

Short manual for the JDQZ-package

Diederik R. Fokkema* Martin B. van Gijzen[‡]

July 11, 2012

1 Introduction

The JDQZ algorithm computes a number of eigenpairs near a target value of the generalized eigen problem

$$\beta \mathbf{A} \mathbf{x} = \alpha \mathbf{B} \mathbf{x}. \quad (1)$$

The algorithm is described in the paper "Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils" by D.R. Fokkema, G.L.G. Sleijpen, and H.A. van der Vorst [?]. The paper can be obtained from the homepage of Gerard Sleijpen (<http://www.math.ruu.nl/people/sleijpen/>) or from the homepage of Henk van der Vorst (<http://www.math.ruu.nl/people/vorst/>). This package is a Fortran 77 implementation of the algorithm. The code has been developed by Diederik Fokkema and has been modified and is currently maintained by Martin van Gijzen.

Permission to copy all or part of this code is granted, provided that the copies are not made or distributed for resale.

THE CODE IS PROVIDED ON AN "AS IS" BASIS. THE AUTHORS PROVIDE NO WARRANTY WHATSOEVER, EITHER EXPRESSED OR IMPLIED, REGARDING THE WORK, INCLUDING WARRANTIES WITH RESPECT TO ITS MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

*ISE Integrated Systems Engineering AG, Technopark Zürich, Technoparkstrasse 1, CH-8005 Zürich, Switzerland. E-mail: fokkema@ise.ch.

[‡]Departement of Mathematics, Utrecht University, P.O. Box 80.010, NL-3508 TA Utrecht, The Netherlands. E-mail: sleijpen@math.ruu.nl, vorst@math.ruu.nl.

2 Obtaining the code

To obtain the code you can send an e-mail to: vangyzen@math.ruu.nl. In return you will receive a uuencoded, gzipped and tared file. Save this file as jdqz.tar.gz.uue. You can install the code by typing the following commands:

```
uudecode jdqz.tar.gz.uue
gunzip jdqz.tar.gz
tar xvf jdqz.tar
```

This has created a directory `jdqz`, with two subdirectories `jdlib` and `jdtest`. You can find this manual in the directory `jdqz`, the code is in the subdirectories. Makefiles are supplied, so if the necessary libraries are present, in particular LAPACK, typing `make` should make the `jdqz-library`.

3 Usage of the package

The package is written in Fortran 77 and uses the double complex data-type. It calls routines from the LAPACK and BLAS libraries. These libraries are installed on many machines. LAPACK and BLAS routines can also be obtained from Netlib (<http://www.netlib.org/index.html>). The directory `jdlib` contains the actual JDQZ-code, the directory `jdtest` contains some examples that may be of help in using the code.

3.1 User supplied subroutines

The user has to supply three problem dependent routines: one for the multiplication of a vector with the operator **A**, one for multiplication with **B**, and one for performing the preconditioning operation. The subroutine to multiply with **A** must be called `AMUL` and must have the following header:

```
      subroutine AMUL( n, q, r )
C.....
C...      Subroutine to compute r = Aq
C.....
      integer          n
      double complex q(n), r(n)
```

\mathbf{q} is the input vector, \mathbf{r} the output vector. n is the dimension of the problem. The subroutine to multiply with \mathbf{B} must be called **BMUL** and must have the following header:

```

      subroutine BMUL( n, q, r )
c.....
c...      Subroutine to compute  $\mathbf{r} = \mathbf{B}\mathbf{q}$ 
c.....
      integer          n
      double complex q(n), r(n)

```

Finally, the routine to perform the preconditioning operation must be called **PRECON** and must have the header

```

      subroutine PRECON( n, q )
c.....
c...      Subroutine to compute  $\mathbf{q} = \mathbf{K}^{-1} \mathbf{q}$ 
c.....
      integer          n
      double complex q(n)

```

The preconditioning matrix should be an approximation of the matrix $\mathbf{A} - \tau\mathbf{B}$, with τ the prechosen target value. Preconditioning within the JDQZ algorithm is described in section 3.4 of [?]. Preconditioning is not essential for the correct behavior of the algorithm. It should improve the rate of convergence, but leaving the vector \mathbf{q} untouched should have no influence on the correctness of the results.

Note that data for the matrices, like the nonzero coefficients and their indices, should be passed via **common** blocks, not via the parameter lists.

3.2 Calling JDQZ

JDQZ itself can be called with the statement

```
call JDQZ( alpha, beta, eivec, wanted, n, target, eps,  
$      kmax, jmax, jmin, method, m, l, mxmv, maxstep,  
$      lock, order, testspace, zwork, lwork )
```

The parameters must be of the following data types:

alpha, beta	double complex array, size jmax
eivec	two dimensional double complex array, size $n \times kmax$
wanted	logical, scalar
n	integer, scalar
target	double complex, scalar
eps	double precision, scalar
kmax	integer, scalar
jmax	integer, scalar
jmin	integer, scalar
method	integer, scalar
m	integer, scalar
l	integer, scalar
mxmv	integer, scalar
maxstep	integer, scalar
lock	double precision, scalar
order	integer, scalar
testspace	integer, scalar
zwork	two dimensional double complex array, size $n \times lwork$
lwork	integer, scalar

The meaning of most of the parameters can be found in [?] and we will refer to the appropriate section if possible.

<code>alpha, beta</code>	Obvious from equation (??)
<code>wanted</code>	Compute the converged eigenvectors (if <code>wanted = .true.</code>)
<code>eivec</code>	Converged eigenvectors if <code>wanted = .true.</code> , else converged Schur vectors
<code>n</code>	The size of the problem
<code>target</code>	The value near which the eigenvalues are sought
<code>eps</code>	Tolerance of the eigensolutions, $\ \beta \mathbf{A}\mathbf{x} - \alpha \mathbf{B}\mathbf{x}\ / \alpha/\beta < \epsilon$
<code>kmax</code>	Number of wanted eigensolutions, on output: number of converged eigenpairs
<code>jmax</code>	Maximum size of the search space
<code>jmin</code>	Minimum size of the search space
<code>method</code>	Linear equation solver:
1:	GMRES _m , [?]
2:	BiCGstab(ℓ), [?]
<code>m</code>	Maximum dimension of searchspace of GMRES _m
<code>l</code>	Degree of GMRES-polynomial in Bi-CGstab(ℓ)
<code>mxmv</code>	Maximum number of matrix-vector multiplications in GMRES _m or BiCGstab(ℓ)
<code>maxstep</code>	Maximum number of Jacobi-Davidson iterations
<code>lock</code>	Tracking parameter (section 2.5.1)
<code>order</code>	Selection criterion for Ritz values:
0:	nearest to <code>target</code>
-1:	smallest real part
1:	largest real part
-2:	smallest imaginary part
2:	largest imaginary part
<code>testspace</code>	Determines how to expand the testspace \mathbf{W}
1:	$\mathbf{w} = \text{"Standard Petrov"} \times \mathbf{v}$ (Section 3.1.1)
2:	$\mathbf{w} = \text{"Standard 'variable' Petrov"} \times \mathbf{v}$ (Section 3.1.2)
3:	$\mathbf{w} = \text{"Harmonic Petrov"} \times \mathbf{v}$ (Section 3.5.1)
<code>zwork</code>	Workspace
<code>lwork</code>	Size of workspace, $\geq 4+m+5jmax+3kmax$ if GMRES _m is used, $\geq 10 + 6\ell + 5jmax + 3kmax$ if Bi-CGstab(ℓ) is used.

4 Guidelines for choosing the parameters.

In this section we will try to give some sound guidelines how to choose the parameters. Optimal parameters are of course problem and system dependent, but we will give "on average" reasonable values, based on practical experience.

eps	10^{-9} , don't take it too large. A relatively large value for eps may cause problems when computing many eigen solutions. Moreover, convergence from moderate to high accuracy is fast.
kmax	It may be wise to take kmax a bit larger than the actual number of eigensolutions you want, to avoid missing one
jmax	very problem and memory dependent, reasonable value: $3 \times \mathbf{kmax}$ with a minimum of 20
jmin	$2 \times \mathbf{kmax}$
method	2 (BiCGstab(ℓ))
m	30
l	2
mxmv	Very problem dependent, any value in the range 5-100 is reasonable. Suggestion: 100
maxstep	1000
lock	Take it small to avoid missing eigensolutions, eg. lock = 10^{-9} .
testspace	3 if a reasonable value for target is known, else take 2

5 Example

In this section we will illustrate the usage of the code by a very simple example. The example can be found in the directory `jdtest`. For **A** and **B** we take diagonal matrices. The action of **A** and **B** is described by the following piece of Fortran code:

```

      subroutine AMUL( n, q, r )
c.....
c...      Subroutine to compute r = Aq
c.....
      integer          n, i
      double complex q(n), r(n)
c
      do i = 1, n

```

```

        r(i) = i*q(i)
    end do
c
    end

    subroutine BMUL( n, q, r )
c.....
c...    Subroutine to compute r = Bq
c.....
        integer          n, i
        double complex q(n), r(n)
c
        do i = 1, n
            r(i) = q(i)/i
        end do
c
    end

```

Obviously, we have for the eigenvalues α/β of (??)

$$\alpha/\beta = i^2 \quad (i = 1, \dots, n) \quad (2)$$

We want to compute the five eigenvalues closest to 31, so **target** = 31, and **kmax** = 5. The preconditioning operation should mimic the action of the matrix $(\mathbf{A} - \tau\mathbf{B})^{-1}$. Since we are dealing with diagonal matrices we can take the exact inverse. The preconditioning operation is then described by

```

    subroutine PRECON( n, q )
c.....
c...    Subroutine to compute q = K^-1 q
c.....
        integer          n, i
        double complex q(n)
c
        do i = 1, n
            q(i) = i*q(i)/(i*i-31)
        end do
c
    end

```

We take for the problem size $n = 100$. We are searching near a target, hence `order = 0`. For the rest of the parameters we take the values suggested in the previous section:

```

eps          10-9
jmax         20
jmin         10
method       2
m            30 (does not matter, is not used)
l            2
mxmv         100
maxstep      1000
lock         10-9.
testspace    3

```

At the end we want to compute the norms of the residuals

$$\|\mathbf{r}_i\| = \|\beta_i \mathbf{A} \mathbf{x}_i - \alpha_i \mathbf{B} \mathbf{x}_i\| \quad (3)$$

and therefore need the eigenvectors. Hence `wanted = .true..` The parameter `lwork` is set to 137, the minimum allowed for the choice of parameters. The following piece of code

```

      call JDQZ(alpha, beta, eivec, wanted, n, target, eps,
$      kmax, jmax, jmin,
$      method, m, l, maxnmv, maxstep,
$      lock, order, testspace, zwork, lwork )
c
      elapse = etime( tarray )
c
c...      Compute the norms of the residuals:
c...      tmp is a double complex array of size n
      do j = 1, kmax
         call amul ( n, eivec(1,j), residu )
         call zscal ( n, beta(j), residu, 1)
         call bmul ( n, eivec(1,j), tmp )
         call zaxpy( n, -alpha(j), tmp, 1, residu, 1 )
         print '("lambda(",i2,"): (" ,1p,e11.4," ,",e11.4,
$           " )")', j,alpha(j)/beta(j)
         print '(a30,d13.6)', '||beta Ax - alpha Bx||:',
$           dznrm2( n, residu, 1 )
      end do
      write(*,10) tarray(1), elapse

```


c

```
10 format(1x,'END JDQZ AFTER ',f6.2,' SEC. CPU-TIME AND ', f6.2,  
$      ' SEC. ELAPSED TIME' )
```

gives as output

```
lambda( 1): ( 3.6000E+01, 0.0000E+00 )  
           ||beta Ax - alpha Bx||: 0.109282D-10  
lambda( 2): ( 2.5000E+01, 0.0000E+00 )  
           ||beta Ax - alpha Bx||: 0.200064D-09  
lambda( 3): ( 1.6000E+01, 0.0000E+00 )  
           ||beta Ax - alpha Bx||: 0.207683D-10  
lambda( 4): ( 4.9000E+01, 0.0000E+00 )  
           ||beta Ax - alpha Bx||: 0.373422D-09  
lambda( 5): ( 9.0000E+00, 0.0000E+00 )  
           ||beta Ax - alpha Bx||: 0.526224D-10  
END JDQZ AFTER    0.84 SEC. CPU-TIME AND    0.96 SEC. ELAPSED TIME
```

The preconditioning reduces the computing time considerably. JDQZ without preconditioning gives the same eigenvalues, but for the timing it gives

```
END JDQZ AFTER    3.97 SEC. CPU-TIME AND    4.13 SEC. ELAPSED TIME
```

hence, preconditioning gives a reduction of the computing time of about a factor four for this example.

References

- [1] D.R. Fokkema, G.L.G. Sleijpen, and H.A. van der Vorst, *Jacobi-Davidson style QR and QZ algorithms for the partial reduction of matrix Pencils*. Preprint 941, Department of Mathematics, Utrecht University, January, 1996.
- [2] Y. Saad and M.H. Schultz. GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.
- [3] G.L.G. Sleijpen and D.R. Fokkema. BiCGstab(ℓ) for linear equations involving matrices with complex spectrum. *ETNA* 1:11–32, 1994