BASIC MANUAL FOR THE MATRIX MANAGER

MatMan

Ramon Codina

November 22, 1999

Abstract

`MatMan` is a Fortran 77 library of subroutines to deal with the linear systems of equations arising in finite element analysis. The idea is to minimize the interaction between the finite element code and all the tasks related to management of matrices needed to arrive to the final algebraic system and its solution. `MatMan` basically performs three different operations: memory allocation for all the arrays involved in the management of matrices, the construction of the matrix of the system, and the solution of this linear system.

Contents

1 Introduction

1.1 What is `MatMan` ?

`MatMan` is a Fortran library of subroutines to solve linear systems of equations of the form

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}, \tag{1}$$

where A, B, C and D are system matrices, x and y are the vectors of unknowns and f and g are given right-hand-side vectors. In particular, `MatMan` is intended to deal with linear systems arising in *finite element calculations*, in which the matrices in (1) may be computed from the assembly of the element matrices.

The reason for writing the linear system in the form of (1) is that variables x and y are assumed to have the meaning of primal and dual variables, respectively, in a mixed problem, and therefore they deserve a different treatment. Of course `MatMan` may deal with the unknowns as if they were all primal, even in a mixed problem, but this is at the expense of loosing efficiency in some of the operations `MatMan` performs.

The basic objective of `MatMan` is to avoid the burden of allocating memory for the system matrix, renumbering its entries, constructing it and solving system (1) every time a new finite element code is written. Indeed, `MatMan` is in charge of *all* these tasks.

A very important feature of `MatMan` is that it is designed to deal *with several problems at the same time*. The total number of problems to be considered is defined in the file `MatMan.h` described in the following, and can be changed at the user's convenience.

1.2 Where does it come from ?

I started to think about `MatMan` as I was developing the CFD code `Fantom`, in which several problems are dealt with (velocity–pressure, temperature, turbulent variables...). It became clear that it would be interesting to have a library in charge of all the operations related to the matrices arising in finite element calculations.

Initially, `MatMan` was a satellite tight to `Fantom`, but the link weakened until its present form, resulting in a plug that can be connected to any finite element code.

This genesis already anticipates that *I did not write the whole package*, but rather created the conglomerant to glue together routines that I already had, and provided by several people the list of who I am unable to record. First versions of routines to renumerate and solve systems with the system matrix stored in skyline based on the implementation by R.L. Taylor were provided to me by M. Cervera (who in turn inherited some of them), some routines to deal with banded matrices were written by O. Soto (who collaborated with me in writing the last version of `Fantom`), iterative algorithms with the system matrix stored in compressed sparse row (CSR) format are based on the SPARSKIT Library developed by Y. Saad, and direct solution methods with the matrix also in sparse format are based on the Y12M package by Z. Zlatev, J. Wasniewski and K. Schaumburg.

This diversity of origins has as a consequence a variety of coding styles which, together with some patches that I had to write to couple routines, makes `MatMan` a little messy to read. Nevertheless, the hope is that it can be used as a true black box, and thus *looking at* it is unnecessary...

2 Basic calls

2.1 Subroutine call

The call to `MatMan` must be as follows:

```
 call MatMan(lnods,iffix,rhsid,unkno,ndofn,nnode,nnodt,
.            nnodw,nelem,npoin,ntotv,nsist,lures,kfact,
.            kimpo,iprob,inte1,inte2,real1,itask)
```

The meaning of the variables is explained next. It is suggested to use dummy arguments for those that are not used for a certain task.

2.2 Tasks

For each problem to be dealt with, the solver library `MatMan` must be called *four times*, with an additional fifth call that is optional. These calls are the following:

Task 1: ***Number or renumber of equations***. This call must be done at the beginning of the code or whenever the finite element mesh changes.

Task 2: ***Memory allocation***. Usually, this call is only performed once after the call with Task 1. But, in cases in which it is not affordable to keep the memory for all the problems at the same time, it is possible to deallocate the memory for a problem and then to reallocate it when it must be solved again. For direct solvers using either a banded or a profile storage, the memory for the matrix allocated is only the maximum needed for each problem, even if this memory has not been released.

Task 3: ***Construction of the system matrix***. `MatMan` constructs the system matrix from the contributions given to its components, either one by one or directly from an element matrix.

Task 4: ***Solution of the linear system***. Obvious.

Task 5: ***Deallocate memory***. Usually, this is unnecessary, unless the memory of all the problems cannot be kept at the same time (see Task 2). This is the only optional task.

3 Data parameters for `MatMan`

3.1 Meaning of the variables

The variables `MatMan` needs are described below. The order in which they appear is the same as the order of the arguments of `MatMan.f`, and therefore it is necessary to go back and forth to get a complete description of the data to be sent to `MatMan` . The unknown x in (1) is considered as the main variable, whereas unknown y is the secondary variable or Lagrange multiplier, which can have *one* degree of freedom per node at most.

`LNODS`: Matrix of nodal connections. Dimension (NNODT,NELEM). It must be filled with 0's for the elements with less nodes than NNODT (see below the meaning of NNODT). For nodes with degrees of freedom that have been condensed by the user, their value must be negative.

`IFFIX`: Vector of fixity conditions. Dimension (NTOTV), although only the first NDOFN * NPOIN components are used (it is assumed that the "multiplier" degrees of freedom are always free).

`RHSID`: RHS vector. Dimension (NTOTV,NSIST).

`UNKNO`: Vector of unknowns. Dimension (NTOTV,NSIST). For iterative solvers it is assumed that the initial guess is also given in this space.

`NDOFN`: Number of d.o.f. of the main nodes.

`NNODE`: Maximum number of main nodes per element.

`NNODT`: Maximum total number of nodes per element (main nodes plus secondary nodes associated to the "multiplier" y).

`NNODW`: Maximum number of active nodes per element:

- `= NNODE`: Only the main nodes (with NDOFN d.o.f.) are active
- `= NNODT`: The main nodes and NNODS = NNODT – NNODE secondary nodes are active. These nodes have only 1 d.o.f. and no fixity conditions.

The need for this variable arises because in the same run it may be necessary to deal with problems with and without secondary variables.

`NELEM`: Number of elements.

`NPOIN`: Number of main nodal points.

`NTOTV`: Total number of d.o.f = NDOFN * NPOIN + NPOIS, where NPOIS is the number of secondary points.

`NSIST`: Number of systems to be solved with the same matrix and different RHS.

`LURES`: Output unit. The user must be in charge of opening it. The information given by `MatMan` is written there.

`KFACT`: Code for matrix factorization. Suppose for the sake of simplicity that there are no secondary variables, and thus the system to be solved is $Ax = f$. If matrix A comes from a partial differential equation, it is singular until Dirichlet boundary conditions are prescribed. Let x_p be the set of prescribed d.o.f., A_p the submatrix of the global matrix A multiplying x_p and A_0 the submatrix of the global matrix A multiplying the free unknowns x_0. After prescribing the boundary condition, the modified algebraic system will be of the form $A_0 x_0 = f_0 - A_p x_p$. This modification may be done either by the user or by `MatMan` (see the meaning of KIMPO).

- `= 0`: Compute the matrix product $A_p x_p$ and do not factorise A. This option IS NOT allowed for EE iterative solvers (see below). It has to be used when A has not changed but the boundary conditions have.
- `= 1`: Compute $A_p x_p$ and factorise the system matrix A. Both A and the boundary conditions are assumed to have changed in this case.
- `= 2`: Use previous values of $A_p x_p$ and do not factorise A. In this case, neither the boundary values nor the system matrix have changed.

`KIMPO`: Code to prescribe fixity conditions.

- `= 0`: The RHS is assumed to be already modified on input.
- `= 1`: The RHS is modified internally. The product $A_p x_p$ is computed and subtracted from the RHS.

`IPROB`: Problem number. *All* the tasks must be done for each problem dealt with in the run (see `MatMam.h` for the maximum number of problems that can be treated at the same time).

`INTE1`: Its meaning is different according to the task.

For Task 3:

- `= NVALU`: For direct solvers: Number of values to be assembled. See the meaning of INTE2 for Task 3.
- `= NVALO`: For EE iterative solvers: dimension of the matrix to be assembled.

For Task 4:

- `= ISTEP`: Time step counter.
- `= 0 :` The time step will not be indicated in the output file.

`INTE2`: For Task 3:

- `= IDEST`: For direct solvers: destination of the values to be assembled. Its dimension is (2*INTE1) and must be built up by the main code. Its components are:

  IDEST(2*IVALU – 1) : Row destination of IVALU.

  IDEST(2*IVALU) : Column destination of IVALU.

- `= IELEM`: For EE iterative solvers: element to which the given matrix (stored in VALUE) belongs.

For Task 4:

- `= IITER`: Iteration counter .
- `= 0 :` The iteration will not be indicated in the output file.

`REAL1`: For Task 3:

- `= VALUE`: For direct solvers: Values to be put in the global matrix. Dimension (INTE1).

  For EE iterative solvers: Element matrix to be stored. Dimension (INTE1,INTE1).

For Tasks 1 and 4:

- `= CPUTI`: Cpu time (to be used by the main code, if wanted).

`ITASK`: Task number to be performed by `MatMan` :

- `= 1`: Renumber of the equations.
- `= 2`: Memory allocation for the system matrix of problem IPROB.
- `= 3`: Construction of the matrix.
- `= 4`: Solution of a linear system.
- `= 5`: Deallocate memory for problem IPROB.

3.2 Variables to be used in each call

The variables to be employed in each call are indicated below. To emphasize which are used in a particular task and which are not, the argument for those that are not has been called `dummy` for real variables and `idumy` for integers.

```
 call MatMan(lnods,iffix,dummy,dummy,ndofn,nnode,nnodt,
.            idumy,nelem,npoin,ntotv,idumy,lures,idumy,
.            idumy,iprob,idumy,idumy,cputi,    1)

 call MatMan(lnods,idumy,dummy,dummy,ndofn,nnode,nnodt,
.            nnodw,nelem,npoin,ntotv,nsist,lures,idumy,
.            kimpo,iprob,idumy,idumy,cputi,    2)

For direct solvers:

 call MatMan(idumy,idumy,dummy,dummy,idumy,idumy,idumy,
.            idumy,idumy,idumy,idumy,idumy,idumy,idumy,
.            idumy,iprob,nvalo,idest,value,    3)

For iterative solvers:

 call MatMan(idumy,idumy,dummy,dummy,ndofn,nnode,nnodt,
.            nnodw,nelem,idumy,idumy,idumy,idumy,idumy,
.            idumy,iprob,nvalo,ielem,value,    3)

 call MatMan(lnods,iffix,rhsid,unkno,ndofn,nnode,nnodt,
.            nnodw,nelem,npoin,ntotv,nsist,lures,kfact,
.            idumy,iprob,istep,iiter,cputi,    4)

 call MatMan(idumy,idumy,dummy,dummy,idumy,idumy,idumy,
.            idumy,nelem,idumy,idumy,idumy,idumy,idumy,
.            idumy,iprob,idumy,idumy,cputi,    5)
```

4 Options available

4.1 List of options

The options to be used for each problem must be defined in a routine at the beginning of the code that must be written by the user. This routine must have the structure described next. Note that it must have the information for *all the problems* to be dealt with when it is called, and it can be called *only once*. Concerning the type of schemes for solving linear systems available, the options are indicated next, where the standard acronyms used in computational linear algebra are employed and the numbers in parenthesis refer to the remarks below:

Direct methods:

Solver type:

LDU decomposition (1)

LDU decomposition with pivoting (2)

Matrix storage:

Banded

Profile (= Skyline)

Sparse

Iterative methods:

EE methods (3):

Conjugate gradients

GMRES

Preconditioning for EE methods:

No preconditioning

Diagonal

SK methods (4):

Conjugate Gradient Method

CG Method (Normal Residual equation)

Bi-Conjugate Gradient Method

BCG with partial pivoting

BCG stabilized

Transpose-Free Quasi-Min. Res. meth.

Full Orthogonalization Method

Generalized Minimum RESidual method

Flexible version of GMRES

Direct versions of Quasi Gener. Min.

Preconditioning for SK methods:

No preconditioning

Diagonal

Left ILUT

Right ILUT

Left and right ILUT

Remarks:

(1) Possible storage: banded, skyline. For banded storage, the system matrix must be positive definite.

(2) Possible storage: sparse

(3) These methods employ an element by element storage of the system matrix. We refer to them as EE methods.

(4) These schemes are based on the SPARSKIT library.

4.2 Subroutine to load the options

The subroutine to be written must have the form described next (the name is obviously optional). This routine must be included by the user in his main code. Its only purpose is to inform `MatMan` of the options to be used for each problem. These options are stored in the file `MatMan.h` described later on.

```
      subroutine inisol
c***********************************************************************
c*
c*    This subroutine initializes the varibles of MatMan.h. All the
c*    arrays correspond to variables that must be provided for each
c*    problem to be solved.
c*
c***********************************************************************
      implicit none
      include 'MatMan.h'

c     WRITE THE DECLARATION OF VARIABLES HERE

c
c***  Length of pointers. It may be either 4 or 8, depending on the
c***  architecture of the computer:
c
      lengp =
c
c***  The following variable is used both for the matrix storage and
c***  the preconditioning strategy. For each problem to be solved,
c***  give the value to KMSIP:
c
      kmsip(1) = !  0   Banded                    -
                 !  1   Skyline                   | Direct solvers
                 !  2   Sparse                    -
                 ! -1   No preconditioning        | EE Iterative solvers
                 ! -2   Diagonal preconditioning  -
                 !  0   No preconditioning        |
                 !  1   Left ILUT precond.        | SK Iterative solvers
                 !  2   Right ILUT precond.       |
```

```
                ! 3   Left and right ILUT       -
c
c***  Code for matrix symmetry (repeat for each problem to be solved):
c
      ksymm(1) = ! 0    Symmetric problem
                 ! 1    Non symmetric problem
c
c***  Code for the renumbering strategy (only used for direct methods):
c
      krenu(1) = ! 0    Do not renumber
                 ! 1    Do renumber
c
c***  Multiplier for sparse matrix dimension. The coefficient to estimate
c***  the fill-in produced in the elimination process when a sparse matrix
c***  storage is used is loaded in:
c
      nmult(1) = ! 3    is recommended
c
c***  Type of solver (repeat for each problem to be solved):
c***  10 < KITES < 21   Direct methods
c***  20 < KITES < 31   Iterative methods with element by element
c***                    storage of the system matrix
c***  30 < KITES < 40   Iterative methods with sparse storage of the
c***                    system matrix. They are based on the SPARSKIT
c***                    library.
c
      kites(1) = ! 11   LDU decomposition
                 ! 12   LDU decomposition with pivoting
                 ! 21   Conjugate gradients
                 ! 22   GMRES
                 ! 31   CG  Conjugate Gradient Method
                 ! 32   CGNR Conjugate Gradient Method (Normal Residual
                 !      equation)
                 ! 33   BCG Bi-Conjugate Gradient Method
                 ! 34   DBCG BCG with partial pivoting
                 ! 35   BCGSTAB  BCG stabilized
                 ! 36   TFQMR Transpose-Free Quasi-Minimum Residual
                 !      method
                 ! 37   FOM Full Orthogonalization Method
                 ! 38   GMRES Generalized Minimum RESidual method
                 ! 39   FGMRES  Flexible version of Generalized Minimum
                 !      method
                 ! 40   DQGMRES Direct versions of Quasi Generalize
                 !      Minimum
c
c***  Maximum number of iterations for iterative solvers (not used
```

```
c***  for direct methods).
c
      itmax(1) =
c
c***  Tolerance for iterative solvers (not used for direct methods).
c
      toler(1) =
c
c***  Krylov sub-space for GMRES
c
      kkryl(1) =
c
c***  Threshold for ILUT preconditioners
c
      thres(1) =
c
c***  Fill-in parameter for ILUT preconditioners
c
      lfill(1) =
c
c***  Title of the problem
c
      wosol(1) ='                '
c
c***  Bubble nodes elimination. Inform MatMan if the degrees of
c***  of freedom corresponding to nodes internal to the element
c***  have been condensed or not:
c
      conde(1) = ! .false. or .true.

      end
```

4.3 The header MatMan.h

The file MatMan.h contains COMMON blocks used by MatMan . It is listed only for the user's information.

```
c*********************************************************************
c*
c*   This file contains the parameters needed by MatMan
c*
c*********************************************************************
      integer       mprob          ! maximum number of problems to
      parameter    (mprob=6)       ! be considered
```

```
c
c***  Pointers for the solver
c
      integer         lengp
      common/lenpoin/ lengp
      integer       iamat,         ! addresses for the matrices
     .              incol,         ! address for LNCOL
     .              inrow,         ! address for LNROW
     .              ialpn,         ! address for LPNTN
     .              ialpo,         ! address for LPONT
     .              iapma,         ! address for matrix A_p
     .              iapii,         ! address for ROFFS of A_p
     .              iapij,         ! address for APCOL of A_p
     .              iafma,         ! address for matrix A_f
     .              iafii,         ! address for ROFFS of A_f
     .              iafij,         ! address for APCOL of A_f
     .              irhsx          ! address for RHSAX
      common/poinsoll/ iamat,incol,inrow,ialpn,ialpo,iapma,iapii,
     .                 iapij,iafma,iafii,iafij,irhsx
c
c***  Output unit
c
      integer        lusol
      common/outsol/ lusol
c
c***  Memory for the system matrix (ITASK=1)
c
      integer       nband(mprob), ! half-bandwidth
     .              kmsip(mprob), ! Type of matrix storage
     .              ksymm(mprob), ! Symmetric (=0) or Unsymmetric (=1)
     .              nonze(mprob), ! Number of non-zero elements in matrix
     .              nmult(mprob), ! Dimension multiplier for sparse matrix
     .              svome(mprob), ! Volatile memory employed
     .              speme(mprob), ! Permanent memory
     .              nzecp(mprob), ! Non-zero components of matrix A_p
     .              nzecf(mprob)  ! Non-zero components of matrix A_f
      common/buil/  nband,kmsip,ksymm,nonze,nmult,svome,speme,
     .              nzecp,nzecf
c
c***  Renumber (ITASK=2)
c
      integer       nequa(mprob), ! number of equations
     .              krenu(mprob)  ! renumbering required or not
      common/ren/   nequa, krenu
      logical*1     conde(mprob)  ! condensation of bubble nodes
      common/renl/  conde
```

```
c
c***  Iterative solvers
c
      integer       kites(mprob), ! type of iterative solver
     .              itmax(mprob), ! maximum number of iterations
     .              kkryl(mprob), ! Krylov subspace for GMRES
     .              lfill(mprob)  ! Fill-in for ILUT
      common/iter/  kites, itmax, kkryl
      real*8        toler(mprob), ! tolerance for convergency
     .              thres(mprob)  ! Threshold for ILUT
      common/rite/  toler,thres
      character*15  wosol(mprob)
      common/cite/  wosol
c
c***  Other parameters and flags
c
      real*8        zeroc
      parameter    (zeroc=1.0e-15) ! zero for comparisons
      integer       kmadi(mprob)   ! initialization of matrices
      common/oflag/ kmadi
```

5 How to use MatMan

The following indications of how to use MatMan assume that it is being installed in a UNIX environment. The modifications needed for other type of operating systems must be done by the user.

- Install the package in a directory called `matman.free` in your account. If you download it from my web site, you just have to untar the file `matman.free.tar` (`tar xvf matman.free.tar`).
- Include the path to find `MatMan.h` in your`.cshrc` file. It must be `[path of matman.free]/Sources/include`. Alternatively, you can put the whole path to the file `MatMan.h` in the routine `inisol.f` described above.
- 'Make' MatMan . In the directory `matman.free` you will find a simple `Makefile` that has been tested in many UNIX platforms (I had some more elegant versions, but this one always works!). This `Makefile` can be modified at your convenience to create a set of libraries of the form `Lib_matman_(option).a`.
- Write the routine `inisol.f` (or any name you like) in your finite element code. This is going to be *the only link* between your code and MatMan . As you have seen from the listing of this routine suggested above, the only thing you have to

do is to fill in the data that `MatMan` needs and that you have had to read from your code. Note that this routine contains the file `MatMan.h`.

- In your code, include the necessary calls to `MatMan` .
- Link your code together with the libraries `Lib_matman_(option).a`.
- ...enjoy!