Computation of thermo-acoustic modes in combustors

Jan-Willem van Leeuwen
Master's Thesis for Applied Mathematics
Delft, June 5th 2007

Abstract

When constructing airplanes, safety is the most important issue. Engines are a critical part of airplanes, and have to be very robust to provide safety. Thermo-acoustic instabilities are a possible weakness of these engines. Unfortunately, it is very hard to predict these instabilities. At CERFACS, Toulouse, a method has been developed to use the wave equation to model the acoustic pressure inside combustion chambers. To solve this equation, the equation is discretized and written as a nonlinear eigenvalue problem. The complexity of the geometry of the combustion chamber leads to very complicated grids with many gridpoints. This means large, sparse, nonlinear eigenvalue problems have to be solved. So far, no method has been designed for this specific problem. Currently, the problem has to be rewritten and solved using a Picard iteration. The resulting linear problem can be easily solved by the state-of-the-art method for large, sparse eigenvalue problems called Arnoldi's method. Another method for solving eigenvalue problems is Jacobi-Davidson. Theoretically, the Jacobi-Davidson method for linear problems is easily extended to quadratic and nonlinear eigenvalue problems, without rewriting the problem. We believe Jacobi-Davidson has the potential to solve nonlinear problems very fast in comparison to the current method. To test this potential, we have compared the Jacobi-Davidson method with Arnoldi on three levels: linear problems, quadratic problems and nonlinear problems. All these problems have been formulated to closely resemble the actual problem of thermo-acoustic instability. For linear problems, Arnoldi has proven to be faster than Jacobi-Davidson. This was expected, since Arnoldi is a very simple yet powerful method, whereas Jacobi-Davidson is more complex. For quadratic problems however, Jacobi-Davidson fulfills its potential by finding the desired solution much faster than Arnoldi. We have not been able to find acceptable solutions of fully nonlinear problems with Jacobi-Davidson. However, we believe that this is where Jacobi-Davidson has the most potential and should be researched further.

Acknowledgments

I would like to express my gratitude to the department of numerical mathematics at the Technological University in Delft, especially my supervisor Martin van Gijzen. He has helped me throughout the entire development of this thesis, providing me with information, wisdom and friendship. Jok Tang, Fang Fang and Elise van Aken for helping me during the periods I was sharing an office with them. I am also greatly indebted to the people at CERFACS for letting me use their facilities and helping me with their knowledge and experience. Special thanks go out to Claude Sensiau for his friendship and dedication to the project, to Thierry Poinsot and Franck Nicoud for their supervision and to Serge Gratton and the parallel algorithm team for providing me with help and support on the mathematical part of the research. Finally, I would like to thank my parents, family and friends for mental and financial support during my studies at the TU Delft, and I thank God for everything I have.

Contents

Chapter 1

Introduction

In combustion chambers, acoustic pressure levels may oscillate under the influence of the heat release and geometrical aspects of the chamber. This can cause instabilities which might damage the combustion chamber. Many different approaches have been used in the past to predict these instabilities. A promising method is to model the acoustic pressure fluctuations using the wave equation. When discretized, this gives an eigenvalue problem where the eigenvalue is related to the frequency of the oscillation. This value is very important in analyzing the design of combustion chamber, and being able to predict it before building a prototype for testing could save valuable time and money.

The goal of this thesis is to make a comparison between two state-of-the-art methods for solving the large, sparse eigenvalue problems that arise from the combustion problems described above. This comparison will be based on theoretical analysis of both methods, as well as numerical results from implementations specifically designed for this type of problem. The most important testing criterion will be the time it takes to solve eigenvalue problems arising from realistic grids. By implementing a method that is faster a lot of time can be saved in the design phase of engines.

The theoretical part of the study has been done in Delft at the Technical University. We have conducted the practical research part of the thesis at CERFACS in Toulouse, France, where the solution method described above is used to aid the design of airplane and helicopter combustors. CERFACS stands for 'Centre Europeèn de Recherche et de Formation Avancée en Computation Scientifique', or in English: European Center for Research and Advanced Training in Scientific Computing. CERFACS is a research organization that aims to develop advanced methods for the numerical simulation and the algorithmic solution of large scientific and technological problems of interest for research as well as industry, and that requires access to the most powerful computers presently available. CERFACS hosts interdisciplinary teams, both for research and advanced training that are comprised of: physicists, applied mathematicians, numerical analysts, and software engineers. Approximately 100 people work at CERFACS, including about 90 researchers and engineers, coming from 10 different countries. They work on specific projects in six main research areas: parallel algorithms, aerodynamics, combustion, climate and environment, data assimilation, and electromagnetism.

A software package called AVSP [11] has been developed at CERFACS to easily discretize realistic problems. A mathematical routine is then called to solve the eigenvalue problems related to the realistic problems. The current numerical method implemented into this mathematical routine is based on Arnoldi's method [1]. This algorithm, as implemented in the mathematical package ARPACK [8] is currently state-of-the-art for solving eigenvalue problems. However, there is a drawback. The particular application studied at CERFACS gives rise to large, nonlinear eigenvalue problems. Solving these mathematical problems with traditional methods such as Arnoldi's method is often costly, because Arnoldi's method cannot solve these problems directly but has to transform them, increasing computing time and often causing the matrices involved to lose their nice properties.

In recent years, a new method, called Jacobi-Davidson, has been developed to deal with these drawbacks. It is based on the same idea as Arnoldi, but gives it an interesting twist that provides faster convergence and removes the need to transform nonlinear eigenvalue problems. These two methods have been tested and compared at CERFACS, as part of this master's thesis. The results of these comparisons indicate that the two methods perform almost the same for linear problems, but for quadratical problems Jacobi-Davidson is much faster.

We start in chapter 2 by describing the problem in detail, and the underlying modeling of the combustion effects. We show how the eigenvalue problem is derived, starting from a coupled system for the important variables. This system is rewritten to a Helmholtz equation for the fluctuating acoustic pressure. This equation is discretized using finite elements.

In chapters 3, 4 and 5 we describe various solution methods for linear, generalized, quadratic and nonlinear eigenvalue problems. We go into detail in explaining how Arnoldi's method and Jacobi-Davidson work, including the earlier methods they were based on. We show how variations of both methods can be constructed to deal with more complex eigenvalue problems, and how they relate to each other. We point out what the strengths and weaknesses of the methods are.

In chapters 6 and 7 the methods are compared. We define several academic testcases, and a realistic testcase. We use implementations of both methods in MATLAB and FORTRAN to be able to compare the methods as fairly as possible. We also explain the choices for the necessary parameters, and discuss the validity of the results. The conclusion of our research, based on the numerical results, is given in chapter 8, and in the last chapter we discuss possibilities for future research.

Chapter 2

Problem Formulation

The physical background of the research presented in this paper is based on the research done by Nicoud, Benoit, Sensiau and Poinsot in [11] where a method to solve the problem of oscillating pressure in combustion chambers is described. Because the acoustics in a gas chamber are coupled to the heat produced by combustion, fluctuations in combustion may start to resonate according to their modes and grow over time. This will cause the pressure inside the combustor to become unstable. The most dangerous oscillations are the ones with the smallest frequencies. Until now, this behavior was hard, if not impossible to predict by simulation during the design stage, because of a lack of sufficient computational power and the complex geometrical details of modern gas chambers. However, it is important to be able to predict and control these oscillations as early as possible.

2.1 Solution methods

Several very different methods have been proposed. Unfortunately, most methods are either too demanding in terms of computational work, or they are based on a greatly simplified model. An example of the first kind is to perform Large Eddy Simulations (LES), that are based on the full three-dimensional unsteady Navier Stokes equations. Another popular method is to model the geometry of the combustor, leaving out details, and to suppose the flame to be infinitely thin. In some cases, this method works good enough, but in general these assumptions are too restrictive.

An intuitive method is to linearize the Navier-Stokes equations. The combustion oscillations have to be taken into account in the energy equation. If the heat release caused by the combustion oscillations can be modeled, the system of equations is closed. We can then proceed in two ways: we can solve the discretized PDE's in the time domain, or in the frequency domain using the Helmholtz equation instead of the wave equation. An important drawback of the first option is that it only offers information about the most unstable modes. The second method yields an eigenvalue problem which will become non-linear when combustion occurs or when there is impedance on the boundary. In this study, we will follow the research from [11] of the second method, that is: to find a way of solving the Helmholtz equation arising from the thermo-acoustic problem numerically by writing it as a nonlinear eigenvalue problem. This is also the method used at CERFACS.

2.2 Derivation of the equation

In this section, the acoustic problem will be described by the wave equation. Then, the equation will be transformed into a Helmholtz equation. The numerical solution methods in the following section will be aiming to approximate the solution of this equation.

2.2.1 Basic equations

Under certain assumptions, further specified in [11], we can describe the physical model by the following equations for mass density, momentum and entropy, together with the state equation and entropy expression:

$$\frac{D\rho}{Dt} = -\rho \nabla \cdot u \tag{2.1}$$

$$\rho \frac{Du}{Dt} = -\nabla p \tag{2.2}$$

$$\frac{Ds}{Dt} = \frac{rq}{p} \tag{2.3}$$

$$\frac{p}{\rho} = rT \tag{2.4}$$

$$s - s_{st} = \int_{T_{st}}^{T} \frac{C_p(T')}{T'} dT' - r \ln\left(\frac{p}{p_{st}}\right) \tag{2.5}$$

The variables that we introduced are: ρ is the density, u is the flow speed, p stands for the pressure, q is the heat release, s is the entropy and T is the temperature. $\frac{D}{Dt}$ is the total derivative with respect to time. The subscript st indicates a state variable.

We linearize this by writing $p = p_0 + p_1$, $\rho = \rho_0 + \rho_1$ and $s = s_0 + s_1$. The second (fluctuating) term in these definitions is of order ϵ compared to the first (steady) term, where $\epsilon \ll 1$. In the linearization of u we assume that u_0/c_0, the Mach number, is practically zero. We write $u = u_1$ where $\sqrt{u_1 \cdot u_1}/c_0$ is also of order ϵ. In this equation, $c_0 = \sqrt{\gamma p_0/\rho_0}$ is the mean speed of sound, with γ the heat capacity per unit mass at fixed pressure divided by the heat capacity per unit mass at fixed volume. With the assumptions of the zero Mach number and the neglection of the heat capacity fluctuations the following set of linear equations for the fluctuating quantities ρ_1, u_1, s_1 and p_1 is obtained:

$$\begin{aligned}
\frac{\partial \rho_1}{\partial t} + u_1 \cdot \nabla \rho_0 + \rho_0 \nabla \cdot u_1 &= 0 &(2.6)\\
\rho_0 \frac{\partial u_1}{\partial t} + \nabla p_1 &= 0 &(2.7)\\
\frac{\partial s_1}{\partial t} + u_1 \cdot \nabla s_0 &= \frac{r q_1}{p_0} &(2.8)\\
\frac{p_1}{p_0} - \frac{\rho_1}{\rho_0} - \frac{T_1}{T_0} &= 0 &(2.9)\\
C_p \frac{T_1}{T_0} - r p_1 p_0 &= s_1 &(2.10)
\end{aligned}$$

Note that the total derivative has been reduced to the partial derivative with respect to time, because with the assumption that $u_0 = 0$ the nonlinear convective terms are always of second order in ϵ. In these equations, also the fluctuating unknowns q_1 and T_1 play a role. To close the set of equations for the fluctuating quantities we need an equation that expresses q_1 in the other variables.

2.2.2 Flame response

Finding a suitable equation for q_1 is, from a physical point of view, the most difficult part of the modeling phase. A choice has to be made between working with global heat release from the whole flame zone or using a local flame model. For modern efficient combustors, the first model does not suffice since the flame is not acoustically compact, that is, the flame region is not small enough compared to the characteristic acoustic wavelength. The second model relates the local unsteady heat release to a reference acoustic velocity in the injector mouth. In equation form, we find that:

$$\frac{q_1(x,t)}{q_{tot}} = n_u(x) \frac{u_1(x_{ref}, t - \tau_u(x)) \cdot n_{ref}}{U_{bulk}} \qquad (2.11)$$

where $n_u(x)$ is the directional interaction field and $\tau_u(x)$ is the field of time lag and n_{ref} is a fixed unitary vector defining the direction of the reference velocity. $n_u(x)$ has been made dimensionless by scaling u_1 and q_1 by U_{bulk}, the bulk velocity, and q_{tot}, the total heat release, respectively. The difficulty with this equation is that the fields of parameters $\tau_u(x)$ and $n_u(x)$ are hard to approximate empirically. The alternative to find reasonable values is to use compressible reacting LES.

By substituting q_1 in equation (2.8) using equation (2.11) we find the following equation:

$$\frac{\partial s_1}{\partial t} + u_1 \cdot \nabla s_0 = \frac{r}{p_0} n_u(x) \frac{q_{tot}}{U_{bulk}} u_1(x_{ref}, t - \tau_u(x)) \cdot n_{ref} \qquad (2.12)$$

2.2.3 Helmholtz equation

Combining equations (2.6), (2.7), (2.9), (2.10) and (2.12) we arrive at the wave equation for p_1:

$$\nabla \cdot \left(\frac{1}{\rho_0} \nabla p_1 \right) - \frac{1}{\gamma p_0} \frac{\partial^2 p_1}{\partial t^2} = - \frac{\gamma - 1}{\gamma p_0} \frac{\partial q_1}{\partial t}. \tag{2.13}$$

Since this equation is linear, we can safely assume that all fluctuating variables are oscillating harmonically at a complex frequency $f = \omega/(2\pi)$. Therefore, we can introduce harmonic variations for pressure, velocity and local heat release perturbations:

$$p_1 = \mathcal{R}(\hat{p}(x) e^{-i\omega t}) \tag{2.14}$$
$$u_1 = \mathcal{R}(\hat{u}(x) e^{-i\omega t}) \tag{2.15}$$
$$q_1 = \mathcal{R}(\hat{q}(x) e^{-i\omega t}). \tag{2.16}$$

If we translate equations 2.11 and 2.13 into the frequency domain we have all the tools necessary to describe the transformed pressure field $\hat{p}$ by the following Helmholtz equation:

$$\nabla \cdot \left(\frac{1}{\rho_0} \nabla \hat{p} \right) + \frac{\omega^2}{\gamma p_0} \hat{p} = \frac{\gamma - 1}{\gamma p_0} \frac{q_{tot}}{\rho_0(x_{ref}) U_{bulk}} n_u(x) e^{i\omega\tau_u(x)} \nabla \hat{p}(x_{ref}) \cdot n_{ref} \tag{2.17}$$

2.2.4 Boundary conditions

An important part in modeling the thermo-acoustic behavior in a gas chamber is the effect of the boundary conditions. Three different types of conditions are possible on the boundary $\partial\Omega$:

Zero pressure: This corresponds to boundaries that are fully reflective, and where the pressure should be equal to the outer pressure. This means that there can be no fluctuations, or in mathematical terms: $\hat{p} = 0$ on $\partial\Omega_D$

Zero normal velocity: This corresponds to boundaries where there can be no fluctuation in the velocity of the flow through the boundary. This happens at walls (where there is no flow at all) or at inlets where the velocity is supposed to be constant. Consequently, $\hat{u} \cdot n_{BC} = 0$. Combined with equation (2.7) this gives for the pressure: $\nabla \hat{p} \cdot n_{BC} = 0$ on $\partial\Omega_N$

Imposed reduced complex impedance: On boundaries where neither the pressure fluctuation nor the normal velocity fluctuation is zero, there will be a combination of both boundary conditions: $c_0 Z \nabla \hat{p} \cdot n_{BC} - i\omega \hat{p} = 0$ on $\partial\Omega_Z$ where Z is the imposed reduced complex impedance, that may depend on the frequency ω.

With these boundary conditions equation (2.17) will lead to a nonlinear eigenvalue problem. This will be further elaborated in the next section.

2.3 The numerical method

In this section, we will see how the Helmholtz equation (2.17) translates into a nonlinear eigenproblem. Various ways to solve these problems are described in section (5). As discretizing the equation is not a simple task, we will first consider a simplified version of equation (2.17) where we don't take the flame response into account. That is, we will discretize the equation

$$\nabla \cdot \left(\frac{1}{\rho_0} \nabla \hat{p} \right) + \frac{\omega^2}{\gamma p_0} \hat{p} = 0$$

combined with the boundary conditions defined before.

2.3.1 Discretization of the equation

To discretize the above problem, we use the finite element method. We divide the domain Ω into tetrahedra and define a piecewise linear function ϕ_j for every vertex v_j. The testfunction ϕ_j equals 1 on its respective vertex v_j and 0 on the other nodes, with linear interpolation. So, for every tetrahedron there are only four test functions that are not zero on the entire area. We can then approximate $\hat{p}$ by $\hat{p}(x) \approx \sum_j \hat{p}_j \phi_j(x)$, where $p_j = p(v_j)$. Because we already know that $\hat{p} = 0$ on $\partial\Omega_D$ from the boundary condition on $\partial\Omega_D$, we can restrict ourselves to the set of vertices S_v of the mesh that do not belong to $\partial\Omega_D$:

$$\hat{p}(x) \approx \sum_{j:v_j \in S_v} \hat{p}_j \phi_j(x).$$

There are now N unknowns, where N is the number of vertices belonging to $\Omega \setminus \partial\Omega_D$. Now the continuous function $\hat{p}$ is discretized: all that is left is to determine the complex coefficients p_j. This can be done by using the Galerkin method. Starting from the Helmholtz equation:

$$\nabla \cdot \left(\frac{1}{\rho_0} \nabla \hat{p} \right) + \frac{\omega^2}{\gamma p_0} \hat{p} = 0$$

we replace $\hat{p}$ by its approximation, multiply with the test function and integrate over Ω, and obtain $\forall k : v_k \in S_v$:

$$\int_\Omega \phi_k \nabla \left(\frac{1}{\rho_0} \nabla \cdot \sum_{j:v_j \in S_v} \hat{p}_j \phi_j(x) \right) dx + \omega^2 \int_\Omega \frac{\phi_k}{\gamma p_0} \sum_{j:v_j \in S_v} \hat{p}_j \phi_j(x) dx = 0.$$

Interchanging the summation and integration operands, and taking out the constants p_j gives $\forall k : v_k \in S_v$:

$$\sum_{j:v_j \in S_v} \int_\Omega \frac{1}{\rho_0} \phi_k \nabla \cdot (\nabla \phi_j) dx \hat{p}_j + \omega^2 \sum_{j:v_j \in S_v} \int_\Omega \frac{1}{\gamma p_0} \phi_k \phi_j dx \hat{p}_j = 0.$$

The first integral is integrated by parts:

$$\int \phi_k \nabla \cdot \nabla \phi_j dx = -\int \nabla \phi_k \nabla \phi_j dx + \oint \phi_k \nabla \phi_j \cdot n d\xi$$

So the equation becomes $\forall k : v_k \in S_v$:

$$\sum_{j:v_j \in S_v} \left(-\int_\Omega \frac{1}{\rho_0} \nabla \phi_k \nabla \phi_j dx + \int_{\partial\Omega} \frac{1}{\rho_0} \phi_k \nabla \phi_j \cdot n d\xi + \omega^2 \int_\Omega \frac{1}{\gamma p_0} \phi_k \phi_j dx \right) \hat{p}_j = 0.$$

Note that $\phi_k(x) = 0 \, \forall k : v_k \in S_v, x \in \partial\Omega_D$ so $\partial\Omega_D$ does not contribute to the boundary integral. On $\partial\Omega_N$ we can rewrite the boundary integral by using the boundary condition $\nabla \hat{p} \cdot n = 0$:

$$\begin{array}{rcl} \sum_{j:v_j \in S_v} \int_{\partial\Omega} \frac{1}{\rho_0} \phi_k \nabla \phi_j \cdot n dx \hat{p}_j & = & \int_{\partial\Omega} \frac{1}{\rho_0} \phi_k \nabla (\sum_{j:v_j \in S_v} \phi_j \hat{p}_j) \cdot n d\xi \\ & = & \int_{\partial\Omega} \frac{1}{\rho_0} \phi_k \nabla \hat{p} \cdot n d\xi \\ & = & 0 \end{array} .$$

Showing that only $\partial\Omega_Z$ contributes to the boundary integral. Rewriting the integral over $\partial\Omega_Z$ in the same way as for $\partial\Omega_N$ and substituting $\nabla \hat{p} \cdot n = i\omega / c_0 Z \sum \phi_j \hat{p}$ gives us our final equation $\forall k : v_k \in S_v$:

$$\sum_{j:v_j \in S_v} \left(-\int_\Omega \frac{1}{\rho_0} \nabla \phi_k \nabla \phi_j dx + i\omega \int_{\partial\Omega_Z} \frac{1}{\rho_0 c_0 Z} \phi_k \phi_j d\xi + \omega^2 \int_\Omega \frac{1}{\gamma p_0} \phi_k \phi_j dx \right) \hat{p}_j = 0.$$

Combining all N equations for all N unknowns p_j into a matrix equation gives:

$$AP + \omega B(\omega) P + \omega^2 CP = 0 \tag{2.18}$$

where P is the vector containing the unknowns p_j and A, B and C are symmetric matrices with generic element

$$A_{kj} = -\int_\Omega \frac{1}{\rho_0} \nabla \phi_k \nabla \phi_j dx$$

$$B_{kj} = \int_{\partial\Omega_Z} \frac{i}{\rho_0 c_0 Z} \phi_k \phi_j d\xi$$

$$C_{kj} = \int_\Omega \frac{1}{\gamma p_0} \phi_k \phi_j dx.$$

To get rid of the nonlinearity that is caused by the fact that B depends on Z and therefore on ω Nicoud et al. suggest in [11] to model the impedance by $1/Z = 1/Z_0 + Z_1 \omega + Z_2 / \omega$. This way it will be possible to rewrite the problem as a quadratic problem, because of the multiplication by ω. The quadratic equation is easier to solve with traditional methods. However, solution methods for fully nonlinear eigenvalue problems have been developed, as stressed in chapter 5.

2.3.2 Incorporating the flame response

In the context of the finite elements method described above it is not at all hard to incorporate the flame response. Equation (2.17) needs to be discretized completely, whereas we only did two terms in the section before. So all that is left is to discretize is $\frac{\gamma-1}{\gamma p_0} \frac{q_{tot}}{\rho_0(x_{ref})U_{bulk}} n_u(x) e^{i\omega\tau_u(x)} \nabla \hat{p}(x_{ref}) \cdot n_{ref}$ in terms of $\hat{p}_j$. Following the same strategy as before, this term gives us another term DP in the matrix equation (2.18) with generic element

$$D_{kj} = \int_\Omega \frac{\gamma - 1}{\gamma} \frac{q_{tot}}{\rho_0(x_{ref})} U_{bulk} n_u(x) \exp^{i\omega\tau_u(x)} \phi_k \nabla \phi_j(x_{ref}) \cdot n_{ref} dx.$$

As is noted in [11], this term is nonlinear in ω and cannot be as easily be rewritten in such a way that we are provided with a quadratic eigenvalue problem, as can be done for the nonlinearity of the impedance. So, we must find a way to solve the following nonlinear eigenvalue problem:

$$(A - D(\omega))P + \omega B(\omega)P + \omega^2 CP = 0.$$

Nicoud et al suggest in [11] to use a simple Picard iteration:

$$(A - D(\omega_{k-1}))P + \omega_k B(\omega_{k-1})P + \omega_k^2 CP = 0,$$

or with the assumption about the impedance that $1/Z = 1/Z_0 + Z_1\omega + Z_2/\omega$:

$$(\mathcal{A} - D(\omega_{k-1}))P + \omega_k \mathcal{B}P + \omega_k^2 \mathcal{C}P = 0 \qquad (2.19)$$

where $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are altered versions of A, B and C to take the modelled impedance in account. However, we can solve the nonlinear eigenproblems without rewriting them as linear problems by using Jacobi-Davidson's method that will be presented in this report.

Chapter 3

Linear Eigenvalue Problems

The aim of this report is to compare methods that can be used to solve the type of eigenvalue problems that arise from combustion. These problems are often quadratic or even more non-linear. Not many solution methods have been developed for this type of problems. A basic strategy would therefore be to rewrite the problem as a linear problem, for which many highly optimized methods can be used. In this chapter two state-of-the-art methods, Arnoldi's method and Jacobi-Davidson, are presented. These methods can be used to efficiently solve linear eigenvalue problems. Jacobi-Davidson can also easily be extended to solve quadratic and non-linear problems. But before we go into detail about either method we will explain the simple eigenvalue problem and basic strategies to solve it.

An eigenvector of a transformation is a non-null vector whose direction remains unchanged when the transformation is applied to that vector. The length of the vector may be changed. The factor with which the vector is multiplied is called the eigenvalue. For example, imagine a 2D picture that is transformed so that its height is doubled, its width remains the same and it is mirrored in the y-axis. The eigenvectors will be the unit vectors $[1, 0]^T$ and $[0, 1]^T$ and the eigenvalues are -1 and 2 respectively.

The standard eigenvalue problem is how to find pairs of eigenvalues and eigenvectors for a given linear transformation. We can write this in matrix notation as follows:

$$Av = \lambda v$$

where A is the square matrix associated with the transformation, v is the eigenvector and λ is the eigenvector. Another way to describe the eigenvalues is as the roots of the characteristic polynomial of A: Since $(A - \lambda I)v = 0$ we must have that $\det(A - \lambda I) = 0$. From basic algebra we know that a polynomial must have n roots, this shows that any $n \times n$ matrix has n eigenvalues. However, these eigenvalues need not be distinct. In some cases A has less than n distinct eigenvectors. We call such a matrix defective.

Although the mathematical formulation is simple, solving it is hard, especially for large matrices. But since the problem arises from many practical situations studying it is very important. That is why several solution methods have been developed. We will describe three methods, namely the Power method, the Arnoldi method and the Jacobi-Davidson method.

3.1 Power Method

The Power Method is the most basic method to approximate eigenvectors and eigenvalues. The information given here about the power method has largely been taken from [14]. The method creates a sequence of vectors by multiplying by A and scaling. Or mathematically: $v_k = \frac{1}{\alpha_k} A v_{k-1}$. It is easily seen that if the sequence converges, then it must converge to an eigenvector. If we write $v = \lim_{k\to\infty} v_k$ then:

$$Av = A \lim_{k\to\infty} v_k = \lim_{k\to\infty} Av_k = \lim_{k\to\infty} \alpha_k v_{k+1} = \lim_{k\to\infty} \alpha_k \lim_{k\to\infty} v_{k+1} = \lim_{k\to\infty} \alpha_k v.$$

This means that we immediately have found an eigenvalue, namely the limit of α_k. It can be shown that the sequence only converges under the assumption that there is one and only one eigenvalue with the largest modulus. Unfortunately, this cannot be checked beforehand. Another drawback of this method is that when the starting vector is orthogonal to the eigenvector associated with the eigenvalue with the largest modulus then the sequence will not converge to that eigenvector.

Convergence speed is another problem for the power method. To explain this clearly we will first point out a nice property of eigenpairs. Since we know that $Av_k = \lambda_k v_k$, we can combine this for all k in a matrix equation (assuming that A is not defective):

$$[Av_1, \ldots, Av_k] = [\lambda_1 v_1, \ldots, \lambda_k v_k] \;\Rightarrow\; AV = V\Lambda$$

where Λ is the diagonal matrix containing the eigenvalues. This shows we can write $A = V\Lambda V^{-1}$ so $A^k = V\Lambda^k V^{-1}$, which implies that multiplying a vector by A^k will be dominated by the largest eigenvalues. This means that convergence speed depends on how much larger the largest eigenvalue is than the second-largest one. If the relative difference is small, convergence will be slow. It also shows that the power method will only converge to the eigenvalue with the largest modulus.

To make the use of the Power Method more attractive, a few possible amendments have been developed. We will introduce the Shift-and-Invert technique and Deflation.

3.1.1 Shift-and-Invert

The Shift-and-Invert technique is designed to improve performance of the power method, and to make it possible to find other eigenvalues than the one with the largest modulus. It uses two properties of matrices: If (λ, x) is an eigenpair of A then $(\lambda + \sigma, x)$ is an eigenpair of $(A + \sigma I)$, and λ^{-1} is an eigenvalue of A^{-1}. Adding a constant diagonal is called *shifting* of a matrix, and σ is the *shift*. It is clear that when the shift is close to an eigenvalue λ_k of A then $(\lambda - \sigma)^{-1}$ is the largest eigenvalue of $(A - \sigma I)^{-1}$. So, if we can efficiently use a method on the matrix $(A - \sigma I)^{-1}$ we can target certain eigenvalues. The closer the shift is to λ, the better the convergence. However, if the shift is chosen equal to an eigenvalue, the matrix will be singular. This is no problem, since we have found the desired eigenvalue.

If we use the Shift-and-Invert technique in combination with the power method, we will have to calculate $(A - \sigma I)^{-1}x = y$. Because inverting a matrix is expensive, it is better to write $(A - \sigma I)y = x$ and solve for y by decomposing $(A - \sigma I)$. The standard decomposition is the LU-decomposition. It may seem dangerous to use this technique of shifting and decomposing, because the shift could cause the matrix $(A - \sigma I)$ to be almost singular when it's close to an eigenvalue. Solving for y using an LU-decomposition will blow up round-off errors. However, this only happens in the direction of the eigenvector that belongs to the desired eigenvalue. But since a multiple of an eigenvector is again an eigenvector belonging to the same eigenvalue, this error has hardly any influence on convergence speed, even if we change the shift to be as close to the approximated eigenvalue as possible during the iterations. This makes it seem like we should update the shift as often as possible. However, every time the shift is changed, we will have to re-calculate this decomposition. Note that decomposing a matrix is more costly than solving a decomposed system, so there is a trade-off in convergence speed between the amount of work needed per iteration and the number of iterations until convergence. This also depends on the size of A. The larger the matrix, the more work is needed to decompose it. In case A is very large it may be better to use an iterative method to solve the system.

Iterative methods on the other hand, are often more prone to the possible problems arising from the fact that the matrix becomes almost singular. This means that changing the shift so that it is closer to an eigenvalue might even slow down convergence. The most important problem is that iterative methods are generally based on multiplying vectors by A, or in this case the shifted version $A - \sigma I$. The closer σ is to λ the *slower* these iterative methods will converge, even though the two largest eigenvalues of $(A - \sigma I)^{-1}$ are further apart the closer σ is to λ. This is a crucial weakness of the shift-and-invert method.

Deflation

A second technique to change the matrix such that another eigenvalue has the largest modulus is the so-called *deflation* technique. It is similar to the shift-and-invert technique in that it adds a matrix to A to change the eigenvalues, but it does not invert. The idea is to add a matrix that changes only one eigenvalue. To use it, only the right eigenvector corresponding to the eigenvalue with largest modulus is needed. We shift the matrix A by subtracting $\sigma u_1 v^H$ where u_1 is the known eigenvector and v^H is the

Hermitian transpose of a vector v such that $v^H u_1 = 1$. σ is the desired shift: the eigenvalues of $A - \sigma u_1 v^H$ will be $\{\lambda_1 - \sigma, \lambda_2, \dots, \lambda_n\}$. In [14] it has been shown empirically that, assuming the eigenvectors are normalized when they are found, choosing $v = u_1$ speeds up convergence in comparison to choosing a random vector with the property that $v^H u_1 = 1$. When already m eigenpairs are known, we can create an $n \times m$ matrix Q containing the eigenvectors as columns, and an $m \times m$ diagonal matrix Σ with the desired shifts as non-zero elements. It can be shown that the eigenvalues of the shifted matrix $A - Q\Sigma Q^H$ are $\{\lambda_1 - \sigma_2, \dots, \lambda_m - \sigma_m, \lambda_{m+1}, \dots, \lambda_n\}$. We can use this technique to find the eigenvalues of A, starting with the one with largest modulus and working our way down. Unfortunately, because the eigenvalues found are only approximations there will be some rounding errors that cause an increased error for each eigenvalue found.

3.2 Search spaces and Ritz Values

Although the shift-and-invert technique makes the Power method more powerful, it still has a weakness. The Power method only uses the last approximation to compute a new one. This means that all information from previous approximation is not used. An obvious improvement would be to work with subspaces. Important methods that use subspaces are Arnoldi and Jacobi-Davidson. As a supporting technique, we introduce the concept of Ritz values: θ_k is Ritz value of A with respect to the subspace $\mathcal{V}_k$ with Ritz vector u_k if

$$u_k \in \mathcal{V}_k, u_k \neq 0,\ Au_k - \theta_k u_k \perp \mathcal{V}_k.$$

This means that if we solve the projected eigenvalue problem on the subspace $\mathcal{V}_k$, we find Ritz values of A. A Ritz pair (u_k, θ_k) is a solution of the equation

$$W_k^* A V_k u_k = \theta_k W_k^* V_k u_k$$

which is found by writing $x = V_k u_k$ and multiplying the equation by an appropriate matrix W_k^*. Usually, we choose $W_k = V_k$.

V_k is a matrix of column vectors spanning $\mathcal{V}_k$. Note that we can choose V_k to consist of the orthonormal basis of $\mathcal{V}_k$, in which case $V_k^* V_k = I$ and the problem above reduces to an $k \times k$ eigenvalue problem: $(V_k^* A V_k - \theta I)u_k = 0$.

3.3 Basic Arnoldi

A more sophisticated algorithm than the power method is the Arnoldi method. This method originally was developed last century by Arnoldi in [1] to transform dense matrices into Hessenberg form. Arnoldi himself already noted that this method could be used to approximate certain eigenvalues, even without finishing the transformation algorithm. Later on, the method was used to find eigenvalues of large sparse matrices.

Arnoldi's method is a Krylov subspace method. Krylov subspaces are based on the simple power method. They are formed as followed: first a starting vector is chosen, say v_0. Then the Krylov subspace is defined as:

$$\mathcal{K}_k(A, v) \equiv \text{span}\{v, Av, A^2v, \dots, A^{k-1}v\}.$$

Notice that $\mathcal{K}_k$ is k-dimensional, while $A \in \mathbb{R}^{n\times n}$. Of course, k is always chosen (much) smaller than n.

Arnoldi's method creates k orthonormal vectors that form a basis for $\mathcal{K}_k$. The vectors are combined in an $n \times m$-matrix V_k, while an $k \times k$-Hessenberg matrix H_k is formed by $h_{ij} = (Av_j, v_i)$. The exterior eigenvalues of H_k can be used as approximations of eigenvalues of A. Usually, this will be a small fraction of the k eigenvalues of H_k. Note that by construction, $H_k = V_k^* AV_k$.

The basic algorithm is as follows:

Algorithm 1 Arnoldi

```
 1: Start: Choose an initial vector $v_1$ of length one,
 2: for $j = 1, 2, \dots, m$ do
 3:     $h_{ij} = (Av_j, v_i),\ i = 1, 2, \dots, j$, {1}
 4:     $w_j = Av_j - \sum_{i=1}^{j} h_{ij}v_i$, {2}
 5:     $h_{j+1,j} = ||w_j||_2$,
 6:     if $h_{j+1,j} = 0$ then
 7:         stop
 8:     end if
 9:     $v_{j+1} = w_j/h_{j+1,j}$.
10: end for
```

At line 3, we have to do a matrix-vector multiplication. This is a costly operation. We can store the result to use it again at line 4, where we perform Gram-Schmidt orthogonalization. In practice, this orthogonalization will be modified Gram-Schmidt. Keep in mind that this algorithm is only a very simple version, that only shows the idea of using a Krylov subspace as a search space for eigenvectors. For practical implementation, a lot of improvements can be made.

3.3.1 Restart

When working with a Krylov-subspace algorithm, we build a subspace from approximations of eigenvectors. When convergence is slow, for instance due to the starting vector, the subspace will continue to grow. This gives two problems. The first is that a growing subspace means larger matrices to work with, and this increases the cost in terms of computation work and time quadratically. The second problem is that we need a lot of memory to store all the information we have about the subspace. To solve these problems, a technique called *restart* has been developed. This is a general name for techniques that reduce the size of the Krylov subspace when a certain size is reached. Some methods restart with a one-dimensional subspace spanned by the latest approximation of the eigenvector. Other methods use several approximations as a basis of the starting subspace. Especially when working with the Jacobi-Davidson method, that will be introduced in the next section, it is common to restart with a subspace with dimension larger than 1. For Arnoldi's method, extensive research has been done to improve performance with clever restarts. Many of the results are implemented in the well-known Computational package ARPACK [8], that is used in e.g. MATLAB to calculate eigenvalues.

3.4 Jacobi-Davidson

Similar to the Arnoldi method, Jacobi-Davidson is a subspace method. It was first published in [17]. However, it does not use a Krylov subspace as search space for the eigenvector. In this method, we also construct matrices V_k and H_k, and use eigenvalues of H_k as approximations of the eigenvalues of A. The main difference is the way in which the matrix V_k is constructed. Instead of using the power method to create a Krylov subspace, we combine ideas of Jacobi and Davidson to look at the orthogonal projection of A onto the complement of our current approximation u_k to create the search space. Before we go deeper into the Jacobi-Davidson method, we will introduce the Jacobi and the Davidson method.

3.4.1 Jacobi

Jacobi published in [6] a method to find eigenvalues of a diagonally dominant matrix A of which the largest diagonal element is $a_{1,1} = \alpha$. The idea is to write the eigenvalue problem as:

$$A \begin{bmatrix} 1 \\ z \end{bmatrix} = \begin{bmatrix} \alpha & c^T \\ b & F \end{bmatrix} \begin{bmatrix} 1 \\ z \end{bmatrix} = \lambda \begin{bmatrix} 1 \\ z \end{bmatrix}.$$

This can be written as a system of two equations, writing θ as the approximation of λ:

$$\begin{array}{rcl} \alpha + c^T z & = & \theta \\ (F - \theta I)z & = & -b \end{array}.$$

If we start with a vector z we can compute θ from the first equation, and insert this value in the second equation and solve for z, thus iteratively approximating λ by θ. To use Jacobi for an arbitrary matrix A, one needs to diagonalize the matrix first (or at least do a few steps in order to make it a diagonally dominant matrix) and exchange some rows and columns to get the largest diagonal-value on the right place.

3.4.2 Davidson

Davidson's method [2] creates a subspace that is built from subsequent approximations of the desired eigenvector. Suppose that we have a subspace $\mathcal{K}$ of dimension k, with basis $v_1, \ldots, v_k$. We can compute the Ritz value θ_k and the Ritz vector u_k of the matrix A over this subspace. The residual is $r_k = Au_k - \theta_k u_k$. The method of expanding the search space is first to compute $t = (D_A - \theta_k I)^{-1} r_k$. Here, D_A is the diagonal of A, which is chosen because the cost of inverting $(D_A - \theta_k I)$ is drastically lower than for $(A - \theta_k I)$. Notice that in this way, we are actually approximating a shift-and-invert step. But because we use a diagonal approximation of $(A - \theta_k I)$, we don't have to worry about decomposing, avoiding shift-changes or iteratively solving for t. The vector t is made orthogonal to $v_1, \ldots, v_k$, and the resulting vector will be v_{k+1}, expanding $\mathcal{K}$. u_k will approximate an eigenvector of A, and θ_k again approximates the corresponding eigenvalue.

3.4.3 Jacobi-Davidson

The Jacobi-Davidson method finds inspiration in ideas from both methods described above. Starting with an initial guess, we search for a correction for the approximate eigenvector in the directions orthogonal to the current approximation, that is, in the subspace $u_k^\perp$. We will do so by first finding an approximation for the eigenvalue using the current approximation of the eigenvector. Then we use the approximated eigenvalue to find an approximation of the eigenvector. Clearly, this idea is similar to Jacobi's.

The approximation θ_k for the eigenvalue λ is found as follows:

$$Ax = \lambda x \Rightarrow x^* Ax = x^* \lambda x$$

Inserting u_k as approximation for x we will define $\theta_k = u_k^* A u_k$, where we assume that u_k has been normalized.

To approximate the eigenvector, we use the idea from Davidson to look in the subspace $u_k^\perp$. We want to find a correction $v \in u_k^\perp$ such that:

$$A(u_k + v) = \lambda(u_k + v)$$

$$\Rightarrow\ (A - \lambda I)v = -(A - \lambda I)u_k = -r_k + (\lambda - \theta_k)u_k \tag{3.1}$$

where $r_k = (A - \theta_k I)u_k$ is the residual from the latest approximation. We project this equation on the subspace $u_k^\perp$ by multiplying on the left side by $I - u_k u_k^*$. We use the following observations:

$$\begin{array}{rcl}
(I - u_k u_k^*)v &=& v, \\
(I - u_k u_k^*)r_k &=& (I - u_k u_k^*)(A - \theta_k I)u_k \\
&=& r_k - u_k(u_k^* A u_k - \theta_k) \\
&=& r_k, \\
(I - u_k u_k^*)u_k &=& 0
\end{array} .$$

If we now multiply (3.1) with $I - u_k u_k^*$ and replace the unknown λ by the known approximation θ_k, we find the so called *Jacobi-Davidson correction equation*:

$$(I - u_k u_k^*)(A - \theta_k I)(I - u_k u_k^*)v = -r_k \tag{3.2}$$

Since the $(A - \theta_k I)$ is transformed to be in $u_k^\perp$ the rank is less than n and the equation is in fact ill-posed. But we are only interested in the part of the solution that is in the same direction as v, so we will use this equation to iteratively approximate the correction v by $\hat{v}$, and the next approximation for the eigenvector will be $u_k + \hat{v}$. A popular method to use when solving iteratively is GMRES [15].

It is clear that the correction equation is similar to a shift-and-invert step. Remember that a shift-and-invert step is to find an update by multiplying the current approximation by the shifted and inverted matrix: $(A - \theta_k I)^{-1} v_k = v_{k+1}$. The difference between Jacobi-Davidson and Shift-and-invert is that Jacobi-Davidson restricts itself to a particular search direction. The matrix $A - \theta_k I$ will become almost singular when θ_k is close to an eigenvalue of A. The projected matrix in the correction equation however, is restricted to the direction that we have no information about in our search space. This means that matrix is projected away from the direction in which the singularity would occur, avoiding the problem of singularity when the shift is close to an eigenvalue.

3.4.4 Search spaces and Harmonic Ritz values

An obvious adaption of the Jacobi-Davidson method is to use the correction v not to correct the current approximation u_k directly, but to store the k-th correction as v_k. We can then build a subspace $\mathcal{V}_k = \text{span}\{v_1, \ldots, v_k\}$ of dimension k and use it to compute Ritz pairs of A. In this way, we obtain approximations of several eigenpairs at once. We can choose to focus on one value, or we can try to let more than one value converge. A problem with this technique is that it will only converge to exterior eigenvalues. When researching instabilities in combustion only the smallest eigenvalues are needed. For other applications however, interior eigenvalues are also important. These can be found by shifting the matrix A, and to keep convergence speed high, we could use so-called *harmonic Ritz values*. Harmonic Ritz values were introduced in [12]. μ_k is a harmonic Ritz value of A with respect to some linear subspace $\mathcal{W}_k$ if μ_k^{-1} is a Ritz value of A^{-1} with respect to $\mathcal{W}_k$. If we choose $\mathcal{W}_k = \text{span}\{Av_1, \ldots, Av_k\}$, $W_k = AV_k$, $x = AV_k y$ and $\mu = \lambda^{-1}$ then we can write, starting from the original eigenvalue problem:

$$\begin{array}{rrcl} & Ax & = & \lambda x \\ \Rightarrow & \mu x & = & A^{-1}x \\ \Rightarrow & \mu A V_k y & = & V_k y \\ \Rightarrow & \mu W_k^* W_k y & = & W_k^* V_k y \end{array}$$

We now have a generalized eigenvalue problem, which we will treat later in this report. Solving this problem for μ will give us an approximation of the smallest eigenvalues of the original problem. By combining harmonic Ritz values with a shift of the matrix (see section 3.1.1) we can efficiently obtain interior values. As mentioned before this technique will not be used in our implementation for quadratical eigenvalue problems, because the application does not make require it.

3.4.5 The Jacobi-Davidson Algorithm

The basic algorithm for calculation of a single eigenvalue of the standard eigenproblem using Jacobi-Davidson is provided by Gerard L.G. Sleijpen et al in [18]. The algorithm is:

Algorithm 2 Jacobi-Davidson Method for λ_{max} of A

```
1: Start with $t = v_0$, starting guess
2: for i=1,...,k-1 do
3:     $t = t - (t^* v_i) v_i$
4: end for
5: $v_k = t/||t||_2$
6: $v_k^A = A v_k$
7: for i=1,...,m-1 do
8:     $M_{i,k} = v_i^* v_k^A$
9:     $M_{k,i} = v_k^* v_i^A$
10: end for
11: $M_{k,k} = v_k^* v_k^A$
12: Compute the largest eigenpair of the eigenproblem $Ms = \theta s$ of the $k \times k$ matrix
    $M$, $(||s||_2 = 1)$
13: $u = Vs$
14: $u^A = V^A s$
15: $r = u^A - \theta u$
16: if $||r||_2 \leq \epsilon$ then
17:     $\tilde{\lambda} = \theta, \tilde{x} = u$
18:     STOP
19: end if
20: Solve $t \perp u$ (approximately) from $(I - uu^*)(A - \theta I)(I - uu^*)t = -r$
```

There are three important points in the algorithm that need some explaining. First, the latest expansion to the basis of the search space is orthogonalized using Gram-Schmidt. The basis vectors v_k together form the matrix V. Then, at lines 7-11, we construct the matrix V^*AV, which we use to calculate the Ritz values of A in the next step of the algorithm. At line 20 we solve the correction equation, the fundamental equation of the Jacobi-Davidson theory. Also note (again) that the correction equation is in fact a transformed shift-and-invert step with shift θ. The shift was obtained as a Ritz value at line 11. We can actually target any eigenvalue by choosing another Ritz value. Of course, this means that we need a good method to find the Ritz values. A commonly used method is the QR algorithm for eigenvalues, see [4]. We can solve for the eigenvalues directly in this case because it is typically a small problem (size k).

In the same paper, Gerard L.G. Sleijpen et al proposed a more advanced algorithm based on harmonic Ritz values and vectors, including restart and deflation techniques. The algorithm is given on the next page. This algorithm is very advanced, and has been implemented by Sleijpen in both MATLAB and Fortran. It is, however, a very general implementation. This makes it somewhat inefficient for the application at hand. Moreover, since this thesis focuses on quadratical eigenvalue problems we decided to write a code that could handle both linear, generalized and quadratic eigenvalue problems. We will go into more detail about this in the next few chapters.

Algorithm 3 Jacobi-Davidson Method for k_{max} eigenvalues of A close to τ

```
 1: Start with $t = v_0$, $k = 0$, $m = 0$, $Q = [\,]$, $R = [\,]$.
 2: while $k < kmax$ do
 3:   for $i = 1, \dots, m$ do
 4:     $t = t - (v_i^* t) v_i$
 5:   end for
 6:   $m = m + 1$, $v_m = t/||t||_2$, $v_m^A = Av_m - \tau v_m$, $w = v_m^A$
 7:   for $i = 1, \dots, k$ do
 8:     $w = w - (q_i^* w) q_i$
 9:   end for
10:   for $i = 1, \dots, m - 1$ do
11:     $M_{i,m}^A = w_i^* w$, $w = w - M_{i,m}^A w_i$
12:   end for
13:   $M_{m,m}^A = ||w||_2$, $w_m = w/M_{m,m}^A$
14:   for $i = 1, \dots, m - 1$ do
15:     $M_{i,m} = w_i^* v_m$, $M_{m,i} = w_m^* v_i$
16:   end for
17:   $M_{m,m} = w_m^* v_m$
18:   Make a $QZ$ decomposition $M^A S^R = S^L T^A$, $MS^R = S^L T$, $S^R, S^L$ unitary and
      $T^A, T$ upper triangular, such that: $|T_{i,i}^A / T_{i,i}| \leq |T_{i+1,i+1}^A / T_{i+1,i+1}|$
19:   $u = Vs_1^R, u^A = V^A s_1^R, \vartheta = \bar{T_{1,1}} \cdot T_{1,1}^A,$
20:   $r = u^A - \vartheta u, \tilde{a} = Q^* r, \tilde{r} = r - Q\tilde{a}$
21:   while $||\tilde{r}||_2 \leq \epsilon$ do
22:     $R = \begin{pmatrix} R & \tilde{a} \\ 0 & \vartheta + \tau \end{pmatrix}$, $Q = [Q, u],\ k = k + 1$
23:     if $k = k_{max}$ then
24:       STOP
25:     end if
26:     $m = m - 1$
27:     for $i = 1, \dots, m$ do
28:       $v_i = Vs_{i+1}^R$, $v_i^A = V^A s_{i+1}^R$, $w_i = Ws_{i+1}^L$, $s_i^R = s_i^L = e_i$
29:     end for
30:     $M^A,\ M$ are the lower $m \times m$-blocks of $T^A, T$ respectively
31:     $u = v_1, u^A = v_1^A, \vartheta = \bar{M_{1,1}} \cdot M_{1,1}^A$
32:     $r = u^A - \vartheta u, \tilde{a} = Q^* r, \tilde{r} = r - Q\tilde{a}$
33:   end while
34:   if $m \geq m_{max}$ then
35:     for $i = 2, \dots, m_{min}$ do
36:       $v_i = Vs_i^R$, $v_i^A = v^A s_i^R$, $w_i W s_i^L$
37:     end for
38:     $M^A,\ M$ are the leading $m_{min} \times m_{min}$-blocks of $T^A, T$ respectively
39:     $v_1 = u,\ v_1^A = u^A,\ w_1 = Ws_1^L, m = m_{min}$
40:   end if
41:   $\theta = \vartheta + \tau$, $\widetilde{Q} = [Q, u]$
42:   Solve $t \perp Q$ (approximately) from $(I - \widetilde{Q}\widetilde{Q}^*)(A - \theta_k I)(I - \widetilde{Q}\widetilde{Q}^*)t = -\tilde{r}$
43: end while
```

We will briefly explain some parts. The first part is used to orthogonalize the latest correction t against the current search space, and to expand the matrices and vectors accordingly. Then we use a QZ decomposition for both M^A and M using the same matrices S^R and S^L: $M^A S^R = S^L T^A$ and $MS^R = S^L T$. See the next section for more information about the QZ algorithm.

After this we compute new approximations for the eigenpair and the residual at line 21, and if we find that norm of the residual has fallen below a given threshold, we are satisfied with the approximation $\vartheta + \tau$ for the eigenvalue. We store this value in the matrix R, together with the vector $\tilde{a}$ such that R is an upper triangular matrix. Another matrix Q is formed such that $AQ = QR$.

From line 23 to line 39 the algorithm checks whether a restart is needed, and if a restart is necessary the variables are changed appropriately. This is the case if either the residual is below a certain threshold ϵ, meaning that we are satisfied with the current approximation, or when the dimension of the search space gets too large. In the first case we restart by removing the converged eigenvector from the basis, so that the dimension of the search space decreases by one. We also need to make appropriate choices for the other variables, see lines 27-32 . If we don't reach the threshold before $m \leq m_{max}$ we restart with $m = m_{min}$, where, m_{min} is the predetermined minimal number of vectors that forms the new basis. Also here, we need to restrict the other variables, see lines 35-39 . If neither condition is met, then we simply continue by updating θ, $\widetilde{Q}$ and t.

3.5 The QZ decomposition

In the algorithm for Jacobi-Davidson, we used the so called QZ decomposition. This decomposition is a more general form of the *Schur* decomposition. In fact, it is also referred to as the generalized Schur decomposition. The Schur decomposition aims to write a square matrix $A = Q^*UQ$ where Q is a unitary matrix containing the orthogonalized eigenvectors of A and U upper triangular with the eigenvalues of A on the main diagonal. The QZ decomposition decomposes two matrices using the same unitary matrices for both matrices. We will give this statement as a theorem:

Theorem 3.5.1 *If A and B are in $\mathbb{C}^{n\times n}$, then there exist unitary Q and Z such that $Q^*AZ = T$ and $Q^*BZ = S$ are upper triangular. If for some k, t_{kk} and s_{kk} are both zero, then $\lambda(A,B) = \mathbb{C}$. Otherwise $\lambda(A,B) = t_{ii}/s_{ii} : s_{ii} \neq 0$.*

$\lambda(A,B)$ is the spectrum (subset of $\mathbb{C}$ containing the eigenvalues) of the generalized eigenproblem $Ax = \lambda Bx$ described in the next section. This theorem and its proof can be found as theorem 7.7.1 in [4], as well as more information about these and other decompositions.

Chapter 4

Generalized and Quadratic Eigenvalue Problems

The standard eigenvalue problem is to find a vector and a scalar for a certain transformation such that the transformed vector is the same as the value multiplied with the vector. An obvious generalization of this problem is to transform the right hand side as well, so for a given pair of matrices A, B (representing transformations) we want to find a vector x and value λ such that:

$$Ax = \lambda Bx.$$

If B is I then we get our original eigenvalue problem back. This more general problem is called the *generalized eigenvalue problem*. It is usually preferred to define $\lambda = \alpha/\beta$ and writing:

$$(\beta A - \alpha B)x = 0,$$

since these numbers are even valid when β is zero, while λ will go to infinity in this case. We will call (α, β) an eigenvalue of the problem $(\beta A - \alpha B)x = 0$.

Generalized eigenproblems arise frequently from applications. If we expand the generalized eigenproblem by a quadratic term, we can deal with even more practical problems. This gives us the *quadratic eigenvalue problem*:

$$\lambda^2 Cx + \lambda Bx + Ax = 0$$

We will discuss both types of problems, restricting ourselves to what we need to understand how to apply search space methods to these problems. The theory presented in this section is for a large part taken from [14].

4.1 Generalized Eigenproblems

When dealing with Generalized Eigenproblems, a few problems may occur. For instance, there are infinitely many eigenvalues (α, β), because we can simply multiply with a constant to find another eigenvalue of the same eigenvector. There are several ways to deal with this: we could just define β to be 1, which could be established for any eigenvalue with $\beta \neq 0$ by scaling. Even though the case that $\beta = 0$ is rare, it is not needed to discard the option. A better solution is to scale the found eigenvalues such that $|\alpha|^2 + |\beta|^2 = 1$.

Another problem is that the matrix pair (A, B) may be singular, that is det($\beta A - \alpha B$) might be zero for all (α, β). In that case, any pair of numbers is an eigenvalue. This is not very interesting of course. It is important to realize that it is well possible for a matrix pair to be regular (non-singular) even when one or both of the matrices involved are singular. In fact, if either A or B is regular, we can write the Generalized eigenproblem as a standard eigenproblem by multiplying with the inverted regular matrix. In case A is singular, we also need to multiply by $1/\lambda$ to get a standard eigenvalue problem.

4.1.1 Properties of the Generalized Eigenvalue Problem

Before we discuss methods of solving generalized eigenproblems, we will look at two properties of these problems. We will see how certain transformations influence the eigenvectors and eigenvalues of the problem. When we multiply both A and B from the left with the same non-singular matrix Y, the right eigenvectors are preserved, while the left eigenvectors are multiplied by Y^{-*}. The first statement is trivial. The second statement follows from:

$$\begin{array}{rrcl} & v^*(\beta A - \alpha B) & = & 0 \\ \Leftrightarrow & v^* Y^{-1} Y(\beta A - \alpha B) & = & 0 \\ \Leftrightarrow & (Y^{-*}v)^*(\beta Y A - \alpha Y B) & = & 0. \end{array}$$

Here, we have assumed that v is a left eigenvector of the original problem, and 0 denotes the zero row vector of appropriate length. In the same way, we can show that when we multiply both matrices from the right with a non-singular matrix X the left eigenvectors are preserved, while the right eigenvectors are multiplied by X^{-1}. In both cases, the eigenvalues remain the same. We say that for non-singular X and Y the pair (YAX, YBX) is equivalent to (A, B).

The second interesting property is found as a theorem in [14]. The theorem states:

Theorem 4.1.1 *Let (A, B) be any matrix pair and consider the transformed matrix pair (A_1, B_1) defined by:*

$$A_1 = \tau_1 A - \sigma_1 B,\ B_1 = \tau_2 B - \sigma_2 A$$

for any four scalars $\tau_1, \tau_2, \sigma_1, \sigma_2$ such that the 2×2 matrix

$$\Omega = \begin{pmatrix} \tau_2 & \sigma_1 \\ \sigma_2 & \tau_1 \end{pmatrix}$$

is non-singular. Then the pair (A_1, B_1) has the same eigenvectors as the pair (A, B). An associated eigenvalue $< \alpha^{(1)}, \beta^{(1)} >$ of the transformed matrix pair is related to the eigenvalue $< \alpha, \beta >$ of the original pair by:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \Omega \begin{pmatrix} \alpha^{(1)} \\ \beta^{(1)} \end{pmatrix}$$

4.1.2 Projection Methods

Previously, we discussed projection (or search space) methods like Arnoldi and Jacobi-Davidson. We would like to find a way to solve generalized eigenproblems using these methods. In general, projection methods search for an eigenvalue (α, β) with an eigenvector u in a subspace $\mathcal{K}$ such that:

$$(\beta A - \alpha B)u \perp \mathcal{L}$$

for given subspaces $\mathcal{K}$ and $\mathcal{L}$. If we know two bases $V = v_1, \ldots, v_m$ and $W = w_1, \ldots, w_m$ of $\mathcal{K}$ and $\mathcal{L}$ respectively, we can rewrite the search space problem as a normal generalized eigenproblem of dimension m:

$$(\beta W^H AV - \alpha W^H BV)y = 0$$

where we have replaced u by Vy. The way in which the subspaces $\mathcal{K}$ and $\mathcal{L}$ are formed depends on the choice of method. Unfortunately, for the Arnoldi method this is not possible, since we cannot construct an appropriate Krylov subspace, so we have no $\mathcal{K}$. To make this more clear, remember that Arnoldi uses a Krylov subspace as a search space. To construct the Krylov subspace, the current approximation is multiplied by A and orthogonalized. In the case of a generalized eigenproblem, multiplying by A would not help because we also have a matrix B. The appropriate Krylov subspace would be $(v, B^{-1}Av, \ldots, (B^{-1}A)^k v)$, which is the Krylov subspace associated with the matrix $B^{-1}A$. In other words, to use Arnoldi we will have to write the problem as a standard eigenvalue problem $B^{-1}Ax = \lambda x$, except when we can find X and Y such that $YBX = I$ (or $YAX = I$), in which case we write $YAX = \lambda x$ where the eigenvectors are transformed as described before. With Jacobi-Davidson, we can easily extend the method to work for generalized or even quadratic problems, as will be shown in the following section. Finally, an outline for an algorithm will be given in the section about nonlinear eigenproblems. This algorithm can be easily adapted to fit generalized or polynomial eigenproblems.

4.2 Quadratic Eigenvalue Problems

Quadratic eigenvalue problems are of the form $\lambda^2 Cx + \lambda Bx + Ax = 0$. These problems often arise from practical applications, like a spring system. For Arnoldi, it is impossible to solve these problems directly. Usually, one will rewrite the problem to generalized form (linearization). We will present the most common way of linearization, and then turn our attention to directly solving a polynomial problem with Jacobi-Davidson.

4.2.1 Rewriting into a Generalized Eigenvalue Problem

When dealing with a quadratic eigenproblem it is always possible to rewrite the problem to a generalized eigenproblem. We can rewrite the problem as:

$$\lambda \begin{pmatrix} B & C \\ I & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda x \end{pmatrix} = \begin{pmatrix} -A & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} x \\ \lambda x \end{pmatrix}.$$

Notice that the Identity Matrix can be replaced by any matrix. This will affect performance of the method that is used to solve the generalized eigenproblem. The most popular alternative is to use C instead of I. This will usually give better performance, especially when all matrices involved are Hermitian (which happens often when the matrices arise from numerically discretized equations), since then the blockmatrices will also be Hermitian. If C is a symmetric positive definite matrix (which is the case for certain classes of physical problems) it may be better to rewrite the quadratic problem as follows:

$$\begin{pmatrix} -B & -A \\ I & 0 \end{pmatrix} \begin{pmatrix} \lambda x \\ x \end{pmatrix} = \lambda \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \lambda x \\ x \end{pmatrix}$$

This gives us a symmetric positive definite matrix on the left-hand side which can be inverted cheaply or, by using the right pre- and postmultiplications, be transformed to I. Notice that the size of the problem has increased to $2n$. For general polynomial eigenproblems of degree k the number of eigenvalues is kn if C is non-singular.

4.2.2 Direct Solving of the Quadratic Eigenproblem

In one of G.L.G. Sleijpen's articles [16] he indicates how to solve a quadratic eigenproblem (or more general: a polynomial eigenproblem) with a Jacobi-Davidson method. The general polynomial eigenproblem is to find a non-trivial eigenvector x and its associated eigenvalue $\lambda \in C$ such that:

$$A_0 x + \lambda A_1 x + \cdots + \lambda^n A_n x = 0 \tag{4.1}$$

For ease of notation, we write this in terms of a matrix-valued polynomial:

$$\Psi(\lambda)x = 0 \text{ where } \Psi(\vartheta) = A_0 + \vartheta A_1 + \cdots + \vartheta^n A_n$$

We proceed much like we did for the standard eigenproblem. We suppose we have a m-dimensional search subspace $\mathcal{V}_m$ and a m-dimensional projection subspace $\mathcal{W}_m$. We can then compute an approximation u of x with associated approximation ϑ of λ, by solving the following projected problem:

$$u \in \mathcal{V}_m, \vartheta \in \mathbb{C} \text{ such that } \Psi(\vartheta)u \perp \mathcal{W}_m. \tag{4.2}$$

The residual r is defined by $r \equiv \Psi(\theta)u$, and for some arbitrary $\tilde{u}$ we correct the approximation u by z_1, where z_1 is an approximate solution of the correction equation:

$$z \perp u \text{ and } \left(I - \frac{\tilde{w}w^*}{w^*\tilde{w}}\right)\Psi(\vartheta)\left(I - \frac{u\tilde{u}^*}{\tilde{w}^*u}\right)z = -r$$

for relevant choices of w and $\tilde{w}$. The speed of convergence depends heavily on the choice of these vectors. In their article, G.L.G. Sleijpen et al have suggested a couple of possibilities. We will present their two most straightforward choices: $\tilde{w} = w = \tilde{u} = u$ giving linear convergence, and $w = \tilde{u} = u$ and $\tilde{w} = T'(\vartheta)$ giving quadratic convergence [16].

The quadratic eigenvalue problems that arise from thermo-acoustic combustion applications are generally modelled with $C = I$, and only the eigenvalues with real part larger than zero are interesting. To efficiently search for these eigenvalues, we implemented the algorithm for quadratical eigenproblems as presented on the next page.

Algorithm 4 Jacobi-Davidson Method for quadratic eigenproblems arising from combustion

```
1: Start with $t = v_0$, starting guess.
2: $Conv = 0$, the number of converged eigenvalues.
3: $v_1 = t/\|t\|$
4: for $Iter = 1, \ldots, MaxIter$ do
5:   for $k = Conv + 1, \ldots, MaxSpace$ do
6:     $v_k^A = Av_k$
7:     $v_k^B = Bv_k$
8:     for i=1,...,m-1 do
9:       $M_{i,k}^A = v_i^* v_k^A$
10:      $M_{k,i}^A = v_k^* v_i^A$
11:      $M_{i,k}^B = v_i^* v_k^B$
12:      $M_{k,i}^B = v_k^* v_i^B$
13:    end for
14:    $M_{k,k}^A = v_k^* v_k^A$
15:    $M_{k,k}^B = v_k^* v_k^B$
16:    $H = \begin{pmatrix} -M^B & -M^A \\ I & O \end{pmatrix}$
17:    Compute the eigenpairs of the eigenproblem $Hy = \theta y$ of the $2k \times 2k$ matrix $H$.
18:    Select $\theta$ with smallest positive real part, and $s$, the normalized lower half of the corresponding eigenvector $y$
19:    $u = Vs$
20:    $u^A = V^A s$
21:    $u^B = V^B s$
22:    $r = u^A + \theta u^B + \theta^2 u$
23:    if $||r||_2 \leq$ tolerance then
24:      $Conv = Conv + 1$
25:      $Eival(Conv) = \theta$
26:      $Eivec(:, Conv) = u$
27:      if $Conv =$ number of desired eigenvalues then
28:        STOP
29:      end if
30:      Select new, unconverged eigenvalue $\theta$ with smallest positive real part, and $s$, the normalized lower half of the corresponding eigenvector $y$
31:      Determine new residual $r$ and vector $u$ as on lines 18-21
32:    end if
33:    if $k = MaxSpace$ then
34:      $V(:, 1 : Conv) = Eivec(:, 1 : Conv)$
35:      $V(:, Conv + 1) = u$
36:      Orthogonalize $V$
37:      Calculate $M^A = V^*AV$ and $M^B = V^*BV$
38:    end if
39:    Solve $t \perp u$ (approximately) from $(I - uu^*)(A + \theta B + \theta^2 I)(I - uu^*)t = -r$
40:    for $i = 1, \ldots, k$ do
41:      $t = t - (v_i^* t)v_i$
42:    end for
43:    $t = t/\|t\|_2$
44:  end for
45: end for
```

A few points might need more explanation. First of all, some parameters need to be set. The values of $MaxIter$ and $MaxSpace$, which represent the maximal number of iterations and the maximal search space size, need to be set by the user. Also, the tolerance needs to be defined. Usually the tolerance depends on the current approximation of the eigenvalue. More information about choices of the parameters and their effects is provided in chapter 7.

At lines 16 and 17 a small eigenvalue problem is constructed and solved. This can be done in several ways. The most obvious way is to linearize as in step 16 and solve using any available efficient method. Another possibility is not to linearize but to write the problem as a small quadratic eigenvalue problem. Methods to solve these problems are described in the following chapter about non-linear eigenproblems, where also methods to solve small non-linear problems are described. However, it is unlikely that this could speed up convergence since linearizing a small system is inexpensive.

At lines 23 - 32 we describe the way the algorithm handles converged eigenpairs. We could also check for the possibility that two eigenpairs converge during the same iteration, but because of the size of the problem this is very unlikely and would make the algorithm less efficient. Directly after this, at lines 33 - 38, we check whether we need to restart. This makes the upper bound of the for loop at line 5 redundant, but technically we do need an upper bound so we leave it as it is. During the restart, we decide to construct a new search space with the converged eigenvectors, together with the latest Ritz vector. This is a good choice, but it is certainly not optimal. We leave research about this topic to our successor(s).

Possibly the most important part of the algorithm is line 39, where we solve the correction equation. This doesn't have to be done exactly; in fact it will be inefficient to do so. We only need to approximate the vector t for a good convergence. There is a clear trade-off here. A better approximation of the actual solution of the correction equation is more expensive, but also means we need fewer iterations of Jacobi-Davidson. In our implementation we have used a highly optimized version of GMRES [3], provided by the parallel algorithms section of CERFACS.

Chapter 5

Nonlinear Eigenvalue Problems

The most difficult challenge in the context of eigenvalue problems is solving the most general form: the nonlinear eigenvalue problem. We write the problem as:

$$T(\lambda)x = 0 \tag{5.1}$$

where T is a matrix function of λ. This is obviously a generalization of the eigenproblems we have discussed so far. For example, if $T := \lambda^2 C + \lambda B + A$ then (5.1) reduces to a quadratic eigenproblem. Various physical problems are known to reduce to nonlinear eigenvalue problems, for example in acoustic modeling.

5.1 Numerical Methods for Nonlinear Eigenproblems

We will investigate how the two methods we have been studying so far, Arnoldi and Jacobi-Davidson, can be extended to the nonlinear case. It is known that these methods perform better for sparse matrices. For dense nonlinear eigenproblems, several other methods have been developed, some of which are discussed by Mehrmann and Voss in [19] and in [9]. The same articles research the iterative projection methods we are interested in: an Arnoldi-type method for nonlinear eigenproblems, as well as an extension of the Jacobi-Davidson method to the nonlinear case. A third projection is also proposed, namely the Rational Krylov method. We will give a summary of this research, with the exclusion of the Rational Krylov method as this falls outside the scope of our own research.

5.2 Newton Type Methods for Small Dense Problems

Before we turn our attention to subspace methods, we will describe some of the methods for small nonlinear eigenvalue problems presented in [19] and in [9]. Subspace methods are all based on constructing small eigenproblems and using the solutions of these problems to find a good approximation of the eigenvalues of the large problem. This means that we need ways to solve these small eigenproblems. Using linearization techniques or approximation by Picard iteration we only need to solve linear eigenvalue problems. However, it is likely that we can find these solutions faster by directly solving the small, nonlinear problems. We will present the methods of inverse iteration, residual inverse iteration and successive linear problems as possibilities to do so. However, we have only implemented the third method. We will leave comparing the different possibilities as a future research topic.

5.2.1 Inverse Iteration

From the linear case where $T(\lambda) = A - \lambda I$ it is known that the inverse iteration is equivalent to Newton's method applied to the nonlinear system [9]:

$$\begin{pmatrix} T(\lambda)x \\ v^*x - 1 \end{pmatrix} = 0.$$

Here, $v \in \mathbb{C}$ is any vector that satisfies $v^*x = 1$. If we let $T(\lambda)$ be nonlinear and apply one step of Newton's method to the same system we find:

$$\begin{pmatrix} T(\lambda_k) & T'(\lambda_k)x_k \\ v^* & 0 \end{pmatrix} \begin{pmatrix} x_{k+1} - x_k \\ \lambda_{k+1} - \lambda_k \end{pmatrix} = \begin{pmatrix} T(\lambda_k)x_k \\ v^*x_k - 1 \end{pmatrix}.$$

We can rewrite this in terms of x_{k+1}:

$$x_{k+1} = -(\lambda_{k+1} - \lambda_k)T(\lambda_k)^{-1}T'(\lambda_k)x_k \quad (5.2)$$

$$v^*x_{k+1} = 1. \quad (5.3)$$

The first expression gives us a direction u_{k+1} for the new approximation of the eigenvector:

$$u_{k+1} := T(\lambda_k)^{-1}T'(\lambda_k)x_k.$$

Multiplying the first equation with v^* and combining with the second gives $1 = -(\lambda_{k+1} - \lambda_k)v^*u_{k+1}$ or:

$$\lambda_{k+1} = \lambda_k - \frac{1}{v^*u_{k+1}}.$$

So we obtain an approximation for both the eigenvalue and (after normalization) the eigenvector.

5.2.2 Residual Inverse Iteration

A drawback of the inverse iteration is that in every iteration we need to decompose $T(\lambda)$ in order to find an update direction u_{k+1}. This can be solved by using $T(\sigma)$ instead of $T(\lambda)$, where σ is a fixed shift close to the desired eigenvalue λ. However, Mehrmann and Voss argue that simply substituting this in the inverse iteration will lead to misconvergence in the nonlinear case.

The residual inverse iteration proposed in [10] fixes this. According to the inverse iteration method we have:

$$\begin{aligned} x_k - x_{k+1} &= x_k + (\lambda_{k+1} - \lambda_k)T(\lambda_k)^{-1}T'(\lambda_k)x_k \\ &= T(\lambda_k)^{-1}(T(\lambda_k) + (\lambda_{k+1} - \lambda_k)T'(\lambda_k))x_k \\ &= T(\lambda_k)^{-1}T(\lambda_{k+1})x_k + \mathcal{O}(|\lambda_{k+1} - \lambda_k|^2) \end{aligned}$$

$$\Rightarrow x_{k+1} \approx x_k - T(\lambda_k)^{-1}T(\lambda_{k+1})x_k \tag{5.4}$$

where we can replace λ_k with the shift σ. Notice that in the third step we used a Taylor expansion of $T(\lambda_{k+1})$ around $T(\lambda_k)$.

The three important steps in the residual inverse iteration method are the following: first we solve λ_{k+1} from $v^*T(\sigma)^{-1}T(\lambda_{k+1})x_k = 0$, then we compute the residual $r_k = T(\lambda_{k+1})x_k$ and finally we solve $T(\sigma)d_k = r_k$ for d_k which we can use to find our next approximation x_{k+1}. Note that in this case we have to solve two systems per iteration instead of one as in the inverse iteration algorithm. However, since the system involves $T(\sigma)$ both times we only have to compute a decomposition once, before we start the algorithm.

5.2.3 Successive linear problems

A first order approximation of the non-linear eigenproblem is given by:

$$T(\lambda)x \approx (T(\tilde{\mu}) - \theta T'(\tilde{\mu})x = 0,\ \theta = \tilde{\mu} - \lambda.$$

Ruhe suggested in [13] the method of successive linear problems, based on this approximation. This method alternatively updates the approximation of the eigenvalue and the approximation of the eigenvector, just like the Residual inverse iteration. We start with an approximation of the eigenvalue, λ_1. Then we solve the linear eigenproblem $T(\lambda_k)u = \theta T'(\lambda_k)u$. We choose an eigenvalue θ of this problem which is smallest in modulus, and update the eigenvalue by setting $\lambda_{k+1} = \lambda_k - \theta$. It is obvious that if $\lambda_{k+1} = \lambda_k$ then $\theta = 0$ and $T(\lambda_k)u = 0$, which means (λ_k, u) is an eigenpair of the non-linear problem. The following theorem shows that the method converges quadratically:

Theorem 5.2.1 *Let T be twice continuously differentiable, and let $\hat{\lambda}$ be a solution of the nonlinear eigenvalue problem $T(\lambda) = 0$ such that $T'(\hat{\lambda})$ is nonsingular and 0 is an algebraically simple eigenvalue of $T'(\hat{\lambda})^{-1}T(\hat{\lambda})$. Then the method of successive linear problems converges quadratically to $\hat{\lambda}$.*

Voss supplies the proof in [19] as follows:

Proof Let $\hat{x}$ be an eigenvector corresponding to $\hat{\lambda}$, and let $v \in \mathbb{C}^n$ such that $v^H\hat{x} = 1$. Let $U(\hat{\lambda})$ be a neighborhood of $\hat{\lambda}$, and $\Phi : \mathbb{C}^n \times \mathbb{C} \times U(\hat{\lambda}) \to \mathbb{C}^{n+1}$ be defined by

$$\Phi(x, \theta, \lambda) := \begin{pmatrix} T(\lambda)x - \theta T'(\lambda)x \\ v^H x - 1 \end{pmatrix}$$

Then $\Phi(\hat{x}, 0, \hat{\lambda}) = 0$, and the matrix

$$\frac{\partial}{\partial(x,\theta)}\Phi(\hat{x},0,\hat{\lambda}) = \begin{pmatrix} T(\hat{\lambda}) & -T'(\hat{\lambda})\hat{x} \\ v^H & 0 \end{pmatrix}$$

is nonsingular. This will be proven later. By the implicit function theorem $\Phi(x,\theta,\lambda) = 0$ defines differentiable functions $x : U(\hat{\lambda}) \to \mathbb{C}^n$ and $\theta : U(\hat{\lambda}) \to \mathbb{C}$ on a neighborhood of $\hat{\lambda}$ again denoted by $U(\hat{\lambda})$ such that $\Phi(x(\lambda), \theta(\lambda), \lambda) = 0$ for every $\lambda \in U(\hat{\lambda})$. With this function the method of successive linear problems can be rewritten as the fixed point iteration $\lambda_{n+1} = \phi(\lambda_n) := \lambda_n - \theta(\lambda_n)$, which converges quadratically if $\phi'(\hat{\lambda}) = 0$. From the implicit function theorem it follows that

$$\begin{aligned} \tfrac{d}{d\lambda}\begin{pmatrix} x \\ \theta \end{pmatrix}(\hat{\lambda}) &= -\tfrac{\partial}{\partial(x,\theta)}\Phi(\hat{x},0,\hat{\lambda})^{-1}\tfrac{\partial}{\partial\lambda}\Phi(\hat{x},0,\hat{\lambda}) \\ &= -\begin{pmatrix} T(\hat{\lambda}) & -T'(\hat{\lambda})\hat{x} \\ v^H & 0 \end{pmatrix}^{-1}\begin{pmatrix} T'(\hat{\lambda})\hat{x} \\ 0 \end{pmatrix} \end{aligned}$$

which yields $\theta'(\hat{\lambda}) = 1$, and therefore $\phi'(\hat{\lambda}) = 0$.

All that is now left to prove is that the matrix $\frac{\partial}{\partial(x,\theta)}\Phi(\hat{x},0,\hat{\lambda})$ is nonsingular. Let

$$\frac{\partial}{\partial(x,\theta)}\Phi(\hat{x},0,\hat{\lambda})\begin{pmatrix} z \\ \mu \end{pmatrix} = \begin{pmatrix} (\hat{\lambda}) & -T'(\hat{\lambda})\hat{x} \\ v^H & 0 \end{pmatrix}\begin{pmatrix} z \\ \mu \end{pmatrix} = 0.$$

If $\mu = 0$ then it follows from the first component $T(\hat{\lambda})z = 0$. Hence, $z = \alpha x$ for some $\alpha \in \mathbb{C}$, and $0 = v^H z = \alpha v^H x = \alpha$ yields $z = 0$. If $\mu \neq 0$ then the first component reads $T(\hat{\lambda})z = -\mu T'(\hat{\lambda})x$. Multiplying by $T(\hat{\lambda})^{-1}$ yields $T'(\hat{\lambda})^{-1}T(\hat{\lambda})z = -\mu x \neq 0$, from which we obtain

$$T'(\lambda)^{-1}T(\lambda))^2 z = -\mu T'(\lambda)^{-1}T(\lambda)x = 0$$

contradicting the fact that 0 is assumed to be an algebraically simple value of $T'(\lambda)^{-1}T(\lambda)$. ∎

5.3 Iterative Projection Methods

When discussing Arnoldi and Jacobi-Davidson type methods, we will follow the same path we did before: assuming that we have a search space of dimension m and an orthonormal basis V, we try to find a new approximation of the desired eigenvector and -value. To find the update, we use the basis of the search space to reduce the dimension of the original problem to m, that is we solve the nonlinear eigenproblem

$$V_k^H T(\vartheta_k) V_k y_k = 0.$$

If we have found a solution $(\theta_k, \tilde{y}_k)$ of this problem, and we write $x_k = V_k \tilde{y}_k$ then (θ_k, x_k) is a Ritz pair of T. We can use this Ritz pair to find an update for the approximated eigenpair. Also, we need these values to expand the search space. The way in which this is done makes the difference between Arnoldi and Jacobi-Davidson.

5.3.1 A General Algorithm for Nonlinear Eigenproblems

A general algorithm for projective methods is outlined as follows:

Algorithm 5 General projection method for nonlinear eigenproblems

```
start with an initial shift $\sigma$ and an orthonormal basis $V$
while $m \leq m_{max}$ do
    compute appropriate eigenvalue $\theta$ and corresponding eigenvector $y$ of the
    projected problem $V^H T(\lambda) V y = 0$
    determine Ritz vector $u = Vy$ and residual $r = T(\theta)u$
    if $||r||/||u|| < \epsilon$ then
        accept approximate eigenpair $\lambda_m = \theta$, $x_m = u$
        $m = m + 1$
        choose new shift $\sigma$
        restart if necessary
        determine approximations $\theta$ and $u$ of next desired eigenvalue and vector
        compute residual $r = T(\theta)u$
    end if
    determine expansion of search space
end while
```

When this algorithm is used for Arnoldi, we need a preconditioner M, which will be shown later. This matrix has to be determined every time we change the shift σ. It is used only to determine the vector that expands the search space. For Jacobi-Davidson we do this using the correction equation, which can be improved by adding a preconditioner. See Sleijpen et al [16], section 7.1 for a discussion on preconditioning the correction equation.

5.3.2 Arnoldi type methods

It is not possible to extend the original Arnoldi method for standard eigenvalue problems to the nonlinear case because we cannot construct a Krylov subspace to use as a search space. However, the method that is proposed by Voss in [20] and [21] is named after Arnoldi because the new search direction is orthonormalized against the previous vectors. The expansion of the current basis V_k can be chosen as $\hat{v}_{k+1} = x_k - T(\sigma)^{-1}T(\theta_k)x_k$ or equivalently as $v_{k+1} = T(\sigma)^{-1}T(\theta)x_k$ since x_k is the Ritz vector and therefore contained in V_k. This choice is based on the residual inverse iteration method, see equation (5.4). A drawback of this method is that we need to solve a large system because of the presence of $T(\sigma)^{-1}$ in the equation. When the problem under consideration is large (which is often the case) then solving will be too expensive. In this case the Arnoldi type method is only efficient if a reasonable preconditioner $M \approx T(\sigma)^{-1}$ is available.

5.3.3 Jacobi-Davidson type methods

A good alternative for Arnoldi is Jacobi-Davidson, especially when there is no preconditioner available. Again we assume to have a Ritz pair (θ_k, x_k). We can simply use the correction equation 3.2 to find an expansion of our basis:

$$\left(I - \frac{p_k x_k^*}{x_k^* p_k}\right) T(\theta_k) \left(I - \frac{x_k x_k^*}{x_k^* x_k}\right) z_{k+1} = -r_k, \; z_{k+1} \perp x_k$$

where we choose $p_k := T'(\theta_k)x_k$ and $r_k := T(\theta_k)x_k$. If we solve this equation approximately for z_{k+1} we can use this vector to expand V_k. Writing out the correction equation, and using that $x_k^* z_{k+1} = 0$ we find that:

$$T(\theta_k)z_{k+1} - \alpha_k p_k = -r_k, \text{ with } \alpha_k := \frac{x_k^* T(\theta_k) z_{k+1}}{x_k^* p_k}$$

Solving for z_{k+1} gives that $z_{k+1} = -x_k + \alpha_k T(\theta_k)^{-1}T'(\theta_k)x_k$, and shows that $\tilde{z}_{k+1} := T(\theta_k)^{-1}T'(\theta_k)x_k \in \text{span}[V_k, z_{k+1}]$. In [20] this vector $\tilde{z}_{k+1}$ is shown to be the direction in which the inverse iteration method with shift θ searches after one step. This means that just as in the linear case, Jacobi-Davidson is a subspace method based on shift-and-inverse iteration, giving quadratic convergence.

5.4 Picard iteration methods

The extensions described above for Arnoldi and Jacobi-Davidson are theoretically sound, but hard to implement. It is especially difficult to find a good way to solve the small nonlinear eigenvalue problem. In our comparison of Arnoldi and Jacobi-Davidson, we have decided to use a Picard iteration to deal with nonlinearities in the eigenvalue problem. This is done in different ways for Arnoldi and Jacobi-Davidson. We will describe both methods as we used them.

5.4.1 Arnoldi with Picard iteration

In chapter 4 we discussed how Arnoldi's method for standard eigenvalue problems can be extended to generalized and quadratic problems. To deal with nonlinear problems, we cannot simply write a linear version of the same problem. A possibility is to use the method of successive linear problems combined with Arnoldi, that is to repeatedly solve the problem $T(\omega_i)u = \theta T'(\omega_i)u$ and the update equation $\omega_{i+1} = \omega_i - \theta$. This method is only useful when the derivative of T with respect to ω is easy to calculate. However, in our application we use the fact that combustion problems are typically structured as quadratic eigenvalue problems. we can first write the problem as a quadratic problem, and then add the remaining nonlinearity. Using Picard, we then have to repeatedly solve equation (2.3.2):

$$(\mathcal{A} - D(\omega_{k-1}))P + \omega_k \mathcal{B} P + \omega_k^2 \mathcal{C} P = 0$$

We solve this equation as described in chapter 4.

5.4.2 Jacobi-Davidson with Picard iteration

In Jacobi-Davidson, we have no problems with applying the algorithm to nonlinear problems. The only difficulty is that the algorithm requires us to find solutions of the small nonlinear eigenvalue problem $V^H T(\lambda) V y = 0$. It will be very hard to find all eigenvalues, which is easy for small quadratic or linear problems. The method of successive linear problems has the possibility to provide one or a few eigenvalues. Another option is to apply the same kind of Picard iteration to this small problem as we did for the large problem with Arnoldi. First we will write the small problem as a quadratic eigenproblem, and then use Picard to deal with the remaining nonlinearities. In the case of Jacobi-Davidson, this means that we will have to repeatedly solve a small problem, whereas with Arnoldi we have to repeatedly solve a large problem. This will cause Arnoldi to suffer more from this adaption, which again shows the limitation of Arnoldi's method when it comes to more complex eigenvalue problems.

Chapter 6

MATLAB Tests

We implemented Jacobi-Davidson in MATLAB as a prototype for the implementation in Fortran, and to make a comparison with Arnoldi for academic test problems. In particular, we made the MATLAB implementation suitable even for nonlinear eigenvalue problems, which we have not done in Fortran. In MATLAB, ARPACK is implemented using Arnoldi's method in a highly optimized fashion. This makes a fair comparison hard - we would have to implement Arnoldi ourselves. We decided to use MATLAB to design and test our implementation of Jacobi-Davidson, and to convert this implementation to Fortran so that a fair comparison can be made. This is done in the next chapter. In this chapter we will present results from various small test problems. In Fortran we focused on realistic problems with large grids, restricted to a quadratic nature. In MATLAB we have solved truly nonlinear eigenvalue problems, with non-constant impedance boundary conditions. With Fortran we only used test cases where the impedance was constant, giving quadratic eigenvalue problems. In Fortran however, we are able to handle much larger problems because calculating in Fortran is faster and more memory efficient.

6.1 Description of the Test Problems

Our goal was to treat three different types of academic problems as suggested by CERFACS. The first is simply solving the wave equation with fully reflective boundaries numerically. This results in a generalized eigenvalue problem as described in chapter 2. The second problem is to change the boundary condition on one or more boundaries to an imposed reduced complex impedance boundary condition. This can be written as a Robin condition, or in physical terms a partially absorbing and partially reflecting wall. The relation between absorbing and reflecting is determined by the impedance parameter. Leaving it constant results in a purely quadratic eigenvalue problem. The third problem is a truly non-linear eigenvalue problem, which we have to solve when the complex impedance depends on the frequency of the wave. This frequency is related to the eigenvalue of the problem, as shown in equation 2.14, which is of course only known by approximation during the calculation, so the impedance is not known exactly either. This is more realistic than keeping the impedance constant during the calculation, however.

6.1.1 Physical Parameters

We have to decide on several parameters. For the geometry, we have taken a rectangular area of 0.5 m $\times$ 0.1 m. We use a coarse grid of 500 nodes and a fine grid of 8000 nodes. To simulate a realistic combustion chamber we would need to work with very large matrices which is not feasible in MATLAB. We have done a realistic simulation with the Fortran code treated in chapter 7. The grid with 8000 points is shown in the following image, as well as a zoomed version:

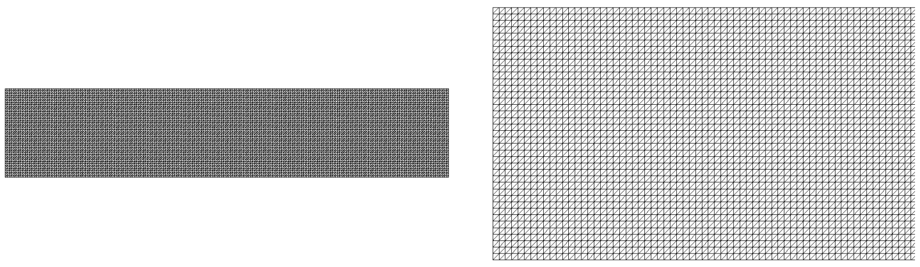
Figure 6.1: The two-dimensional grid of 8000 points, left image is the overview and right image zoomed in

The thermodynamic parameters are:

$$\begin{cases} T &= 300\,K \\ p_0 &= 101.325\text{Pa} = 1\,atm \\ \gamma &= 1.4 \\ r &= 287\,J\,K^{-1}\,kg^{-1} \end{cases}$$

where T is the temperature, p_0 the constant part of the pressure (the eigenvalue problem will yield the fluctuating part), γ is the adiabatic coefficient, or the heat capacity per mass unit at fixed pressure divided by the heat capacity per mass unit at fixed volume, and finally r is the difference in those heat capacities. These choices result in the following values for the speed of sound c and the density ρ: $c = 347\,m\,s^{-1}$ and $\rho = 1.1768\,kg\,m^{-3}$.

6.2 Implementation

We have implemented the two methods in roughly the same way. The main program loads the prefabricated matrices into the workspace, and sets the parameters. It then either calls Arnoldi's method or Jacobi-Davidson. We implemented the latter ourselves, whereas Arnoldi is the method used by the

MATLAB function `eigs`. This function uses the implementation of Arnoldi called ARPACK. Our implementation of Jacobi-Davidson constructs the matrices W^*AW, W^*BW and W^*CW, where W is a matrix containing basis vectors that span the current search space. It then uses the QR algorithm 3.4.5 to solve the small generalized eigenvalue problem:

$$\begin{pmatrix} -W^*BW & -W^*AW \\ I & O \end{pmatrix} \begin{pmatrix} \omega s \\ s \end{pmatrix} = \omega \begin{pmatrix} W^*CW & O \\ O & I \end{pmatrix} \begin{pmatrix} \omega s \\ s \end{pmatrix}.$$

The algorithm we used for Jacobi-Davidson is given in section 4.2.2.

An alternative to linearizing and using Arnoldi's method is to solve the non-linear eigenproblem $W^*AWs+\omega W^*BWs+\omega^2W^*CWs = 0$. This can be done using the successive linear problems method as described in section 5.2.3

From the different eigenpairs we find by solving the small eigenproblem we select the one of which the eigenvalue is closest to the target. If the stopping criterion is met then we have found a converged eigenpair. If not, then we take the residual r and use GMRES to approximate the solution of the correction equation $(I - uu^*)(A - \theta I)(I - uu^*)t = -r$ for the vector $t \perp u$, where $u = Ws$. This vector is then orthogonalized against W and expands the search space.

6.2.1 Numerical Parameters

We have set the tolerance of Jacobi-Davidson at $tol = 10^{-8}$. The stopping criterion is $\|r\| <= tol|\omega|$, where r is the residual and ω is the approximation of the eigenvalue. For Arnoldi we use the MATLAB function `eigs`, which uses ARPACK with shift-and-invert techniques. The stopping criterion for Arnoldi is the default criterion of ARPACK. The starting vector as well as the minimal and the maximal search space size are also the defaults suggested by ARPACK for Arnoldi. For Jacobi-Davidson the starting vector is the normalized version of a random vector. We set the maximal search space size at 30, and the minimal size depends completely on the number of converged eigenpairs. Jacobi-Davidson uses GMRES to find the solution to the correction equation. The amount of GMRES iterations is 30 for linear and quadratic problems, but only 5 for nonlinear problems. For the linear problems, we used JDQZ (see algorithm 3). We have used the default parameters suggested by the programmer, with the exception of the number of iterations between restarts (we chose 15 instead of 5) and the number of GMRES iterations to solve the correction equation. We decided to take 30 GMRES iterations, because throughout testing with both MATLAB and Fortran this value seemed to work very good for acoustic problems.

6.3 Results

We present the results of the different tests. The first table shows the eigenvalues found by both methods for a problem where the matrix B is zero. This kind of problem can easily be rewritten as a linear problem, which gives us the opportunity to compare the values found numerically with the actual frequencies which can be determined analytically. In the second table we show the results of the quadratic problem. This problem is the same as the linear one, but with B a constant non-zero matrix, depending on the constant-valued impedance. This means that we cannot rewrite it to a linear problem, but it is still possible to find the analytical eigenvalues. This is also the case for the third problem, which is nonlinear because of the dependence of the impedance on the eigenvalue.

6.3.1 Linear Problem

For the linear problem, we have used JDQZ and compared it with the MATLAB function `eigs` which uses ARPACK. The results are given in the next table.

	Small		Large	
	AR	JD	AR	JD
λ_1	0	0	0	0
λ_2	2181	2181	2181	2181
λ_3	4360	4360	4363	4363
λ_4	6535	6535	6544	6544
λ_5	8703	8703	8725	8725
λ_6	10855	10855	10904	10904
time in s	0.46	3.18	2.57	65.35

Table 6.1: Results for linear problems using ARPACK and JDQZ

This table shows that the built-in MATLAB is much faster in solving this particular problem. The eigenvalues that are found are the same. ARPACK is much faster because of the optimizations, and because it uses shift-and-invert techniques. The following plot shows the convergence of Jacobi-Davidson.

These plots show that during the first iterations convergence is relatively slow, but once an eigenpair has converged a new one is found every 10 iterations. An interesting detail is the trouble that Jacobi-Davidson has in both cases with the 6th eigenvalue. It can be assumed that a bad restart takes place, meaning that the method has to restart because the search space is getting too big, but that useful information is being thrown away during the restart. This is almost inevitable when restarting, and further research is needed to find a way of restarting in a better way.

We can also use the methods that we have written for quadratic problems to solve linear problems. This not smart to do in general, but the results for

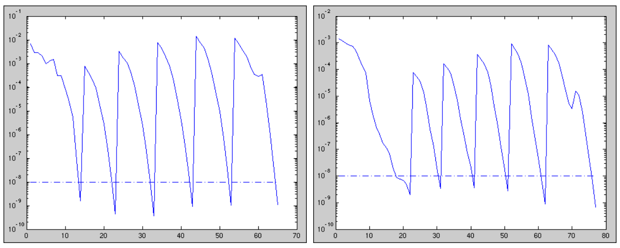

Figure 6.2: Convergence history for JDQZ, left is for the small matrix, right for the large matrix

Jacobi-Davidson in the case of a large matrix are surprisingly much better! The reformulation of the quadratic eigenvalues has caused the eigenvalues to become imaginary instead of real, but otherwise the values are equal to those found by JDQZ and 'linear Arnoldi'.

	Small		Large	
	AR	JD	AR	JD
λ_1	0	0	0	0
λ_2	$\pm 2181i$	$\pm 2181i$	$\pm 2181i$	$\pm 2181i$
λ_3	$\pm 4360i$	$\pm 4360i$	$\pm 4363i$	$\pm 4363i$
λ_4	$\pm 6535i$	$\pm 6535i$	$\pm 6544i$	$\pm 6544i$
λ_5	$\pm 8703i$	$\pm 8703i$	$\pm 8725i$	$\pm 8725i$
λ_6	$\pm 10855i$	$\pm 10855i$	$\pm 10904i$	$\pm 10905i$
time in s	0.644	2.71	8.173	29.74

Table 6.2: Results for linear problems with methods for quadratic problems

6.3.2 Quadratic Problem

For quadratic problems naturally we can only use the methods that are designed for these problems. We have checked the validity of the results. The eigenvalues with negative imaginary part found by both methods correspond directly with analytic eigenvalues. Notice that also eigenvalues with positive imaginary part are provided by the numerical methods, but these correspond to negative frequencies, which cannot be deducted from the analytic eigenfrequencies, because those are always positive.

As mentioned before, the eigenvalues found are correct. It's clear that Arnoldi is again faster, but this can again be explained by the optimal implementation of ARPACK. The fact that the difference between the methods is smaller in this case gives good hope that when implemented in a similar way, Jacobi-Davidson can be faster.

	Small		Large	
	AR	JD	AR	JD
λ_1	0	0	0	0
λ_2	$-240.6 + 272.7i$	$-241.7 + 271.9i$	$-241.4 + 272.1i$	$-241.4 + 272.1i$
λ_3	$-240.6 - 1908.5i$	$-246.7 - 1920.6i$	$-240.6 - 1908.8i$	$-240.6 - 1908.8i$
λ_4	$-240.7 + 2453.9i$	$-241.2 + 2454.3i$	$-240.7 + 2454.1i$	$-240.7 + 2454.1i$
λ_5	$-240.5 - 4087.4i$	$-242.0 - 4089.1i$	$-240.6 - 4090.1i$	$-240.6 - 4090.2i$
λ_6	$-240.6 + 4633.0i$	$-240.8 + 4633.0i$	$-240.7 + 4635.5i$	$-240.7 + 4635.5i$
time in s	0.775	1.20	11.74	26.70

Table 6.3: Results for quadratic problems

6.3.3 Nonlinear Problem

The easiest way to extend the method we used for quadratic methods to solve general nonlinear problems is to implement Picard iterations. For Arnoldi, this means taking an initial guess for the desired eigenvalue, and setting it constant where the equation is nonlinear. This results in a quadratic problem, which we can solve with ARPACK. The eigenvalue is then taken as a new guess, which leads to a new quadratic problem. This continues until the residual $r = T(\lambda)$ is considered small enough, according to a predetermined tolerance. For Jacobi-Davidson, we can do the same, but we can also avoid having to solve a large problem over again, by implementing the Picard-iteration in the inner loop. This means using the initial guess when the small projected problem is constructed, providing a small quadratic eigenproblem. We then simply create a new small quadratic eigenproblem using the result of the previous. This Picard iteration is much cheaper.

	Small		Large	
	AR	JD	AR	JD
λ_1	0	0	0	N/A
λ_2	$380.3 - 627.0i$	$380.4 - 627.0i$	$380.3 - 628.7i$	N/A
λ_3	$760.5 - 1254.2i$	$760.8 - 1254.1i$	$760.8 - 1253.6i$	N/A
λ_4	$1140.4 - 1880.6i$	$1524.8 - 2505.2i$	$1142.1 - 1880.1i$	N/A
λ_5	$1524.2 - 2506.6i$	$1524.7 - 2505.4i$	$1522.3 - 2507.8i$	N/A
λ_6	$1885.7 - 3135.1i$	$1524.1 - 2505.9i$	$1903.5 - 3135.8i$	N/A
time	177.5	31.4	2451	x

Table 6.4: Results for nonlinear problems

These results look worse than those of the quadratic problem. Jacobi-Davidson finds one eigenvalue three times, skipping two others that are found by Arnoldi. On the other hand, Arnoldi takes a very long time to solve these problems. The nonlinearity increases computing time drastically. Jacobi-Davidson takes less time, but if the problem becomes larger it fails to even find one eigenvalue. This indicates that nonlinear eigenvalue problems are very hard to solve indeed.

6.4 Nonlinear problems with direct solving of the small eigenproblem

We have spent a lot of time trying to find a way to deal with nonlinear eigenproblems. An obvious way to improve the performance of Jacobi-Davidson is to replace the Picard-iteration in the inner loop by another way of finding an eigenpair of the small nonlinear problem. This can be done using the method of successive linear problems (SLP), as described in section 5.2.3. This is a Newton method and should provide quadratic convergence. However, when implemented in Jacobi-Davidson for quadratic problems the performance of JD decreases, and when used to solve nonlinear problems convergence is halted before the stop-criterion is met. Jacobi-Davidson with SLP has a lot of potential, but needs to be studied upon more before a working implementation can be provided. Especially the accuracy is below an acceptable level.

6.5 Provisional Conclusions

From the results presented above we can conclude that Arnoldi and Jacobi-Davidson both handle quadratic problems quite well. Because of the optimized form of Arnoldi as implemented in ARPACK, this method is still faster for almost all testcases. However, Jacobi-Davidson is far from fully optimized in this implementation. The fact that both methods find the correct eigenvalues in a comparable amount of time indicates that we need to compare them at a lower-end level of programming, which makes it easier to reach the same kind of optimization for both methods. For the nonlinear test case, Jacobi-Davidson is even faster when using the coarse grid, but does not converge at all when using the finer grid. Arnoldi finds the same eigenvalues in both cases, indicating that the results are correct. However, the long time it takes to find the results for the finer grid motivates further research to find a method that performs better.

Chapter 7

Fortran Tests

In this section, we wish to compare the Arnoldi method and the Jacobi-Davidson method for solving large quadratic eigenvalue problems in a realistic setting. We will do so by performing several numerical tests using AVSP, a tailor-made software product by CERFACS that is specialized in solving the wave equation by transforming the equation to a Helmholtz equation. This program is a linear finite elements code with triangular or tetraedrical elements and lumped mass matrices. Before performing these tests, we need to perform preliminary tests to determine the optimal values of the parameters of the two methods. We cannot compare correctly if one of the methods is not performing optimally. Unfortunately, the optimal speed would only be obtained by using preconditioned GMRES in Jacobi-Davidson to solve the correction equation and using shift-and-inverted Arnoldi. This is difficult to do, since our implementation is matrix-free. Instead, we use a subroutine to calculate the matrix-vector product every time we need the vector Ax, Bx or Cx for given x. For this reason, we will leave preconditioning for Jacobi-Davidson and shift-and-invert techniques for Arnoldi outside the scope of the research. This way, we can make a fair comparison as it is. After finding the optimal parameters we compare the two methods for an academic two-dimensional problem, a three-dimensional problem, and for a real combustion chamber. We will apply three different boundary conditions in each case. We will use several gridsizes for each problem. We will look for the time it takes to converge to a fixed tolerance with optimal parameters, and we will investigate the number of matrix-vector products and scalar products used by each method.

7.1 Description of the testcases

We have performed numerical tests on nine different problems. Each problem is based on the wave equation. This equation is then discretized on a given grid. The implementation is matrix-free, for every matrix-vector multiplication AVSP simply determines the result by calculating the value in every element. This means the eigenvalue problem must be solved using matrix-vector product routines instead of the matrices themselves. ARPACK was already used by AVSP to give the solution of the eigenproblem, and we have branched Jacobi-Davidson into AVSP at the same place to compare the performance of both methods at a level comparable to industrial application. The only thing we did not do is to parallelize the program, but in [5] it is shown that Jacobi-Davidson is very well parallelizable. Therefore, the results for a serial implementation will be a strong indication for performance in a parallel environment.

The first three problems are two-dimensional with the same geometry but different boundary conditions. We define a rectangular area of 1 meter by 0.2 meter. The speed of sound will in all testcases be assumed constant at 340 m/s. The three sets of boundary conditions are chosen to represent three levels of difficulty, as experienced with ARPACK. The first is to simply take the normal velocity to be zero on all boundaries, in other words to have fully reflecting walls. We know the analytical eigenvalues, which makes it easy to check whether both methods have converged. In this case, the resulting eigenvalue problem, with the assumption that the mass matrix is lumped, reduces to a standard, linear eigenvalue problem. The second and third set of boundary conditions only differ from the first by imposing an impedance boundary condition on the inlet wall. This results in a quadratic eigenvalue problem. We define two different (but constant) values for the impedance, because from experience it is known that the performance of AVSP using Arnoldi's method is sensitive to this value. We solve the resulting equations on four different regular grids with an increasing number of cells. We have used respectively 500, 2000, 8000 and 32000 nodes. The structure of these grids are the same as in the matlab tests where we also used grids with 500 and 8000 points.

The second set of three problems is closely related to the first. The geometry is a rectangular box with dimensions 1m x 0.2m x 0.1m. We use the same boundary conditions as in the two-dimensional problem, resulting in a linear eigenproblem and two quadratic eigenproblems of varying difficulty. We constructed regular grids of approximately 500, 2000, 8000 and 32000 nodes. Finally we tested the two methods on the grid of an actual combustion chamber named Arrius. This grid contains a little more than 22000 nodes. In this testcase, we again applied three different sets of boundary conditions in the same way as for the academic testcases.

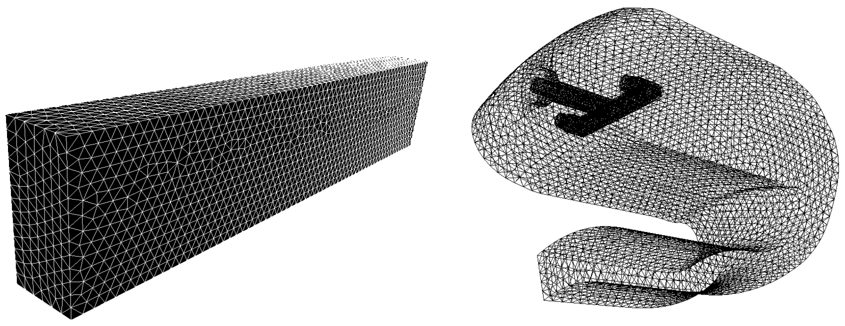

Figure 7.1: Three-dimensional grids

7.2 Parameter settings

The parameters involved for ARPACK are the number of desired eigenpairs, the minimal and maximal size of the search space and the tolerance. For Jacobi-Davidson the latter three parameters are important as well, together with the maximum size of the subspace used by GMRES to solve the correction equation. The number of desired eigenvalues is not as important for Jacobi-Davidson. In ARPACK, the minimal size of the subspace depends on the number of desired eigenpairs, which means that when looking for e.g. 10 eigenvalues the first 4 may be found in a different amount of time then when looking for 4 eigenvalues. In our implementation of Jacobi-Davidson, the minimal search space size only depends on the number of converged eigenpairs. This choice implies that the time needed to find a certain amount of eigenpairs does not depend on how many are desired.

Other parameters, that do not influence the results in favor of either method, are the tolerance and the starting vector. The tolerance is a constant used in the stopping criterion, which is described in the following section. We have decided to choose $tol = 10^{-4}$ for the linear tests and 10^{-5} for the quadratic tests to ensure adequate convergence. We found that decreasing the tolerance any more doesn't give more accurate eigenvalues. In fact, the difference between eigenvalues found with 'low' tolerance and 'high' tolerance is in the relative order of 0.1 %.

As starting vector we could not choose the constant vector because this is an eigenvector with eigenvalue 0. For the application, we are not interested in this eigenvalue, which means that choosing the eigenvector as starting vector could cause problems. We decided to take the alternating vector $(1, 0, 1, 0, \dots)$ which is never an eigenvector. A random vector would only be possible if we could keep the random seeds equal for Arnoldi and Jacobi-Davidson. This seemed unnecessarily complicated. Moreover, a known starting vector makes it easier to compare different tests and to solve possible programming mistakes.

7.2.1 Stopping criterion

Of course, the stopping criterion should be equal between the two methods to make a fair comparison. This cannot be done straightforward because the problem is linearized when using Arnoldi making the residual in Arnoldi's method twice the size of the residual in Jacobi-Davidson. To make sure both methods give equally precise answers we will rewrite the equation to calculate the norm of the residual for Jacobi-Davidson:

$$\begin{aligned}\|r_{JD}\|_2 &= \left\|Ap_{JD} + \omega Bp_{JD} + \omega^2 Cp_{JD}\right\|_2 \\ &= \left\|\begin{pmatrix} -Ap_{JD} - \omega Bp_{JD} - \omega^2 Cp_{JD} \\ 0 \end{pmatrix}\right\|_2 \\ &= \left\|\begin{pmatrix} -B & -A \\ I & 0 \end{pmatrix}\begin{pmatrix} \omega p_{JD} \\ p_{JD} \end{pmatrix} - \omega \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} \omega p_{JD} \\ p_{JD} \end{pmatrix}\right\|_2\end{aligned}$$

This can be easily compared to the Arnoldi residual, which is given by:

$$\|r_{AR}\|_2 = \left\|\begin{pmatrix} -B & -A \\ I & 0 \end{pmatrix}\begin{pmatrix} \omega p_{AR} \\ p_{AR} \end{pmatrix} - \omega \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} \omega p_{AR} \\ p_{AR} \end{pmatrix}\right\|_2$$

The only thing left to do is to scale the equation for the Jacobi-Davidson residual so that the vector $(\omega p_{JD},\ p_{JD})'$ has norm 1. Since we know that $\|p_{JD}\|_2 = 1$ this can simply be done by multiplying the equation by $(1 + |w|^2)^{-0.5}$. This means that $(1 + |w|^2)^{-0.5}\|r_{JD}\|_2 \approx \|r_{AR}\|_2$.

For the stopping criterion used for Arnoldi, we stick to the criterion proposed in ARPACK: $||r_{AR}|| < tol \cdot |\omega|$, where ω is the current approximation of λ. For Jacobi-Davidson this becomes $||r_{JD}|| < tol \cdot |\omega| \cdot \sqrt{1 + |\omega|^2}$. Also, since we are not interested in the eigenvalue zero or in eigenvalues with negative real part we choose to ignore any eigenvalue that falls under this category. Note that the condition for Jacobi-Davidson is almost the same as Arnoldi's for linear eigenproblems. This can be seen by writing the linear problem $Ax + \mu x = 0$ as $-Ax = \lambda^2 x,\ \lambda^2 = \mu$. Arnoldi (for linear problems) uses the stopping criterion $||r_{AR}|| < tol \cdot |\omega_\mu|$ where ω_μ is the approximation of μ, while Jacobi-Davidson (for quadratic problems) uses $||r_{JD}|| < tol \cdot |\omega_\lambda| \cdot \sqrt{1 + |\omega_\lambda|^2}$ where ω_λ is the approximation of λ. For eigenvalues (much) larger than 1 it's obvious that $|\omega_\lambda| \cdot \sqrt{1 + |\omega_\lambda|^2} \approx \omega_\lambda^2 \approx \lambda^2 \approx \omega_\mu$.

7.2.2 Maximal search space size

Both Arnoldi and Jacobi-Davidson construct search spaces in which to look for eigenvectors. For very large and possibly ill-conditioned problems it will be difficult to find the eigenvectors. To prevent constructing search spaces that are too memory-demanding, it is necessary to implement restarts. A restart takes place when the search space has reached a given size, after which a small search space is taken as a new start for the search. This means that we have to think about the minimal and maximal size of the search space. Typically, the minimal size is based on the number of converged eigenvectors or on the desired number of eigenvalues. A less obvious parameter is the maximal size. When we take this number relatively large, for example larger than 100, we need less iterations to converge. However, this requires a larger amount of memory, and more time per iteration. From preliminary tests it appeared that the optimal value for this parameter is very problem dependent, and is restricted by memory issues. In general, the best performance in terms of time will be obtained by increasing this parameter when the problem size increases. In the tests we tried 5 different values for both methods to make sure the comparison was made based on optimal parameters.

For the three-dimensional testcase with 8000 nodes we have plotted the behavior of Arnoldi and Jacobi-Davidson when the maximal search space is changed. The performance of Arnoldi is shown using the red, dashed lines and Jacobi-Davidson with the continuous blue line.

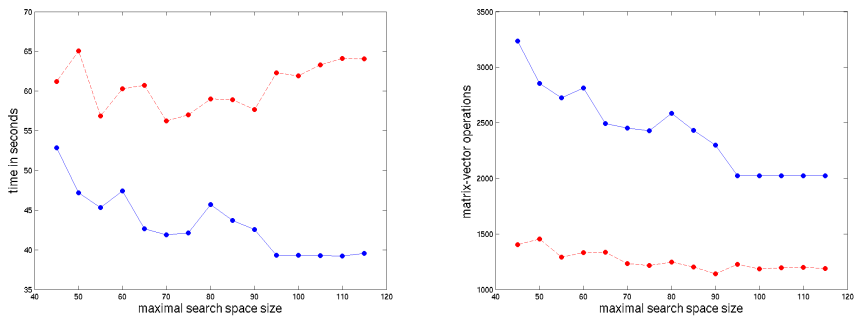

Figure 7.2: The grid with 8000 points

It seems that these test results indicate that with Arnoldi the expected trade-off appears, but with Jacobi-Davidson it is simply a matter of choosing the maximal search space size to be bigger than the (expected) number of iterations, causing the algorithm to find the answer without restarting. We suspect that this is not true for much larger problems, but we did not have the opportunity to test this hypothesis. Also, some work might have to be done to improve the current method of restarting which now takes the converged eigenvectors and the last Ritz vector as a base for the search space.

From the graphs depicted above we can also conclude that Arnoldi's method takes much more time per matrix-vector operation than Jacobi-Davidson. This can be explained from the observation that Arnoldi orthogonalizes a growing set of vectors every time it does a matrix-vector operation. This causes Arnoldi's method to use $(n_{max} - n_{min})n_{max}$ vector operations per iteration (an Arnoldi-iteration is the procedure of building an entire search space, not just adding one vector to the base as with Jacobi-Davidson), where n_{max} is the maximal and n_{min} the minimal size of the search space. Every Arnoldi-iteration uses only $n_{max} - n_{min}$ matrix-vector products, leaving us with n_{max} vector operations per matrix-vector product. This explains why increasing the maximal search space size over 100 increases calculation time but not the amount of matrix-vector products. Most of Jacobi-Davidson's work is done in the GMRES algorithm that approximates the solution to the correction equation. In this algorithm, like in Arnoldi's, we need $n_{GMRES} - 1$ vector products per matrix-vector operation. But n_{GMRES} typically has a value below 40 (we explain more about this value in the next section), which means Jacobi-Davidson takes less vector operations per matrix-vector product, especially when n_{max} is chosen large. This difference is only made larger by the fact that for quadratical problems the size of vectors doubles for Arnoldi's method. Which means that every vector product in Arnoldi's method gives two times more work than in Jacobi-Davidson. Even though Jacobi-Davidson also has a growing set of vectors that serves as a base for the search space, and need to be orthogonalized, this amount is small per matrix-vector operations because there are 40 matrix-vector operations needed to get one new vector, effectively adding one or two vector operations per matrix-vector operation.

7.2.3 Maximal GMRES subspace size

For Jacobi-Davidson the maximal GMRES-subspace size is another important parameter. GMRES is used to approximate the answer of the correction equation. The better the approximation, the fewer iterations Jacobi-Davidson will need to find the desired eigenvalues. On the other hand, increasing the maximal GMRES subspace size will require more memory and CPU-time to find the approximation. From various tests we found that this parameter is almost problem independent in our range of test cases, with an optimal value of 30. Sometimes a slight change of this value improves performance a little, but in general we can trust this value to be good enough. This consistency is off course a good property of a numerical method for industrial application, where it is more important that parameters are easy to predict than that they are absolutely optimal. In our comparison with Arnoldi we have used 30 as a standard value for the maximal GMRES subspace size.

7.3 Numerical Results

To make a fair comparison between the Arnoldi method and the Jacobi-Davidson method we have written a Fortran code for Jacobi-Davidson, using standard BLAS and LAPACK routines for the basic numerical operations and a GMRES code provided by CERFACS [3]. At CERFACS, a FORTRAN version of ARPACK is currently used to solve large sparse eigenvalue problems. To compare the two methods, we made two versions of AVSP, one of which used Jacobi-Davidson to solve the eigenvalue problem, the other used ARPACK. We then compared with which method the program found the desired eigenvalues faster.

7.3.1 Academic two-dimensional testcase

For the two-dimensional case, the results for the CPU-time are presented in the following table. Directly after that we give a table with the values of the maximal search space size parameter for each test case.

	$u = 0$		$y = 0.4+0.3$ i		$y = 3+2$ i	
	AR	JD	AR	JD	AR	JD
500	0.12	0.53	0.7	1.3	1.4	1.05
2000	0.78	2.1	7.16	4.92	15.84	4.96
8000	8.05	14.69	60.98	29.82	167.04	33.49
32000	85.5	116.46	760.49	272.4	2675.51	299.03

Table 7.1: CPU-time in seconds for 2-dimensional test cases until 10 eigenvalues have been found, for 4 different gridsizes

	$u = 0$		$y = 0.4+0.3$ i		$y = 3+2$ i	
	AR	JD	AR	JD	AR	JD
500	25	25	40	30	50	35
2000	30	40	60	35	60	55
8000	30	50	60	70	75	95
32000	50	130	60	150	90	150

Table 7.2: Maximal search space size parameter

From the upper table we see that in case of an impedance boundary condition ARPACK's performance decreases drastically, whereas Jacobi-Davidson only needs about twice the amount of time to solve these more difficult problems. Especially for fine grids Jacobi-Davidson is much faster than ARPACK. The performance deterioration with Arnoldi is logical, since then the problem becomes quadratic in which case Arnoldi has to solve a problem twice the size of the original problem. In practice, a different implementation of Arnoldi is needed to deal with this rewriting of the problem. An important result for Jacobi-Davidson is that the value of the impedance has almost no influence on calculation-time, as long as it's constant. The currently implemented method to deal with non-constant impedance is to iterate using a Picard-iteration. This means that the value for the impedance is chosen based on information from the last iteration, but taken constant during the iteration. In other words, we do the same calculation for different values for the impedance. If the calculation time would depend strongly on this value then the total calculation time becomes very unpredictable. The second table shows that it is hard to predict what value should be chosen for the maximal search space size. To keep a balance between speed and memory-efficiency the search space should not be chosen much larger than 120. The value of 150 for the grid with 32000 points is chosen purely to get the solution as fast as possible, but with even larger problems memory restrictions will play a larger role.

Another interesting aspect is the speed with which the different eigenvalues are found. We will give the amount of time needed to find each of the 10 eigenvalues for the 2D testcase with 32000 nodes.

	u = 0		y = 0.4+0.3 i		y = 3+2 i	
	AR	JD	AR	JD	AR	JD
λ_1	55.23	53.64	523.35	57.43	1358.35	55.49
λ_2	55.8	58.27	518.14	74.81	1918.77	83.77
λ_3	57.23	62.12	507.39	95.53	1750.79	103.56
λ_4	57.23	65.09	507.39	113.02	1358.35	120.77
λ_5	62.78	70.19	528.77	135.34	1802.81	150.45
λ_6	83.00	100.62	683.43	161.34	1384.73	174.51
λ_7	76.49	102.05	698.34	204.43	2675.51	225.07
λ_8	75.12	106.24	727.22	229.43	1581.31	252.81
λ_9	85.5	111.96	760.49	253.4	1892.99	276.06
λ_{10}	76.49	116.46	746.36	272.4	2059.02	299.03

Table 7.3: CPU-time needed to find the different eigenvalues

This table shows that Arnoldi finds the eigenvalues in groups. This is because the method only checks for convergence after the maximal size of the search space is reached. It can do so since the expansion of the search space doesn't depend on the residual, as with Jacobi-Davidson. This method of building up the search space independent of one specific residual (which is calculated using only one Ritz vector) causes all Ritz vectors to converge at the same time. This is contrary to Jacobi-Davidson, where only one vector is converging at a time. We see that in the above results because for Jacobi-Davidson the time between finding two eigenvectors is often of the same order. Also, Jacobi-Davidson converges to the smallest eigenvalue first, and then finds them in order. ARPACK often finds the eigenvalues in a different order. Since for many applications only the smallest eigenvalues are important we see this as a nice property of Jacobi-Davidson.

To get more insight in the convergence behavior of Jacobi-Davidson, we have made a graph depicting the convergence, for the 8000-point grid with two different values for the impedance. Notice that only the norm of the residual of the non-converged eigenpair closest to the target is plotted. We plot this norm against the time, because not every iteration takes the same amount of time. These plots show again the slow start.

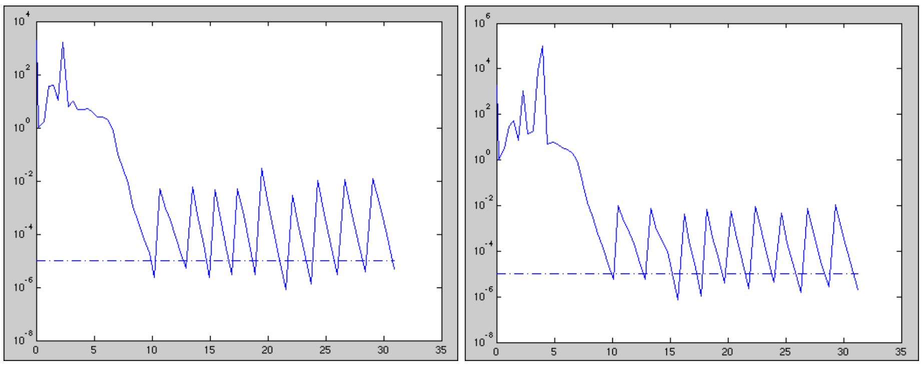

Figure 7.3: Convergence history for Jacobi-Davidson, left has low impedance and right has high impedance

7.3.2 Academic three-dimensional testcase

For the three-dimensional case, the results are as follows:

	$u = 0$		y = 0.4+0.3 i		y = 3+2 i	
	AR	JD	AR	JD	AR	JD
500	0.32	1.10	0.72	1.42	2.02	1.44
2000	1.21	5.35	4.66	6.25	22.42	7.38
8000	6.53	26.27	32.27	32.57	172.36	49.15
32000	44.04	190.65	330.62	160.06	N/A	264.23

Table 7.4: CPU-time in seconds for 3-dimensional test cases

	$u = 0$		y = 0.4+0.3 i		y = 3+2 i	
	AR	JD	AR	JD	AR	JD
500	40	40	50	35	70	35
2000	40	60	60	60	75	50
8000	60	90	80	90	80	90
32000	60	120	80	140	N/A	140

Table 7.5: Maximal search space size parameter

We see in this table the same behavior that we saw for the two-dimensional test cases. When an impedance boundary condition is imposed, the performance of Arnoldi decreases drastically whereas CPU-time needed with Jacobi-Davidson hardly increases. The difficulties that Arnoldi has with large quadratic problems become clear, especially for the case with $y = 3+2i$. Since our computing time per test was limited to 900 seconds, Arnoldi did not finish in the case with 32000 gridpoints and the difficult impedance. For the three-dimensional quadratic problems with approximately 8000 points we also give the convergence graphs. Notice the effect of restarting in the middle of the plots.

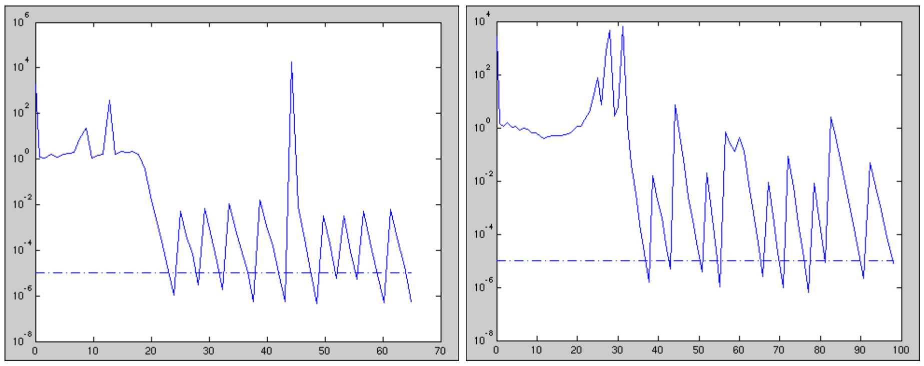

Figure 7.4: Convergence history for Jacobi-Davidson, left has low impedance and right has high impedance

7.3.3 ARRIUS combustion chamber

The results for the academic testcases indicate that Jacobi-Davidson performs much better than Arnoldi for quadratic eigenvalue problems. For industrial application however we will need to show that the same is true for an actual combustion chamber. At CERFACS several grids of combustion chambers are available. We chose a relatively small chamber called ARRIUS with 22000 nodes, to prevent calculation times to be too high. To give the reader an idea of the way the pressure fluctuates according to different eigenmodes we plot the eigenvectors belonging to the eigenvalues with the smallest real part. The total pressure in the combustion chamber is a combination of the constant part of the pressure and the sum of the fluctuating parts that follow from the eigenvalue problem. These plots are shown in appendix A; Here we give the numerical results of the comparison of the two methods. A value of 10^{-6} is used for the tolerance, Arnoldi uses a search space spanned by at most 100 vectors, and Jacobi-Davidson builds a search space using at most 170 vectors.

	$u = 0$		$y = 0.4+0.3$ i		$y = 3+2$ i	
	AR	JD	AR	JD	AR	JD
λ_1	131	131	1158	284	2106	310
λ_2	125	152	1171	507	2106	573
λ_3	125	171	1117	668	3091	755
λ_4	131	195	1171	1095	3913	1406
λ_5	131	223	1211	1266	2302	1556
λ_6	143	246	1251	1446	2251	2145
λ_7	143	298	1290	1725	3141	2544
λ_8	155	391	1290	1990	3888	2706
λ_9	191	435	1675	2952	6059	3271
λ_{10}	197	513	1662	3127	7724	3644
Matvecs	2669	7291	8018	39882	37747	46857

Table 7.6: CPU-time needed to find each eigenvalue of the ARRIUS combustion chamber, and the total number of matrix-vector multiplications

This table shows the time at which each eigenvalue is found by the two methods, and the total number of matrix-vector operations (matvecs) for three different sets of boundary conditions. The results are comparable to the results for the academic test cases, with the exception that when more than 5 eigenvalues are needed, Arnoldi now performs better for the case where the impedance is relatively small. For the case with large impedance Jacobi-Davidson is faster. Comparing the number of matvecs, we see that Jacobi-Davidson still needs a lot more than Arnoldi, but for Arnoldi this amount increases fast for larger impedances.

Chapter 8

Conclusions

Arnoldi's method is known as a powerful method to solve eigenvalue problems, and this report confirms that. The ARPACK package is currently a state-of-the-art algorithm to find the eigenvalues of large, sparse linear eigenvalue problems. This algorithm can be used to solve quadratic eigenvalue problems as well, albeit in their linearized form, with satisfactory performance. In comparison with Jacobi-Davidson however, the performance is behaving too unpredictable. Some problems are solved reasonably fast, but others take a lot of time. This depends on the shape the spectrum of the linearized problem; Arnoldi is not very well suited to find inner eigenvalues, although combining Arnoldi with shift-and-invert techniques boosts performance for these problems.

In this report, we have compared the recently developed Jacobi-Davidson method for different types of eigenvalue problems with Arnoldi's method. As expected, linear problems are solved faster using ARPACK then when using the JDQZ implementation of Jacobi-Davidson. Jacobi-Davidson however is more easily extended to quadratic or nonlinear eigenvalue problems. This statement is proven by our test, that show that eigenpairs of quadratic eigenvalue problems are found almost as fast as those of linear problems with the same grid, only with different boundary conditions.

For nonlinear problems, we believe Jacobi-Davidson has the potential to become the most important solution method. At the same time, our results reveal that there are still some problems to be taken care of: even though Jacobi-Davidson is faster than Arnoldi, it can fail in finding all desired eigenvalues. This is inherent to the fact that Jacobi-Davidson is a Newton-type method. A possible solution might be to combine Jacobi-Davidson with a good method of finding eigenvalues of the small projected eigenvalue problem. Our tests with the method of successive linear problems failed because of a lack of accuracy of the eigenpairs provided by this method. For implementation in AVSP, we believe that a parallel version of Jacobi-Davidson is necessary to deal with quadratic eigenvalue problems arising from thermo-acoustic problems. It could replace the current method of linearization and using parallel ARPACK.

Chapter 9

Future Research

A lot of questions are still unanswered about Jacobi-Davidson. First of all, we have not been able to develop a working method for the fully nonlinear problem arising from the combustion problems that are dealt with at CERFACS. These problems have a quadratic structure, but with added nonlinear dependence on the eigenvalue because of both the boundary conditions and the flame response. We have worked on a possible solution in MATLAB for the nonlinearity caused by a non-constant impedance in the boundary condition, but the results weren't satisfactory. It must remain a goal of research to find a method, possibly an extension of Jacobi-Davidson, that can solve these fully nonlinear problems without having to use a Picard iteration or linearizing the quadratic structure.

Possible improvements in performance of our current Jacobi-Davidson algorithm are a more sophisticated restart strategy, using a preconditioner when solving the correction equation, and a smarter selection method of the Ritz pairs. This method has to select which pair is best suited to use when searching for the next expanding vector, to prevent switching the target of convergence. Finally, a parallel version of Jacobi-Davidson is being developed at CERFACS, and is expected to be a straightforward adaption of the current serial version.

Appendix A

Plots

We present here plots of the eigenmodes of two cases: the 2D academic case, and the realistic 3D ARRIUS chamber. These eigenmodes indicate how the acoustic pressure fluctuates for each separate (complex) eigenfrequency. The real part of these frequencies are the physical frequencies of the eigenmodes, the imaginary part is linked with the damping of the oscillation. Unfortunately, for the 3D cases, we aren't able to provide the frequencies.

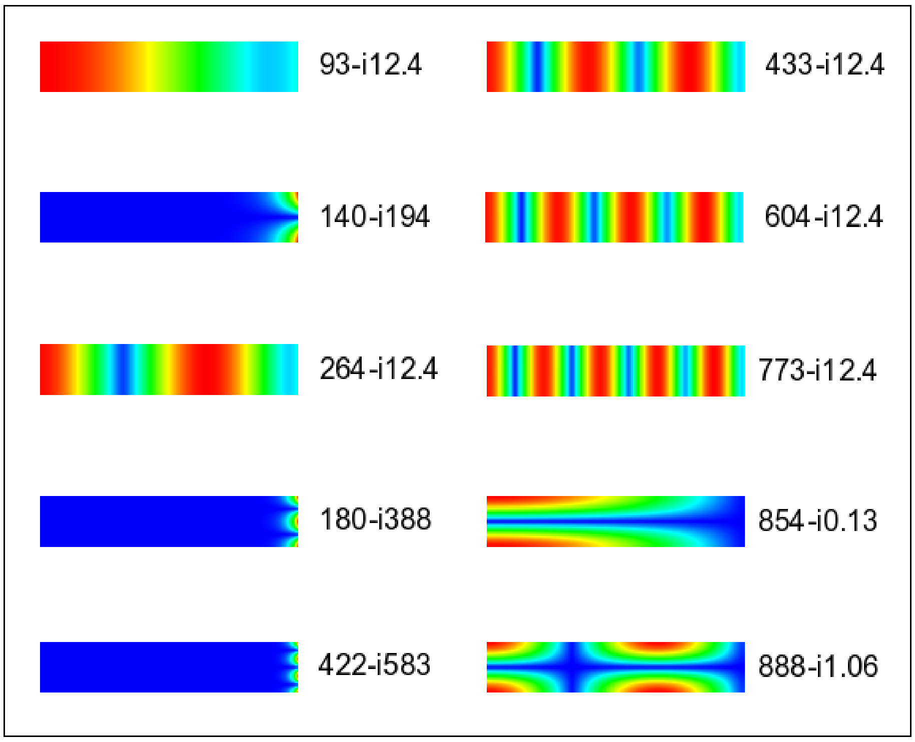

Figure A.1: pressure levels for ten eigenfrequencies

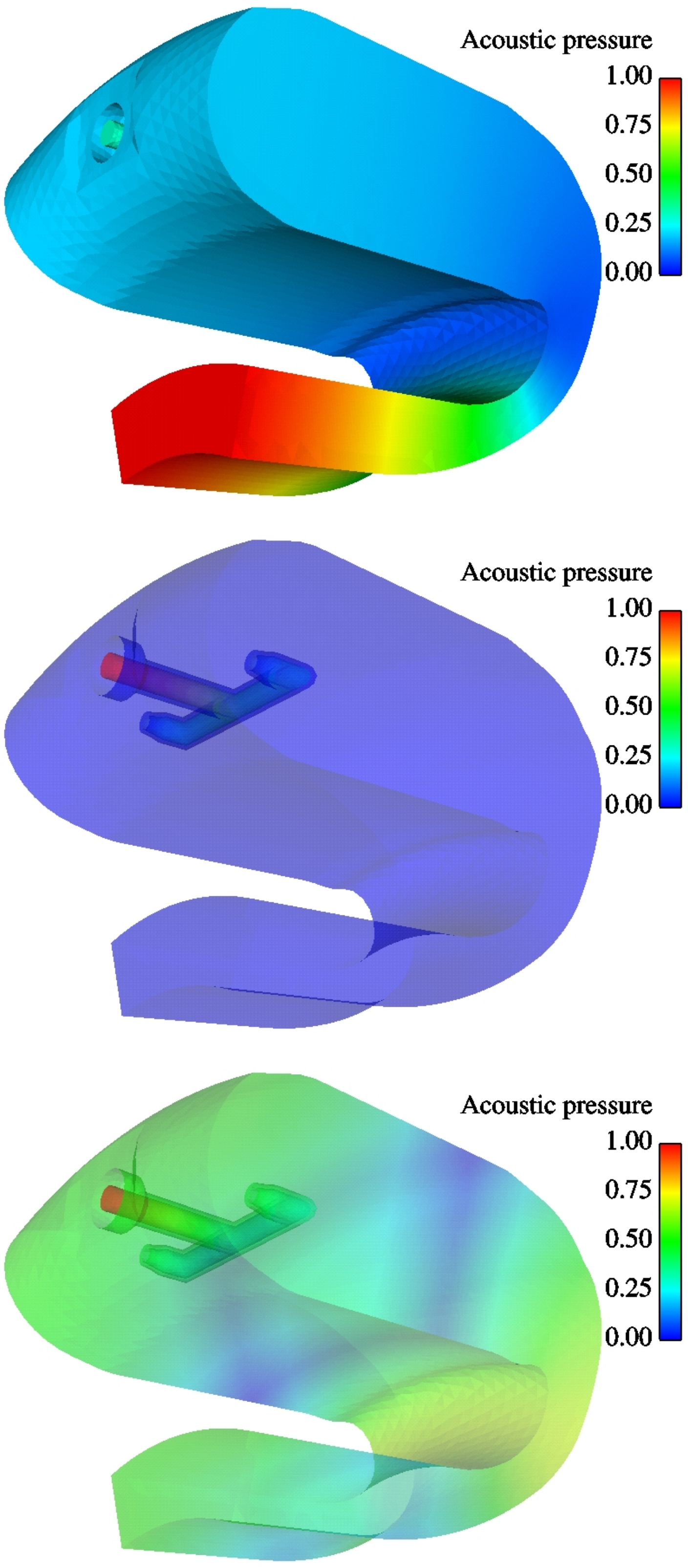

Figure A.2: pressure levels for the three smallest eigenfrequencies

Appendix B

Matlab code

We present the code that we used as Jacobi-Davidson algorithm, and as a blueprint for the Fortran code presented in the next chapter. The code consists of the functions `qjd`, `select` and `GMRES`.

```
function [eival, eivec, converged, relres] = qjd( K, C, M, num, target, tol, n_gmres, rsta
%
% Dimension of the problem:
n = length(K);

% Search spaces:
V = zeros(n,rstart);
KV = zeros(n,rstart);
MV = zeros(n,rstart);
CV = zeros(n,rstart);

% Projected matrices:
H_K = zeros(rstart,rstart);
H_C = zeros(rstart,rstart);
H_M = zeros(rstart,rstart);

% Space to solve small augmented eigenproblem
H1 = zeros(2*rstart,2*rstart);
H2 = zeros(2*rstart,2*rstart);
z = zeros(2*rstart,2*rstart);
alpha = zeros(2*rstart);
theta = zeros(num+1);

% Initialization
V(:,1) = rand(n,1);
V(:,1) = ones(n,1);
V(:,1) = V(:,1)/norm(V(:,1));

iter = 0;
converged = 0;
ritzval = target;

while ( iter <= maxit & converged < num )
   for ( inner =  converged+1:rstart )
      iter = iter + 1;
%
```

```
% Multiply new basis vector with matrices
      KV(:,inner) = K*V(:,inner);
      CV(:,inner) = C*V(:,inner);
      MV(:,inner) = M*V(:,inner);
%
% Calculate new row and column of projected matrix:
      for i = 1:inner-1
         H_K(i,inner) = dot( V(:,i), KV(:,inner) );
         H_K(inner,i) = dot( V(:,inner), KV(:,i) );
         H_C(i,inner) = dot( V(:,i), CV(:,inner) );
         H_C(inner,i) = dot( V(:,inner), CV(:,i) );
         H_M(i,inner) = dot( V(:,i), MV(:,inner) );
         H_M(inner,i) = dot( V(:,inner), MV(:,i) );
      end
      H_K(inner,inner) = dot( V(:,inner), KV(:,inner) );
      H_C(inner,inner) = dot( V(:,inner), CV(:,inner) );
      H_M(inner,inner) = dot( V(:,inner), MV(:,inner) );
%
% Projected augmented system:
      O = zeros(inner);
      I = eye(inner);
      H1 = [-H_C(1:inner,1:inner) -H_K(1:inner,1:inner); I O];
      H2 = [H_M(1:inner,1:inner) O; O I];

%
% Calculate eigenvalues/vectors of small augmented system
      [z alpha] = eig(H1,H2);
      alpha = diag(alpha);
%
% Find the converged and the first nonconverged ritz values and small vectors
      [s theta] = select( z, alpha, converged, target );
      ritzval = theta(converged+1);
%
% New Ritzvector:
      t = s(:,converged+1)/norm(s(:,converged+1));
      ritzvec = V(:,1:inner)*t;
%
% New residual:
      r = (ritzval^2*MV(:,1:inner) + ritzval*CV(:,1:inner) + KV(:,1:inner))*t;
      rn = norm(r);
      relres(iter) = rn;
       if ( rn < tol*abs(ritzval) )
%      if ( rn < tol )
%
% Found new converged eigenvalue
         converged = converged + 1;
         eival(converged) = ritzval;
         eivec(:,converged) = ritzvec;
         if ( converged == num ) break; end;
% Find the new target ritz vector
         [s theta] = select( z, alpha, converged, target );
         ritzval = theta(converged+1);
%
% New Ritzvector:
         t = s(:,converged+1)/norm(s(:,converged+1));
         ritzvec = V(:,1:inner)*t;
%
% New residual:
         r = (ritzval^2*MV(:,1:inner) + ritzval*CV(:,1:inner) + KV(:,1:inner))*t;
      end
      if ( inner == rstart )
%
% Throw away all vectors except converged ones + restart vector:
         V(:,1:converged+1) = V*s;
%
% Orthogonalize the remaining basis vectors
         for j = 1:converged+1
            for i = 1:j-1
               beta = dot( V(:,i), V(:,j) );
               V(:,j) = V(:,j) - beta*V(:,i);
            end
            V(:,j) = V(:,j)/norm(V(:,j));
         end
%
% Multiply remaining basis vectors (only converged) with matrices
         KV(:,1:converged) = K*V(:,1:converged);
         CV(:,1:converged) = C*V(:,1:converged);
         MV(:,1:converged) = M*V(:,1:converged);
%
% Compute projected matrices:
         H_K(1:converged,1:converged) = V(:,1:converged)'*KV(:,1:converged);
         H_C(1:converged,1:converged) = V(:,1:converged)'*CV(:,1:converged);
         H_M(1:converged,1:converged) = V(:,1:converged)'*MV(:,1:converged);
      else
%
% Compute new basis vector
         if ( n_gmres == 0 )
            V(:,inner+1) = r;
         else
%
% Jacobian
            J = K + ritzval*C + ritzval^2*M;
            w = 2*ritzval*(M*ritzvec) + C*ritzvec;
            V(:,inner+1) = gmres(J, -r, ritzvec, w, n_gmres, droptol );
         end
%
% Orthogonalize the new basis vector
         for j = 1:inner
            beta = dot( V(:,j), V(:,inner+1) );
            V(:,inner+1) = V(:,inner+1) - beta*V(:,j);
         end
         V(:,inner+1) = V(:,inner+1)/norm(V(:,inner+1));
      end
   end
end
```

```
function [s, theta] = select(z, alpha, converged, target )
n = length(alpha);
mask = zeros(n,1);
%  Find the converged and the first unconverged Ritz value
for i = 1:converged+1
   mindist = 1e10;
   for j = i:n
      if ( mask(j) == 0 )
         if ( abs(alpha(j)-target  ) < mindist )
            mindist = abs( alpha(j)-target ) ;
            loc = j;
         end
      end
   end
   s(:,i) = z(n/2+1:n,loc);
   theta(i) = alpha(loc);
   mask(loc) = 1;
end
return
```

```
function [x] = gmres(J, b, u, w, m, droptol );

if ( droptol < 0 )
   precon = 0;
else
   precon = 1;
end

n = length(b);
x = 0*b;

uw = dot( u, w );
v = b;

if ( precon )
   [L U] = luinc(J,droptol );
   v = v - dot( u, v )*u;
   v = U\(L\v);
   v = v - (dot(u,v)/uw)*w;
end

beta = norm(v);
v = v/beta;
V = zeros(n,m);
H = zeros(m+1,m);

for k = 1:m,
   V(:,k) = v;
   v = v - dot( u, v )*u;
   v = J*v;
   v = v - (dot(u,v)/uw)*w;

% Preconditioning
   if ( precon )
      v = v - dot( u, v )*u;
      v = U\(L\v);
      v = v - (dot(u,v)/uw)*w;
   end
%
%  Orthogonalize the new basis vector
   for j = 1:k
      H(j,k) = dot(V(:,j),v);
      v = v - H(j,k) * V(:,j);
   end
   alpha = norm(v);
   H(k+1,k) = alpha;
   v = v/alpha;
end;
%
%  Solve small system
e1 = zeros(k+1,1); e1(1)=beta; y =H\e1;
x = V*y;

return;
```

Appendix C

Fortran Code

We provide the code that we have written as an implementation of Jacobi-Davidson. We have used a routine called `QJD.f` as a portal to choose the correct version of the algorithm. In practice, we have only used one algorithm (`RQJD`, Regular Quadratic Jacobi-Davidson) because we have always taken $C = I$ and a constant (or zero) impedance boundary condition for our tests. For this reason, we only present the routine `RQJD.f`. The other three only differ slightly, and can easily be recreated from the basic algorithm. We have also used several smaller subroutines. Some of those routines were written by us, we present those as well. Other routines, that we don't present, are BLAS or LAPACK routines, available through the net. The subroutines `init_ZGMRES` and `Drive_ZGMRES` are available via the CERFACS website as part of the GMRES package.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C    Subroutine QJD calls any of the four possible
C    Jacobi-Davidson Algorithms, depending on the
C    User input for the variable ProbType.
C
C    This implementation of the Jacobi-Davidson algorithm
C    is designed to solve quadratic eigenvalue problems of
C    the form A*P + z*w*B*P + w^2*C*P = 0
C    where A, B and C are square matrices of equal size,
C    z is the complex impedance, P is the desired eigenvector
C    and w the desired eigenvalue. When C = I or z = constant,
C    the memory requirements can be lowered by adapting the
C    variable Probtype (more on this below)
C
C    Programmers:
C    Jan-willem van Leeuwen
C    Martin van Gijzen
C
C    Version 1.5, finished 26-01-06
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      Subroutine QJD(Eival, Eivec, N, Target, Tol, Neig, MaxSpace,
     +               MaxGMRES, MaxIter, ProbType, Conv)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C    Info about the program:
C    Call the algorithm with the statement:
C    Call QJD(Eival, Eivec, N, Target, Tol, Neig, MaxSpace,
C     +       MaxGMRES, MaxIter, ProbType)
C
C    The Jacobi-Davidson method is implemented in a matrix-free
C    way. To let the method function, the user should supply 4 Subroutines:
C
C    AMUL(N, X, Y) for the multiplication Y = A*X
C    BMUL(N, X, Y) for the multiplication Y = B*X
C    CMUL(N, X, Y) for the multiplication Y = C*X
C    Double complex function impedance(w) for the calculation
C    of the complex non-constant impedance. The matrix B will
C    be multiplied by this value, not divided!
C
C    When C = I, the subroutine CMUL still has to be supplied, but won't
C    be referenced. The same is true for the function for the impedance.
C
C    Info about the Parameters:
C
C    Eival:    Double Complex (Neig) Array.
C              Contains on output an array of Neig eigenvalues.
C    Eivec:    Double Complex (N, Neig) Array.
C              Contains the Neig eigenvectors, with the i-th column
C              corresponding to the eigenvalue Eival(i)
C    N:        Integer. Size of the problem
C    Target:   Double Complex number. The eigenvalues that are returned
C              are closest to target
C    Tol:      Double Precision number. Tolerance for the Residu. Higher
C              Tolerance will increase convergence speed, but decreases
```

```
C              accuracy. Advised value: between 1d-6 and 1d-9.
C    Neig:     Integer. On input: number of eigenvalues wanted with
C              positive complex part. On output: Number of converged
C              Eigenvectors/Eigenvalues with pos. complex part.
C    MaxSpace: Integer. Maximal allowed size for the search space.
C              Advised value: between 20 and 40. For problems
C              where A is non-symmetrical, higher values of Maxspace
C              are more efficient.
C    MaxGMRES: Integer. Maximal allowed GMRES iterations per
C              Jacobi-Davidson iteration
C              Advised value: between 10 and 30
C    MaxIter:  Integer. Maximal allowed number of iterations.
C              Advised value: between 200 and 600
C    Probtype: Logical (2) array. Sets the problem type.
C              Set Probtype(1) = .TRUE. when the mass matrix C
C              is equal to the identity matrix
C              Set Probtype(2) = .TRUE. if impedance is constant
C              make sure to add the impedance in the BMUL subroutine!
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      Implicit none

      Integer N, Neig, MaxSpace, MaxGMRES, MaxIter, MVCounter, Conv
      Double Precision Tol
      Double Complex Target, Eivec(N,Neig), Eival(Neig)
      Logical Probtype(2)

C-----   Check validity of the parameters

      If(N.LE.0.OR.Neig.LE.0.OR.MaxSpace.LE.0.OR.
     +   MaxIter.LE.0.OR.Tol.LE.0) Then
         Print*, 'An important parameter is negative or zero'
         Return
      End if

C-----   Call the correct subroutine

      If (Probtype(1)) Then
         If (Probtype(2)) Then
            Call RQJD(Eival, Eivec, N, Target, Tol, Neig,
     +                Conv, MaxSpace, MaxGMRES, MaxIter)
         Else
            Call IQJD(Eival, Eivec, N, Target, Tol, Neig,
     +                Conv, MaxSpace, MaxGMRES, MaxIter)
         End If
      Else
         If (Probtype(2)) Then
            Call GQJD(Eival, Eivec, N, Target, Tol, Neig,
     +                Conv, MaxSpace, MaxGMRES, MaxIter)
         Else
            Call NLJD(Eival, Eivec, N, Target, Tol, Neig,
     +                Conv, MaxSpace, MaxGMRES, MaxIter)
         End If
      End If

      Print*, Eival

      End
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   Subroutine RQJD supplies the eigenvectors and eigenvalues
C   for problems with C = I and constant impedance
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      Subroutine RQJD(Eival, Eivec, N, Target, Tol, Neig, Conv,
     +               MaxSpace, MaxGMRES, MaxIter)

      Implicit none

C-----   Declare In- and output parameters

      Integer N, Neig, Conv, MaxSpace, MaxGMRES, MaxIter
      Double Precision Tol
      Double Complex Target, Eivec(N,Neig), Eival(Neig)

C-----   Declare program variables

      Integer I, I2, J, J2, Iter, Iseed(2), INFO

      Double Precision Delta, Limit
      Double Precision, allocatable :: Rwork(:)

      Double Complex One, Zero, Tempval(2), Ritzval, Gamma, VL(1,1)
      Double complex, dimension(:), allocatable ::
     +   P, R, W1, Ritzvec, T, Work, Y1
      Double complex, dimension(:,:), allocatable ::
     +   Y, W, H1, AW, WAW, BW, WBW

C-----   Declare Functions

      Double Precision Norm
      Double Complex zdotc
      Real Etime

C-----   Allocate memory

      Allocate (Rwork(8*MaxSpace))
      Allocate (P(N), R(N), W1(N), Ritzvec(N), T(2*MaxSpace),
     +          Work(8*MaxSpace), Y1(MaxSpace))
      Allocate (Y(2*MaxSpace, 2*MaxSpace), W(N, MaxSpace),
     +          H1(2*MaxSpace, 2*MaxSpace),
     +          AW(N, MaxSpace), WAW(MaxSpace, MaxSpace),
     +          BW(N, MaxSpace), WBW(MaxSpace, MaxSpace))

C------   Initialize variables.

      Delta = 1d-3
      Limit = 1d-10
      iseed = (3, 8)
c      call zlarnv( 1, Iseed, N, P)
      Do J = 1,N
         P(J) = cmplx(mod(J,2),0d0)
      End Do
      P = P/norm(N,P)
      I = 0
      One = (1d0,0d0)
      Zero = (0d0, 0d0)
      W(:,1) = p
      Call Zeros(2*MaxSpace, 2*MaxSpace, Y)

      Conv = 0
      Iter = 0

C------   Start the main loop.
      DO WHILE ( Iter .LT. MaxIter )

C------   Start the inner loop.
         DO I = Conv+1, MaxSpace
            Iter = Iter + 1
            I2=2*I

            Call Amul(N, W(1,I), AW(1,I))
            Call Bmul(N, W(1,I), BW(1,I))

            Do J = 1,I-1
               WAW(J,I) = zdotc(N, W(1,J), 1, AW(1,I), 1);
               WAW(I,J) = zdotc(N, W(1,I), 1, AW(1,J), 1);
               WBW(J,I) = zdotc(N, W(1,J), 1, BW(1,I), 1);
               WBW(I,J) = zdotc(N, W(1,I), 1, BW(1,J), 1);
            End Do
            WAW(I,I) = zdotc(N, W(1,I), 1, AW(1,I), 1);
            WBW(I,I) = zdotc(N, W(1,I), 1, BW(1,I), 1);

C------   Construct the small matrix.

            H1(1:I,1:I) = -WBW(1:I,1:I)
            H1(1:I,I+1:I2) = -WAW(1:I,1:I)
            Call Eye(I,H1(I+1:I2,1:I))
            Call Zeros(I,I,H1(I+1:I2,I+1:I2))

C------   Solve the small eigenproblem.

            Call ZGEEV( 'N', 'V', I2, H1, 2*MaxSpace, T, VL, 1, Y,
     +         2*MaxSpace, WORK, 4*MaxSpace, RWORK, INFO )

            If (INFO .NE. 0) Then
               Print*, 'Info=', Info
            Endif

C------   Select the ritzvalue closest to the target.

            Call Select(I2, Y(1:I2,1:I2), T(1:I2), Target,
     +         Conv, Ritzval, Y1(1:I), Neig, Eival)

C------   Determine the ritzvector.

            Call zgemv('N', N, I, ONE, W, N, Y1, 1,
     +                   ZERO, Ritzvec, 1)

C------   Calculate the residu.

            Call zeros( N, 1, R)
            Do J=1,I
```

```
               Do J2=1,N
                  R(J2) = R(J2) + (AW(J2,J) +
     +               Ritzval * BW(J2,J) +
     +               Ritzval**2 * W(J2,J))*Y1(J)
               End Do
            End Do


C------   Test for convergence.

            If (norm(N,R) .LE. Tol*abs(ritzval)**2) Then
               Conv = Conv+1

               Print*, 'Found eigenvalue #', Conv, Ritzval
               Eival(Conv) = Ritzval
               Eivec(:,Conv) = Ritzvec
               If (Conv.EQ.Neig) Then
                  Deallocate (p, R, W1, Ritzvec, T, Work, Y1)
                  Deallocate (Y, W, H1, AW, WAW, BW, WBW)
                  Return
               End If

C------   Find the new target ritzvector

               Call Select(I2, Y(1:I2,1:I2), T(1:I2), Target,
     +            Conv, Ritzval, Y1(1:I), Neig, Eival)

               Call zgemv('N', N, I, ONE, W, N, Y1, 1,
     +                   ZERO, Ritzvec, 1)

C------   Calculate Residu

               Call zeros( N, 1, R)
               Do J=1,I
                  Do J2=1,N
                     R(J2) = R(J2) + (AW(J2,J) +
     +                  Ritzval * BW(J2,J) +
     +                  Ritzval**2 * W(J2,J)) * Y1(J)
                  End Do
               End Do

            End if

C------   Check for restarting.

            If (I.EQ.MaxSpace) Then
               W(:,1:Conv) = Eivec(:,1:Conv)
               W(:,Conv+1) = Ritzvec
               Do J = 1,Conv+1
                  Do J2 = 1,J-1
                     Gamma = zdotc(N, W(1,J2), 1, W(1,J), 1)
                     W(:,J) = W(:,J) - Gamma*W(:,J2)
                  End Do
                  W(:,J) = W(:,J)/norm(N, W(1,J))

                  Call Amul( N, W(1,J), AW(1,J) )
                  Call Bmul( N, W(1,J), BW(1,J) )
               End Do
               Call zgemm('C', 'N', Conv+1, Conv+1, N,
     +            One, W, N, AW, N, Zero, WAW, MaxSpace)
               Call zgemm('C', 'N', Conv+1, Conv+1, N, One,
     +            W, N, BW, N, Zero, WBW, MaxSpace)
            Else

C------   Find the next expanding vector.

               If (MaxGMRES.GT.0) Then
                  call zeros( N, 1, W1 )
                  Do J = 1, I
                     W1 = W1 + (2*ritzval*W(:,J)+BW(:,J))*Y1(J)
                  End do
                  Call GMRES(N, MAXGMRES, -R, Ritzvec, W1,
     +               .TRUE., Ritzval, Ritzval**2, P)
               Else
                  P = R
               End If

C------   Orthogonalize the new vector.

               Do J=1,I
                  P = P - zdotc(N, W(1,J), 1, P, 1)*W(:,J)
               End Do
               W(:,I+1) = P/Norm(N, P, 1)

            End if
         End do
      End do

      Deallocate (p, R, W1, Ritzvec, T, Work, Y1)
      Deallocate (Y, W, H1, AW, WAW, BW, WBW)

      End
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   Subroutine Select selects the ritzpair of which the ritzvalue is
C   the converged+1st closest to target
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      Subroutine Select(I2, Y, T, Target, Converged, Ritzval, Y1,
     +                  Neig, eival)

      Implicit none

      Integer Converged, Idamin, I, I2, J, J1, J2,
     +        K, Neig
      Double Complex Y(I2,I2), Ritzval, Y1(I2/2), Target, T(I2),
     +               eival(Neig)
      Double Precision Norm, T2(I2), T3(I2), Tmax

C----   First move converged eigenvalues away from target

      Do J = 1,I2
         T3(J) = abs(T(J)-Target)
      End Do
      Tmax = T3(1)
      Do J=2,I2
         If (T3(J).GT.Tmax) Then
            Tmax = T3(J)
         End If
      End do

C-----   For our application, we ignore the zero-valued eigenvalues

      Do J1 = 1, I2
         if (real(T(J1)).lt.1d0) then
         T3(J1)=Tmax+1d0
         endif
      end do

C-----   Now ignore the values close to converged ones.

      Do J1 = 1, Converged
         Do J = 1,I2
            T2(J) = abs(T(J)-Eival(J1))
         End Do
         K = Idamin(I2, T2, 1)
         T3(K) = Tmax+2d0
      end do

      K = Idamin(I2, T3, 1)
      Ritzval = T(K)
      I=I2/2
      Y1(1:I) = Y(I+1:I2,K)
      Y1(1:I) = Y1(1:I)/Norm(I, Y1(1:I))

      End
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  Subroutine GMRES finds an approximate solution to the correction equation
C  This Subroutine uses the GMRES driver supplied by the CERFACS algo-team
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      Subroutine GMRES(N, MAXGMRES, B, U1, W1, NOC, C, C1, P)

      Implicit none

C-----    Declare Subroutine Parameters

      Integer N, MAXGMRES
      Double Complex B(N), U1(N), W1(N), C, C1, P(N)

C-----    Declare Calculation Variables

      Integer LWORK, INFO(3), IRC(5), ICNTL(8), J, StartX,
     +   StartZ
      Double Precision CNTL(5), RINFO(2)
      Double Complex X(N), Z(N), Temp(N), UU, UW
      Double Complex, Allocatable :: Work(:)
      Logical NOC

C-----    Declare called functions

      Double Complex zdotc
      Double Precision Norm

C------   Allocate GMRES Workspace

      LWORK = MaxGMRES**2+MaxGMRES*(N+5)+5*N+2
      Allocate (Work(LWORK))

C------   Initialize GMRES Variables

      Call Init_zgmres(ICNTL, CNTL)
      ICNTL(2) = 0
      ICNTL(4) = 0
      ICNTL(6) = 0
      ICNTL(7) = MaxGMRES

      Work(N+1:2*N) = B(:)
      IRC(1) = 1

C------   Determine iteration-specific variables

      UW = zdotc(N, U1, 1, W1, 1)
      UU = zdotc(N, U1, 1, U1, 1)

C------   Start the reverse communication loop

      Do While(IRC(1).NE.0)
         Call Drive_ZGMRES(N, N, MAXGMRES, Lwork, work,
     +      IRC, ICNTL, CNTL, INFO, RINFO)

         StartX = IRC(2)
```

```
   StartZ = IRC(4)

   If(IRC(1).EQ.1)Then
      X = Work(StartX:StartX+N-1)
      X = X - zdotc(N, U1, 1, X, 1)*U1
      Call Zeros(N, 1, Z)
      Call Zeros(N, 1, Temp)
      Call Amul(N, X, Z)
      Call Bmul(N, X, Temp)
      Z = Z + C*Temp
      Call Zeros(N, 1, Temp)
      If(NOC)Then
         Temp = X
      Else
         Call Cmul(N, X, Temp)
      End If
      Z = Z + C1*Temp
      Z = Z - (zdotc(N, U1, 1, Z, 1)/UW)*W1
      Work(StartZ:StartZ+N-1) = Z

   Else If(IRC(1).EQ.2)Then
      Work(StartZ:StartZ+N-1) = Work(StartX:StartX+N-1)

   Else If(IRC(1).EQ.3)Then
      Work(StartZ:StartZ+N-1) = Work(StartX:StartX+N-1)

   Else If(IRC(1).EQ.4)Then
      Do J=0,IRC(5)-1
         Work(StartZ+J) = zdotc(N, Work(StartX+J*N), 1,
+           Work(IRC(3)), 1)
      End Do

   End If

End Do

P = Work(1:N)
If(Norm(N,P).LT.1d-16)Then
   P = B
End If
Deallocate (Work)
Return
End
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   Subroutine Eye provides an N x N unity matrix
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      Subroutine Eye(N, A)

      Implicit none

      Integer N, I
      Double Complex A(N,N)

      Call Zeros(N, N, A)

      Do I=1,N
         A(I,I)=(1d0,0d0)
      End Do

      End
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   Subroutine Zeros provides an N x M zero-matrix
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      Subroutine Zeros(N, M, A)

      Implicit None

      Integer N, M, J1, J2
      Double Complex A(N,M)

      Do J1=1,N
         Do J2=1,M
            A(J1, J2)=(0d0,0d0)
         End Do
      End Do

      End


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   The function Norm provides |X| where X is a vector of dimension N
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      Double Precision Function Norm(N, X)

      Implicit none

      Integer N, J
      Double Complex X(N), zdotc

      Norm = Real(zdotc(N, X(1:N), 1, X(1:N), 1))**5d-1

      End
```

Bibliography

[1] W.E. Arnoldi, *The principle of minimized iteration in the solution of the matrix eigenvalue problem*, Quarterly of Applied Mathematics, volume 9, pages 17-25 (1951).

[2] E.R. Davidson, *The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices*, Journal of Computational Physics, volume 17, pages 87-94 (1975).

[3] F. Frayss, L. Giraud, S. Gratton, and J. Langou, *A set of GMRES routines for real and complex arithmetics on high performance computers*, ACM Trans. Math. Softw. , Volume 31(2), pages 228–238 (2005).

[4] G.H. Golub and C.F. van Loan, **Matrix Computations**, The Johns Hopkins University Press, Baltimore and Londen (1983).

[5] Martin B. van Gijzen, *The parallel computation of the smallest eigenpair of an acoustic problem with damping*, Int. Journal for Numerical Methods in Engineering, Volume 45, pages 765-777 (1999).

[6] C.G.J. Jacobi, *˝Uber ein leichtes Verfahren, die in der Theorie der Skularstrungen vorkommenden Gleichungen numerisch aufzulsen* (1846).

[7] C. Lancszos, *An iteration method for the solution of the eigenvalue problem of the linear differential and integral operators*, Journal on Res. Nat. Bur. Stand., Volume 45, pages 51-94 (1950).

[8] R.B. Lehoucq, D.C. Sorensen and C. Yang, *ARPACK Users' Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, Philadelphia, (1998).

[9] V. Mehrmann and H. Voss, *Nonlinear Eigenvalue Problems: A challenge for Modern Eigenvalue Methods*, Report 83, Arbeitsbereich Mathematik, TU Hamburg-Harburg (2004) GAMM Mitteilungen 27, pages 121-152 (2004).

[10] A. Neumaier, *Residual inverse iteration for the nonlinear eigenvalue problem*, SIAM Journal on Numerical Analysis, Volume 22, pages 914-923 (1985).

[11] F. Nicoud, L. Benoit, C. Sensiau and T. Poinsot, *Acoustic modes in combustors with comlex impedances and multidimensional flames*, AIAA Journal, Volume 45, No. 2, pages 426-441 (2007).

[12] C.C. Paige, B.N. Parlett, and H.A. van der Vorst, *Approximate solutions and eigenvalue bounds from Krylov subspaces*, Numerical Linear Algebra Applications, Volume 2, pages 115-134 (1995).

[13] A. Ruhe, *Algorithms for the nonlinear eigenvalue problem*, SIAM Journal on Numerical Analysis, Volume 10, pages 674-689, 1973

[14] Y. Saad, **Numerical Methods for Large Eigenvalue Problems** (1992), Manchester University Press Series in Algorithms and Architectures for Advanced Scientific Computing.

[15] Y. Saad, Martin H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, Volume 7, pages 856-869. Society for Industrial and Applied Mathematics, Philadelphia (1986).

[16] G.L.G. Sleijpen, J.G.L. Booten, D.R. Fokkema and H.A. van der Vorst, *Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems: Part I*, BIT Numerical Mathematics, Volume 36, Number 3, pages 595-633 (1996).

[17] G.L.G. Sleijpen and H.A. van der Vorst, *A Jacobi-Davidson Iteration Method for Linear Eigenvalue Problems* SIAM Journal on Matrix Analysis and Applications, Volume 17, Number 2, pages 401-425 (1996).

[18] G.L.G. Sleijpen, H.A. van der Vorst and Zhaojun Bai *Jacobi-Davidson algorithms for various eigenproblems -A working document-*, Preprint nr. 1114 Department of Mathematics, University Utrecht, (1999).

[19] H. Voss, *Numerical Methods for Sparse Nonlinear Eigenvalue Problems*, Report 70, Arbeitsbereich Mathematik, TU Hamburg-Harburg (2004) Proc. XVth Summer School on Software and Algorithms of Numerical Mathematics, Hejnice, Czech Republic

[20] H. Voss *An Arnoldi method for nonlinear symmetric eigenvalue problems*, Online Proceedings of the SIAM Conference on Applied Linear Algebra, Williamsburg, http://www.siam.org/meetings/laa03/ (2003).

[21] H. Voss *An Arnoldi method for nonlinear eigenvalue problems* BIT Numerical Mathematics 44, pages 387-401 (2004).