

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

Valérie Frayssé^{‡§} Luc Giraud ^{*¶} Serge Gratton ^{*}

CERFACS Technical Report TR/PA/06/09

This report is the users' guide for the new release and supersedes TR/PA/98/07

Abstract

In this report we describe our implementations of the FGMRES 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 FGMRES 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. Furthermore, either implicit or explicit calculation of the residual at restart are possible depending on the actual cost of the matrix-vector product. Finally the implemented stopping criterion is based on a normwise backward error.

1 Introduction

The FGMRES (Flexible Generalized Minimum Residual) method [27] is among the most widely used Krylov solvers for the iterative solution of general large linear systems when variable preconditioning is considered. This numerical algorithm is embedded in many sophisticated package products that are either specialized packages for the solution of PDEs [1, 32] or general purpose packages for the solution of sparse linear systems [21]. Using these sophisticated packages requires to comply with a predefined data structure (that is package-dependent); furthermore many of these packages do not support all the arithmetics and do not offer a wide range of orthogonalisation schemes. Even though a basic version

[‡]CERFACS, Toulouse, France

[§]Current address: Kvasar Technology LLC, Boston, USA

[¶]Current address: ENSEEIHT-IRIT, Toulouse, France

of this numerical algorithm is fairly simple to describe and to implement, we did not find any general implementation for complex matrices when we first looked for it a few years ago. This is the main reason why we decided to develop our own implementation with the objectives of having a parallel portable package with a simple API (Application Program Interface) easily understandable by non specialists in linear algebra, while incorporating all the features enabling one to play with the different possible variants for orthogonalization schemes, restarting strategies and stopping criteria. Four variants of the Gram-Schmidt orthogonalization procedure are implemented, that permit the user to select the best suited variant depending on the target parallel computer and the conditioning of the linear system. We made the dot product calculation available in reverse communication to comply with parallel distributed memory usages where this calculation generally requires a global reduction. In many cases, because of the limited amount of memory or because the cost of the orthogonalization becomes prohibitive, a restart should be performed and a new initial residual should be computed. To address the situations where the matrix-vector product is cheap, as in finite element calculation for instance, or expensive, as in boundary element calculation where the matrix is dense and is often approximated via fast multipole technique, the user can select either an explicit or an implicit calculation of the residual. As suggested in [27] we implement a mechanism enabling us to release a contiguous part of the workspace so that the user might use it to implement the preconditioning step. For instance, the user can implement a few step of restarted GMRES [28] where the size of the restart in set according to the amount of memory available at each step. Finally, the stopping criterion is a key component for the iterative solvers. We have decided to enable the user to choose among all the possible criteria based on normwise backward error in the euclidean norm that are commonly admitted as relevant for iterative solvers [2, 31]. The purpose of this paper is to present the API of the FGMRES routines and to describe several choices that have been made in order to get an efficient and reliable implementation suitable for real and complex arithmetic on any scientific computer. The package has first been made available in the public domain in 1998 and has been so far downloaded more than 900 times both by academic and research institutions and by industrial companies. The most popular arithmetics are the double real and the double complex. This latter seems to be fairly popular in the scientific community of wave propagation and convection diffusion problems as illustrated by the list of papers in that field that reference our package [3, 9, 10, 22, 23, 24, 25, 34]. The selection of Fortran 77 as programming language was mainly governed by the sake of portability and interoperability with other languages. Many users using more sophisticated languages like C or C++ have, at the very most, written a wrapper to encapsulate our solvers in their software. Finally, to illustrate the success of our package we should point out that it has been appearing for many months in the top 10 items returned by Google when a search is performed on the word "FGMRES". Nevertheless, we indicate that our FGMRES package is slightly less popular than our GMRES package [14, 15]. We believe that a main reason is that GMRES is often used while FGMRES should be (because the pre-

conditioner is varied at each step, that is the preconditioning step is computed by solving iteratively an auxiliary problem for instance). However, GMRES still provide the user with a satisfactory solution thanks to its robustness to inaccurate matrix-vector product recently studied in [8, 30, 33, 17].

2 The FGMRES algorithm

2.1 General description

The Generalized Minimum RESidual (GMRES) method was proposed by Saad and Schultz in 1986 [28] in order to solve large, sparse and non Hermitian linear systems. GMRES belongs to the class of Krylov based iterative methods. In 1993, Saad [27] introduced a variant of the GMRES method with right preconditioning that enables the use of a different preconditioner at each step of the Arnoldi process.

For the sake of clarity and readability we first describe the GMRES algorithm with right preconditioner and derive the FGMRES algorithm. Furthermore, for the sake of generality we describe this method for linear systems that are complex, everything also specialises to real arithmetic calculation. Let A be a square nonsingular $n \times n$ complex matrix, and b be a complex vector of length n , that define the linear system

$$Ax = b. \quad (1)$$

Let M be a square nonsingular $n \times n$ complex matrix, we define the right preconditioned linear system

$$AMt = b, \quad (2)$$

where $x = Mt$ is the solution of the unpreconditioned linear system. Let $t_0 \in \mathbb{C}^n$ be an initial guess for this linear system and $r_0 = b - AMt_0$ be its corresponding residual.

The GMRES algorithm builds an approximation of the solution of (2) of the form

$$t_m = t_0 + V_m y \quad (3)$$

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

$$\mathcal{K}_m = \text{span} \{r_0, AMr_0, \dots, (AM)^{m-1}r_0\},$$

and where y belongs to \mathbb{C}^m . The vector y is determined so that the 2-norm of the residual $r_m = b - AMt_m$ 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 $H_m = V_m^H AV_m$ of order m . The Arnoldi process satisfies the relationship

$$A[Mv_1, \dots, Mv_m] = AMV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^H, \quad (4)$$

where e_m is the m^{th} canonical basis vector. Equation (4) can be rewritten as

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

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 defined by Equation (3) verifies

$$\begin{aligned} r_m &= b - AMt_m = b - AM(t_0 + V_m y) \\ &= r_0 - AMV_m y = r_0 - V_{m+1}\bar{H}_m y \\ &= \beta v_1 - V_{m+1}\bar{H}_m y \\ &= V_{m+1}(\beta e_1 - \bar{H}_m y). \end{aligned} \tag{5}$$

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

$$\min_{y \in \mathbb{C}^m} \|\beta e_1 - \bar{H}_m y\|_2. \tag{6}$$

We will denote by y_m the solution of (6). Therefore, $t_m = t_0 + V_m y_m$ is an approximate solution of (2) for which the residual is minimal over \mathcal{K}_m . The GMRES method owes its name to this minimization property that is its key feature as it ensures the decrease of the residual norm. We depict in Algorithm 1 the sketch of the Modified Gram-Schmidt (MGS) variant of the GMRES method with right preconditioner.

In practice, 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 computes the sequence of approximate solutions t_j until t_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. $t_0 = t_m$ to insure the monotonicity property of the residual norms 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 become very time consuming when the size of the Krylov space becomes large.

If the preconditioner involved at step 6 in Algorithm 1 varies at each step, we can still write an equality similar to (4). This equality writes:

$$\begin{aligned} A[M_1 v_1, \dots, M_k v_m] &= A[z_1, \dots, z_m] \\ &= AZ_m \\ &= V_m H_m + h_{m+1,m} v_{m+1} e_m^H \\ &= V_m \bar{H}_m, \end{aligned}$$

Algorithm 1 Right preconditioned GMRES

```
1: Choose a convergence threshold  $\varepsilon$ 
2: Choose an initial guess  $t_0$ 
3:  $r_0 = b - AMt_0 = b$ ;  $\beta = \|r_0\|$ 
4:  $v_1 = r_0/\|r_0\|$ ;
5: for  $k = 1, 2, \dots$  do
6:    $w = AMv_k$ ;
7:   for  $i = 1$  to  $k$  do
8:      $h_{i,k} = v_i^T w$ 
9:      $w = w - h_{i,k}v_i$ 
10:  end for
11:   $h_{k+1,k} = \|w\|$ 
12:   $v_{k+1} = w/h_{k+1,k}$ 
13:  Solve the least-squares problem  $\min \|\beta e_1 - \bar{H}_k y\|$  for  $y$ 
14:  Exit if the convergence is detected
15: end for
16: Set  $x_m = M(t_0 + V_m y)$ 
```

which enables us to get a relation similar to (5). Using $x_m = x_0 + Z_m y$ we have

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

where y is the solution of a least-squares problem similar to (6). Because this GMRES variant allows for flexible preconditioners it is referred to as Flexible GMRES. From an implementation point of view the main difference between right preconditioned GMRES and FGMRES is the memory requirement. In that latter algorithm, both V_k and Z_k need to be stored. Furthermore, while only happy breakdowns might occur in GMRES, FGMRES can break at some step before it has computed the solution. We describe the MGS variant of this method in Algorithm 2 and refer to [27] for a complete description of the convergence theory.

In the following paragraphs, we highlight the main key-points for FGMRES:

- the solution of the least-squares problem (6),
- the construction of the orthonormal basis V_m ,
- the stopping criteria for the iterative scheme, and
- the calculation of the residual at the restart.

Algorithm 2 Flexible GMRES

- 1: Choose a convergence threshold ε
 - 2: Choose an initial guess x_0
 - 3: $r_0 = b - Ax_0 = b$; $\beta = \|r_0\|$
 - 4: $v_1 = r_0/\|r_0\|$;
 - 5: **for** $k = 1, 2, \dots$ **do**
 - 6: $z_k = M_k v_k$;
 - 7: $w = Az_k$;
 - 8: **for** $i = 1$ **to** k **do**
 - 9: $h_{i,k} = v_i^T w$
 - 10: $w = w - h_{i,k} v_i$
 - 11: **end for**
 - 12: $h_{k+1,k} = \|w\|$
 - 13: $v_{k+1} = w/h_{k+1,k}$
 - 14: Solve the least-squares problem $\min \|\beta e_1 - \bar{H}_k y\|$ for y
 - 15: Exit if the convergence is detected
 - 16: **end for**
 - 17: Set $x_m = x_0 + Z_m y$
-

2.2 The least-squares problem

At each step j of FGMRES, one needs to solve the least-squares problem (6). The matrix \bar{H}_j involved in this least-squares problem is a $(j+1) \times j$ complex matrix which is upper Hessenberg. We wish to use an efficient algorithm for solving (6) which exploits the structure of \bar{H}_j .

First, we base the solution of (6) 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 a $(j+1) \times (j+1)$ upper triangular matrix, then the solution y_j of (6) is given by

$$y_j = R(1:j, 1:j)^{-1} R(1:j, j+1). \quad (7)$$

Here, $R(1:j, 1:j)$ denotes the $j \times j$ top left submatrix of R and $R(1:j, j+1)$ stands for the last column of R . Moreover, as V_{j+1} is an orthonormal matrix,

$$\|r_j\|_2 = \|b - Ax_j\|_2 = \|\beta e_1 - \bar{H}_j y_j\|_2 = |r_{j+1, j+1}|. \quad (8)$$

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

The QR factorization of upper Hessenberg matrices can be efficiently performed using Givens rotations, because they enable one to sequentially zero out all elements $\bar{H}_{k+1, k}$, $k = 1, \dots, j$. 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 [4, 6, 7]):

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 $[\tilde{H}_j, \beta e_1] = Q_k R_k$. 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

$$\tilde{R}_j = \begin{pmatrix} + & + & + & * & + \\ 0 & + & + & * & + \\ 0 & 0 & + & * & + \\ 0 & 0 & 0 & * & + \\ 0 & 0 & 0 & * & 0 \end{pmatrix}.$$

2. A Givens rotation that zeros element $\tilde{R}_j(j + 2, j + 1)$ is computed and applied to \tilde{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 [4, 7] 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 FGMRES. It ensures that the convergence is not slowed down or delayed. However, 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 process for the computation of V_m and H_m . In finite precision arithmetic, there might be a severe loss of orthogonality; this loss can be compensated for by selectively iterating the orthogonalization scheme [5, 19] and the resulting QR factorization 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 an efficient manner by gathering the dot products into one dense 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 products is a well-known bottleneck [13, 16, 20, 29].

In our FGMRES 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 [26] to define the criterion for the selective reorthogonalization and set $K = \sqrt{2}$ as suggested by [12] as the value for the threshold.

2.4 Stopping criteria

We have chosen to base our stopping criterion on the normwise backward error. The backward error analysis, introduced by Givens and Wilkinson [35], is a powerful concept for analyzing the quality of an approximate solution:

1. it is independent of the details of round-off propagation: the errors 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 defined by (9) measures the distance between the data of the initial problem and those of a perturbed problem. Dealing with such a distance both requires to choose the data that are perturbed and a norm to quantify the perturbations. For the first choice, the matrix and the right-hand side of the linear systems are natural candidates. In the context of linear systems, classical choices are the normwise and the componentwise perturbations [11, 18]. 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 to the solution $x = A^{-1}b$. Then for α and β given (we refer to the end of this section for a discussion on how to choose those parameters),

$$\begin{aligned} \eta(x_j) &= \min_{\Delta A, \Delta b} \{ \varepsilon > 0 : \|\Delta A\|_2 \leq \varepsilon\alpha, \|\Delta b\|_2 \leq \varepsilon\beta \\ &\quad \text{and } (A + \Delta A)x_j = b + \Delta b \} \\ &= \frac{\|b - Ax_j\|_2}{\alpha \|x_j\|_2 + \beta} \end{aligned} \tag{9}$$

is called the *normwise backward error* associated with x_j . It measures the norm of the smallest perturbations ΔA on A and Δb on b such that $(A + \Delta A)x_j = b + \Delta b$. 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 of the data. Therefore, there is no gain in iterating after the backward error has reached machine precision (or data accuracy). Thanks to Equality (8), 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

$$\eta_A(x_j) = \frac{|r_{j+1,j+1}|}{\alpha \|x_j\|_2 + \beta}$$

as the stopping criterion of the FGMRES iterations. However, it is well-known that, in finite precision arithmetic, the computed residual (8) given from the Arnoldi process may differ significantly from the true residual. Therefore, it is not safe to exclusively use $\eta_A(x_j)$ as the stopping criterion. Our strategy is the following: first we iterate until $\eta_A(x_j)$ becomes lower than the tolerance, then afterwards, we iterate until $\eta(x_j)$ itself becomes lower than the tolerance. We hope in this way to minimize the number of explicit residual computations (involving the computation of matrix-vector products) necessary to evaluate $\eta(x_j)$, while still having a reliable stopping criterion.

How to choose α and β ? Classical choices for α and β that appear in the literature are $\alpha = \|A\|_2$ and $\beta = \|b\|_2$. Any other choice that reflects the possible uncertainty in the data can also be plugged in. In our implementation, default values are used when the user's input is $\alpha = \beta = 0$. Table 1 explains the output information given to the user on the unpreconditioned linear system on return from FGMRES.

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

Table 1: Stopping criterion for the FGMRES method.

2.5 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 techniques or in electromagnetism when a fast multipole expansion is 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 $r_0 = b - Ax_m$ with $x_m = x_0 + V_m y$. We can then observe that

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

It follows that the calculation of the residual amounts to computing a linear combination of the $(m + 1)$ Arnoldi 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 except the last which is equal to $r_{m+1, m+1}$. This non-zero value is a by-product of the solution of the least-squares problem. This calculation of the residual requires $n(2m + 1) + 2m$ floating point operations (flops) and is to be preferred to an explicit calculation whenever the matrix-vector product involving A would use 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 (although it might still enable us to get the solution within an overall shorter amount of computational time). Notice that the implementation of this trick requires the storage of $(m + 1)$ Arnoldi 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 $\eta_A(x_j)$ but not assessed by $\eta(x_j)$.

3 Implementation of FGMRES

3.1 The user interface

For the sake of simplicity and portability, the FGMRES 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 FGMRES in a parallel distributed memory environment, where only the user knows how the data has

been distributed (we refer to [16] 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_SFGMRES for real single precision arithmetic computation,
 DRIVE_DFGMRES for real double precision arithmetic computation,
 DRIVE_CFGMRES for complex single precision arithmetic computation,
 DRIVE_ZFGMRES for complex double precision arithmetic computation.

Finally, to hide the numerical method from the user as much as possible, 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_DFGMRES(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 basis V_m and Z_m as well as the Hessenberg matrix. It is not altered by the subroutine except if it was set by the user to a value larger than N or to a value too large for LWORK. In the first case, it would be reset to N. In the latter case, it would be reset 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:
 $M*M+M*(2*NLOC+5)+5*NLOC+1$ if ICNTL(7)=1, $M*M+M*(2*NLOC+5)+6*NLOC+1$ otherwise.
 The former value should be incremented by M if ICNTL(4)=2 or ICNTL(4)=3.
 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 needs 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 7 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 3 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 FGMRES. Details are given in Section 3.4.
RINFO	is a SINGLE/DOUBLE PRECISION REAL which contains the backward errors for the unpreconditioned linear systems.

3.2 The reverse communication management

The INTEGER array IRC allows the implementation of 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 $z \leftarrow Ax$.
3	The user must perform the right preconditioning $z \leftarrow M_i^{-1}x$.
4	The user must perform one or more scalar products $z \leftarrow x^*y$.

Notice that the value 2 has been skipped to be consistent with the implementation we proposed for GMRES in [14, 15].

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 (i.e. with a leading dimension equal to NLOC). IRC(3) indicates the index in WORK where y should be read. This programming trick permits ones to implement the dot products with a level 2 BLAS 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/reductions and alleviate the cost of the dot product computations.

Finally, IRC(6) indicates the index in WORK where a free workspace of size IRC(7) is available because it is not yet used by the solver. We allocate the space required to store V_m and Z_m at the end of the workspace, as depicted in Figure 1. We refer to Section 4 for an example of use of the driver routine.

3.3 The control parameters

The entries of the array ICNTL control the execution of the DRIVE_FGMRES subroutine. All entries of ICNTL are input parameters and some of them have a

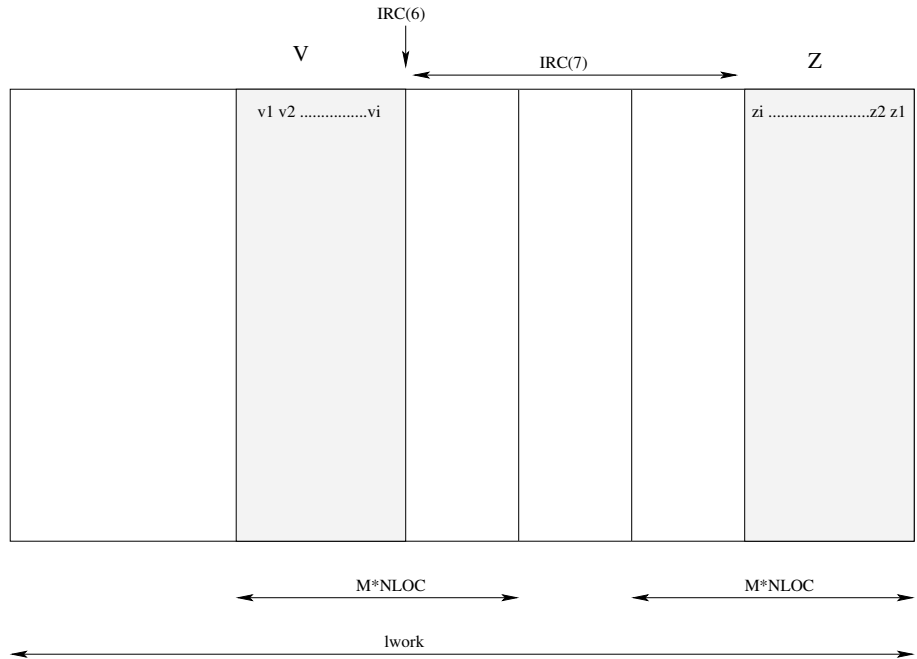


Figure 1: Management of the workspace: picture at the i -th iteration of FGMRES.

default value set by the routine `INIT_FGMRES`.

- 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 than or 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 than or equal to zero. A zero value implies that the convergence history will not be displayed.
- ICNTL(4) determines which orthogonalization scheme to apply (**Default is 0, i.e. MGS**).
- ICNTL(5) 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(5)=0`, the initial guess is set to zero.
- ICNTL(6) is the maximum number of iterations (accumulated over the restarts) allowed (**No Default: Must be set by the user**).
Must be larger than 0.

ICNTL(7) controls the strategy to compute the residual at the restart (**Default is 1**).
Must be equal to either 0 or 1.

Possible values for ICNTL(4) 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(7) 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 2.4) that control the execution of the algorithm:

- CNTL(1) is the convergence tolerance for the backward error (**Default is 10^{-5}**).
Must be greater than or equal to zero.
- CNTL(2) is the normalizing factor α (**Default is 0**).
Must be greater than or equal to zero.
- CNTL(3) is the normalizing factor β (**Default is 0**).
Must be greater than or equal to zero.

Default values are used when the user's input is $\alpha = \beta = 0$; that is $\beta = \|b\|_2$ respectively.

3.4 The information parameters

Once IRC(1) = 0, the entries of the array INFO explain the circumstances under which FGMRES 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(6) iterations.

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 the workspace. If `INFO(1) = -3`, then `INFO(2)` contains the minimal size necessary for the workspace.

If `INFO(1) = 0`, then `RINFO` contains the backward error for the linear system.

3.5 Initialization of the parameters

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

`INIT_SFGMRES` for real single precision arithmetic computation,
`INIT_DFGMRES` for double precision arithmetic computation,
`INIT_CFGMRES` for complex single precision arithmetic computation,
`INIT_ZFGMRES` 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_FGMRES(ICNTL,CNTL)
```

The default value for

`ICNTL(1)` is 6,
`ICNTL(2)` is 6,
`ICNTL(3)` is 0: no convergence history,
`ICNTL(4)` is 0: MGS is used,
`ICNTL(5)` is 0: default initial guess $x_0 = 0$,
`ICNTL(6)` is -1: the user must specify explicitly the maximum number of iterations,
`ICNTL(7)` is 1: the residual is explicitly computed at each restart,
`CNTL(1)` is 1,
`CNTL(2)` is 0,
`CNTL(3)` is 0.

3.6 Automatic correction for invalid parameters

To avoid an exit with an error when some parameters have been wrongly set by the user, we try as far 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` allowed for that size of the workspace. If `M` is lower than 1, we exit with an error.
`ICNTL(4)` is set to an invalid value, we set it back to the default.
`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 to `N`.
`ICNTL(7)` 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 a relevant alternative and consequently we output an error message and return to the calling program. 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 any FGMRES iteration.

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 2. We should also mention that a free implementation of GMRES [14, 15] is also

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 2: BLAS routines called in GMRES.

available at the same URL address.

4 An example of use

We give below an example of use of the FGMRES driver. Here the preconditioner is the GMRES method implemented as in [14, 15]. Note that, in this example, we have chosen not to allocate extra memory for the preconditioner: when the preconditioner is needed for FGMRES, we compute how many steps of GMRES are possible with the part of the workspace which is still free. The inner GMRES can be itself preconditioned: in this example we use a simple a Jacobi (left) preconditioner.

```

    program validation
*
    integer lda, ldstrt, lwork
    parameter (lda= 1000, ldstrt = 60)
    parameter (lwork= ldstrt**2 + ldstrt*(2*lda+5)
&             + 6*lda + ldstrt)
*
    integer i, j, n, m, m2
    integer revcom, colx, coly, colz, nbscal
    integer revcom2, colx2, coly2, colz2, nbscal2
    integer irc(7), icntl(7), info(3)
    integer irc2(5), icntl2(8), info2(3)
*
    integer matvec, preconditionLeft, preconditionRight, dotProd
    parameter (matvec=1, preconditionLeft=2)
    parameter (preconditionRight=3, dotProd=4)
*
    integer nout
*
    complex*16 a(lda,lda), work(lwork)
    real*8      cntl(3), rinfo, rn
    real*8      cntl2(5), rinfo2(2)
*
    complex*16 ZERO, ONE
    parameter (ZERO = (0.0d0, 0.0d0), ONE = (1.0d0, 0.0d0))
*
* 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_zfgmres(icntl,cntl)
    call init_zgmres(icntl2,cntl2)
*
*****
*c Tune some parameters for FGMRES
*****
*
* Tolerance
    cntl(1) = 1.d-9
* Save the convergence history in file fort.20
    icntl(3) = 20
* ICGS orthogonalization
    icntl(4) = 3
* Maximum number of iterations
    icntl(6) = 100
*
*****
*c Tune some parameters for GMRES
*****
*
* Tolerance
    cntl2(1) = 5.d-2
* warning output stream
    icntl2(2) = 0
* Save the convergence history in file fort.20

```

```

        icntl2(3) = 30
* No preconditioning
        icntl2(4) = 0
        print *, ' Inner GMES precondition 0-none, 1:left, 2:right '
        read(*,*) icntl2(4)
* ICGS orthogonalization
        icntl2(5) = 3
* Maximum number of iterations
        icntl2(7) = 6
        print *, ' Max Inner GMES iterations '
        read(*,*) icntl2(7)
*
*****
** Reverse communication implementation
*****
*
10 call drive_zfgmres(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 for FGMRES
*   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.precondRight) then
* perform the right preconditioning for the FGMRES iteration
*
* Check if there is enough space left in the workspace
* to perform few steps of GMRES as right preconditioner
        rn = float(n)
        rx  = rn + 5.0
        rc  = 5.0*rn + 1 - float(irc(7))
*
* Update the linear part of the second order equation to be solved
if ((icntl2(5).eq.2).or.(icntl2(5).eq.3)) then
        rx = rx + 1
endif
* Update the constant part of the second order equation to be solved
*
        if (icntl2(8).eq.0) then
            rc = rc + rn
        endif
        m2 = ifix((-rx+sqrt(rx**2-4.0*rc))/2.0)
*
        if (m2.gt.0) then
* copy colx in the workspace (right hand side location) of
* the inner gmres iteration
        call zcopy(n,work(colx),1,work(irc(6)+n),1)
20 call drive_zgmres(n,n,m2,irc(7),
    &         work(irc(6)),irc2,icntl2,cntl2,info2,rinfo2)
        revcom2 = irc2(1)
        colx2  = irc2(2) + irc(6) -1
        coly2  = irc2(3) + irc(6) -1
        colz2  = irc2(4) + irc(6) -1
        nbscal2 = irc2(5)

```

```

        if (revcom2.eq.matvec) then
* Perform the matrix-vector product for the
* inner GMRES iteration
        call zgemv('N',n,n,ONE,a,lda,work(colx2),1,
&          ZERO,work(colz2),1)
        goto 20
        else if (revcom2.eq.precondRight) then
* perform the precondition for the inner GMRES iteration
        do i =0,n-1
            work(colz2+i) = work(colx2+i)/a(i+1,i+1)
        enddo
        goto 20
        else if (revcom2.eq.precondleft) then
* perform the precondition for the inner GMRES iteration
        do i =0,n-1
            work(colz2+i) = work(colx2+i)/a(i+1,i+1)
        enddo
        goto 20
        else if (revcom2.eq.dotProd) then
* perform the dot-product for the inner GMRES iteration
* work(colz) <-- work(colx) work(coly)
* The statement to perform the dot products can be
* written in a compact form.
*   call zgemv('C',n,nbscal2,ONE,work(colx2),n,
* &           work(coly2),1,ZERO,work(colz2),1)
* For sake of simplicity we write it as a do-loop
        do i=0,nbscal2-1
            work(colz2+i) = zdotc(n,work(colx2+i*n),1,
&          work(coly2),1)
        &
        goto 20

```

```

        endif
        call zcopy(n,work(irc(6)),1,work(colz),1)
        goto 10
        else
* (m2.1e.0)
        call zcopy(n,work(colx),1,work(colz),1)
        goto 10
        endif
        else if (revcom.eq.dotProd) then
* perform the scalar product for the FGMRES iteration
* 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 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
*
*****
* dump the solution on a file
*****
        .....
*

```

References

- [1] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.
- [2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, second edition, 1994.
- [3] O. Batiste, A. Alonso, and I. Mercader. Hydrodynamic stability of binary mixtures in Bénard and thermogravitational cells. *J. Non-equilib. Thermodyn.*, 29:359–375, 2004.
- [4] 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.
- [5] Å. Björck. Numerics of Gram-Schmidt orthogonalization. *Linear Algebra Appl.*, 197–198:297–316, 1994.
- [6] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.
- [7] 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.
- [8] A. Bouras and V. Frayssé. Inexact matrix-vector products in Krylov methods for solving linear systems: a relaxation strategy. *SIAM Journal on Matrix Analysis and Applications*, 26(23):660–678, 2005.
- [9] B. Carpentieri. *Sparse preconditioners for dense linear systems from electromagnetic applications*. PhD thesis, CERFACS, Toulouse, France, April 2002.
- [10] B. Carpentieri, I. S. Duff, L. Giraud, and G. Sylvand. Combining fast multipole techniques and an approximate inverse preconditioner for large electromagnetism calculations. *SIAM Journal Scientific Computing*, 27(3):774–792, 2005.
- [11] F. Chaitin-Chatelin and V. Frayssé. *Lectures on Finite Precision Computations*. SIAM, Philadelphia, 1996.
- [12] 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.

- [13] J. Frank and C. Vuik. Parallel implementation of a multiblock method with approximate subdomain solution. *Appl. Num. Math.*, 30:403–423, 1999.
- [14] V. Frayssé, L. Giraud, S. Gratton, and J. Langou. A set of GMRES routines for real and complex arithmetics on high performance computers. Technical Report TR/PA/03/03, CERFACS, Toulouse, France, 2003. Available on <http://www.cerfacs.fr/algor>.
- [15] V. Frayssé, L. Giraud, S. Gratton, and J. Langou. Algorithm 842: A set of GMRES routines for real and complex arithmetics on high performance computers. *ACM Transactions on Mathematical Software*, 31(2):228–238, 2005.
- [16] 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.
- [17] L. Giraud, S. Gratton, and J. Langou. Convergence in backward error of relaxed GMRES. Technical Report TR/PA/04/132, CERFACS, Toulouse, France, 2004.
- [18] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.
- [19] W. Hoffmann. Iterative algorithms for Gram-Schmidt orthogonalization. *Computing*, 41:335–348, 1989.
- [20] R. B. Lehoucq and 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.
- [21] Z. Li, Y. Saad, and M. Sosonkina. pARMS: a parallel version of the recursive multilevel solver. *Numerical Linear Algebra with Applications*, 10:485–509, 2003.
- [22] E. Meca, I. Mercader, O. Batiste, and L. Ramírez de la Piscina. A blue sky catastrophe in double-diffusive convection. *Physical Review Letters*, 92(23):1–4, 2004.
- [23] E. Meca, I. Mercader, O. Batiste, and L. Ramírez de la Piscina. Complex dynamics in double-diffusive convection. *Theoretical Computational Fluid Dynamics*, 18:231–238, 2004.
- [24] I. Mercader, A. Alonso, and O. Batiste. Numerical analysis of the Eckhaus instability in travelling-wave convection in binary mixtures. *European Physical Journal E*, 15:319–333, 2004.

- [25] I. Mercader, O. Batiste, L. Ramírez de la Piscina, X. Ruiz, S. Rüdiger, and J. Casademunt. Bifurcations and chaos in single-roll natural convection with low Prandtl number. *Phys. Fluids*, 2005. submitted.
- [26] H. Rutishauser. Description of algol 60. handbook for automatic computation. *Springer Verlag, Berlin*, 1.a, 1967.
- [27] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14:461–469, 1993.
- [28] 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.
- [29] 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.
- [30] V. Simoncini and D. B. Szyld. Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM Journal Scientific Computing*, 25:454–477, 2003.
- [31] Z. Strakoš and P. Tichy. Error estimation in preconditioned conjugate gradients. *BIT*, 2006. accepted.
- [32] R. S. Tuminaro, M. Heroux, S. A. Hutchinson, and J. Shadid. Official Aztec User’s Guide - Version 2.1. Tech. Rep. 99-8801J, Sandia National Laboratories, 1999.
- [33] J. van den Eshof and G. L. G. Sleijpen. Inexact Krylov subspace methods for linear systems. *SIAM Journal on Matrix Analysis and Applications*, 26(1):125–153, 2004.
- [34] J. S. Warsa, M. Benzi, T. A. Wareing, and J. E. Morel. Preconditioning a mixed discontinuous finite element method for radiation diffusion. *Numerical Linear Algebra with Applications*, 11:795–811, 2004.
- [35] J. H. Wilkinson. *Rounding errors in algebraic processes*, volume 32. Her Majesty’s stationery office, London, 1963.