

API of a set of FGMRES Routines for Real and Complex Arithmetics on High Performance Computers¹

VALÉRIE FRAYSSÉ

Kvasar Technology LLC, Boston, USA

LUC GIRAUD

ENSEEIH-IRIT, Toulouse, France

SERGE GRATTON

CNES, Toulouse, France

1. FRAMEWORK

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

The FGMRES (Flexible Generalized Minimum Residual) method [Saad 1993] is among the most widely used Krylov solvers for the iterative solution of general large linear systems when variable preconditioning is considered.

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

$$AMz = b \tag{2}$$

with $x = Mz$. In some situation, it might not be possible to explicitly form M and its application to a vector might vary from one iteration to the next (for instance Mt is obtained by iteratively solving the linear system $Ay = t$). Such a preconditioning strategy is referred to as flexible preconditioning. The FGMRES (Flexible Generalized Minimum Residual) method [Saad 1993] is among the most widely used Krylov solvers for the iterative solution of general large linear systems when variable/flexible preconditioning is considered.

¹The major part of this work was carried out while the three authors were working at CERFACS

2. IMPLEMENTATION OF FGMRES

2.1 The user interface

For the sake of simplicity and portability, the FGMRES implementation is developed in Fortran 77 and 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 FGMRES in a parallel distributed memory environment, where only the user knows how the data has been distributed. 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 (See Figure 1 for a illustration of the definition of this parameter). 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. From a rate of convergence point of view, it is generally observed that the larger M the faster the convergence.

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 LWORK_min:

$$\text{LWORK_min} = M*M + M*(2*NLOC + 5) + 5*NLOC + 1 \text{ if } ICNTL(7)=1,$$

$$\text{LWORK_min} = M*M + M*(2*NLOC + 5) + 6*NLOC + 1 \text{ otherwise.}$$
 The above value of LWORK_min 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 solution in output. The following NLOC entries contain the right-hand side b of the linear 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 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 3 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 FGMRES. Details are given in Section 2.4.
RINFO	is a SINGLE/DOUBLE PRECISION REAL which contains the backward error for the linear systems.

In the figure below we illustrate how the parameter NLOC and N should be set for the parallel solution of a linear system in a distributed memory environment using 2 processors. We assume that (x_1, b_1) of size n_1 ((x_2, b_2) of size n_2) are stored in the local memory of the processor P_1 (resp. P_2).

$$\boxed{\begin{array}{l} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad \begin{array}{l} \text{on } P_1 : \text{NLOC} = n_1, N = n_1 + n_2 \\ \text{on } P_2 : \text{NLOC} = n_2, N = n_1 + n_2 \end{array} \end{array}}$$

Fig. 1. Definition of NLOC and N on 2 processors in a parallel distributed environment

2.2 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 $z \leftarrow x^H y$.

Notice that the value 2 has been skipped to be consistent with the implementation we proposed for GMRES in [Frayssé et al. 2003; 2005].

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

When $\text{IRC}(1) = 4$, $\text{IRC}(5)$ gives the number of scalar products to be performed. In this case, x denotes an array of size $\text{NLOC} \times \text{IRC}(5)$ stored column-wise (i.e. with a leading dimension equal to NLOC). $\text{IRC}(3)$ indicates the index in `WORK` where y should be read. This programming trick permits one 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 one to reduce the number of global synchronizations/reductions and alleviate the cost of the dot product computations.

Finally, $\text{IRC}(6)$ indicates the index in `WORK` where a free workspace of size $\text{IRC}(7)$ 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 2. We refer

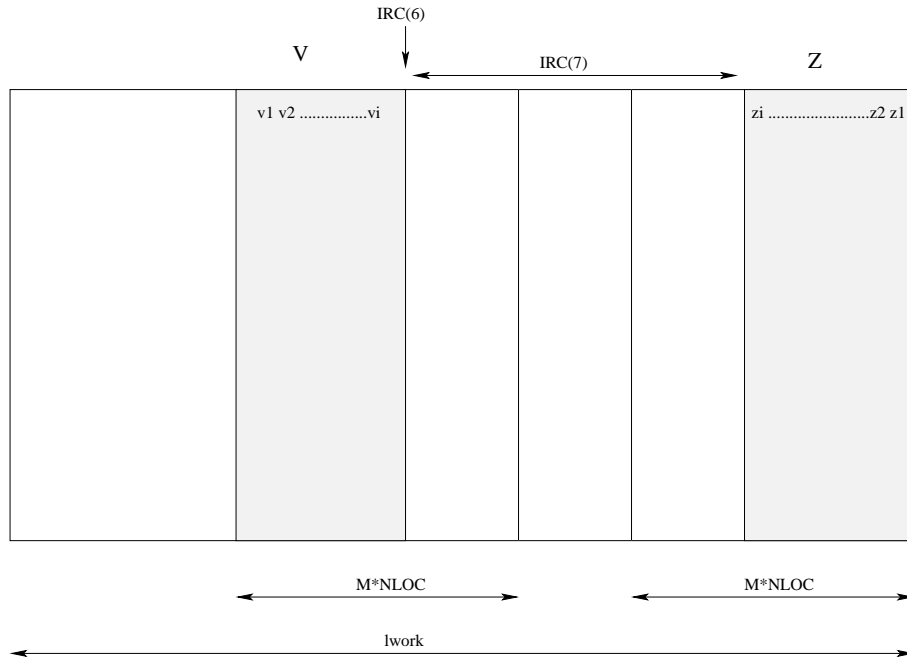


Fig. 2. Management of the workspace: picture at the i -th iteration of FGMRES.

to Section 3 for an example of use of the driver routine.

2.3 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 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 (**Default is arbitrary 100**).
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 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.

2.4 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	<code>LWORK</code> too small.
-4	convergence not achieved after <code>ICNTL(6)</code> 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.

2.5 Initialization of the parameters

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

<code>INIT_SFGMRES</code>	for real single precision arithmetic computation,
<code>INIT_DFGMRES</code>	for double precision arithmetic computation,
<code>INIT_CFGMRES</code>	for complex single precision arithmetic computation,
<code>INIT_ZFGMRES</code>	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

<code>ICNTL(1)</code>	is 6,
<code>ICNTL(2)</code>	is 6,
<code>ICNTL(3)</code>	is 0: no convergence history,
<code>ICNTL(4)</code>	is 0: MGS is used,
<code>ICNTL(5)</code>	is 0: default initial guess $x_0 = 0$,
<code>ICNTL(6)</code>	is arbitrary set to 100,
<code>ICNTL(7)</code>	is 1: the residual is explicitly computed at each restart,
<code>CNTL(1)</code>	is 1,
<code>CNTL(2)</code>	is 0,
<code>CNTL(3)</code>	is 0.

2.6 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(7)** is set to an invalid value, we set it back to the default.

2.7 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 the 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 I. We

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 I. BLAS routines called in GMRES.

should also mention that a free implementation of GMRES [Frayssé et al. 2003; 2005] is also available at the same URL address.

3. 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 [Frayssé et al. 2003; 2005]. 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 to compute the largest possible restart
        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 to compute the largest possible restart
    *
        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.le.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

- FRAYSSÉ, V., GIRAUD, L., GRATTON, S., AND LANGOU, J. 2003. A set of GMRES routines for real and complex arithmetics on high performance computers. Technical Report TR/PA/03/03, CERFACS, Toulouse, France. Available on <http://www.cerfacs.fr/algor>.
- FRAYSSÉ, V., GIRAUD, L., GRATTON, S., AND LANGOU, J. 2005. Algorithm 842: A set of GMRES routines for real and complex arithmetics on high performance computers. *ACM Trans. Math. Softw.* 31, 2, 228–238.
- SAAD, Y. 1993. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.* 14, 461–469.
- SAAD, Y. AND SCHULTZ, M. 1986. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.* 7, 856–869.