

A Set of GMRES Routines for Real and Complex Arithmetics

Valérie Frayssé * Luc Giraud * Serge Gratton *

CERFACS Technical Report TR/PA/97/49

Abstract

In this report we describe the implementations of the GMRES algorithm for both real and complex, single and double precision arithmetics suitable for serial, shared memory and distributed memory computers. For the sake of simplicity, flexibility and efficiency the GMRES solvers have been implemented using the reverse communication mechanism for the matrix-vector product, the preconditioning and the dot product computations. For distributed memory computation several orthogonalization procedures have been implemented to reduce the cost of the dot product calculation, that is a well-known bottleneck of efficiency for the Krylov methods. Finally the implemented stopping criterion is based on a normwise backward error. After a short presentation of the GMRES methods and of the solution of the least-squares problems in real and complex arithmetic, we give a detailed description of the user interface.

Keywords : linear systems, Krylov methods, GMRES, reverse communication, distributed memory.

1 The GMRES algorithm

1.1 General description

The Generalized Minimum RESidual (GMRES) method was proposed by Saad and Schultz in 1986 [6] in order to solve large, sparse and nonsymmetric (or non Hermitian) linear systems. GMRES belongs to the class of Krylov based iterative methods.

Let A be a square nonsingular $n \times n$ complex matrix, and b be a complex vector of length n , defining the linear system

$$Ax = b \tag{1}$$

to be solved. Let $x_0 \in \mathbf{C}^n$ be an initial guess for this linear system and $r_0 = b - Ax_0$ be its corresponding residual.

The GMRES algorithm builds an approximation of the solution of (1) under the form

$$x_m = x_0 + V_m y \tag{2}$$

*CERFACS, 42 av. Gaspard Coriolis, 31057 Toulouse Cedex, France.
Email : fraysse,giraud,gratton@cerfacs.fr

where V_m is an orthonormal basis for the Krylov space of dimension m

$$\mathcal{K}_m = \text{span} \{r_0, Ar_0, \dots, A^{m-1}r_0\},$$

and where y belongs to \mathbf{C}^m . The vector y is determined so that the 2-norm of the residual $r_m = b - Ax_m$ is minimal over \mathcal{K}_m .

The basis V_m for the Krylov subspace \mathcal{K}_m is obtained via the well-known Arnoldi process. The orthogonal projection of A onto \mathcal{K}_m results in an upper Hessenberg matrix $H_m = V_m^*AV_m$ of order m . The Arnoldi process satisfies the relationship

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T, \quad (3)$$

where e_m is the m^{th} canonical vector. Equation (3) can be rewritten as

$$AV_m = V_{m+1} \bar{H}_m$$

where

$$\bar{H}_m = \begin{bmatrix} & & & & & \\ & & & & & \\ & & & & & \\ & & & H_m & & \\ & & 0 & \dots & 0 & \\ & & & & & h_{m+1,m} \end{bmatrix}$$

is an $(m+1) \times m$ matrix.

The Arnoldi process may breakdown if $h_{j+1,j} = 0$ for a certain j , which means that the subspace \mathcal{K}_j is invariant under A . However, this is an happy breakdown since the exact solution is obtained.

Let $v_1 = r_0/\beta$ where $\beta = \|r_0\|_2$. The residual r_m associated with the approximate solution (2) verifies

$$\begin{aligned} r_m &= b - Ax_m = b - A(x_0 + V_m y) \\ &= r_0 - AV_m y = r_0 - V_{m+1} \bar{H}_m y \\ &= \beta v_1 - V_{m+1} \bar{H}_m y \\ &= V_{m+1} (\beta e_1 - \bar{H}_m y). \end{aligned}$$

Since V_{m+1} is an orthonormal matrix, the residual norm $\|r_m\|_2 = \|\beta e_1 - \bar{H}_m y\|_2$ is minimal when y solves the linear least-squares problem

$$\min_{y \in \mathbf{C}^m} \|\beta e_1 - \bar{H}_m y\|_2. \quad (4)$$

We will denote by y_m the solution of (4). Therefore, $x_m = x_0 + V_m y_m$ is an approximate solution of (1) for which the residual is minimal over \mathcal{K}_m . GMRES owes its name to this minimization property.

In exact arithmetic, GMRES converges in at most n steps. However, in practice, the convergence may be slow and the storage of the orthogonal basis V_m may become prohibitive. The restarted GMRES method is designed to cope with this memory drawback. Given a fixed m , the restarted GMRES method computed the sequel of approximate solutions x_j until x_j is acceptable or $j = m$. If the solution was not found, then a new starting vector is chosen on which GMRES is applied again. Often, GMRES is restarted from the last computed approximation, i.e. $x_0 = x_m$. The process is iterated until a good enough approximation is found. We will denote by GMRES(m) the restarted GMRES algorithm for a projection size of at most m .

In the following paragraphs, we enlight the main key-points for GMRES:

- the solution of the least-squares problem (4),
- the construction of the orthonormal basis V_m , and
- the stopping criteria for the iterative scheme.

1.2 The least-squares problem

At each step of GMRES, one needs to solve the least-squares problem (4). The matrix \bar{H}_m involved in this least-squares problem is an $(m+1) \times m$ complex matrix which is upper Hessenberg. We wish to use an efficient algorithm for solving (4) which exploits the structure of \bar{H}_m .

First, we base the solution of (4) on the QR factorization of the matrix $[\bar{H}_m, \beta e_1]$: if $QR = [\bar{H}_m, \beta e_1]$ where Q is an orthonormal matrix and $R = (r_{ij})$ is an $(m+1) \times (m+1)$ upper triangular matrix, then the solution y_m of (4) is given by

$$y_m = R(1:m, 1:m)^{-1} R(1:m, m+1). \quad (5)$$

Here, $R(1:m, 1:m)$ denotes the $m \times m$ first submatrix of R and $R(1:m, m+1)$ stands for the last column of R . Moreover, it is easy to see that

$$\|r_m\|_2 = \|b - Ax_m\|_2 = \|\beta e_1 - \bar{H}_m y_m\|_2 = |r_{m+1, m+1}|. \quad (6)$$

Therefore, the residual of the linear system need not be computed explicitly.

There are two classical ways of reliably performing the QR factorization of a matrix: one is based on Householder reflections, the other on Givens rotations. Householder reflections zero out all the components but one in a column vector, whereas Givens rotations zero out a selected entry only. This second solution should be preferred here because the matrix \bar{H}_m is already in the Hessenberg form.

Givens rotations are matrices of the form

$$G = G(i, k, \theta) = \begin{pmatrix} I & & & & \\ & c & & -s & \\ & & I & & \\ & s & & c & \\ & & & & I \end{pmatrix}$$

where $G_{ii} = c = \cos \theta$, $G_{ik} = -s = -\sin \theta$ and the I s are identity matrices of appropriate dimensions. Let z be a *real* vector such that $z_i \neq 0$ and $z_k \neq 0$. Then for θ satisfying

$$\cos \theta = \frac{z_i}{\sqrt{z_i^2 + z_k^2}} \quad \text{and} \quad \sin \theta = \frac{z_k}{\sqrt{z_i^2 + z_k^2}}$$

the vector $w = G(i, k, \theta)^T z$ is such that $w_k = 0$. For a reliable computation of $\cos \theta$ and $\sin \theta$, the reader is referred to [5]. However, this definition of a Givens rotation works only in the case of real numbers. In complex arithmetic, we define the Givens rotations as

$$G = G(i, k, \theta) = \begin{pmatrix} I & & & & \\ & c & & -s & \\ & & I & & \\ & \bar{s} & & c & \\ & & & & I \end{pmatrix}$$

where $G_{ii} = c$ is a *real* coefficient and $G_{ik} = -s$ is *complex*. Let z be a complex vector with $z_k \neq 0$. The vector $w = G^T z$ is such that $w_k = 0$ if we choose

$$c = \frac{|z_i|}{\sqrt{z_i^2 + z_k^2}} \quad \text{and} \quad s = c \frac{z_k}{z_i} \quad \text{if} \quad z_i \neq 0,$$

or $c = 0$ and $s = 1$ otherwise.

The QR factorization of the matrix $[\bar{H}_m, \beta e_1]$ is not fully performed at each step m of the algorithm: it is updated from that of $[\bar{H}_{m-1}, \beta e_1]$ obtained at the previous step.

1.3 Computation of V_m

Most of the time, the Arnoldi algorithm is implemented through the Modified Gram-Schmidt (MGS) process for the computation of V_m and H_m . However, in finite precision arithmetic, there might be a severe loss of orthogonality in the computed basis¹; this loss can be compensated by iterating the orthogonalization scheme [2]. The resulting algorithm is called Iterative Modified Gram-Schmidt (IMGS). The drawback of IMGS is the increased number of dot products.

The Classical Gram-Schmidt (CGS) algorithm can be implemented in a cheap manner by gathering the dot products into one matrix-vector product, but it is well-known that CGS is numerically worse than MGS. However, iterating CGS (ICGS) results in an algorithm of the same numerical quality as IMGS. Therefore, ICGS is particularly attractive in a parallel distributed environment, where the computation of the dot product is a well-known bottleneck [7].

In our GMRES implementation, we have chosen to give the user the possibility of using any of the four different schemes quoted above : CGS, MGS, ICGS and IMGS.

1.4 Preconditioning

We allow left and/or right preconditioning so that we actually solve the linear system

$$M_1^{-1} A M_2^{-1} z = M_1^{-1} b \tag{7}$$

with $x = M_2^{-1} z$. The selection and the computation of the preconditioning step is left to the user. The use of preconditioners has some consequences on the stopping criterion. We discuss these points in the next paragraph.

1.5 Stopping criteria

We have chosen to base our stopping criterion on the normwise backward error [3]. The backward error analysis, introduced by Givens and Wilkinson [8], is a powerful concept for analyzing the quality of an approximate solution:

1. it is independent from the details of round-off propagation: the error introduced during the computation are interpreted in terms of perturbations of the initial data, and the computed solution is considered as exact for the perturbed problem;
2. because round-off errors are seen as data perturbations, they can be compared with errors due to numerical approximations (consistency of numerical schemes) or to physical measurements (uncertainties on data coming from experiments for instance).

¹Whether this loss of orthogonality inhibits the convergence of GMRES is still a research topic.

The backward error measures in fact the distance between the data of the initial problem and those of the perturbed problem; therefore it relies upon the choice of the data allowed to vary and the norm to measure these variations. In the context of linear systems, classical choices are the normwise and the componentwise perturbations [3]. These choices lead to explicit formulas for the backward error (often a normalized residual) which is then easily evaluated. For iterative methods, it is generally admitted that the normwise model of perturbation is appropriate [1].

Let x_m be an approximation of the solution $x = A^{-1}b$. Then

$$\begin{aligned}\eta(x_m) &= \min \{ \varepsilon > 0; \|\Delta A\|_2 \leq \varepsilon\alpha, \|\Delta b\|_2 \leq \varepsilon\beta \text{ and } (A + \Delta A)x_m = b + \Delta b \} \\ &= \frac{\|b - Ax_m\|_2}{\alpha \|x_m\|_2 + \beta}\end{aligned}$$

is called the *normwise backward error* associated with x_m . The best one can require from an algorithm is a backward error of the order of the machine precision. In practice, the approximation of the solution is acceptable when its backward error is lower than the uncertainty on the data. Therefore, there is no gain in iterating after the backward error has reached machine precision (or data accuracy). Thanks to Equality (6), we see that the 2-norm of the residual is given directly in the algorithm during the solution of the least-squares problem. Therefore, the backward error can be obtained at a low cost and we can use

$$\eta_A = \frac{|r_{m+1,m+1}|}{\alpha \|x_m\|_2 + \beta}$$

as the stopping criterion of the GMRES iterations. However, it is well-known that, in finite precision arithmetic, the computed residual (6) given from the Arnoldi process may differ significantly from the true residual. Therefore, it is not safe to use exclusively η_A as the stopping criterion. Our strategy is the following: first we iterate until η_A becomes lower than the tolerance, then afterwards, we iterate until η becomes itself lower than the tolerance. We hope in this way to minimize the number of residual computations (involving the computation of matrix-vector product) necessary to evaluate η , while having a reliable stopping criterion.

When GMRES is used in conjunction with preconditioning, then our stopping criterion is based on the backward error for the preconditioned system (7):

$$\eta^P = \frac{\|M_1^{-1}AM_2^{-1}z_m - M_1^{-1}b\|_2}{(\alpha^P \|x_m\|_2 + \beta^P)}$$

with $x_m = M_2^{-1}z_m$. We denote by

$$\eta_A^P = \frac{|r_{m+1,m+1}|}{\alpha^P \|x_m\|_2 + \beta^P}$$

the stopping criterion for the preconditioned GMRES. As previously, we stop the iterations when the computed values of η_A^P and then η^P satisfy the prescribed tolerance. We prefer to stop the iterations on the preconditioned linear system and not on the original linear system because the residual which is readily available in the algorithm is that of the *preconditioned* system. It would be too expensive to compute the residual of the unpreconditioned system at each iteration. For the user's information, we also give the value of the backward error for the unpreconditioned system on return from the solver.

Although there is a priori no order between the backward error of the preconditioned system and that of the unpreconditioned system, we noticed in our experiments that η (or η_A) is usually

α^P	β^P	Stopping criterion
0	0	$\frac{\ M_1^{-1}AM_2^{-1}z_m - M_1^{-1}b\ _2}{\ M_1^{-1}b\ _2}$
0	$\neq 0$	$\frac{\ M_1^{-1}AM_2^{-1}z_m - M_1^{-1}b\ _2}{\beta^P}$
$\neq 0$	0	$\frac{\ M_1^{-1}AM_2^{-1}z_m - M_1^{-1}b\ _2}{\alpha^P \ x_m\ _2}$
$\neq 0$	$\neq 0$	$\frac{\ M_1^{-1}AM_2^{-1}z_m - M_1^{-1}b\ _2}{\alpha^P \ x_m\ _2 + \beta^P}$

Table 1: Stopping criterion for the preconditioned GMRES method

smaller than η^P (or η_A^P). It is therefore recommended to use a larger tolerance for the preconditioned system than one would have used on the unpreconditioned one.

How to choose α , β , α^P and β^P ? Classical choices for α and β that appears in the literature are $\alpha = \|A\|_2$ and $\beta = \|b\|_2$. Similarly, α^P and β^P should be chosen such as $\alpha^P \sim \|M_1^{-1}A\|_2$ and $\beta^P \sim \|M_1^{-1}b\|_2$. Any other choice that reflects the possible uncertainty on the data can also be plugged in. In our implementation, default values are used when the user's input is $\alpha = \beta = 0$ or $\alpha^P = \beta^P = 0$. Table 1 lists the stopping criteria for different choices of α^P and β^P . Similarly, Table 2 explains the output information given to the user on the unpreconditioned linear system on return from GMRES.

2 Implementation of GMRES

2.1 The user interface

For the sake of simplicity and portability, the GMRES implementation is based on the reverse communication mechanism

- for implementing the numerical kernels that depend on the data structure selected to represent the matrix A and the preconditioners,
- for performing the dot products.

This last point has been implemented to allow the use of GMRES in a parallel distributed memory environment, where only the user knows how he has spread his data (we refer to [4] where examples of parallel distributed performances are reported). We have one driver per arithmetic, and we use the BLAS and LAPACK terminology that is

DRIVE_SGMRES for real single precision arithmetic computation,
DRIVE_DGMRES for real double precision arithmetic computation,
DRIVE_CGMRES for complex single precision arithmetic computation,
DRIVE_ZGMRES for complex double precision arithmetic computation.

α	β	Information on the unpreconditioned system
0	0	$\frac{\ Ax_m - b\ _2}{\ b\ _2}$
0	$\neq 0$	$\frac{\ Ax_m - b\ _2}{\beta}$
$\neq 0$	0	$\frac{\ Ax_m - b\ _2}{\alpha \ x_m\ _2}$
$\neq 0$	$\neq 0$	$\frac{\ Ax_m - b\ _2}{\alpha \ x_m\ _2 + \beta}$

Table 2: Stopping criterion for the preconditioned GMRES method

Finally, to hide as much as possible the numerical method from the user, only a few parameters are required by the drivers, whose interfaces are similar for all arithmetics. Below we present the interface for the real double precision driver:

```
CALL DRIVE_DGMRES(N,NLOC,M,LWORK,WORK,IRC,ICNTL,CNTL,INFO,RINFO)
```

- N** is an INTEGER variable that must be set by the user to the order n of the matrix A . It is not altered by the subroutine.
- NLOC** is an INTEGER variable that must be set by the user to the size of the subset of entries of b and x that are allocated to the calling process in a distributed memory environment. For serial or shared memory computers NLOC should be equal to N. It is not altered by the subroutine.
- M** is an INTEGER variable that must be set by the user to the projection size m (restart parameter). This parameter controls the amount of memory required for storing the Krylov basis and the Hessenberg matrix. It is not altered by the subroutine.
- LWORK** is an INTEGER variable that must be set by the user to the size of the workspace WORK. WORK must be greater than or equal to $M*M+M*(NLOC+5)+5*NLOC+1$. It is not altered by the subroutine.
- WORK** is a SINGLE/DOUBLE PRECISION REAL/COMPLEX array of length LWORK. The first NLOC entries contain the initial guess x_0 in input and the computed approximation of the unpreconditioned solution in output. The following NLOC entries contain the right-hand side b of the unpreconditioned system. The remaining entries are used as workspace by the subroutine.
- IRC** is an INTEGER array of length 5 that need not be set by the user. This array controls the reverse communication. Details of the reverse communication management are given in Section 2.2.
- ICNTL** is an INTEGER array of length 7 that contains control parameters that must be set by the user. Details of the control parameters are given in Section 2.3.
- CNTL** is a SINGLE/DOUBLE PRECISION REAL array of length 5 that contains control parameters that must be set by the user. Details of the control parameters are given in Section 2.3.

INFO is an INTEGER array of length 3 which contains information on the reasons of exiting GMRES. Details are given in Section 2.4.

RINFO is a DOUBLE PRECISION REAL array of length 2 which contains the backward errors for the preconditioned and the unpreconditioned linear systems.

2.2 The reverse communication management

The INTEGER array IRC permits to implement the reverse communication. None of its entries need be set by the user.

On each exit, IRC(1) indicates the action that must be performed by the user before invoking the driver again. Possible values of IRC(1) and the associated actions are as follows:

- 0 Normal exit.
- 1 The user must perform the matrix vector product $z \leftarrow Ax$.
- 2 The user must perform the left preconditioning $z \leftarrow M_1^{-1}x$.
- 3 The user must perform the right preconditioning $z \leftarrow M_2^{-1}x$.
- 4 The user must perform one or more scalar products $z \leftarrow X^*y$.

On each exit with IRC(1) > 0, IRC(2) indicates the index in WORK where x should be read and IRC(4) indicates the index in WORK where z should be written.

When IRC(1) = 4, IRC(5) gives the number of scalar products to be performed. In this case, X denotes an array of size NLOC \times IRC(5) stored column-wise. IRC(3) indicates the index in WORK where y should be read. This programming trick permits to realize the dot products with a BLAS 3 routine: this happens when the orthogonalization scheme is either CGS or ICGS. Furthermore, on distributed memory computers, this allows to reduce the number of global synchronization and alleviate the cost of the dot product computations.

2.3 The control parameters

The entries of the array ICNTL control the execution of the DRIVE_GMRES subroutine. All entries of ICNTL are input parameters.

- ICNTL(1) is the stream number for the error messages.
- ICNTL(2) is the stream number for the warning messages.
- ICNTL(3) is the stream number for the convergence history.
- ICNTL(4) controls the type of preconditioning.
- ICNTL(5) determines which orthogonalization scheme to apply.
- ICNTL(6) controls whether the user wishes to supply an initial guess of the solution vector. If ICNTL(6)=0, the initial guess is set to zero.
- ICNTL(7) is the maximum number of iterations allowed.

Possible values for ICNTL(4) are

- 0 no preconditioning,
- 1 left preconditioning,
- 2 right preconditioning,
- 3 double side preconditioning,
- 4 error, default set in the routine INIT_GMRES.

Possible values for ICNTL(5) are

- 0 modified Gram-Schmidt orthogonalization (MGS),
- 1 iterative modified Gram-Schmidt orthogonalization (IMGS),
- 2 classical Gram-Schmidt orthogonalization (CGS),
- 3 iterative classical Gram-Schmidt orthogonalization (ICGS).

The entries of the CNTL array define the tolerance and the normalizing factors (see Section 1.5) that control the execution of the algorithm:

- CNTL(1) is the convergence tolerance for the backward error (see Section 1.5 for details).
- CNTL(2) is the normalizing factor α .
- CNTL(3) is the normalizing factor β .
- CNTL(4) is the normalizing factor α^P .
- CNTL(5) is the normalizing factor β^P .

2.4 The information parameters

Once IRC(1) = 0, the entries of the array INFO explain the circumstances under which GMRES was exited. All entries of INFO are output parameters.

Possible values for INFO(1) are

- 0 normal exit. Convergence has been observed.
- 1 erroneous value $n < 1$.
- 2 erroneous value $m < 1$.
- 3 LWORK too small.
- 4 convergence not achieved after ICNTL(7) iterations.
- 5 preconditioning type not set by user.

If INFO(1) = 0, then INFO(2) contains the number of iterations performed until achievement of the convergence and INFO(3) gives the minimal size for workspace. If INFO(1) = -3, then INFO(2) contains the minimal size necessary for the workspace.

If INFO(1) = 0, then RINFO(1) contains the backward error for the preconditioned linear system and RINFO(2) contains the backward error associated with the unpreconditioned linear system.

2.5 Initialization of the parameters

An initialization routine is available to the user for each arithmetic:

- INIT_SGMRES for real single precision arithmetic computation,
- INIT_DGMRES for double precision arithmetic computation,
- INIT_CGMRES for complex single precision arithmetic computation,
- INIT_ZGMRES for complex double precision arithmetic computation.

These routines set the input control parameters ICNTL and CNTL defined above to default values. The generic interface is

```
CALL INIT_GMRES(ICNTL,CNTL)
```

The default value for

ICNTL(1) is 6,
 ICNTL(2) is 6,
 ICNTL(3) is 0: no convergence history,
 ICNTL(4) is 4: erroneous value. The user must specify explicitly the type of preconditioning,
 ICNTL(5) is 0: MGS is used,
 ICNTL(6) is 0: default initial guess $x_0 = 0$,
 ICNTL(7) is -1: the user must specify explicitly the maximum number of iterations,
 CNTL(1) is 1,
 CNTL(2) is 0,
 CNTL(3) is 0,
 CNTL(4) is 0,
 CNTL(5) is 0.

3 Availability of the software

The code is written in FORTAN 77 and makes calls to BLAS routines, as indicated in Table 3. The code is free for non-commercial use only. The source code is available from the WEB at the URL

<http://www.cerfacs.fr/algor/>

together with the software license agreement.

Simple precision		Double precision	
real	complex	real	complex
SDOT	CDOTC	DDOT	ZDOTC
SSCAL	CSSCAL	DSCAL	ZDSCAL
SAXPY	CAXPY	DAXPY	ZAXPY
SGEMV	CGEMV	DGEMV	ZGEMV
SCNRM2	SCNRM2	DNRM2	DZNRM2
SCOPY	CCOPY	DCOPY	ZCOPY
STRSV	CTRSV	DTRSV	ZTRSV

Table 3: BLAS routines called in GMRES

4 An example of use

```

%
  program validation
*
  integer lda, ldstrt, lwork
  parameter (lda = 1000, ldstrt = 60)
  parameter (lwork = ldstrt**2+ldstrt*(lda+5)+5*lda+1)
  integer i, j, n, m
  integer revcom, colx, coly, colz, nbscal
  integer irc(5), icntl(7), info(3)
*
  integer matvec, preconditionLeft, preconditionRight
  parameter(matvec=1, preconditionLeft=2)
  parameter(preconditionRight=3, dotProd=4)
*
  complex*16 a(lda,lda), work(lwork)
  real*8 cntl(5), rinfo(2)
*
  complex*16 ZERO, ONE
  parameter (ZERO = (0.0d0, 0.0d0), ONE = (1.0d0, 0.0d0))
*
  complex*16 zdotc
  external zdotc
*
  * Initialize the matrix
*
  ....
*
  * Set the right-hand side b such that b_i = 1+sqrt(-1)
  do i = 1,n
    work(i+n) = (1.d0,1.d0)
  enddo
*
  *****
  * Initialize the control parameters to default values
  *****
  call init_zgmres(icntl,cntl)
*
  *****
*
  * Tune some parameters for GMRES
  *****
  *
  * Tolerance
  cntl(1) = 1.d-10
  * Save the convergence history in file fort.20
  icntl(3) = 20
  * No preconditioning
  icntl(4) = 1
  * ICGS orthogonalization
  icntl(5) = 3
  * Maximum number of iterations
  icntl(7) = 100
  *
  *****
  * reverse communication implementation
  *****
  *
  10 call drive_zgmres(n,n,m,lwork,work,irc,icntl,cntl,info,rinfo)
  revcom = irc(1)
  colx = irc(2)
  coly = irc(3)
  colz = irc(4)
  nbscal = irc(5)
  *
  if (revcom.eq.matvec) then
  * perform the matrix vector product
  * work(colz) <-- A * work(colx)
  call zgemv('N',n,n,ONE,a,lda,work(colx),1,
  & ZERO,work(colz),1)
  goto 10
  *
  else if (revcom.eq.preconditionLeft) then
  * perform the left preconditioning
  * work(colz) <-- M_1^{-1} * work(colx)
  call zcopy(n,work(colx),1,work(colz),1)
  goto 10
  *

```

```

        else if (revcom.eq.precondRight) then
* perform the right preconditioning
* work(colz) <-- M_2^{-1} * work(colx)
    call zcopy(n,work(colx),1,work(colz),1)
    goto 10
*
        else if (revcom.eq.dotProd) then
* perform the scalar product
* work(colz) <-- work(colx) work(coly)
*
* The statement to perform the dot products can be
* written in a compact form.
*   call zgemv('C',n,nbscal,ONE,work(colx),n,
*   &           work(coly),1,ZERO,work(colz),1)
* For the sake of simplicity we write it as a do-loop
    do i=0,nbscal-1
        work(colz+i) = zdotc(n,work(colx+i*n),1,
&                       work(coly),1)
    enddo
    goto 10
*
endif
*
if (info(1).eq.0) then
    write(*,*) ' Normal exit'
    write(*,*) ' Convergence after ', info(2),
&             ' iterations'
    write(*,*) ' Backward error - preconditioned
&             system', rinfo(1)

```

```

    write(*,*) ' Backward error - unpreconditioned
&             system', rinfo(2)
    write(*,*) ' Solution : '
    do j=1,n
        write(*,*) work(j)
    enddo
    write(*,*) ' Optimal size for workspace ', info(3)
else if (info(1).eq.-1) then
    write(*,*) ' Bad value of n'
else if (info(1).eq.-2) then
    write(*,*) ' Bad value of m'
else if (info(1).eq.-3) then
    write(*,*) ' Too small workspace. '
    write(*,*) ' Minimal value should be ', info(2)
else if (info(1).eq.-4) then
    write(*,*) ' No convergence after ', icntl(7),
&             ' iterations'
else if (info(1).eq.-5) then
    write(*,*) ' Type of preconditioner not specified'
endif
*
*****
* dump the solution on a file
*****
    ....
*
    stop
end

```

Acknowledgments

The authors wish to thank Hatim Kharraz-Aroussi from ENSIAS (Rabat, Morocco) for his implementation of ICGS during his training period at CERFACS.

References

- [1] M. Arioli, I. S. Duff, and D. Ruiz. Stopping criteria for iterative solvers. *SIAM J. Matrix Anal. Appl.*, 13:138–144, January 1992.
- [2] Å. Björck. Numerics of Gram-Schmidt orthogonalization. *Linear Algebra Appl.*, 197–198:297–316, 1994.
- [3] F. Chaitin-Chatelin and V. Frayssé. *Lectures on Finite Precision Computations*. SIAM, Philadelphia, 1996.
- [4] V. Frayssé, L. Giraud, and H. Kharraz-Aroussi. On the influence of the orthogonalization scheme on the parallel performance of GMRES. Tech. Rep. TR/PA/98/07, CERFACS, Toulouse, France, 1998. Submitted to EUROPAR’98.
- [5] G. H. Golub and C. Van Loan. *Matrix computations*. Johns Hopkins University Press, 1989. Second edition.
- [6] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.
- [7] J. N. Shadid and R. S. Tuminaro. A comparison of preconditioned nonsymmetric Krylov methods on a large-scale MIMD machine. *SIAM J. Sci. Comp.*, 14(2):440–459, 1994.
- [8] J. H. Wilkinson. *Rounding errors in algebraic processes*, volume 32. Her Majesty’s stationery office, London, 1963.