A Set of Flexible-GMRES Routines for Real and Complex Arithmetics

Valérie Frayssé * Luc Giraud * Serge Gratton *

CERFACS Technical Report TR/PA/98/20

Abstract

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

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

1 The FGMRES algorithm

1.1 General description

The Generalized Minimum RESidual (GMRES) method was proposed by Saad and Schultz in 1986 [7] in order to solve large, sparse and nonsymmetric (or non Hermitian) linear systems. In 1993, Saad [6] 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.

In the sequel, we start by briefly describing the standard GMRES algorithm with right preconditioning and then show the modification which allows to use a different preconditioning at each GMRES iteration.

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, $r_0 = b - Ax_0$ be its corresponding residual and M^{-1} the right preconditioner.

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

The GMRES algorithm with right preconditioning solves the modified system:

$$AM^{-1}(Mx) = b.$$

The algorithm is classically described by

1. **Start** Choose an initial guess x_0, m the dimension of the Krylov subspaces.
   Setup a $(m+1) \times m$ matrix $\bar{H}_m$ to zero.
2. **Arnoldi process:**
   (a) Compute $r_0 = b - Ax_0$ and $v_1 = \dfrac{r_0}{\|r_0\|}$
   (b) For j=1,..,m
   - Compute $z = M^{-1}v_j$
   - Compute $w = Az$
   - For $i = 1, \ldots, j$ $\begin{cases} h_{i,j} =< w, v_i > \\ w = w - h_{i,j}v_i \end{cases}$
   - Compute $h_{j+1,j} = \|w\|$ and $v_{j+1} = \dfrac{w}{\|w\|}$

   (c) Define $V_m = [v_1, \ldots, v_m]$
3. **Update the approximate solution:**
   Compute $x_m = x_0 + M^{-1}V_m y_m$ where $y_m = \text{argmin} \left\| \beta e_1 - \bar{H}_m y \right\|$
4. **Restart** If convergence stop, else $x_0 = x_m$, goto 2.

Algorithm 1: GMRES(m) with right preconditioning.

The Arnoldi process simply constructs an orthogonal basis of the preconditioned Krylov subspace:

$$\mathcal{K}_m = \text{span}\left\{r_0, AM^{-1}r_0, \ldots, (AM^{-1})^{m-1}r_0\right\},$$

using a modified Gram-Schmidt orthogonalization. Step 3 of the above algorithm updates the solution using a linear combination of the preconditioned vectors $z_i = M^{-1}v_i$. When all these vectors are obtained by applying the same M^{-1} to the v_i we do not need to store them. But if we want to vary the preconditioner at each step by using M_i^{-1}, we can still update the solution x at step 3 at the price of storing the sequence of vectors z_i. In this respect we end-up with the Flexible variant of GMRES (FGMRES) described in [6].

The only difference with the classical GMRES is that we have to store the preconditioned vectors z_i and perform the update of the solution with these vectors.

We do not pursue further the description of this algorithm but refer to [6] for a complete exposition of the convergence theory; we only notice that despite the classical GMRES a general convergence theorem cannot be proved.

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

- the solution of the least-squares problem (2),
- the construction of the orthonormal basis V_m, and
- the stopping criteria for the iterative scheme.

1. **Start** Choose an initial guess x_0, m the dimension of the Krylov subspaces.
   Setup a $(m+1) \times m$ matrix $\bar{H}_m$ to zero.

2. **Arnoldi process:**

   (a) Compute $r_0 = b - Ax_0$ and $v_1 = \dfrac{r_0}{\|r_0\|}$

   (b) For j=1,..,m

   - Compute $z_j = M_j^{-1} v_j$
   - Compute $w = Az_j$
   - For $i = 1, \ldots, j$ $\begin{cases} h_{i,j} =< w, v_i > \\ w = w - h_{i,j} v_i \end{cases}$
   - Compute $h_{j+1,j} = \|w\|$ and $v_{j+1} = \dfrac{w}{\|w\|}$

   (c) Define $Z_m = [z_1, \ldots, z_m]$

3. **Update the approximate solution:**
   Compute $x_m = x_0 + Z_m y_m$ where

$$y_m = \operatorname{argmin} \left\| \beta e_1 - \bar{H}_m y \right\| \tag{2}$$

4. **Restart** If convergence stop, else $x_0 = x_m$, goto 2.

Algorithm 2: FGMRES(m) with right preconditioning.

1.2 The least-squares problem

At each step of GMRES, one needs to solve the least-squares problem (2). The matrix $\bar{H}_m$ involved in this least-squares problem is an $(m+1) \times m$ complex matrix which is upper Hessenberg. We wish to use an efficient algorithm for solving (2) which exploits the structure of $\bar{H}_m$.

First, we base the solution of (2) on the QR factorization of the matrix $[\bar{H}_m,\ \beta e_1]$: if $QR = [\bar{H}_m,\ \beta e_1]$ where Q is an orthonormal matrix and $R = (r_{ij})$ is an $(m+1) \times (m+1)$ upper triangular matrix, then the solution y_m of (2) is given by

$$y_m = R(1:m, 1:m)^{-1} R(1:m, m+1). \tag{3}$$

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

$$\|r_m\|_2 = \|b - Ax_m\|_2 = \left\|\beta e_1 - \bar{H}_m y_m\right\|_2 = |r_{m+1,m+1}|. \tag{4}$$

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

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

Givens rotations are matrices of the form

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

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

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

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

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

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

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

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

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

1.3 Computation of V_m

Most of the time, the Arnoldi algorithm is implemented through the Modified Gram-Schmidt (MGS) process for the computation of V_m and H_m. However, in finite precision arithmetic, there might be a sever loss of orthogonality in the computed basis[1]; this loss can be compensated by iterating the orthogonalization scheme [2]. The resulting algorithm is called Iterative Modified Gram-Schmidt (IMGS). The drawback of IMGS is the increased number of dot products.

The Classical Gram-Schmidt (CGS) algorithm can be implemented in a cheap manner by gathering the dot products into one matrix-vector product, but it is well-known that CGS is numerically worse than MGS. However, iterating CGS (ICGS) results in an algorithm of the same numerical quality as IMGS. Therefore, ICGS is particularly attractive in a parallel distributed environment, where the computation of the dot product is a well-known bottleneck [8].

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.

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

1.4 Stopping criteria

We have chosen to base our stopping criterion on the normwise backward error [3]. The backward error analysis, introduced by Givens and Wilkinson [9], 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 [3]. These choices lead to explicit formulas for the backward error (often a normalized residual) which is then easily evaluated. For iterative methods, it is generally admitted that the normwise model of perturbation is appropriate [1].

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

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

is called the *normwise backward error* associated with x_m. The best one can require from an algorithm is a backward error of the order of the machine precision. In practice, the approximation of the solution is acceptable when its backward error is lower than the uncertainty on the data. Therefore, there is no gain in iterating after the backward error has reached machine precision (or data accuracy). Thanks to Equality (4), we see that the 2-norm of the residual is given directly in the algorithm during the solution of the least-squares problem. Therefore, the backward error can be obtained at a low cost and we can use

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

as the stopping criterion of the FGMRES iterations. However, it is well-known that, in finite precision arithmetic, the computed residual (4) given from the Arnoldi process may differ significantly from the true residual. Therefore, it is not safe to use exclusively η_A as the stopping criterion. Our strategy is the following: first we iterate until η_A becomes lower than the tolerance, then afterwards, we iterate until η becomes itself lower than the tolerance. We hope in this way to minimize the number of residual computations (involving the computation of matrix-vector product) necessary to evaluate η, while having a reliable stopping criterion.

How to choose α and β ? Classical choices for α and β that appears in the literature are $\alpha = \|A\|_2$ and $\beta = \|b\|_2$. Any other choice that reflects the possible uncertainty on the data can also be plugged in. In our implementation, default values are used when the user's input is $\alpha = \beta = 0$. Table 1 lists the stopping criteria for different choices of α and β.

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

Table 1: Stopping criterion for the FGMRES method

2 Implementation of FGMRES

2.1 The user interface

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

- for implementing the numerical kernels that depend on the data structure of 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 he has spread his data. 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 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_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 bases V_m and Z_m and the Hessenberg matrix. It is not altered by the subroutine.
`LWORK`	is an INTEGER variable that must be set by the user to the size of the workspace `WORK`. `WORK` must be greater than or equal to `M*M+M*(2*NLOC+5)+5*NLOC+1`. It is not altered by the subroutine.
`WORK`	is a SINGLE/DOUBLE PRECISION REAL/COMPLEX array of length `LWORK`. The first `NLOC` entries contain the initial guess x_0 in input and the computed approximation of the unpreconditioned solution in output. The following `NLOC` entries contain the right-hand side b of the unpreconditioned system. The remaining entries are used as workspace by the subroutine.
`IRC`	is an INTEGER array of length 7 that need not be set by the user. This array controls the reverse communication. Details of the reverse communication management are given in Section 2.2.
`ICNTL`	is an INTEGER array of length 6 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 DOUBLE PRECISION REAL which contains the backward error for the linear system.

2.2 The reverse communication management

The INTEGER array `IRC` permits to implement the reverse communication. None of its entries need be set by the user.
On each exit, `IRC(1)` indicates the action that must be performed by the user before invoking the driver again. Possible values of `IRC(1)` and the associated actions are as follows:

`0`	Normal exit.
`1`	The user must perform the matrix vector product $z \leftarrow Ax$.
`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 [4].

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

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

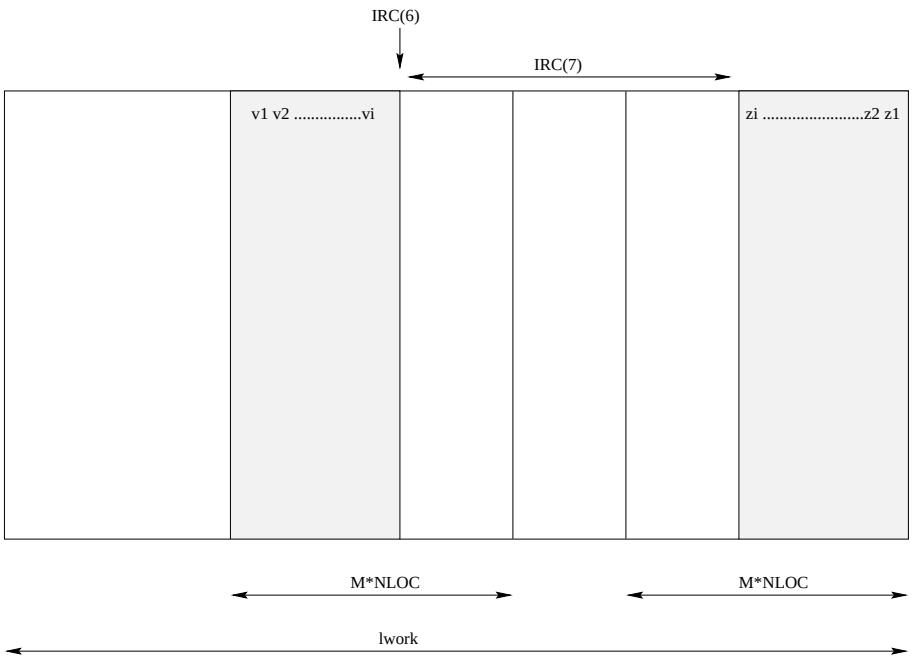

Figure 1: Management of the workspace: picture at the ith iteration of FGMRES

2.3 The control parameters

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

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

Possible values for `ICNTL(4)` are

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

The entries of the `CNTL` array define the tolerance and the normalizing factors (see Section 1.4) that control the execution of the algorithm:

`CNTL(1)` is the convergence tolerance for the backward error (see Section 1.4 for details).
`CNTL(2)` is the normalizing factor α.
`CNTL(3)` is the normalizing factor β.

2.4 The information parameters

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

Possible values for `INFO(1)` are

`0`	normal exit. Convergence has been observed.
`-1`	erroneous value $n < 1$.
`-2`	erroneous value $m < 1$.
`-3`	`LWORK` too small.
`-4`	convergence not achieved after `ICNTL(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 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:

`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,
`CNTL(1)`	is 1,
`CNTL(2)`	is 0,
`CNTL(3)`	is 0.

3 Availability of the software

The code is written in FORTAN 77 and makes calls to BLAS routines, as indicated in Table 2. The code is free for non-commercial use only. The source code is available from the WEB at the URL

`http://www.cerfacs.fr/algor/`

together with the software license agreement.

Simple precision		Double precision	
real	complex	real	complex
SAXPY	CAXPY	DAXPY	ZAXPY
SGEMV	CGEMV	DGEMV	ZGEMV
SCNRM2	SCNRM2	DNRM2	DZNRM2
SCOPY	CCOPY	DCOPY	ZCOPY
STRSV	CTRSV	DTRSV	ZTRSV

Table 2: BLAS routines called in FGMRES

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 [4]. Note that, in this example, we have chosen not to allocate extra memory for the preconditioner: when the right 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 propose a Jacobi (left) preconditioner.

```
      program validation
*
      integer lda, ldstrt, lwork
      parameter (lda = 1000, ldstrt = 60)
      parameter (lwork = ldstrt**2 + ldstrt*(2*lda+5) + 5*lda + 1)
*
      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(7), info2(3)
*
      integer matvec, precondLeft, precondRight, dotProd
      parameter (matvec=1, precondLeft=2)
      parameter (precondRight=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 precond 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 the FGMRES iteration
*     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)
      m2 = ifix((-5.0-rn+sqrt((rn+5.0)**2-4.0*(5.0*rn
  &                   +1.0-float(irc(7)))))/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 preconditioning 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 preconditioning 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 scalar 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

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

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

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

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

[5] G. H. Golub and C. Van Loan. *Matrix computations.* Johns Hopkins University Press, 1989. Second edition.

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

[7] 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.

[8] J. N. Shadid and R. S. Tuminaro. A comparison of preconditioned nonsymmetric Krylov methods on a large-scale MIMD machine. *SIAM J. Sci. Comput.*, 14(2):440–459, 1994.

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