Computing eigenvalues occurring in continuation methods with the Jacobi-Davidson QZ method

Jos L.M. van Dorsselaer*

February 1997

Abstract. Continuation methods are a well-known technique for computing several stationary solutions of problems involving one or more physical parameters. In order to determine whether a stationary solution is stable, and to detect the bifurcation points of the problem, one has to compute the rightmost eigenvalues of a related, generalized eigenvalue problem. The recently developed Jacobi-Davidson QZ method can be very effective for computing several eigenvalues of a given generalized eigenvalue problem. In this paper we will explain how the Jacobi-Davidson QZ method can be used to compute the eigenvalues needed in the application of continuation methods. As an illustration, the two-dimensional Rayleigh-Bénard problem has been studied, with the Rayleigh number as a physical parameter. We investigated the stability of stationary solutions, and several bifurcation points have been detected. The Jacobi-Davidson QZ method turns out to be very efficient for this problem.

1 Introduction

In physical applications one is often interested in stationary solutions of partial differential equations, and how their behaviour depends on (some) physical parameter(s) in the model. For instance, one would like to know whether stationary solutions (if they exist) are stable. At some critical values of the physical parameter, the so-called *bifurcation points*, a stable solution may become unstable and vice versa; further the number of stationary solutions can change at bifurcation points. Clearly, the computation of stationary solutions and bifurcation points is important for analyzing the physical problem under consideration.

In practice, continuation methods [10] are often used to compute stationary solutions for different values of the physical parameter(s). With this approach one may also find unstable stationary solutions; these unstable stationary solutions have no physical relevance, but they might change into stable ones, when the physical parameter passes a bifurcation point (an example of this will be given in Section 4.2). For the investigation of stability, and the determination of bifurcation points, one has to compute some eigenvalues of a certain generalized eigenvalue problem

$$Aq = \lambda Bq; \tag{1.1}$$

a stationary solution is stable if all eigenvalues λ of (1.1) have negative real parts; if at least one of the eigenvalues has a positive real part, the stationary solution is unstable. When one

*Mathematical Institute, Utrecht University, P.O. Box 80.010, NL-3508 TA Utrecht, The Netherlands. This research has been supported by the Netherlands Organization for Scientific Research (N.W.O.).

of the eigenvalues of (1.1) equals zero (i.e. the matrix A is singular), the physical parameter is a bifurcation point, and the number of stationary solutions may change [10].

In one continuation step one has to solve a system of non-linear algebraic equations (in order to obtain a stationary solution) and to compute some eigenvalues (the rightmost ones and those closest to zero) of (1.1). The determination of these eigenvalues is the most expensive part of the computation, both in CPU-time and memory requirements. In this paper we will focus on the computation of these eigenvalues.

For small problems (1.1), one can compute all eigenvalues with the QZ method (see e.g. [6]). However, this is not feasible for larger problems (1.1), e.g. those obtained from partial differential equations in 2- or 3D (the size of the matrices A and B is equal to the number of unknowns obtained after discretizing the partial differential equations). For these problems one should use other methods; a well-known technique is the power method (see e.g. [6]), and block versions of this method (like SIT, see [13, 3]) have been used to obtain more eigenvalues of a problem equivalent to (1.1) (see, e.g. [3]). However, these methods can be very slow in practice. The fact that (1.1) is a generalized eigenvalue problem (i.e. B is not the identity matrix I; in fact, B is often singular) may cause some extra complications when applying these methods. In the last decade some promising eigenvalue methods have been developed (see [7] for an overview of and references to such methods). One of these methods is the so-called Jacobi-Davidson QZ method developed by Fokkema, Sleijpen and Van der Vorst [5]. The main purpose of this paper is to show that this Jacobi-Davidson QZ method can be very efficient for computing the rightmost eigenvalues of (1.1), and those closest to zero.

As an application of the Jacobi-Davidson QZ method, we consider the two-dimensional Rayleigh-Bénard problem, with the Rayleigh number as a physical parameter. The bifurcation behaviour of this problem has been studied extensively in [3]; in this paper the SIT method has been used to compute some of the rightmost eigenvalues to (1.1). Our experiments show that the Jacobi-Davidson QZ method is more efficient than the SIT method for this example. The successful application of the Jacobi-Davidson QZ method to the 2D Rayleigh-Bénard problem suggests that this method might be suitable for investigating the bifurcation behaviour of 3D flow problems in the near future.

This paper is organized as follows. In Section 2.1 we describe how stationary solutions can be found by using continuation methods. The relation between stability of stationary solutions and the eigenvalue problem (1.1), as well as the concept of bifurcation points, will be discussed in Section 2.2. Section 3 deals with the Jacobi-Davidson QZ method and its application to continuation methods. The Jacobi-Davidson method, an essential ingredient of the Jacobi-Davidson QZ method, will be described in Section 3.1, and the Jacobi-Davidson QZ method is presented in Section 3.2. The application of the Jacobi-Davidson QZ method in combination with continuation methods will be discussed in Section 3.3. In Section 4 we present an illustration of the Jacobi-Davidson QZ method. The two-dimensional Rayleigh-Bénard problem, and its discretization will be described in Section 4.1. Our numerical experiments are given in Section 4.2, and the main conclusions of the paper are summarized in Section 4.3.

2 Continuation methods and the related eigenvalue problem

2.1 Computing stationary solutions with continuation methods

Consider the systems of differential equations

$$BY'(t) = F(Y(t), \mu) \quad \text{for} \quad t \geq 0, \tag{2.1}$$

with $B \in \mathbb{R}^{n,n}$, $\mu \in \mathbb{R}$, $Y(t) \in \mathbb{R}^n$ for $t \geq 0$, and $F : \mathbb{R}^{n+1} \to \mathbb{R}^n$ is a smooth function. For $y \in \mathbb{R}^n$ the $n \times n$ matrix

$$F'(y, \mu)$$

stands for the Jacobian matrix of the function F with respect to y, and the vector $F_\mu(y, \mu) \in \mathbb{R}^n$ contains the partial derivatives $(\partial F_j / \partial \mu)(y, \mu)$.

In (2.1), μ stands for a 'physical' parameter. Although the solutions to (2.1) depend on μ, we will not express this dependence in order to keep the notation transparent. Also (systems of) time-dependent partial differential equations lead to (2.1), after discretizing the spatial derivative(s) with e.g. a finite difference or a finite element method.

In many applications one is interested in finding stationary solutions to (2.1), for different values of μ. Let $y = y(\mu)$ be such a stationary solution, i.e.

$$F(y(\mu), \mu) = 0. \tag{2.2}$$

Continuation methods [10] are often used for computing stationary solutions. A typical example of a continuation method is given in Algorithm 1. Suppose a stationary solution $y_0 = y(\mu_0)$ and $\Delta\mu \neq 0$ are given.

$\mu_1 = \mu_0 + \Delta\mu$
$\tilde{y}_1 = y_0 + \Delta\mu \cdot y'(\mu_0)$
`solve` $F(y_1, \mu_1) = 0$ `with Newton's method; use` $\tilde{y}_1$ `as starting vector`
`for` $j = 2, 3, \ldots, M$ `do`
  $\mu_j = \mu_{j-1} + \Delta\mu$
  $\widetilde{y}_j = 2y_{j-1} - y_{j-2}$
  `solve` $F(y_j, \mu_j) = 0$ `with Newton's method; use` $\widetilde{y}_j$ `as starting vector`
`end for`

Algorithm 1. *A continuation method.*

Algorithm 1 generates M stationary solutions $y_j = y(\mu_j)$. The vector $y'(\mu_0)$ is the solution to $F'(y_0, \mu_0)y'(\mu_0) = -F_\mu(y_0, \mu_0)$; this follows from differentiating (2.2) with respect to μ. Instead of $\widetilde{y}_j$ one might use $y_{j-1} + \Delta\mu \cdot y'(\mu_{j-1})$ as a starting vector for computing y_j; both are $\mathcal{O}(\Delta\mu^2)$ approximations to y_j. The determination of $\widetilde{y}_j$ requires less computational costs (no linear system has to be solved), and therefore $\widetilde{y}_j$ is used as a starting vector in Algorithm 1.

In practice one often uses (pseudo-)arclength continuation (cf., e.g., [8, 10]). An advantage of this technique is that it can handle turning points [10], while Algorithm 1 cannot. A major drawback of arclength continuation is that two linear systems of order n have to be solved in each Newton step (see [3]) — instead of one in Algorithm 1.

More discussion about continuation methods can be found in [10, Chapter 4].

The linear systems which occur in Algorithm 1 have the matrix $F'(y,\mu)$ as a coefficient matrix. When (2.1) is obtained from a set of partial differential equations, this matrix is usually large and sparse. For some of these problems it is possible to solve the linear systems with a direct solver, using important properties of the Jacobian matrix (like sparsity, small bandwidth etc.). But, for large general problems one has to solve these linear systems iteratively, using e.g. a Krylov subspace method (such as GMRES [9] or BiCGstab(ℓ) [12]) in combination with a suitable preconditioner. Finally we note that there exist methods for solving nonlinear equations which combine the ideas of Newton iteration with Krylov subspace techniques (see, e.g., [2, 4]). These methods may be more efficient for solving $F(y_j,\mu_j) = 0$ in Algorithm 1 than Newton's method.

2.2 Stability and bifurcation points

We call a stationary solution y to (2.1) *stable* if $\lim_{t\to\infty} Y(t) = y$ for all $Y(0)$ close to y. In order to investigate whether y is stable, equation (2.1) is linearized around y (note that $F(y,\mu) = 0$):

$$BY'(t) = F'(y,\mu)\cdot(Y(t) - y).$$

If all eigenvalues of the problem

$$F'(y,\mu)q = \lambda Bq \tag{2.3}$$

have strictly negative real parts, then, when certain technical conditions are satisfied (cf., e.g., [10, p. 20]), the equilibrium y is stable. When at least one eigenvalue of (2.3) has a positive real part, the equilibrium y is unstable [10, p. 20]. Hence investigating the stability of equilibria amounts to computing the eigenvalues (with largest real part) of (2.3).

The number μ is called a *bifurcation point* if the matrix $F'(y,\mu)$, with $y = y(\mu)$ a stationary solution, is singular (cf., e.g., [10]). This matrix is singular if and only if 0 is an eigenvalue of (2.3). The number of stationary solutions to (2.1) may change at a bifurcation point.

In order to investigate the stability of stationary solutions, and the computation of bifurcation points, we have to determine in each continuation step the rightmost eigenvalues to (2.3), and those closest to 0. The Jacobi-Davidson QZ method, to be described in the next section, is well suited for this.

3 The Jacobi-Davidson QZ (JDQZ) method and its application in continuation methods

The JDQZ method has been developed recently by Fokkema, Sleijpen and Van der Vorst [5]. In this section, the JDQZ method will be described briefly; for more details and discussion, see [5].

With the JDQZ method one can compute several eigenvalues (and eigenvectors) of the generalized eigenvalue problem

$$\beta Aq = \alpha Bq; \tag{3.1}$$

here A, B are $n \times n$ matrices with complex entries, and α, $\beta \in \mathbb{C}$. The pair $\langle \alpha, \beta \rangle$ is called an eigenvalue, and q is the corresponding eigenvector. We write the eigenvalue problem in the form (3.1), instead of (1.1) (note that $\lambda = \alpha/\beta$), because $\beta = 0$ is possible; when B is singular, $\beta = 0$ for at least one eigenvalue. In the Rayleigh-Bénard problem (see Section 4), and in many other applications from fluid dynamics, the matrix B is indeed singular.

In Section 3.1 we describe the Jacobi-Davidson method (see [11]), a method to compute one eigenvalue of (3.1); this method is an essential ingredient of the JDQZ method. The topic of Section 3.2 is the JDQZ method. The application of the JDQZ method in combination with continuation methods will be discussed in Section 3.3.

3.1 The Jacobi-Davidson (JD) method

With the JD method [11] one tries to compute an approximation $\langle \tilde{\alpha}, \tilde{\beta} \rangle \approx \langle \alpha, \beta \rangle$, close to a specified target τ (i.e. $\tilde{\alpha}/\tilde{\beta}$ should be close to τ), and an approximate eigenvector $\tilde{q} \approx q$. In each step a search subspace span$\{V\}$ containing the vector $\tilde{q}$ and a test subspace span$\{W\}$ are constructed; V and W are complex $n \times j$ matrices with $j \ll n$ and $V^*V = W^*W = I$. The vector $\tilde{q}$ and $\langle \tilde{\alpha}, \tilde{\beta} \rangle$ are obtained from the projected eigenvalue problem

$$\tilde{\beta} W^* A V u = \tilde{\alpha} W^* B V u. \tag{3.2}$$

The eigenpair $(\langle \tilde{\alpha}, \tilde{\beta} \rangle, u)$ closest to the target τ is selected (i.e. $\tilde{\alpha}/\tilde{\beta}$ should be as close as possible to τ); the vector $\tilde{q} = Vu$ is an approximation to the eigenvector q. Throughout this paper, the approximate eigenvalue $\langle \tilde{\alpha}, \tilde{\beta} \rangle$ is scaled such that $|\tilde{\alpha}|^2 + |\tilde{\beta}|^2 = 1$.

In order to improve the approximations, the spaces span$\{V\}$ and span$\{W\}$ will be expanded in the next step; compute the residual $r = \tilde{\beta} A\tilde{q} - \tilde{\alpha} B \tilde{q}$ and the vector $\tilde{z} = \kappa_0 A \tilde{q} + \kappa_1 B \tilde{q}$ with

$$\kappa_0 = \left(1 + |\tau|^2\right)^{-1/2} \qquad \text{and} \qquad \kappa_1 = -\tau \left(1 + |\tau|^2\right)^{-1/2},$$

and scale the vectors $\tilde{q}$ and $\tilde{z}$ such that $\|\tilde{q}\|_2 = \|\tilde{z}\|_2 = 1$ ($\|\cdot\|_2$ stands for the Euclidean norm). Note that (3.2) is an eigenvalue problem of small size, so one can use e.g. the QZ method (see, e.g., [6]) to compute all eigenvalues and eigenvectors to (3.2).

The space span$\{V\}$ is expanded with the vector v which is orthogonal to $\tilde{q}$ and satisfies

$$(I - \tilde{z}\tilde{z}^*)(\tilde{\beta} A - \tilde{\alpha} B)(I - \tilde{q}\tilde{q}^*) v = -r, \tag{3.3}$$

and span$\{W\}$ is expanded with the vector $w = \kappa_0 A v + \kappa_1 B v$. The vector v is orthogonalized with respect to the columns of the $n \times j$ matrix V, and then added to the matrix V. In a similar way, the matrix W is enlarged with the vector w. This procedure is repeated until $\|r\|_2$ is small enough.

When the space span$\{V\}$ becomes too large, it is possible to restart the JD method; one might, e.g., replace span$\{V\}$ by the vector $\tilde{q}$ and span$\{W\}$ by $\tilde{z}$, and repeat the procedure described above. A more efficient restarting procedure will be discussed in Section 3.2.

The JD method converges quadratically, if (3.3) is solved exactly. Solving (3.3) is not trivial, because of the different projections involved; see Section 3.2 for details.

Other choices for κ_0, κ_1, and the projections in front of and after $\tilde{\beta} A - \tilde{\alpha} B$ in (3.3) can be found in [5, 11]; experiments indicate that the choices described above are adequate (cf., e.g., [5, 11]).

3.2 The JDQZ method

The purpose of the JDQZ method is to determine a *partial generalized Schur form*

$$AQ_k = Z_k S_k, \qquad BQ_k = Z_k T_k; \tag{3.4}$$

here Q_k and Z_k are $n \times k$ matrices with $Q_k^* Q_k = Z_k^* Z_k = I$, and S_k and T_k are $k \times k$ upper triangular matrices. From (3.4) one easily obtains k eigenvalues of (3.1) (and, optionally, the corresponding eigenvectors): note that $\beta S_k x = \alpha T_k x$ ($x \in \mathbb{C}^k$) implies $\beta A\, Q_k x = \alpha B\, Q_k x$. The columns of the matrix Q_k are called *generalized Schur vectors*.

The first column of Q_k ('the first Schur vector') is an eigenvector of (3.1), and we use the JD method to compute this Schur vector. Suppose a partial Schur form

$$AQ_{k-1} = Z_{k-1} S_{k-1}, \quad BQ_{k-1} = Z_{k-1} T_{k-1}$$

is known already. The question is how to compute the next Schur vector q? In [5] it is shown that q satisfies $Q_{k-1}^* q = 0$ and

$$(I - Z_{k-1} Z_{k-1}^*)(\beta A - \alpha B)(I - Q_{k-1} Q_{k-1}^*) q = 0. \tag{3.5}$$

Observe that (3.5) is a generalized eigenvalue problem, and we will use the JD method to compute an eigenvector q (which is a Schur vector) of (3.5).

To apply the JD method we construct $n \times j$ matrices V and W with $V^* V = W^* W = I$, and the extra condition $V^* Q_{k-1} = W^* Z_{k-1} = 0$. Let $\tilde{q} \in \mathrm{span}\{V\}$ and $\langle \tilde{\alpha}, \tilde{\beta} \rangle \approx \langle \alpha, \beta \rangle$ be selected from the projected eigenvalue problem

$$\begin{aligned} &\tilde{\beta} W^* (I - Z_{k-1} Z_{k-1}^*) A (I - Q_{k-1} Q_{k-1}^*) V u = \\ &\tilde{\alpha} W^* (I - Z_{k-1} Z_{k-1}^*) B (I - Q_{k-1} Q_{k-1}^*) V u, \end{aligned} \tag{3.6}$$

which is the same problem as (3.2) (in exact arithmetic), because $V^* Q_{k-1} = W^* Z_{k-1} = 0$. Compute $r = (I - Z_{k-1} Z_{k-1}^*)(\tilde{\beta} A - \tilde{\alpha} B)(I - Q_{k-1} Q_{k-1}^*) \tilde{q}$ and $\tilde{z} = \kappa_0 A \tilde{q} + \kappa_1 B \tilde{q}$, and scale the vectors $\tilde{q}$ and $\tilde{z}$ such that $\|\tilde{q}\|_2 = \|\tilde{z}\|_2 = 1$. The search space $\mathrm{span}\{V\}$ will be expanded with the vector v satisfying

$$\begin{aligned} &Q_{k-1}^* v = 0, \ \tilde{q}^* v = 0, \ \text{and} \\ &(I - \tilde{z}\tilde{z}^*)(I - Z_{k-1} Z_{k-1}^*)(\tilde{\beta} A - \tilde{\alpha} B)(I - Q_{k-1} Q_{k-1}^*)(I - \tilde{q}\tilde{q}^*) v = -r, \end{aligned} \tag{3.7}$$

and $\mathrm{span}\{W\}$ is expanded with $w = \kappa_0 A v + \kappa_1 B v$. The vectors v and w are orthogonalized and added to V and W, respectively. When $\|r\|_2$ is less than a given tolerance, an acceptable approximation for a new Schur vector q has been detected. This vector will be added to the matrix Q_{k-1}: $Q_k = [Q_{k-1}, \tilde{q}]$ and, further, $Z_k = [Z_{k-1}, \tilde{z}]$. The procedure above can be repeated (with Q_{k-1} replaced by Q_k etc.). Before we explain how (3.7) can be solved, we will discuss how to determine a new matrix V after the detection of a Schur vector.

When a Schur vector has been found, we have to restart the JDQZ proces with a different matrix V, because the relation $V^* Q_k = 0$ is violated. One might replace V by a vector v satisfying $v^* Q_k = 0$, but with this choice one might discard information in V regarding the new Schur vectors. Also when the space V becomes too large one would like to restart without losing valuable information. Both kind of restarts can be done efficiently, using the generalized Schur form related to the projected system (3.2) (which is equivalent to (3.6)):

$$W^* A V \mathcal{Q} = \mathcal{Z} \mathcal{S}, \qquad W^* B V \mathcal{Q} = \mathcal{Z} \mathcal{T}; \tag{3.8}$$

here $\mathcal{Q}$ and $\mathcal{Z}$ are $j\times j$ matrices with $\mathcal{Q}^*\mathcal{Q} = \mathcal{Z}^*\mathcal{Z} = I$ and $\mathcal{S}$ and $\mathcal{T}$ are $j\times j$ upper triangular matrices with diagonal elements s_i and t_i, respectively. The generalized Schur form (3.8) will be ordered such that

$$|s_1/t_1 - \tau| \le |s_2/t_2 - \tau| \le \cdots \le |s_j/t_j - \tau| \tag{3.9}$$

(this is possible, cf., e.g., [5]). Note that $\tilde{q} = V\mathcal{Q}_1$ ($\mathcal{Q}_1$ is the first column of $\mathcal{Q}$), and from the orthogonality of $V\mathcal{Q}$ it follows that $\tilde{q}$ is perpendicular to the other columns of $V\mathcal{Q}$. Therefore we restart with $V := V\mathcal{Q}_{2,j}$ ($\mathcal{Q}_{2,j}$ is the matrix consisting of the $2^{\text{nd}}, 3^{\text{rd}}, \ldots, j$th column of $V\mathcal{Q}$); this new matrix V satisfies $V^*V = I$, $V^*Q_k = 0$, and contains as much information of the old V as possible. Further we set $W := W\mathcal{Z}_{2,j}$. In a similar fashion we can restart when the matrix V becomes too large. From (3.9) we may argue that the first columns of $V\mathcal{Q}$ contain more important information about the Schur vectors to be detected than the last columns. One might replace V by $V\mathcal{Q}_{1,j_{\min}}$, and W by $W\mathcal{Z}_{1,j_{\min}}$, where $j_{\min} < j$, and continue the process.

It is not clear how to solve (3.7) in practice, because of the projections involved. Let K be a nonsingular $n\times n$ matrix and denote $\widetilde{Q}_k = [Q_{k-1}, \tilde{q}]$ (the matrix Q_{k-1} expanded by $\tilde{q}$), $\widetilde{Z}_k = [Z_{k-1}, \tilde{z}]$ (Z_{k-1} expanded by $\tilde{z}$), $\widetilde{Y}_k = K^{-1}\widetilde{Z}$ and $\widetilde{H}_k = \widetilde{Q}_k^*\widetilde{Y}_k$. It is shown in [5] that (3.7) is equivalent to

$$\widetilde{Q}_k^* v = 0 \quad\text{and}\quad (I - \widetilde{Y}_k\widetilde{H}_k^{-1}\widetilde{Q}_k^*)K^{-1}(\tilde{\beta}A - \tilde{\alpha}B)(I - \widetilde{Y}_k\widetilde{H}_k^{-1}\widetilde{Q}_k^*)v = -\hat{r}, \tag{3.10}$$

with $\hat{r} = (I - \widetilde{Y}_k\widetilde{H}_k^{-1}\widetilde{Q}_k^*)K^{-1}r$. The projections in front of and after $K^{-1}(\tilde{\beta}A - \tilde{\alpha}B)$ are the same, so that Krylov subspace methods like GMRES [9] or BiCGstab(ℓ) [12] can be used to solve (3.10). The performance of Krylov subspace methods can often be improved by using some kind of preconditioning. The matrix K in (3.10) may be interpreted as a preconditioner, and in [5] it is proposed to take $K \approx A - \tau B$. This preconditioner K can be used for different pairs $\langle\tilde{\alpha}, \tilde{\beta}\rangle$, so it may be useful to put some effort in the computation of an effective preconditioner. In some cases it might be worth wile to determine a complete factorization of the matrix $A - \tau B$, for one fixed value of τ (only one preconditioner is constructed for the whole JDQZ process); the costs of this complete factorization may be amortized in some cases, because several equations of type (3.10) have to be solved in the JDQZ method.

A pseudo-code for the JDQZ method is given in Algorithm 2. In order to apply the JDQZ method, the user has to supply some parameters, viz.

$$\varepsilon, \quad \tau, \quad k_{\max}, \quad j_{\min}, \quad j_{\max}, \tag{3.11}$$

and a starting vector $\tilde{q} \in \mathbb{C}^n$. The parameter ε is a stopping tolerance for the JD iteration (the JD iteration will be stopped when $\|r\|_2 \le \varepsilon$), τ is a target (the JDQZ method is supposed to compute the eigenvalues closest to τ), $k_{\max}$ is the number of eigenvalues one would like to compute, and $j_{\min}$ and $j_{\max}$ are the minimal (after restart) and maximal dimension of the search space span$\{V\}$, respectively. If there is no approximate eigenvector known, one can take a random starting vector $\tilde{q}$. The notation $Q_k = [Q_{k-1}, \tilde{q}]$ means that Q_k is the matrix obtained from expanding Q_{k-1} with the vector $\tilde{q}$. For $k = 1$ the matrices Q_{k-1} and Z_{k-1} are not defined, and we adopt the conventions $I - Q_0Q_0^* = I$, $Q_1 = [Q_0, \tilde{q}] = \tilde{q}$ etc. In case $k_{\max} = 1$, Algorithm 2 reduces to the JD method (see Section 3.1).

The JDQZ method may converge very fast, even for interior eigenvalues and double eigenvalues (see [5]). In Section 3.1 we mentioned that the JD method converges quadratically,

k=1; j=1;
choose a starting vector $\tilde{q}$
$\kappa_0 = \left(1 + |\tau|^2\right)^{-1/2}$ and $\kappa_1 = -\tau\left(1 + |\tau|^2\right)^{-1/2}$
$v = \tilde{q}$ and $w = \kappa_0 A v + \kappa_1 B v$
$V = v/\|v\|_2$ and $W = w/\|w\|_2$
solve the eigenvalue problem $\tilde{\beta} W^* A V u = \tilde{\alpha} W^* B V u \qquad (|\tilde{\alpha}|^2 + |\tilde{\beta}|^2 = 1)$
select $\langle \tilde{\alpha}, \tilde{\beta} \rangle$ and u; $\tilde{q} = V u$
$\tilde{z} = \kappa_0 A \tilde{q} + \kappa_1 B \tilde{q}$
$\tilde{q} = \tilde{q}/\|\tilde{q}\|_2$ and $\tilde{z} = \tilde{z}/\|\tilde{z}\|_2$
$r = (\tilde{\beta} A - \tilde{\alpha} B)\tilde{q}$
while $k \leq k_{\max}$ do
  while $\|r\|_2 > \varepsilon$ do
    if $j = j_{\max}$ then (restart)
      $V = V \mathcal{Q}_{1,j_{\min}}$ and $W = W \mathcal{Z}_{1,j_{\min}}$
      $j = j_{\min}$
    else
      obtain v from solving (3.10) (or (3.7)) approximately
      $w = \kappa_0 A v + \kappa_1 B v$
      expand V with v and W with w; make V and W orthogonal
      $j = j + 1$
    end if
    solve the eigenvalue problem $\tilde{\beta} W^* A V u = \tilde{\alpha} W^* B V u \qquad (|\tilde{\alpha}|^2 + |\tilde{\beta}|^2 = 1)$
    select $\langle \tilde{\alpha}, \tilde{\beta} \rangle$ and u; $\tilde{q} = V u$
    $\tilde{z} = \kappa_0 A \tilde{q} + \kappa_1 B \tilde{q}$
    $\tilde{q} = \tilde{q}/\|\tilde{q}\|_2$ and $\tilde{z} = \tilde{z}/\|\tilde{z}\|_2$
    $r = (I - Z_{k-1} Z_{k-1}^*)(\tilde{\beta} A - \tilde{\alpha} B)(I - Q_{k-1} Q_{k-1}^*)\tilde{q}$
  end while
  $Q_k = [Q_{k-1}, \tilde{q}]$ and $Z_k = [Z_{k-1}, \tilde{z}]$
  $k = k + 1$
  if $k < k_{\max}$
    $V = V \mathcal{Q}_{2,j}$ and $W = W \mathcal{Z}_{2,j}$
    $j = j - 1$
    solve the eigenvalue problem $\tilde{\beta} W^* A V u = \tilde{\alpha} W^* B V u \qquad (|\tilde{\alpha}|^2 + |\tilde{\beta}|^2 = 1)$
    select $\langle \tilde{\alpha}, \tilde{\beta} \rangle$ and u; $\tilde{q} = V u$
    $\tilde{z} = \kappa_0 A \tilde{q} + \kappa_1 B \tilde{q}$
    $\tilde{q} = \tilde{q}/\|\tilde{q}\|_2$ and $\tilde{z} = \tilde{z}/\|\tilde{z}\|_2$
    $r = (I - Z_{k-1} Z_{k-1}^*)(\tilde{\beta} A - \tilde{\alpha} B)(I - Q_{k-1} Q_{k-1}^*)\tilde{q}$
  end if
end while

Algorithm 2. *JDQZ: $k_{\max}$ eigenvalues close to a target τ are computed. ε is a stopping tolerance. $j_{\min}$ and $j_{\max}$ determine the smallest (after restart) and largest dimension of the search subspace* $\mathrm{span}\{V\}$*, respectively.*

if the correction equation (3.3) (or (3.7), (3.10) in the JDQZ setting) is solved accurately. Experiments show (see, e.g., [5]) that it is not necessary to solve (3.10) accurately in the beginning of the JDQZ process for obtaining fast convergence. The correction equation (3.10) can be solved iteratively, which allows the JDQZ method to be applicable to very large eigenvalue problems. An important question is how to choose a good stopping criterion for the iterative solution of (3.10); solving (3.10) accurately may reduce the number of steps in the JDQZ method, but the execution time of a single step may become higher. It is not clear which strategy leads to the best overall performance of the JDQZ method. In [5] it is suggested to solve (3.10) with a Krylov subspace method, and to stop the iterative proces when

$$\|r_i\|_2 < 2^{-j}\|r_0\|_2 \,, \tag{3.12}$$

where r_i is the ith residual of the Krylov subspace method, and j the iteration number of the JDQZ step (i.e. the dimension of span$\{V\}$). This choice leads to an efficient method.

We refer to [5] for more details, discussions, variants, and illustrations of the JDQZ method.

Remark 3.1. For standard eigenvalue problems (i.e. $B = I$ in (3.1)) one can simplify the method described above. In [5], the Jacobi-Davidson QR (JDQR) method is proposed for computing a partial Schur form $AQ_k = Q_k R_k$ (here R_k is a $k \times k$ upper triangular matrix, and Q_k is as above). Roughly speaking, this JDQR method can be obtained from the JDQZ method by replacing W by V, Z_k by Q_k, and $\langle\tilde{\alpha}, \tilde{\beta}\rangle$ by $\tilde{\lambda} = \tilde{\alpha}/\tilde{\beta}$. Hence, in the JDQR method one might save both computation time and memory storage in comparison with the JDQZ method. See [5] for more details.

3.3 Using JDQZ in continuation methods

In continuation methods we have to compute the rightmost eigenvalues to (2.3), and investigate whether an eigenvalue equals zero or not (see Section 2.2). In many physical applications, most of the eigenvalues have negative real parts, and only a few of them (if any) have a non-negative real part. In this paper we will consider this situation, which means that eigenvalues close and equal to zero belong to the rightmost ones. We now discuss how the parameters (3.11) in the JDQZ algorithm should be chosen; we will focus on the target τ (note that A is the Jacobian matrix $F'(y, \mu)$).

Since we are interested in eigenvalues close to 0, one might choose $\tau = 0$. However, it may be safer to have τ in the right half plane, in order to avoid missing eigenvalues with positive real parts. On the other hand, when $|\tau|$ is too large, the JDQZ method may not be able to designate between different eigenvalues close to 0, which may lead to a slower convergence rate (or even no convergence at all). Hence the choice $\tau = 1$ seems reasonable. Moreover, in case $\tau = 1$, the approximated eigenvalues $\langle\tilde{\alpha}, \tilde{\beta}\rangle$ closest to τ correspond to the in modulus largest approximated eigenvalues of the matrix $(A - B)^{-1}(A + B)$ (see [5]), and the dominant eigenvalues of $(A - B)^{-1}(A + B)$ determine whether a stationary solution is stable or not (cf., e.g., [3]). Therefore we suggest to take $\tau = 1$ when applying the JDQZ method in continuation methods.

The choice of $k_{\max}$ depends on the number of bifurcation points one expects to compute; it is advised to take $k_{\max}$ slightly larger than this number. It is possible to change $k_{\max}$ during the continuation process. Standard choices for the other parameters can be found in [5] and Section 4.2.

In general one uses a randomly chosen vector $\tilde{q}$ as a starting vector for the JDQZ method. But, when using JDQZ in combination with continuation, one has already computed Schur vectors in the previous continuation step(s). Using these Schur vectors may lead to faster convergence (cf. the selection of a starting vector for computing stationary solutions in Algorithm 1). One might e.g. take the Schur vector which was computed first in the previous continuation step (this Schur vector is also an eigenvector) as a starting vector $\tilde{q}$. Instead of starting JDQZ with one single vector, one might also start with a search subspace span$\{V\}$, e.g. the space spanned by the Schur vectors from the previous continuation step. Note that it is not necessary to compute eigenvectors (apart from the first one, which is also a Schur vector); in fact, the columns of V have to be orthogonal. (Also extrapolation of Schur vectors from different continuation steps is possible.) Although these kind of starting procedures look attractive, we observed in our experiments that there is not much difference (in CPU-time) between starting with an arbitrary vector $\tilde{q}$ or the first Schur vector of the previous continuation step. Starting with the subspace spanned by the old Schur vectors even leads to higher computation times. See Section 4.2.2 for our experiments and more discussion.

A possible disadvantage of using previously computed Schur vectors is that they may be close to Schur vectors in the current step which do *not* correspond to the rightmost eigenvalues. This may slow down the convergence of the JDQZ method (because 'wrong' Schur vectors are selected first), or, the JDQZ method may converge to undesired eigenvalues (those corresponding to Schur vectors close to the Schur vectors used for starting JDQZ). To illustrate this, consider, e.g., the 2×2 matrices $A = \mathrm{diag}(-1, \mu)$ and $B = I$. The Schur vector $(1, 0)^{\mathrm{T}}$ is a Schur vector for all $\mu \in \mathbb{R}$, but corresponds only to the rightmost eigenvalue for $\mu < -1$, and *not* for $\mu > -1$. However, this phenomenon did not occur in our experiments.

4 An application: Rayleigh-Bénard convection

4.1 The Rayleigh-Bénard problem

In order to illustrate the JDQZ method in combination with continuation methods, we consider the 2D Rayleigh-Bénard problem, which has been studied extensively in the literature (see e.g. [3]). A liquid layer in a two-dimensional rectangular box, with length 10 and height 1, is heated from below. The temperature on the top and bottom of the box is constant, the sidewalls are isolated and all velocities are zero on the boundaries (no-slip condition). The horizontal and vertical velocity are denoted by u and w respectively, p stands for the (scaled) pressure, and the temperature is denoted by T. This leads to the following system of partial differential equations:

$$\mathrm{Pr}^{-1}\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + w\frac{\partial u}{\partial z}\right) = -\frac{\partial p}{\partial x} + \nabla^2 u, \tag{4.1}$$

$$\mathrm{Pr}^{-1}\left(\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + w\frac{\partial w}{\partial z}\right) = -\frac{\partial p}{\partial z} + \nabla^2 w + \mathrm{Ra}\, T, \tag{4.2}$$

$$\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z} = 0, \tag{4.3}$$

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + w\frac{\partial T}{\partial z} = \nabla^2 T, \tag{4.4}$$

with boundary conditions:

$$
\begin{aligned}
&u = w = \partial T/\partial x = 0 && \text{at} \quad x = 0,\,10, \\
&u = w = 0 \ \text{ and } \ T = 1 && \text{at} \quad z = 0, \\
&u = w = T = 0 && \text{at} \quad z = 1.
\end{aligned} \tag{4.5}
$$

Here Pr is the Prandtl number and Ra is the Rayleigh number; in this paper we take Pr = 5.5, and Ra will be our continuation parameter μ [3]. Note that p is not uniquely determined; this leads to, after discretization of the spatial variables, a Jacobian which is always singular. It is clear from Section 2 that this is not attractive for continuation, and therefore we prescribe p at a certain point: we set $p(5, \frac{1}{2}) = 0$.

The equations (4.1) – (4.5) are discretized on a staggered grid with uniform mesh sizes, using finite difference approximations. For the nonlinear terms we use first order upwind, and the other terms are discretized by second order central differences. In the grid cell containing the point $(5, \frac{1}{2})$ we replace the discretization of (4.3) by $p(5, \frac{1}{2}) = 0$. The discretized system obtained in this way can be written in the form (2.1), where $Y(t)$ contains the velocities u, w, the pressure p and the temperature T at certain gridpoints, and $\mu = \mathrm{Ra}$. The dimension n of the system (2.1) equals $4n_x n_z$, where n_x and n_z stand for the number of grid cells in the horizontal and vertical direction, respectively. The matrix B is a diagonal matrix, which is singular due to the boundary conditions (4.5) and the absence of time derivatives in (4.3). The unknowns are numbered per grid cell, and a grid cell is only coupled to six of its neighbours. The grid cells are ordered by column, from bottom to top, beginning with the first column. This leads to a Jacobian matrix of which the bandwidth is equal to $4n_z + 3$; when n_z is small, it is feasible to compute LU-factorizations of this Jacobian $F'(y, \mathrm{Ra})$ (needed for the Newton process in Algorithm 1), and the preconditioner $K = A - \tau B$ for the JDQZ method.

Remark 4.1. In [3] the Rayleigh-Bénard problem was formulated using the temperature T, the streamfunction ψ and the vorticity ω (with $u = \partial\psi/\partial z$, $w = -\partial\psi/\partial x$ and $\omega = \partial w/\partial x - \partial u/\partial z$) as unknown quantities. This leads to a system of three partial differential equations — instead of four — and this approach is more efficient from a numerical point of view. Our reason for using primitive variables is that this approach can easily be extended to 3D, while this is not possible for the streamfunction-vorticity formulation.

4.2 Numerical experiments

In this section we will describe our experiments. In Section 4.2.1 we will give the actual results of our continuation code, and the performance of the JDQZ method (for different choices of parameters) will be discussed in Section 4.2.2.

4.2.1 The continuation code applied to the 2D Rayleigh-Bénard problem

In Table 1 we have listed the first two bifurcation points obtained with our code (for different grid sizes), and we compared these to the corresponding values from [3].

From Table 1 we see that $n_x = 129$ and $n_z = 33$ gives the most accurate results. Due to memory limitations, we were not able to perform experiments with smaller grid sizes. In particular, the memory requirements for the LU-factorization of $A - \tau B$ can be severe; the number of (possible) nonzero entries of L and U equals $\mathcal{O}(n_x n_z^2)$. The experiments with

$n_z = 17$ suggest that is not useful to take $n_x > 129$. We took $n_x = 129$ and $n_z = 33$ in the experiments described in this section.

n_x	n_z	Ra_1	Ra_2
129	17	1698.3	1701.7
257	17	1698.8	1701.8
129	33	1720.0	1724.0
[3]		1731.3	1734.1

Table 1. *Values of the first two bifurcation points* (Ra_1 *and* Ra_2) *for different grid sizes, and the corresponding values obtained in* [3].

The following bifurcation behaviour for the 2D Rayleigh-Bénard problem has been found in [3]. The trivial, motionless, solution $u \equiv w \equiv 0$ and $T \equiv 1 - z$ is a stationary solution for all $\mathrm{Ra} > 0$. For $\mathrm{Ra} < \mathrm{Ra}_1$ it is the only stationary solution, and this stationary solution is also stable for $\mathrm{Ra} < \mathrm{Ra}_1$. At $\mathrm{Ra} = \mathrm{Ra}_1$ a 10-cell solution (this is a nontrivial solution) branches off, which is stable for $\mathrm{Ra} > \mathrm{Ra}_1$, and the motionless stationary solution becomes unstable for $\mathrm{Ra} > \mathrm{Ra}_1$. At $\mathrm{Ra} = \mathrm{Ra}_2$ an unstable 9-cell solution branches off, and this solution splits into a stable and an unstable stationary solution at another bifurcation point. The motionless solution splits again at a third bifurcation point, which results in an 11-cell solution.

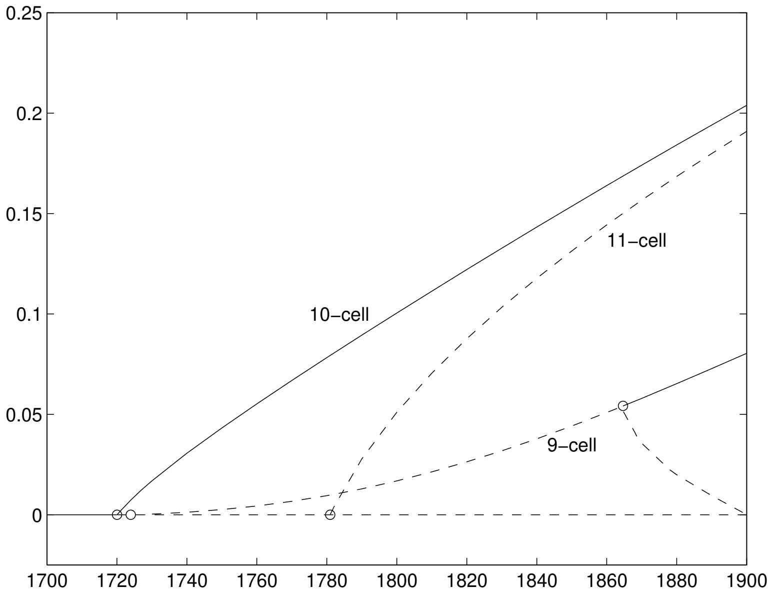

Figure 1. *A bifurcation diagram. The Rayleigh number* Ra *is on the horizontal axis, and the vertical velocity* w *at* $(0.0362, 0.75)$ *(the same point as in [3]) is on the vertical axis. Each curve corresponds to a stationary solution; stable stationary solutions are indicated with a solid line, and unstable ones with a dashed line. The bifurcation points are given by small circles* ($\circ$).

With our code we found the same bifurcation behaviour. The 9-cell solution splits at $\mathrm{Ra} = 1864.6$, and the 11-cell solution branches off at $\mathrm{Ra} = 1781.0$. Our results are visualized in

Figure 1, and three nontrivial stationary solutions are displayed in Figure 2. The pictures in Figure 2 look similar to the corresponding ones in [3]. Hence we may conclude that our code produces meaningful results.

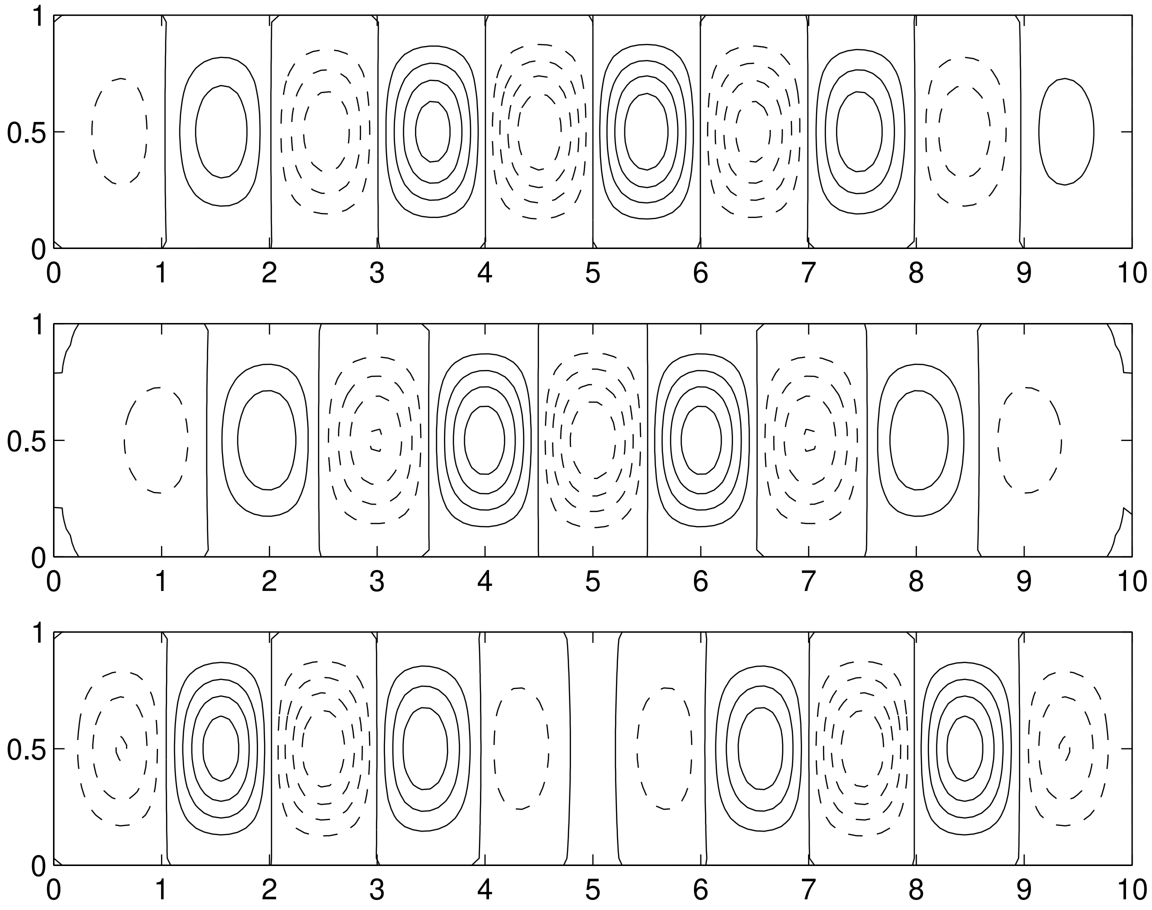

Figure 2. *Three solutions, viz. a 10-cell solution at* Ra = 1721 *(the first picture), a 9-cell solution at* Ra = 1725 *(the second picture), and an 11-cell solution at* Ra = 1782. *In each picture, the current of the fluid is given; a dashed curve means that the fluid moves clockwise, and a solid curve indicates an anticlockwise direction of the fluid.*

The bifurcation points can be computed as follows. At $\mathrm{Ra} = \mathrm{Ra}_1$, the sign of the largest eigenvalue of (2.3) changes (the rightmost eigenvalues of (2.3) turned out to be real in our case), and the secant method has been used to determine at which value Ra the largest eigenvalue of (2.3) equals 0. To compute the second bifurcation point $\mathrm{Ra} = \mathrm{Ra}_2$, we determined, using the secant method, when the second largest eigenvalue equals 0, etc. In order to determine more bifurcation points at an unstable branch, we need to compute several eigenvalues at each continuation step, and not only the rightmost one.

4.2.2 Performance of the JDQZ method

In this section we will consider the performance of the JDQZ method. We will deal with the case $n_x = 129$ and $n_z = 17$, instead of $n_x = 129$ and $n_z = 33$, because of memory requirements and slower computation times in the latter case (in particular wall clock time, due to swapping).

In Algorithm 1 we set $\Delta\mu = \Delta\mathrm{Ra} = 10$, and the Newton iteration is stopped when two consecutive approximations $y_j^{(k)}$ and $y_j^{(k+1)}$ of the stationary solution y_j satisfy $\|y_j^{(k+1)} - y_j^{(k)}\|_\infty \leq 10^{-6}$ ($\|\cdot\|_\infty$ stands for the maximum norm). In Algorithm 2 (the JDQZ method) we set $\tau = 1$ (cf. our discussion in Section 3.3), $j_{\min} = 10$, $j_{\max} = 20$, and we take different values for $k_{\max}$ and ε (see Table 2). In order to solve the correction equation (3.10) we used two different Krylov subspace methods, viz. GMRES$_m$ (at most m steps with full GMRES, *no* restarts) [9] with $m = 5$, and BiCGstab(ℓ) [12] with $\ell = 2$ and a maximum of 100 matrix-vector multiplications for solving (3.10) per JD step. Further, the stopping criterion (3.12) has been used for both methods. It is likely that (3.10) is solved more accurately with BiCGstab(2) than with GMRES$_5$ (more matrix-vector multiplications are allowed for BiCGstab(2)). Therefore one might expect that less JD steps are needed for BiCGstab(2), but, a single JD step is more expensive. A priori it is not clear which Krylov subspace method leads to the most efficient variant of JDQZ. In order to obtain a subspace span$\{V\}$ as soon as possible, GMRES$_1$ is used to solve (3.10) when $j \leq j_{\min}$ (j is the dimension of span$\{V\}$).

We consider three different strategies to start the JDQZ method, viz. starting with a random vector, starting with the first column of the matrix $Q_{k_{\max}}$ computed in the previous continuation step, and starting with the subspace spanned by the columns of $Q_{k_{\max}}$; we will call these methods JDQZ1, JDQZ2 and JDQZ3 respectively (the latter two have been discussed in Section 3.3).

In Table 2 we listed the CPU-times (in seconds) for an 'average' continuation step. (We performed 15 steps, and the average of the last 10 steps is listed in the table, so that the effect of starting the continuation is ruled out). In each step, 3 Newton iterations were needed to compute a stationary solution with Algorithm 1. The computations were done on a SUN SPARC 1000E with 4 processors.

$k_{\max}$	ε	solving (3.12)	CPU(s) JDQZ1	CPU(s) JDQZ2	CPU(s) JDQZ3
–	–	–	33	33	33
4	10^{-6}	GMRES$_5$	168	162	170
4	10^{-6}	BiCGstab(2)	178	166	173
4	10^{-9}	GMRES$_5$	211	215	284
4	10^{-9}	BiCGstab(2)	245	226	265
6	10^{-6}	GMRES$_5$	207	197	305
6	10^{-6}	BiCGstab(2)	227	210	289
6	10^{-9}	GMRES$_5$	271	262	489
6	10^{-9}	BiCGstab(2)	345	307	437

Table 2. *The CPU-time (in seconds) for an 'average' continuation step. In the first line of the table the CPU-time for one step of Algorithm 1 has been displayed (no eigenvalues have been computed in this case).*

From the results in Table 2 we may conclude that the JDQZ method is well suited for computing several eigenvalues accurately.

When we compare the CPU-times for GMRES_5 and BiCGstab(2) we see that the first one is the most efficient for solving (3.10) when JDQZ1 or JDQZ2 is used. In the experiments with JDQZ1 and JDQZ2 we observed that $K = A - \tau B$ is a very good preconditioner for (3.10); the criterion (3.12) was often satisfied after performing 2 or 3 GMRES steps. When applying BiCGstab(2), one has to perform 4 matrix-vector multiplications per step. Hence BiCGstab(2) requires more matrix-vector multiplications per 'average' step, and this might explain why GMRES_5 is more efficient for JDQZ1 and JDQZ2. On the other hand, when JDQZ3 is used, BiCGstab(2) turns out to be more efficient than GMRES_5. In these experiments we observed that the average number of JD steps for GMRES_5 is significantly larger than for BiCGstab(2), and this might explain why solving (3.10) with BiCGstab(2) is more efficient when JDQZ3 is used.

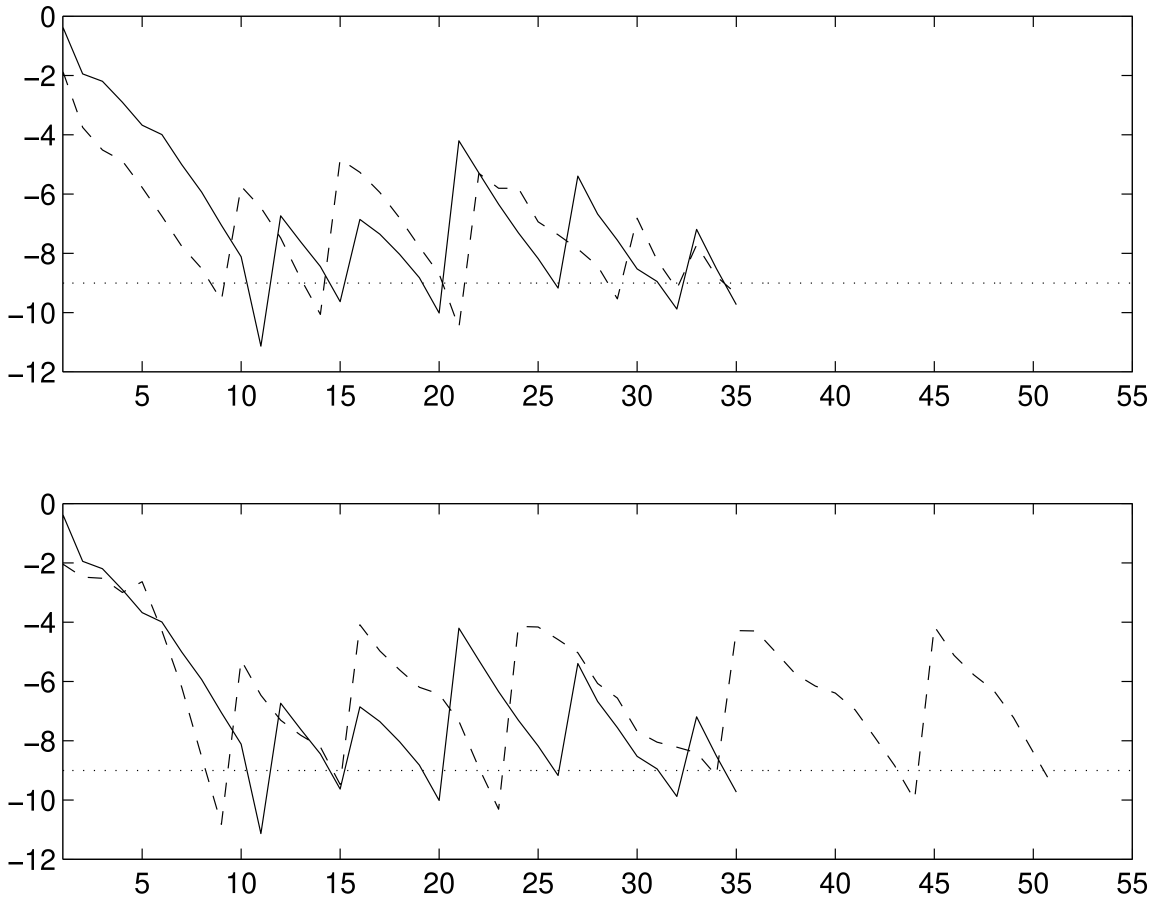

Figure 3. *The convergence of the JDQZ method with different starting strategies for the last continuation step ($k_{\max} = 6$, $\varepsilon = 10^{-9}$ and* GMRES_5 *has been used to solve (3.12)). On the horizontal axis the number of JD steps has been displayed, and* $\log_{10}(\|r\|_2)$ *(r is the residual in the JDQZ method) is on the vertical axis. In both pictures, the solid curve corresponds to JDQZ1; the dashed curve in the upper picture corresponds to JDQZ2, and the dashed curve in the lower picture corresponds to JDQZ3.*

A somewhat surprising result is that starting the JDQZ method with Schur vectors from the previous continuation step does not improve the efficiency of the method much. Starting with the first Schur vector (JDQZ2) leads to (almost) the same CPU-times (in particular when GMRES_5 is used to solve (3.10)), while starting with all Schur vectors (JDQZ3) leads to a significantly slower method. In order to understand this, we have plotted the convergence behaviour of the different JDQZ methods at the last continuation step in Figure 3. When we compare JDQZ1 with JDQZ2 we observe that the first Schur vector is detected earlier with JDQZ2, but JDQZ2 needs more JD steps to compute the other Schur vectors. This might be explained as follows: the first Schur vector of the previous continuation step might have a larger component in the direction of the first Schur vector than a random vector (observe that the norm of the initial residual is smaller), so that this first Schur vector is found earlier. On the other hand, it is likely that a random vector has larger components in the direction of the other Schur vectors, so the subspace span$\{V\}$ (after the detection of the first Schur vector) might contain more information about the other Schur vectors than the corresponding subspace in JDQZ2 (compare the norms of the residuals just after the first restart, when the first Schur vector has been removed from span$\{V\}$). In the upper picture in Figure 3 we see that both methods need about the same number of JD steps in order to find 6 Schur vectors (with corresponding eigenvalues).

For the JDQZ3 method we would not expect such behaviour, because the start subspace contains all Schur vectors computed in the previous continuation step. Again the first Schur vector is found earlier (in comparison with JDQZ1). It is possible that some Schur vectors of the previous continuation step are removed in span$\{V\}$ when this space is reduced, after restarting, from dimension 20 ($= j_{\max}$) to 10 ($= j_{\min}$), but this does not explain why JDQZ3 needs more JD steps than JDQZ1 (or JDQZ2) to discover the other Schur vectors. Perhaps it is better to start JDQZ with the first old Schur vector, and add the second Schur vector of the previous continuation step to span$\{V\}$ when the first new Schur vector has been detected and removed from span$\{V\}$ etc. We have not tried this approach. On the other hand, JDQZ1 performs very well for this example, so it could be hard to construct a method which performs better in this case.

With both $\varepsilon = 10^{-6}$ and $\varepsilon = 10^{-9}$ we obtained the same eigenvalues, so it is not necessary to take ε too small. In applications one might e.g. set $\varepsilon = 10^{-6}$, and switch to $\varepsilon = 10^{-9}$ when an eigenvalue close to the imaginary axis has been found. The 4 eigenvalues computed with $k_{\max} = 4$ were also detected with $k_{\max} = 6$, and they turned out to be the four rightmost ones. This shows that the choice $k_{\max} = 4$ is a reasonable one.

Unfortunately, it in general is not possible to determine a complete LU-factorization of the matrix $A - \tau B$ (consider e.g. problems where the matrices A and B stem from a system of partial differential equations in 3D). For these problems one might consider an incomplete decomposition $K = LU \approx A - \tau B$, or another type of preconditioner. In order to investigate the behaviour of JDQZ when incomplete factorizations are used, we have repeated some of our experiments with an incomplete factorization of $A - \tau B$. An incomplete block LU factorization, based on a repeated red-black ordering [1] is used (cf. [14]); each block is a 4×4 matrix corresponding to the variables u, w, p, and T at a certain grid cell. The linear equations arising from Newton's method in Algorithm 1 have been solved with BiCGstab(8). The CPU-time of an average continuation step with $k_{\max} = 0$ is 388 seconds, while only 33 seconds were needed for a direct factorization of the Jacobian (see Table 2). It is not an easy task to compute eigenvalues with JDQZ, using this preconditioner; we did not find any eigenvalues with the methods and parameters chosen as in Table 2 (i.e. at most 100 matrix-vector multiplications with BiCGstab(2)). Using BiCGstab(8) instead, with a maximum of 1000 matrix-vector multiplications for solving (3.10) (instead of 100), and $k_{\max} = 4$, $\varepsilon = 10^{-6}$, we were able to find the four rightmost eigenvalues at Ra = 1900. The CPU-time of this continuation step was 4793 seconds, while an 'average' continuation step with a direct factorization took less than 180 seconds (cf. Table 2). For the 2D Rayleigh-Bénard problem, this incomplete factorization is not well suited as a preconditioner for solving the correction equation (3.10). Without a good preconditioner it can be very hard to obtain eigenvalues with the JDQZ method within a reasonable computation time.

Finally we compare the JDQZ method to the SIT method [13], which has been used in [3] to compute the rightmost eigenvalues. In [3] the SIT method (which is essentially a block version of the power method, see [13, 3] for the details) has been applied to the matrix $C = (B - A)^{-1}(A + B)$; a stationary solution y is stable when all eigenvalues of the matrix C have a modulus less than 1 (cf., e.g., [3]). For our discretization of the Rayleigh-Bénard problem the SIT method was not able to compute the rightmost eigenvalue accurately; in our experiments we observed that the error in the rightmost eigenvalue is slightly more than 1% — even for the experiments with much higher CPU-times for an 'average' continuation step than those corresponding to the JDQZ method with $k_{\max} = 4$ and $\varepsilon = 10^{-6}$. (An LU-factorization of $B - A$ has always been used to solve the linear systems occurring in the SIT method.) These experiments indicate that the JDQZ method may be much more efficient than the SIT method for computing eigenvalues occurring in continuation methods for problems of the type that we have considered.

4.3 Conclusions

In this section the main conclusions of the paper are summarized.

The JDQZ method can be a very efficient tool for computing several rightmost eigenvalues in continuation methods, provided a good preconditioner for the correction equation (3.10) is available. Without such a preconditioner the JDQZ method may behave rather poorly (cf. also [5]). For some (small) problems a good preconditioner can be constructed by a direct factorization of $A - \tau B$, but this is not feasible for large problems. When such preconditioners become available, the JDQZ method might be suitable for computing eigenvalues related to 3D problems.

In our experiments we observed that the JDQZ method is more efficient than the SIT method [13], which is not surprising because the SIT method is based on a block version of the power method, which converges linearly, while the JDQZ method often shows a quadratic convergence behaviour.

Using one or more Schur vectors from the previous continuation step does not necessarily lead to a faster convergence of the JDQZ method. Starting the JDQZ method with one Schur vector or a random vector gives about the same computation times, while starting with a subspace containing all previously computed Schur vectors led to a significantly slower method.

Acknowledgements. The author wishes to thank D.R. Fokkema and M.B. van Gijzen (Utrecht University, Dept. of Math.) for providing the JDQZ code, G. Tiesinga (Groningen University, Dept. of Math.) for the Rayleigh-Bénard code with the incomplete factorization mentioned in Section 4.2.2, and M. Vellinga, H.A. Dijkstra, and M.J. Molemaker (Utrecht University, IMAU) for the continuation code as used in [3]. Further he is indebted to G.L.G. Sleijpen, H.A. van der Vorst (Utrecht University, Dept. of Math.), and F.W. Wubs (Groningen

University, Dept. of Math.) for stimulating discussions, and suggestions concerning the presentation of this paper.

References

[1] C.W. Brand: An incomplete-factorization preconditioning using repeated red-black ordering. *Numer. Math.* **61**, 433-454 (1992).

[2] P.N. Brown & Y. Saad: Hybrid Krylov methods for nonlinear systems of equations. *SIAM J. Sci. Stat. Comput.* **11**, 450-481 (1990).

[3] H.A. Dijkstra, M.J. Molemaker, A. van der Ploeg & E.F.F. Botta: An efficient code to compute non-parallel steady flows and their linear stability. *Computers and Fluids* **24**, 415-434 (1995).

[4] D.R. Fokkema, G.L.G. Sleijpen & H.A. van der Vorst: Accelerated inexact Newton schemes for large systems of nonlinear equations. Preprint nr. 918, Dept. Math., Utrecht University, Utrecht, July 1995. To appear in SIAM J. Sci. Comput.

[5] D.R. Fokkema, G.L.G. Sleijpen & H.A. van der Vorst: Jacobi-Davidson style QR and QZ algorithms for the partial reduction of matrix pencils. Preprint nr. 941, Dept. Math., Utrecht University, Utrecht, January 1996. To appear in SIAM J. Sci. Comput.

[6] G.H. Golub & C.F. van Loan: *Matrix Computations*, 2nd Ed. Baltimore: The John Hopkins University Press, 1989.

[7] G.H. Golub & H.A. van der Vorst: 150 Years old and still alive: eigenproblems. Preprint nr. 981, Dept. Math., Utrecht University, Utrecht, October 1996. To appear in *The State of the Art in Numerical Analysis* (I.S. Duff & G.A. Watson eds). Oxford: Oxford University Press.

[8] H.B. Keller: Numerical solution of bifurcation and nonlinear eigenvalue problems, in *Applications of Bifurcation Theory* (P.H. Rabinowitz ed.), 359-384. Academic Press, 1977.

[9] Y. Saad & M. Schultz: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.* **7**, 856-869 (1986).

[10] R. Seydel: *Practical Bifurcation and Stability Analysis*, 2nd Ed. New York: Springer, 1994.

[11] G.L.G. Sleijpen, A.G.L. Booten, D.R. Fokkema & H.A. van der Vorst: Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems. *BIT* **36**, 595-633 (1996).

[12] G.L.G. Sleijpen & D.R. Fokkema: BiCGstab(ℓ) for linear equations involving unsymmetric matrices with complex spectrum. *Electronic Transactions in Numerical Analysis* (*ETNA*) **1**, 11-32 (1993).

[13] W.J. Stewart & A. Jennings: A simultaneous iteration algorithm for real matrices. *ACM Trans. Math. Softw.* **7**, 184-198 (1981).

[14] G. Tiesinga: Block preconditioned BiCGstab(2) for solving the Navier-Stokes equations. Z. angew. Math. Mech. **76**, Supplement 1, 563-564 (1996).