization using Householder matrices.

Regarding the nonlinear case we showed two algorithm: HIRK and NLRK (and iterative methods). The first one is sharp and effective algorithm but is it not efficient. A way to solve this problem can be to use the low rank structure of the matrices coefficients, in general we think that also other structures (e.g. semi separability) can be exploited. The NLRK algorithm instead is very fast but it is based on linear interpolations, it means that works well just if the problem is a small perturbation of a linear problem. A possible solution can be to use instead iterative projection methods. In these algorithms an important step is the hard purging that till now it is just proposed heuristically. One can wonder if other restarting strategies can works better or at least give a deeper explanation of the hard purging.

Acknowledgements

I would like to thank professor Dario A. Bini, the scientific advisor of this thesis, for the help, suggestions and support. Moreover I also want to thank the researcher Simone Camarri of the Faculty of Aerospace Engineering, who suggested and explained me the application *stability of a flow in a pipe* and some other eigenproblems of fluid dynamics. I also wish to thank professor Heinrich Voss who provided me some of the matrices of the example *fluid-solid structure interaction.* In conclusion I also thank Axel Ruhe who answered about a technical question regarding rational Krylov algorithm.

Bibliography

[1] Zhaojun Bai, David Day, James Demmel, and Jack Dongarra. A test matrix collection for non-hermitian eigenvalue problems. Technical report, Research report, Department of Mathematics, University of Kentucky, 1995.

[2] Timo Betcke, Nicholas J Higham, Volker Mehrmann, Christian Schröder, and Françoise Tisseur. Nlevp: A collection of nonlinear eigenvalue problems. *ACM Transactions on Mathematical Software (TOMS)*, 39(2):7, 2013.

[3] Peter Cedric Effenberger. *Robust solution methods for nonlinear eigenvalue problems*. PhD thesis, EPFL, 2013.

[4] F. Hecht. New development in freefem++. *J. Numer. Math.*, 20(3-4):251–265, 2012.

[5] Elias Jarlebring et al. Krylov methods for nonlinear eigenvalue problems. *Mémoire de DEA, Royal Institute of Technology, Stockholm, Sweden*, 2003.

[6] Elias Jarlebring and Heinrich Voss. Rational krylov for nonlinear eigenproblems, an iterative projection method. *Applications of Mathematics*, 50(6):543–554, 2005.

[7] Emmanuel Kamgnia, Bernard Philippe, et al. Counting eigenvalues in domains of the complex field. *Electronic Transactions on Numerical Analysis*, 40:1–16, 2013.

[8] MATLAB Manual. Version 6, the math works. *Inc., Natick, MA*, 2000.

[9] Karl Meerbergen. Dangers in changing poles in the rational lanczos method for the hermitian eigenvalue problem. 1999.

[10] Axel Ruhe. Rational krylov sequence methods for eigenvalue computation. *Linear Algebra and its Applications*, 58:391–405, 1984.

[11] Axel Ruhe. Rational krylov: A practical algorithm for large sparse nonsymmetric matrix pencils. *SIAM Journal on Scientific Computing*, 19(5):1535–1551, 1998.

[12] Axel Ruhe. *Templates for the solution of algebraic eigenvalue problems: a practical guide*, volume 11, chapter Rational Krylov Subspace Method. Siam, 2000.

[13] Axel Ruhe. The rational krylov algorithm for nonlinear matrix eigenvalue problems. *Journal of Mathematical Sciences*, 114(6):1854–1856, 2003.

[14] Axel Ruhe. Rational krylov for large nonlinear eigenproblems. In *Applied Parallel Computing. State of the Art in Scientific Computing*, pages 357–363. Springer, 2006.

[15] Youcef Saad. *Numerical methods for large eigenvalue problems*, volume 158. SIAM, 1992.

[16] Peter J Schmid and Dan S Henningson. *Stability and transition in shear flows*, volume 142. Springer, 2001.

[17] Sergey I Solov'ëv. Preconditioned iterative methods for a class of nonlinear eigenvalue problems. *Linear algebra and its applications*, 415(1):210–229, 2006.

[18] Roel Van Beeumen, Karl Meerbergen, and Wim Michiels. A rational krylov method based on hermite interpolation for nonlinear eigenvalue problems. *SIAM Journal on Scientific Computing*, 35(1):A327–A350, 2013.

[19] Henk A van der Vorst. Computational methods for large eigenvalue problems. *Handbook of numerical analysis*, 8:3–179, 2002.

[20] Heinrich Voss. A maxmin principle for nonlinear eigenvalue problems with application to a rational spectral problem in fluid-solid vibration. *Applications of Mathematics*, 48(6):607–622, 2003.

[21] James Hardy Wilkinson, James Hardy Wilkinson, and James Hardy Wilkinson. *The algebraic eigenvalue problem*, volume 155. Oxford Univ Press, 1965.

[22] Kesheng Wu and Horst Simon. Thick-restart lanczos method for large symmetric eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(2):602–616, 2000.

[23] Kesheng Wu and Horst D Simon. Dynamic restarting schemes for eigenvalue problems. Technical report, Lawrence Berkeley National Lab., CA (US), 1999.