A Set of GMRES Routines for Real and Complex Arithmetics on High Performance Computers

Valérie Frayssé [‡] Luc Giraud [§] Serge Gratton [§] Julien Langou [§]

CERFACS Technical Report TR/PA/03/3

This report is the users' guide for the new release and supersedes 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 portability, simplicity, flexibility and efficiency the GMRES solvers have been implemented in Fortran 77 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. This report is the users' guide for the release 2.0 of the package and supersedes [10].

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

1 Introduction

The purpose of this report is to present the API (Application Program Interface) of the GMRES routines and to describe several choices that have been made in order to get an efficient and reliable implementation of the GMRES [16] algorithm suitable for real and complex arithmetic on any scientific computer.

This report is organized as follows. In Section 2 we briefly describe the GMRES method and present the key ingredients on whose we paid a particular attention. In Section 3 we detail the API and explain the meaning of the control parameters enabling the users to select any particular feature of the code. We also indicate what are the output generated by the code as warning and error. We end with a short example that illustrates the use of the reverse communication mechanism and the typical feed-back that is given to the user.

[‡]Wallaware Inc., 146 Smith Street #2, Boston MA 02120, U.S.A. Email: `valerie@spydre.com`. Part of this work was performed while the author was a researcher at CERFACS

[§]CERFACS, 42 av. Gaspard Coriolis, 31057 Toulouse Cedex, France. Email : `giraud, gratton, langou@cerfacs.fr`

2 The GMRES algorithm

2.1 General description

The Generalized Minimum RESidual (GMRES) method was proposed by Saad and Schultz in 1986 [16] in order to solve large, sparse and non Hermitian linear systems. GMRES belongs to the class of Krylov based iterative methods.

For the sake of generality we describe this method for linear systems that are complex, everything also extends to real arithmetic calculation. 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 $x_0 \in \mathbb{C}^n$ 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

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

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

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

and where y belongs to $\mathbb{C}^m$. 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 $H_m = V_m^H A V_m$ of order m. The Arnoldi process satisfies the relationship

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^H, \tag{3}$$

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

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

where

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

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

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 a matrix with orthonormal columns, the residual norm $\|r_m\|_2 = \left\|\beta e_1 - \bar{H}_m y\right\|_2$ is minimal when y solves the linear least-squares problem

$$\min_{y \in \mathbb{C}^m} \left\|\beta e_1 - \bar{H}_m y\right\|_2 . \tag{4}$$

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 that is its key feature as it ensures the decrease of the residual norm.

In exact arithmetic, GMRES converges in at most n steps. However, in practice, n can be very large and the storage of the orthonormal 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. $x_0 = x_m$ to comply with the monotonicity property even when restarting. 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. One possible benefit of using restarted GMRES is that it alleviates the cost of the orthogonalization procedure that can becomes very much time consuming when the size of the Krylov space becomes large.

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,
- the stopping criteria for the iterative scheme, and
- the calculation of the residual at the restart.

2.2 The least-squares problem

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

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

$$y_j = R(1:j, 1:j)^{-1} R(1:j, j+1). \tag{5}$$

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

$$\|r_j\|_2 = \|b - Ax_j\|_2 = \left\|\beta e_1 - \bar{H}_j y_j\right\|_2 = |r_{j+1,j+1}| . \tag{6}$$

Therefore, the norm of the residual of the linear system is the by product value of the algorithm and can be obtained without explicitly evaluating the residual vector.

QR factorization of upper Hessenberg matrices can be efficiently performed using Givens rotations, because they enable to zero out sequentially all elements $\bar{H}_{k+1,k}$, $k = 1, \ldots j$. However,

since $[\bar{H}_{j+1},\ \beta e_1]$ is obtained from $[\bar{H}_j,\ \beta e_1]$ by adding one column c, the R factor R_{j+1} of $[\bar{H}_{j+1},\ \beta e_1]$ is obtained by updating the R factor R_j of $[\bar{H}_j,\ \beta e_1]$ using an algorithm that we briefly outline now, for $j = 3$ (see [3, 5, 6]):

1. Let
$$R_j = \begin{pmatrix} + & + & + & + \\ 0 & + & + & + \\ 0 & 0 & + & + \\ 0 & 0 & 0 & + \end{pmatrix}$$
and $Q_k \in C^{(j+1)\times(j+1)}$ be such that $[\bar{H}_j,\ \beta e_1] = Q_k R_k$. The matrix Q_k is not explicitly computed, only the sine and cosine of the Givens rotations are stored. The vector $w = Q_k^H c$ is then computed by applying the stored Givens rotations, and w is inserted in between the j and $j+1$ columns of R_k, to yield
$$\widetilde{R}_j = \begin{pmatrix} + & + & + & * & + \\ 0 & + & + & * & + \\ 0 & 0 & + & * & + \\ 0 & 0 & 0 & * & + \\ 0 & 0 & 0 & * & 0 \end{pmatrix}$$

2. A Givens rotation that zeros element $\widetilde{R}_j(j+2, j+1)$ is computed and applied to $\widetilde{R}_j$ to produce the matrix
$$R_{j+1} = \begin{pmatrix} + & + & + & + & + \\ 0 & + & + & + & + \\ 0 & 0 & + & + & + \\ 0 & 0 & 0 & + & + \\ 0 & 0 & 0 & 0 & + \end{pmatrix}$$

The computation of the sine and cosine involved in the givens QR factorization use the BLAS routines *ROTG, and we refer the reader to [3, 6] for questions related to the reliability of these transformations.

2.3 Computation of V_j

The orthogonality quality of the V_j plays a central role in GMRES as deteriorating it might slow down or delay the convergence. On the other hand, ensuring a very good orthogonality might be expensive and useless for some applications. Consequently a trade-off has to be found to balance the numerical efficiency of the orthogonalization scheme and its inherent efficiency on a given target computer.

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 basis; this loss can be compensated by selectively iterating the orthogonalization scheme [4, 13]. 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 efficient manner by gathering the dot products into one matrix-vector product, but it is well-known that CGS is numerically worse than MGS. However, CGS with selective reorthogonalization (ICGS) results in an algorithm of the same numerical quality as IMGS. Therefore, ICGS is particularly attractive in a parallel distributed environment, where the global reduction involved in the computation of the dot product is a well-known bottleneck [1, 9, 12, 17].

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. We follow [14] to define the criterion for the selective reorthogonalization and set $K = \sqrt{2}$ as suggested by [8] as the value for the threshold.

2.4 Preconditioning

The convergence of GMRES or GMRES(m) to solve (1) might be slow. To overcome this drawback, one often prefer to solve a transformed linear system that is referred to as the preconditioned linear system. More precisely if $A \approx M_1 M_2$ we actually solve the linear system

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

with $x = M_2^{-1} z$. In our implementation we allow the user to select left and/or right preconditioning. The use of preconditioners has some consequences on the stopping criterion. We discuss these points in the next paragraph.

2.5 Stopping criteria

We have chosen to base our stopping criterion on the normwise backward error [7]. The backward error analysis, introduced by Givens and Wilkinson [18], 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).

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 [7]. 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 [2].

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

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

is called the *normwise backward error* associated with x_j. 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

$$\eta_{A,j} = \frac{|r_{j+1,j+1}|}{\alpha \|x_j\|_2 + \beta}$$

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 $\eta_{A,j}$ as the stopping criterion. Our strategy is the following: first we iterate until $\eta_{A,j}$ becomes lower than the tolerance, then afterwards, we iterate until η_j becomes itself lower than the tolerance. We hope in this way to minimize the number of explicit residual computations (involving the computation of matrix-vector product) necessary to evaluate η_j, while still 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):

$$\eta_j^P = \left\|M_1^{-1}AM_2^{-1}z_j - M_1^{-1}b\right\|_2 / (\alpha^P \left\|x_j\right\|_2 + \beta^P)$$

with $x_j = M_2^{-1}z_j$. We denote by

$$\eta_{A,j}^P = \frac{|r_{j+1,j+1}|}{\alpha^P \left\|x_j\right\|_2 + \beta^P}$$

the stopping criterion for the preconditioned GMRES. As previously, we stop the iterations when the computed values of $\eta_{A,j}^P$ and then η_j^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.

We should notice that for right preconditioner $\eta = \eta^P$ (or $\eta_A = \eta_A^P$); this is the reason why right preconditioning is often preferred in many applications. Otherwise there is a priori no order between the backward error of the preconditioned system and that of the unpreconditioned system. Nevertheless, we noticed in our experiments that η (or η_A) is usually 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 $\alpha^P \sim \left\|M_1^{-1}A\right\|_2$ and $\beta^P \sim \left\|M_1^{-1}b\right\|_2$. 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.

2.6 Computation of the residual at restart

In some applications, the computation of each matrix-vector product can be extremely expensive as for instance in some domain decomposition technique or in electromagnetism when a fast multipole expansions used to evaluate the matrix-vector product. In that case, one would like to avoid the explicit calculation of the residual at each restart of GMRES. Since we then set $x_0 = x_m$, we have

α^P	β^P	Stopping criterion
0	0	$\frac{\left\|M_1^{-1}AM_2^{-1}z_j - M_1^{-1}b\right\|_2}{\left\|M_1^{-1}b\right\|_2}$
0	$\neq 0$	$\frac{\left\|M_1^{-1}AM_2^{-1}z_j - M_1^{-1}b\right\|_2}{\beta^P}$
$\neq 0$	0	$\frac{\left\|M_1^{-1}AM_2^{-1}z_j - M_1^{-1}b\right\|_2}{\alpha^P \left\|x_j\right\|_2}$
$\neq 0$	$\neq 0$	$\frac{\left\|M_1^{-1}AM_2^{-1}z_j - M_1^{-1}b\right\|_2}{\alpha^P \left\|x_j\right\|_2 + \beta^P}$

Table 1: Stopping criterion for the preconditioned GMRES method.

α	β	Information on the unpreconditioned system
0	0	$\frac{\left\|Ax_j - b\right\|_2}{\left\|b\right\|_2}$
0	$\neq 0$	$\frac{\left\|Ax_j - b\right\|_2}{\beta}$
$\neq 0$	0	$\frac{\left\|Ax_j - b\right\|_2}{\alpha \left\|x_j\right\|_2}$
$\neq 0$	$\neq 0$	$\frac{\left\|Ax_j - b\right\|_2}{\alpha \left\|x_j\right\|_2 + \beta}$

Table 2: Stopping criterion for the unpreconditioned GMRES method.

$r_0 = b - Ax_m$ with $x_m = x_0 + V_m y$. We can then observe that

$$\begin{aligned} r_0 &= b - A(x_0 + V_m y_m) \\ &= V_{m+1}(\beta e_1 - \bar{H} y_m) \\ &= V_{m+1} Q_m (Q_m^H \beta e_1 - \begin{bmatrix} R(1:m, 1:m) \\ 0 \end{bmatrix} y_m) \\ &= V_{m+1} Q_m \begin{bmatrix} 0 \\ r_{m+1,m+1} \end{bmatrix} \end{aligned}$$

It follows that the calculation of the residual amounts at computing a linear combination of the $(m+1)$ Arnoldi's vectors. The coefficients of the linear combination are computed by applying the Givens rotations in the reverse order to the vector which has all its entries equal to zero but the last that is equal to $r_{m+1,m+1}$. This non-zero value is a by product of the solution of the least-square. This calculation of the residual requires $n(2m+1) + 2m$ floating point operations (flops) and should be preferred to an explicit calculation if the matrix-vector product involving A implies more than $2n(m+1)$ flops. We should mention that in some circumstances, for instance when the required backward error is close to the machine precision, the use of this trick might slightly delay the convergence (while it might still enable us to get the solution in a shorter period of time). Notice that the implementation of this trick requires to store $(m+1)$ Arnoldi's vectors, while only m have to be stored otherwise. For the sake of robustness, even if this calculation of the residual is selected by the user, we enforce an explicit residual calculation if, in the previous restart, the convergence was detected by $\eta_{A,j}^P$ but not assessed by η_j^P.

3 Implementation of GMRES

3.1 The user interface

For the sake of simplicity and portability, the GMRES implementation is developed in Fortran 77 and 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 the data have been spread (we refer to [12] where examples of parallel distributed performance 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.

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 except if it was set by the user to a value larger than N, it is then set to N; or to a value too large for LWORK. In that latter case, it is set to the maximum possible value permitted by LWORK.

LWORK is an INTEGER variable that must be set by the user to the size of the workspace WORK. LWORK must be greater than or equal to
if ICNTL(5) = 0 or 1
M*M+M*(NLOC+5)+5*NLOC+2 if ICNTL(8) = 1; M*M+M*(NLOC+5)+6*NLOC+2 otherwise.
if ICNTL(5) = 2 or 3
M*M+M*(NLOC+5)+5*NLOC+M+1 if ICNTL(8) = 1; M*M+M*(NLOC+5)+6*NLOC+M+1 otherwise.
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 3.2.

ICNTL is an INTEGER array of length 8 that contains control parameters that must be set by the user. Details of the control parameters are given in Section 3.3.

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 3.3.

INFO is an INTEGER array of length 3 which contains information on the reasons of exiting GMRES. Details are given in Section 3.4.

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

3.2 The reverse communication management

The INTEGER array IRC permits to implement the reverse communication. None of its entries must 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 (ie with a leading dimension equal to NLOC). IRC(3) indicates the index in WORK where y should be read. This programming trick permits to realize the parallel dot products with a BLAS 2 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 synchronizations introduced by the reduction and it alleviates the cost of the parallel dot product computations.

3.3 The control parameters

The entries of the array ICNTL control the execution of the DRIVE_GMRES subroutine. All entries of ICNTL are input parameters and some of them have a default value set by the routine INIT_GMRES.

ICNTL(1) is the stream number for the error messages **(Default is 6)**.
Must be a strictly positive value.

ICNTL(2) is the stream number for the warning messages **(Default is 6)**.
Must be greater of equal to zero. A zero value implies that the warning messages will not be displayed.

ICNTL(3) is the stream number for the convergence history **(Default is 0)**.
Must be greater of equal to zero. A zero value implies that the convergence history will not be displayed.

ICNTL(4) controls the location of preconditioning. **(No Default: Must be set by the user).**

ICNTL(5) determines which orthogonalization scheme to apply **(Default is 0, i.e. MGS)**.

ICNTL(6) controls whether the user wishes to supply an initial guess of the solution vector. **(Default is 0)**.
Must be equal to either 0 or 1. If ICNTL(6)=0, the initial guess is set to zero.

ICNTL(7) is the maximum number of iterations (cumulated over the restarts) allowed. **(No Default: Must be set by the user)**.
Must be equal larger than 0.

ICNTL(8) controls the strategy to compute the residual at the restart (see Section 2.6 for details). **(Default is 1)**.
Must be equal to either 0 or 1.

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) **(Default)**,
1 iterative selective modified Gram-Schmidt orthogonalization (IMGS),
2 classical Gram-Schmidt orthogonalization (CGS),

3	iterative selective classical Gram-Schmidt orthogonalization (ICGS).

Possible values for ICNTL(8) are

0	A recurrence formula is used to compute the residual at each restart, except if the convergence was detected using the Arnoldi residual during the previous restart.
1	The residual is explicitly computed using a matrix vector product **(Default)**.

The entries of the CNTL array define the tolerance and the normalizing factors (see Section 2.5) that control the execution of the algorithm:

CNTL(1)	is the convergence tolerance for the backward error (see Section 2.5 for details). **(Default is** 10^{-5}**)**. Must be greater of equal to zero.
CNTL(2)	is the normalizing factor α. **(Default is 0)**. Must be greater of equal to zero.
CNTL(3)	is the normalizing factor β. **(Default is 0)**. Must be greater of equal to zero.
CNTL(4)	is the normalizing factor α^P. **(Default is 0)**. Must be greater of equal to zero.
CNTL(5)	is the normalizing factor β^P. **(Default is 0)**. Must be greater of equal to zero.

Default values are used when the user's input is $\alpha^P = \beta^P = 0$, $\alpha = \beta = 0$; that is $\beta^P = \left\|M_1^{-1}b\right\|_2$, $\beta = \|b\|_2$ respectively.

3.4 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.

3.5 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,
ICNTL(8)	is 1: the residual is explicitly computed at each restart,
CNTL(1)	is 10^{-7},
CNTL(2)	is 0,
CNTL(3)	is 0,
CNTL(4)	is 0,
CNTL(5)	is 0.

3.6 Automatic correction for invalid parameters

To avoid to exit with an error when some parameters have been wrongly set by the user, we try as much as possible to correct them and generate a warning message in the warning stream. Such a situation might occur when:

M	is set to a value larger than N, we set it to N.
LWORK	is too small for the required M, we then compute the largest possible value of M enables for that size of the workspace. If M is lower than 1, we exit with an error.
ICNTL(5)	is set to an invalid value, we set it back to the default.
ICNTL(6)	is set to an invalid value, we set it back to the default.
ICNTL(7)	is set to an invalid value, we set it to N.
ICNTL(8)	is set to an invalid value, we set it back to the default.

3.7 Unrecoverable invalid parameters

For some invalid values of the input parameters we cannot guess what could be an relevant alternative and consequently we output an error message and return to the calling programme. Such a situation might occur when:

N	is set to a value smaller than 1.
M	is set to a value smaller than 1.
LWORK	is too small to enable to perform any GMRES iteration.
ICNTL(4)	does not correspond to any preconditioner alternative.

4 Availability of the software

For sake of maintenance of the code, only one source file exists and is used to generate the source code for each of the four arithmetics. The final code is written in Fortran 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 and a set of example codes. We should also mention

Simple precision		Double precision	
real	complex	real	complex
SAXPY	CAXPY	DAXPY	ZAXPY
SNRM2	SCNRM2	DNRM2	DZNRM2
SCOPY	CCOPY	DCOPY	ZCOPY
SGEMV	CGEMV	DGEMV	ZGEMV
SROT	CROT	DROT	ZROT
SROTG	CROTG	DROTG	ZROTG
STRSV	CTRSV	DTRSV	ZTRSV

Table 3: BLAS routines called in GMRES.

that a free implementation of Flexible GMRES [11, 15] is also available at the same URL address.

5 The API change since previous version of the package

The **only** change for the users of the previous version is that the integer array control `ICNTL` is now of length 8, while it was 7 before.

6 An example of use

6.1 An example of call

```
%
      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(8), info(3)
*
      integer matvec, precondLeft, precondRight
      parameter(matvec=1, precondLeft=2)
      parameter(precondRight=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.precondLeft) 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
```

6.2 An example of output

In the example below all the output stream were set to the screen

```
guinness% sTestgmres
 *********************************************
 This code is an example of use of GMRES
 in single precision arithmetic
 Results are written in output files
 fort.10 and sol_sTestgmres.
 *********************************************

 Matrix size <   1000
900
 Restart  <  60
4

  WARNING GMRES :
        For M =   4 optimal value
        for LWORK =    9037

                    CONVERGENCE HISTORY FOR GMRES

Errors are displayed in unit:  6
Warnings are displayed in unit:  6
Matrix size:     900
Local matrix size:     900
Restart:       4
Left and right preconditioning
Iterative classical Gram-Schmidt
Default initial guess x_0 = 0
True residual computed at restart
Maximum number of iterations:   100
Tolerance for convergence: 0.10E-06
Backward error on the unpreconditioned system Ax = b:
    the residual is normalised by ||b||
Backward error on the preconditioned system (P1)A(P2)y = (P1)b:
     the preconditioned residual is normalised by ||(P1)b||
Optimal size for the workspace:   9037

Convergence history: b.e. on the preconditioned system
 Iteration   Arnoldi b.e.    True b.e.
    1           0.33E-01         --
    2           0.29E-02         --
    3           0.37E-03         --
    4           0.45E-04         --
    5           0.61E-05         --
    6           0.77E-06         --
    7           0.11E-06         --
    8           0.15E-07       0.49E-07

Convergence achieved
B.E. on the preconditioned system:   0.49E-07
B.E. on the unpreconditioned system: 0.65E-07
info(1) =  0
Number of iterations (info(2)):     8
```

References

[1] R. B. Lehoucq A. G. Salinger. Large-scale eigenvalue calculations for stability analysis of steady flows on massively parallel computers. *Int. J. Numerical Methods in Fluids*, 36:309–327, 2001.

[2] M. Arioli, I. S. Duff, and D. Ruiz. Stopping criteria for iterative solvers. *SIAM J. Matrix Anal. Appl.*, 13:138–144, January 1992.

[3] D. Bindel, J. Demmel, W. Kahan, and O. Marques. On computing givens rotations reliably and efficiently. *ACM Transactions on Mathematical Software (TOMS)*, 28(2):206–238, June 2002.

[4] Å. Björck. Numerics of Gram-Schmidt orthogonalization. *Linear Algebra Appl.*, 197–198:297–316, 1994.

[5] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[6] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of Basic Linear Algebra Subprograms (BLAS). *ACM Transactions on Mathematical Software (TOMS)*, 28(2):135–151, June 2002.

[7] F. Chaitin-Chatelin and V. Frayssé. *Lectures on Finite Precision Computations*. SIAM, Philadelphia, 1996.

[8] W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart. Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization. *Math. Comp.*, 30:772–795, 1976.

[9] J. Frank and C. Vuik. Parallel implementation of a multiblock method with approximate subdomain solution. *Appl. Num. Math.*, 30:403–423, 1999.

[10] V. Frayssé, L. Giraud, and S. Gratton. A set of GMRES routines for real and complex arithmetics. Tech. Rep. TR/PA/97/49, CERFACS, 1997.

[11] V. Frayssé, L. Giraud, and S. Gratton. A set of Flexible-GMRES routines for real and complex arithmetics. Technical Report TR/PA/98/20, CERFACS, 1998.

[12] 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. Preliminary version of the paper published in the proceedings of EuroPar'98, *Lecture Notes in Computer Science, Springer-Verlag, vol. 1470, pp. 751-762.*

[13] W. Hoffmann. Iterative algorithms for Gram-Schmidt orthogonalization. *Computing*, 41:335–348, 1989.

[14] H. Rutishauser. Description of algol 60. handbook for automatic computation. *Springer Verlag, Berlin*, 1.a, 1967.

[15] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14:461–469, 1993.

[16] 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.

[17] 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.

[18] J. H. Wilkinson. *Rounding errors in algebraic processes*, volume 32. Her Majesty's stationery office, London, 1963.