A User's Guide for the CADYF-SPLIB Project*

Xiaoge Wang
Randall Bramley
Department of Computer Science
Indiana University - Bloomington

June 14, 1995

Abstract

The CADYF-SPLIB project is a continuing effort to develop practical linear system solver strategies for difficult problems from incompressible flows in computational fluid dynamics (CFD). CADYF is a Fortran package for solving steady state incompressible flows using the finite element method. The package was developed and directed by Dominique Pelletier of Ecole Polytechnique in Montreal. SPLIB is a heavily instrumented package for solving large, sparse nonsymmetric systems of equations. SPLIB consists of 13 iterative methods and 7 preconditioners, developed and implemented by Randall Bramley and Xiaoge Wang at Indiana University. This report is a user's manual describing the interface between CADYF and various linear solver packages, including SPLIB.

Beginning in 1989, a collaborative effort began between research groups at Ecole Polytechniqe in Montreal and the Center for Supercomputer Research and Development at the University of Illinois. Later, with the move of Randall Bramley to Indiana University followed by the award of an Indiana based NSF CISE Postdoctoral Fellowship to Xiaoge Wang, the second group was centered in the Computer Science Department of Indiana University. The Montreal research group

*Supported by NSF Grant CDA-9309746

is led by Dominique Pelletier and includes Jean-Francoise Hetu and collaborative advice from Andre Garon and Michel Fortin.

The Montreal group has developed a finite element based computer program called CADYF for solving steady state, incompressible fluid flow problems. Compressible flows often can be solved using explicit methods that do not require solving large linear systems, but the constraint of incompressibility together with meshes with high degrees of refinement tend to require implicit methods. Furthermore, steady state problems generally yield linear systems which are extremely difficult to solve, unlike time dependent problems which essentially can have a large multiple of the identity matrix added to the coefficient matrix. This tends to shift the eigenvalues of the coefficient matrix to the right half plane and away from the origin, making the systems easier to solve with iterative methods.

Steady state incompressible flow programs tend to spend the vast majority of computer time in setting up and solving a sequence of these large, sparse linear systems with poor eigenvalue distributions. Because of this, the Indiana group concentrated on developing a testing environment that would allow

1. Modular replacement of linear solver methods in CADYF
2. Rapid testing of new linear solvers and their preconditioners
3. The extraction of realistic (as opposed to mathematically idealized) linear systems, for detailed analysis and sharing with other researchers in numerical linear algebra
4. The development of robust solution strategies for incompressible fluid flows.

This required developing extraction routines, which write out a linear system to a Harwell/Boeing formatted [5] file with the matrix stored in compressed sparse row format (instead of the compressed sparse column required by the original Harwell/Boeing format). In order to provide a wide variety of comparisons for different linear solvers, an interface was developed to allow the use of MA28 [4], SPARSPAK [3], the native skyline solver of CADYF, and SPLIB, a heavily instrumented package of iterative solvers and preconditioners. A fifth package, called STOKES, allows solving the generalized Stokes problems in the so-called mixed formulation, using projection methods developed in [1, 2]. The first four packages can be used to solve the linear systems from either the augmented Lagrangian formulation or a mixed formulation.

This report gives details of the changes and additions needed to develop this research machinery, and provides a rudimentary user's guide for running CADYF with different linear solver packages. Section 1 gives the instructions on how to use the combined package, and some details about the necessary control parameters. Section 2 gives an overview of the structure of CADYF and where the linear solver interface fits in. Section 3 is a listing of the subroutines either changed or added to CADYF to implement the project.

1 Instructions for using CADYF with the interface to several linear solver packages

CADYF with an interface to several linear solver packages is referred to here as CADYF version 7.0. The usage of CADYF version 7.0 is similar to that of previous versions. Version 7.0 contains an extension to allow the direct assembly of matrices into the compressed sparse row (CSR) sparse data structure, and an interface to a set of packages of linear system solvers, including direct and iterative methods. Running CADYF version 7.0 is similar to the execution of earlier versions, with changes that allow specifying the linear solver to use. First we briefly describe the usage of previous CADYF, and then describe the usage of CADYF version 7.0.

1.1 Usage of CADYF

The source code of CADYF is compiled and the executable code is stored in the directory where it will be run by means of the makefile. A script file runs the executable code and manages the placement, linkage, and removal of input and output files. To run the code, use

script_name input_name

where *script_name* is the name of the script file and *input_file* is the name of the input file. The input file has to have *.cadyf* as a suffix. The suffix must not be present in the command line, or the script will look for the file *input_name.cadyf.cadyf* as input.

Currently, the executable files are stored on the Indiana University SGI Power Challenge (caledonia) in the directory

$$/home/bramley/work/cadyf/bin.$$

Three versions, v5.0, v6.0 and v7.0 are currently saved. The script files and executable files for these three versions are:

cadyf: Script file for v5.0.

cadyf.exe: executable file for v5.0.

cadyfs: Script file for v6.0.

cadyfs.exe: executable file for v6.0.

cadyfs_v7: Script file for v7.0.

cadyfs_v7.exe: executable file for v7.0.

The format of *.cadyf* files for versions v5.0 and v6.0 are the same. The difference between v5.0 and v6.0 is that v6.0 supports directly assembling matrices into the sparse data structure CSR. The selection of skyline or CSR data structure is set in the include file *skyline.inc*, and any change of data structure selection requires a recompilation of the executable file. The format of a *.cadyf* file for version 7.0 is slightly different from that of the previous versions. Extra lines in the *.cadyf* file contain parameters that specify that selection of the package, control the execution of the package selected, etc. Note that input files from other CADYF users (Dominique Pelletier) can not be used without modification in v7.0. Details of how to modify those files to run in v7.0 are given in Section 1.3. Similarly, input files generated by the Indiana local version of AMIRAL can not be directly used in v5.0 or v6.0.

The extra information needed can be supplied in two ways: generated by the preprocessing code AMIRAL (local version) or by editing the *.cadyf* file, which is used for CADYF v5.0 or v6.0. The details of these two method are presented in the following subsections respectively.

1.2 Specifying the Interface in AMIRAL Input File

AMIRAL is the mesh generator for CADYF, which parses a simple description of the problem parameters and reads the geometry of the domain and then creates the finite element connectivity and node lists to set up a *.cadyf* file, which in turn is directly read by CADYF.

AMIRAL has been extended to generate the *.cadyf* files with specification of linear solvers. This new version of AMIRAL is currently stored in the directory

$$/home/bramley/work/amiral$$

It includes the source code in subdirectory *amiral*, executables in *bin*, library archive files in *lib*, and other related libraries.

Because CADYF has an innovative adaptive mesh option, there are two ways of using AMIRAL: to generate the $.cadyf$ file for an initial mesh and to generate a $.cadyf$ file for an adaptedmesh.

To generate an initial mesh, use

amiral name_1 name_2

where *amiral* is the name of the script file, *name_1* is the name of method specification file and *name_2* is the name of the grid specification file. To generate an adapted mesh, use

amiral name_1 name_2 name_3

where *amiral* is the name of the script file, *name_1* is the name of method specification file, *name_2* is the name of the grid specification file and *name_3* is the name of the file carrying the requisite interpolation information. *name_1* file has to have the suffix $.amiral$, *name_2* has suffix $.backm2d$ and *name_3* has suffix $.cad5out$. Those suffixes are not typed in the command line that executes the script, instead being supplied by the script file.

The interface information can be specified in *name.amiral* file, Then the specification in the CADYF input file will be automatically set up along with the other data in the corresponding format.

The format of the interface specification in an AMIRAL input file will be explained later. First an example of the specification follows. Note that in the text below, a line number is placed in parentheses to the left of each line so that we can refer to specific lines. However, those line numbers are *not* part of the actual AMIRAL file.

Example: The following example shows the portion of the interface specification in a *.amiral* file. The coefficient matrix of the linear system from the discretization is to be put into a compressed sparse row (CSR) format. The selected package is STOKES, which is the projected Stokes solver. The preconditioner ILU(s) is selected with zero level of fill-ins. The iterative solver is chosen to be GMRES(m) with a stopping tolerance equal to 0.000001. The maximum allowed iterations is 1000. The rank of Krylov subspace to allow in GMRES(m) is $m = 20$. The instrumentation level of STOKES is set to be 2, which causes XGRAPH output files to be generated by STOKES that show the convergence history of the iterative linear solver. All of this information is contained in lines (2) to (13), and is further documented in the code STOKES.F. Note that "fin" is French for "end", and, e.g., "du bloc parameters" means "of block parameters". Franglais is the natural language of CADYF-SPLIB.

```
(1)  Interface
(2)     matrix         = CSR
(3)     package        = STOKES
(4)     var_splib
(5)         precondition = ILUS
(6)         solution     = GMRES
(7)         tol          = 0.000001
(8)         max_iter     = 1000
(9)         krylov_size  = 20
(10)         integer_p    = 0
(11)         real_p       = 0.1
(12)         instlvl      = 2
(13)     fin  ! du bloc parameters
(14)     var_sppak
(15)         order = MMD
(16)     fin  ! du bloc ordering
(17)     var_ma28
(18)         omega = 0.01
(19)         drop  = 1.0d-10
(20)     fin  ! du bloc ordering
(21)     dump  = ( 3, 4 )
(22)  fin
```

Lines (1) and (22) are the start and stop specifiers of the interface block. All of the lines between them contain the interface information. Lines (14) to (16) contain information giving the ordering method selection for the package SPARSPAK, which is called by STOKES to compute the factorization of certain submatrices. SPARSPAK can also be called directly to solve the entire linear system, by specifying package = SPARSPAK. Lines (17) to (20) are for specification of parameters when MA28 is used. Although the package selection above is set to be STOKES in line (3), it is valid to have additional information such as the MA28 parameters. The data so generated into the *.cadyf* file output by AMIRAL will simply be ignored by CADYF. All of the keywords (CSR, ILUS, etc) can be in upper, lower, or mixed case – AMIRAL will convert them internally to be upper case.

A complete interface specification consists of a begin and an end statement as in lines (1) and (22). Each interface block has 4 parts.

(A) Matrix format:

```
matrix = SKYLINE | CSR
```

The valid value of *format* are: SKYLINE and CSR only. All other values will be treated as invalid.

(B) Package selection:

```
package = SKYLINE | MA28 | SPARSPAK | SPLIB | STOKES
```

All others will be considered as unknown packages.

(C) Parameters for package MA28: The specification for the variables of MA28 that need to be set by users begins with the statement *var_ma28* and ends with *fin* as follows:

```
var_ma28
    omega = real
    drop  = real
fin
```

Here *omega* is a parameter for Markovitz pivoting. It can be any value between 0 and 1, and 0.1 is usually recommended [4]. *drop* is the tolerance for numerical dropping. It can be any real number between 0 and 1. Nondiagonal elements computed during the factorization will be dropped if their magnitudes are smaller than this tolerance (see the documentation with MA28 for further details).

(D) Variables of package SPARSPAK: The specification for parameters in SPARSAPK that need to be set by users begins with the statement *var_sppak* and ends with *fin* as follows:

```
var_sppak
    order = MMD | RCM | ONEWAY | RQ | ND
fin
```

The only parameter that needs to be set for SPARSPAK is the selection of reordering method. MMD here stands for minimum degree ordering, RCM for reverse Cuthill McKee ordering, ONEWAY for oneway dissection ordering, RQ for refine quotient tree ordering and ND for nested dissection ordering. All the other names will be considered as unknown method. For the type of problems generated by CADYF, MMD is usually recommended.

(E) Variables of package SPLIB or STOKES: The specification for parameters in SPLIB that need to be set by users begins with the statement *var_splib* and ends with *fin*. Since STOKES uses SPLIB as its internal solver, this block is also needed in STOKES to select the methods for solving the problem. The format of this block is as follows:

```
var_splib
    precondition = NONE | ILUS | MILUS | ILUT | SSOR |
                   TRIDIAG | ILU0 |EIMGS
    solution     = BICG | CBNE | CGNR | CGS | CGSTAB |
                   GMRES | QMRTF | BICGS | GMREST |
                   JACOBI | GAUSS | SOR | ORTHOMIN
    tol          = real
    max_iter     = integer
    krylov_size  = integer
    integer_p    = integer
    real_p       = real
    instlvl      = integer
fin
```

SPLIB has 7 preconditioners and 13 iterative solvers, but only two of the solvers (CGSTAB and GMRES) are available in STOKES. *tol* is the stopping condition test. *max_iter* is the maximum number of iterations allowed. If the norm of relative residual is less than *tol* or the number of iteration reaches *max_iter* then the iterations are stopped. *krylov_size* is the size of Krylov subspace. It must be specified if the solution method is chosen to be GMRES, GMREST or ORTHOMIN. $integer_p$ and $real_p$ are integer and real parameters for the computation of the preconditioner. $integer_p$ is the number of levels of fill-in allowed for ILUS and MILUS, and for ILUT it is the number of fill-in elements allowed in each row. $real_p$ is a dropping tolerance if the preconditioning method uses a numerical dropping strategy to control the fill-in, as ILUT and EMGS do. *instlvl* is the level of the instrumentation for the iterative method. The meaning of each level can be find in the description of the packages SPLIB or STOKES respectively. Since the instrumentation can take a lot time (in some cases doubling the time required), instrumentation level should be set to 0 if details of the convergence behavior are not needed.

(F) Matrix output:

```
dump = ( integer1, integer2 )
```

This statement specifies when the matrix will be written out to external files for later testing and analysis. In CADYF, the nonlinear solver has two phases, each possibly empty: the first one uses a successive substitution method and the second phase uses the Newton-Raphason method. $integer1$ can be 1 or 2 corresponding to the choice of these two phases. $integer2$ can be any integer between 1 and the maximum allowed step of the corresponding phase. For example, if $dump = (1, 2)$, then the matrix will be extracted on the first phase's second step. If either of the integers is zero or out of the range (negative or greater than actual number of steps), then no matrix will be output, which is equivalent to the case that the matrix output is not specified in the file.

Given this information, AMIRAL generates a $.cadyf$ file with additional lines that the CADYF-SPLIB interface routine needs. Next we show how to directly edit an already existing $.cadyf$ file to include that information without using a modified AMIRAL.

1.3 Insert Interface Specification into the CADYF Input File

The second way of specifying the choice of linear solver and related parameters is by inserting the specifications directly into the CADYF input file; this can be useful for modifying existing input files without having to run AMIRAL again. The format is given below.

The data specifying the interface is placed just before the node description. It usually starts from line 17 of the CADYF input file, and the number of added lines varies, depending on the amount of information specified by AMIRAL and the methods chosen. There is at least one line containing 4 integers followed by 4 logical parameters. The value of the four integer control parameters are

1st: matrix format

```
= 0, skyline format.
= 1, CSR format.
```

2nd: package selection:

```
= 1, skyline solver.
= 2, MA28.
= 3, SPARSPAK.
= 4, SPLIB.
= 5, STOKES.
```

3rd: first integer in matrix output specification. It specifies on which phase of nonlinear solution the matrix will be extracted.

4th: second integer in matrix output specification. It specifies on which step number of the nonlinear solution the matrix will be extracted.

The values of the four logical control parameters are:

1st: matrix output:

```
= T, there is a request for output matrix.
= F, there is no request for output matrix.
```

2nd: parameters for SPLIB

```
= T, there is a specification for SPLIB.
= F, there is no specification for SPLIB.
```

3rd: parameters for SPARSPAK.

```
= T, there is a specification for SPARSAPK.
= F, there is no specification for SPARSPAK.
```

4th: parameters for MA28.

```
= T, there is a specification for MA28.
= F, there is no specification for MA28.
```

After the first line with four integer and logical parameters, the number of lines and amount of data on them depends on the values of the logical parameters. If the 2nd logical number is T then 8 numbers will follow specifying the SPLIB parameters. The meaning of these numbers is as follows:

1st: SPLIB parameter: selection of solver (integer).

```
= 1, BICG
= 2, CBNE
= 3, CGNR
= 4, CGS
= 5, CGSTAB
= 6, GMRES
= 7, QMRTF
= 8, BICGS
= 9, GMREST
= 10, JACOBI
= 11, GAUSS
= 12, SOR
= 13, ORTHOMIN
= other, unknown
```

2nd: SPLIB parameter: selection of preconditioner (integer).

```
= 0, NONE
= 1, ILUS
= 2, MILUS
= 3, ILUT
= 4, SSOR
= 5, TRIDIAG
= 6, ILU0
= 7, EIMGS
= other, unknown
```

3rd: SPLIB parameter: integer parameter for preconditioner. It can be the level of fill-ins allowed (ILUS) or the number of fill-in elements in each row (ILUT).

4th: SPLIB parameter: maximum size of the Krylov subspace.

5th: SPLIB parameter: maximum allowed number of iterations (integer).

6th: SPLIB parameter: real parameter for preconditioner. It controls the dropping in ILUT. For EIMGS, this gives the levels of fill to use; if the real parameter is of the form x.y with y ¿ 0, then x levels of fill are allowed in EIMGS. If the real parameter is 0.0 in EIMGS, then zero levels of fill are used. If it is between 0.0 and 1.0, then a nonzero pattern sparser than that of A is enforced on the preconditioner.

7th: SPLIB parameter: tolerance for iteration termination test.

8th: SPLIB parameter: level of instrumentation (integer).

If the third logical number from the first line is T, then an integer number follows to specify the selection of the ordering method in SPARSPAK. The value of the number indicate the method to use:

```
= 1, RCM
= 2, ONEWAY
= 3, RQ
= 4, ND
= 5, MMD
= other, unknown
```

If the fourth logical number is T, then two real numbers follow , *omega* and *drop*, which specify the MA28 parameters.

Note that if, for example, the second and third parameters are both T, then at least 9 numbers should follow the first interface line. Those numbers need not be on a single line, but the modifications to CADYF will attempt to read 9 numbers and errors will occur if they are not provided.

For convenience, a short data file named *append* is stored in the directory. If the user needs to create a input file from an input file for a previous version of CADYF, use a text editor to insert *append* beginning at line 17 the *.cadyf* file and change the values of the parameters as needed. The contents of file *append* are:

```
          0           1           0           0 T T T T
          5           1          10          20        1000
0.1000000000000000      9.9999999999999995E-07            4
          5
1.0000000000000000E-02 1.0000000000000000E-02
```

This contains the maximum possible information ever needed. Some data may not be used for a given run, but it is easy to edit the added lines to handle different cases. For example, the *append* file shown above has as useful data only the first two integers. They specify using skyline format and the skyline solver. Matrix extraction is not requested because the third and fourth integers are both zero.

Suppose next we need to run a test case that uses CSR data structure and an SPLIB solver with ILU(2) as the preconditioning method and CGSTAB as the iterative method. The stopping criteria is to be 10^{-5} and the maximum allowed number of iterations is to be 300. Then change the first integer to 1 and second to 4 and set the numbers in the second and third lines to be

```
         5              1             2             20            300
  0.1000000000000000        1.0000E-05
```

The fourth integer in the second line (20 here) is for the size of Krylov subspace. It is irrelevant for this test case, since CGSTAB does not explicitly store a basis for the Krylov subspace (this kind of information is provided in the documentation for SPLIB). The first real number in the third line (0.1 here) is for preconditioners that require a real number as parameter. It is irrelevant also in this test case, since ILU(s) does not require a real parameter. To use the package STOKES instead, set the second integer to be 5 and all the others identically as in using SPLIB.

To run a test case that uses SPARSPAK with certain preordering, one need only set the first integer to 0 (using skyline data structure) or 1 (using CSR format), the second integer to be 3 and the integer in fourth row to be the number that corresponds to the chosen ordering method. For example,

```
         1              3             0              0 T T T T
         5              1             10            20            1000
  0.1000000000000000        9.9999999999999995E-07              4
         5
  1.0000000000000000E-02 1.0000000000000000E-02
```

uses CSR data structure and SPARSPAK with a minimum degree ordering.

The two real numbers in the last row are for the MA28 package. To run MA28, the second integer in the first row needs to be set to 2.

2 Structure of CADYF-SPLIB Interface

The structure of the interface is shown in the following graph. From the graph we can see that the interface between CADYF and other linear solver packages are simple and clear. First, in the data-read phase of CADYF, the subroutine *inintf* is called to read in the specification of the selection of the package and the control parameters of the chosen package, as described in Section 1. After that it return to CADYF. At the point of subroutine *nrss.f* where CADYF needs the linear solver, it calls *interf.f* and gives control to this interface routine. *interf.f* will set up the data structure according to the selection of the package and then pass control to the package. After the linear system is solved (or the solver fails), the package returns the solution and control to the interface. Interface routine *interf.f* will then transform the solution into the internal form of CADYF and return the solution and control to CADYF. Note that the call to *inintf.f* is only once but typically *interf.f* will be invoked many times because multiple nonlinear iterations, each requiring the solution of a linear system, may be required.

$$
\begin{array}{ccl}
\text{CADYF} & & \text{INTERFACE} \\
\text{start} & & \\
\vdots & & \\
\vdots & & \\
\text{call indata} & \Longleftrightarrow & \text{inintf.f} \\
\vdots & & \\
\vdots & & \\
\text{call nrss} & \Longleftrightarrow & \text{interf.f} \left\{ \begin{array}{lcl} tskl.f & \longrightarrow & SKYLINE \\ tma28.f & \longrightarrow & MA28 \\ tsppak.f & \longrightarrow & SPARSPAK \\ tsplib.f & \longrightarrow & SPLIB \\ tstokes.f & \longrightarrow & STOKES \end{array} \right. \\
\vdots & & \\
\vdots & & \\
\text{end} & &
\end{array}
$$

In the next Section, we recommend you start with the tree hierarchy in looking at changed/added subroutines. So begin with TSPPAK.F, then examine the routines it calls, etc.

3 Added or Modified Routines in CADYF

All the subroutines in the directory */home/user5/xwang/Report/CFD/Cadyf* will be specified in the section. The description includes the following:

Name of the file that contains the subroutine.

Usage of the subroutine.

Function of the subroutine.

List of parameters (calling sequence).

List of include files

List of subroutines or functions called by the subroutine.

The subroutines are listed in alphabetic order. This list is to provide a quick reference and brief information. Details of the implementation of each subroutines are not discussed here.

File Name: blkperm.f:
Usage: call blkperm(n, nv, np, G, cind_G, rptr_G, perm, invp, nerror)
Function: Permute the matrix G to get the form of $\begin{bmatrix} A & B \\ C & 0 \end{bmatrix}$.

Parameters:

n: size of problem G.

nv: number of velocity unknowns.

np: number of presure unknowns.

G, cind_G, rptr_G: CSR form of matrix G.

perm(n): permutation vector. (i,j) permuted to (perm(i), perm(j)).

invp(n): inverse of perm.

nerror: error code.

Include files:

blank.inc tables.inc disks.inc intf.inc

Subroutines:

File Name: caddvr.f
Usage:
Function: Main program of CADYF.
Parameters:
Include files:

blank.inc

Subroutines:

cadyf

File Name: cadyf.f
Usage: call cadyf
Function: CADYF main subroutine.
Include files:

iobyte.inc	itwo.inc	blank.inc	crdtyp.inc	disks.inc
dof.inc	eldof.inc	eqsize.inc	execut.inc	header.inc
neqns.inc	prbsiz.inc	problm.inc	timing.inc	skyline.inc

Subroutines:

ADDTBL	LOCTBL	GETSEC	INITBL	PROB
INDATA	POST1	SETREA	LISTBL	MODSLV
RMVTBL	RECPF	OUTVEL	OUTPF	OUTPUT
SAV3D2	SAV2D2			

File Name: convert_csr.f
Usage: call convert_csr(n, nnz, diag, A, r_point, c_ind, Acsr)
Function: Convert a matrix in skyline format to CSR format, squeezing out explicitly stored zeros from the skyline data structure.
Parameters:

integer n: size of matrix

integer nnz: number of non-zeros. Could be changed if there are explicit zeros in the original skyline matrix form.

n, nnz, diag, A: matrix A in skyline format.

n, r_point, c_ind, Acsr: matrix A in CSR format

Include files:
Subroutines:

File Name: csr2coord1.f
Usage: call csr2coord1(n, nnz, A, rptr, cind, aa, irn, icn)
Function: Set up the coordinatewise data structure for MA28 from a CSR input matrix.
Parameters:

n: size of the matrix.

nnz: number of non zeros of the matrix.

A, rptr, cind: input matrix in CSR format

aa, irn, icn: pointers to the arrays containing the coordinatewise data structure of the input matrix.

Include files:

blank.inc   tables.inc   disks.inc   execut.inc   ma28.inc

Subroutines:

addtbl   dcopy   icopy

File Name: csr2coord2.f
Usage: call csr2coord2(n, nnz, A, rptr, cind, aa, ivect, jvect)
Function: Set up a coordinatewise data structure for MA28 using a previously determined pivoting set. This differs from csr2coor.f by setting up a shorter vector (does not include the working space for computing pivot sequence).
Parameters:

n: size of the matrix.

nnz: number of non zeros of the matrix.

A, rptr, cind: matrix in CSR format.

aa, ivect, jvect: pointers to the tables that contains the coordinate wise data structure of the matrix.

Include files:

blank.inc tables.inc disks.inc execut.inc ma28.inc

Subroutines:

loctbl addtbl dcopy icopy

File Name: csr2spaks.f
Usage: call csr2spaks(n, rptr, cind, spcon, spmap, work, isize)
Function: Input the matrix structure for SPARSPAK from a matrix in CSR format.
Parameters:

n: size of the matrix.

rptr, cind: structure of the matrix in CSR form.

spcon, spmap: array for control parameters passed to SPARSPAK. These are common blocks in the original version of SPARSPAK, changed to arguments here.

work: double precision array for working space in SPARSPAK.

isize: size of working space.

Include files:

sparspak.inc	spksys.inc	crdtyp.inc

Subroutines:

sprspk	ijbegn	inij	ijend	ordrb6
ordra2	ordra4	ordrb4	ordra6	

File Name: csr2spakv.f

Usage: call csr2spakv(n, A, rptr, cind, spcon, spmap, work, isize)

Function: Read in numerical values of the matrix into SPARSPAK from the input matrix in CSR format.

Parameters:

n: size of the matrix.

A, rptr, cind: matrix in CSR format.

spcon, spmap: array for control parameters passing in SPARSPAK.

work: double precision array for working space in SPARSPAK.

isize: size of working space.

Include files:

sparspak.inc spksys.inc

Subroutines:

inaij6 inaij2 inaij4

File Name: dyf.f
Usage:
Function: block data of CADYF.
Parameters:
Include files:

disks.inc

Subroutines:

File Name: formeq.f

Usage: call FORMEQ (prop , mprop , frstel, disdiv, solver, nodes , groupe, nbritr, propg , nparg , numstp, numitr)

Function: driver for computing element matrices for all flows :

- 2-d
- axisymmetric
- cylindrical (axisymmetric swirling flows)
- 3-d

formulations :

mixed

penalty

stationary

Parameters:

disdiv: maximum value of discrete divergence for all elements.

prop(i): array of element group fluid properties

Include files:

mxintp.inc	blank.inc	crdtyp.inc	disks.inc	dof.inc
eldof.inc	elgrp.inc	neqns.inc	option.inc	prbsiz.inc

Subroutines:

ADDTBL	LOCTBL	CAREL	PROB	NS2d
NSAXI	NSCYL	NS3D	RMVTBL	

File Name: get_perm.f
Usage: call get_perm (neq, neqp, idp, perm, invp, nerror)
Function: get permutation vector p of length n from CADYF. The permutation vector then later will be used to permute the matrix G into block A, B and C form.
Parameters:

neq: size of the matrix. (length of vector ide).

neqp: length of vector idp.

idp: vector created by CADYF to store the information of the location of presure unknowns.

perm(n): permutation vector. (i,j) is permuted from (perm(i), perm(j))

invp(n): inverse of perm.

nerror: error code.

Include files:

prbsiz.inc disks.inc

Subroutines:

listbl prob

File Name: getnext.f

Usage:

integer int, getnext int = getnext()

Function: return index of the next available element of the work array from which the memory manager carves out tables in CADYF and SPLIB. This differs from addtbl in that it does not create a new array.

Parameters:

Include files:

blank.inc   tables.inc

Subroutines:

File Name: indata.f
Usage: call INDATA (vrsion)
Function: Reads nodal data and generates equation numbers.
Parameter:

vrsion

Include files:

blank.inc	blocks.inc	crdtyp.inc	disks.inc	dof.inc
echo.inc	eldof.inc	fixdbc.inc	neqns.inc	neuman.inc
option.inc	period.inc	prbsiz.inc	problm.inc	sophie.inc
timint.inc	timvol.inc			

Subroutines:

ADDTBL	CONTRL	INBLOK	INBLOK	INBLOK
INBLOK	INTFNS	RMVTBL	INPERI	ININTF
INNODE	INBLAD	INCONS	INLOAD	INVOL
SETREA	INITAL	INELEM	INGRAF	STRUCT

File Name: inintf.f
Usage: call inintf(nerror)
Function: Read in the control parameter for the interface of linear solvers.
The control data include: `imatrix: = 0: skyline =1: CSR ipakage: selection of pakages. = 1: SKYLINE = 2: Ma28 = 3: SPARSPAK = 4: SPLIB = 5: STOKESOL = other: unknown inumstp, inumitr: both equal 0 if no extration. both not equal 0, indicate the place where data will be extracted. idump: indicate if matrix needs to be dumped rsplib: indicate if the parameter for SPLIB are specified = .true. read in parameters. = .false. no parameter specification. rsppak: indicate if the parameter for SPARSPAK are specified = .true. read in parameters. = .false. no parameter specification. if (rsplib) then isolver: selection of iterative methods . = 1: BICG = 2: CGNE = 3: CGNR = 4: CGS = 5: CGSTAB = 6: GMRES(k) = 7: QMRTF = 8: BiCGS = 9: GMREST = 10: JACOBI = 11: GAUSS = 12: SOR = 13: ORTHMIN = others: unknown iprecon: selection of preconditioners in SPLIB. = 0: none = 1: ILU(s) = 2: MILU(s,real_p) = 3: ILUT(s,real_p) = 4: SSOR(real_p) = 5: TRID(s) = 6: ILU(-1) = 7: ECIMGS iparms(2): integers for preconditions and solvers. real_p: real numbers for preconditions. real_s: tolerance for stoping test. instlvl: instrumentation level. maxit: maximum iterations allowed. end if if (rsppak) then iorder: selection of matrix ordering in SPARSAPK = 0 use default method. (MMD) = 1 RCM = 2 one way dissection = 3 or refined quotient tree = 4 nested dissection = 5 mmd endif if (rma28) then u: omega for pivoting. endif`

Parameter:

nerror: error code.

Include files:

intf.inc	disks.inc	skyline.inc	splib.inc	ma28.inc
sparspak.inc	problm.inc			

Subroutines:

File Name: instr_direct_csr.f

Usage: call instr_direct_csr (n, nnz, space, A, rptr, cind, rhs, sol, t_fac, t_sol, iounit, work1)

Function: Print out performance information of the linear solver. This includes the computation of residual vector. This routine requires that the matrix in CSR format.

Parameters:

size of matrix: n

number of nonzeros: nnz

minimum space rquired: space

I/O unit: iounit

matrix A: A, rptr, cind (in CSR)

right hand side: rhs

solution: sol

time for factorization t_fac

time for solver t_sol

working space: work1(n).

Include files:

Subroutines:

amux dnrm

File Name: instr_direct_skl.f

Usage: call instr_direct_skl (n, nnz, space, A, diag, rhs, sol, t_fac, t_sol, iounit, work1)

Function: Print out performance information of the linear solver. This includes the computation of residual vector. This routine requires that the matrix in skyline format.

Parameters:

size of matrix: n

number of nonzeros: nnz

minimum space rquired: space

I/O unit: iounit

matrix A: A, diag (skyline format)

right hand side: rhs

solution: sol

t_fac: time for factorization

t_sol: time for for/backword solver.

working space: work1(n).

Include files:

Subroutines:

ax_skl dnrm2

File Name: Ax_skl.f
Usage: call Ax_skl(n, A, diag, x, Ax)
Function: Compute matrix vector multiplication $A * x$ for a matrix stored in skyline data structure.
Parameters:

n: size of the matrix A and x.

A, diag: matrix A in skyline format.

x: vector.

Ax: vector of A*x

Include files:
Subroutines:

File Name: interf.f
Usage: call interf(numstp,numitr,totalt)
Function: Create a interface for sparse linear systems solvers. It contains the calls to the following pakages:

(1) SKYLINE solver from cadyf

(2) MA28 direct solver.

(3) SPARSEPAK

(4) SPLIB iterative solvers including preconditioners.

(5) STOKES (projection method)

This can also dump out the data (including the matrix, right hand side vector and a returned solution) in CSR format.
Parameters:

numstp, numitr: parameters specify the nonlinear iteration on which to extract the matrix. Because a linear system is solved inside a nested double loop, numstp is the index of the outer loop and numitr is the index of the inner loop. The two parameters are passed into the interface to determine the position where the data is to be dumped out for further inverstigation.

totalt: total time in solving the linear system. It does not include the time for setting up the proper data structure and other cost. The overhead will be computed by subtracting the total time from the time of interf call.

Include files:

blank.inc	intf.inc	tables.inc	disks.inc	execut.inc
neqns.inc	crdtyp.inc			

Subroutines:

tskl	tma28	tsppak	tsplib	tstokes
prtmt_blocks	skl2csr	prtmt_global		

File Name: memleft.f
Usage: integer memleft ..... remains = memleft()
Function: Return number of integer words left in memory array.
Include files:

tables.inc blank.inc execut.inc

Subroutines:

File Name: mprtmt.f

Usage: call mprtmt(nrow, ncol, a, ja, ia, rhs, sol, guesol, title, key, type, ifmt, job, iounit)

Function: Writes a matrix in Harwell-Boeing format into a file. This assumes that the matrix is stored in COMPRESSED SPARSE ROW FORMAT. Right hand sides in full format can also be written out. It differs from the subroutine prtmt by Saad, but is based on it. This routine can take separate rhs and sol, instead of one combined vector.

Parameters:

nrow = number of rows in matrix

ncol = number of columns in matrix

a = real*8 array containing the values of the matrix stored columnwise

ja = integer array of the same length as a containing the column indices of the corresponding matrix elements of array a.

ia = integer array of containing the pointers to the beginning of the row in arrays a and ja.

rhs = real array containing the right-hand-side (s).

sol = the associated initial guesses or exact solutions in this order. See also guesol for details. the vector rhs will be used only if job .gt. 2 (see below). Only full storage for the right hand sides is supported.

guesol= a 2-character string indicating whether an initial guess (1-st character) and / or the exact solution (2-nd) character) is provided with the right hand side. if the first character of guesol is 'G' it means that an an intial guess is provided for each right-hand sides. These are assumed to be appended to the right hand-sides in the array rhs. if the second character of guesol is 'X' it means that an exact solution is provided for each right-hand side. These are assumed to be appended to the right hand-sides and the initial guesses (if any) in the array rhs.

title = character*72 = title of matrix test (character a*72).

key = character*8 = key of matrix

type = charatcer*3 = type of matrix.

ifmt = integer specifying the format chosen for the real values to be output (i.e., for a, and for rhs-guess-sol if applicable). Full explanation of ifmt is in the routine.

job = integer to indicate whether matrix values and a right-hand-side is available to be written
job = 1 write the arrays ja and ia only.
job = 2 write matrix including values, i.e., a, ja, ia
job = 3 write matrix and one right hand side: a,ja,ia,rhs.
job = nrhs+2 write matrix and nrhs successive right hand sides

iounit = logical unit number where to write the matrix into.

Include files:

Subroutines:

File Name: nrss.f

Usage: call NRSS (slvcod)

Function: iteration driver for iteration by newton-raphson or successive substitution method.

Parameters:

slvcod `= 0 iteration has converged = 1 iteration has diverged = 2 maximum number of iterations reached with no convergence.`

Include files:

mxintp.inc	skyline.inc	eqsize.inc	tables.inc	intf.inc
crdtyp.inc	blank.inc	disks.inc	eldof.inc	elgrp.inc
neqns.inc	prbsiz.inc	solutn.inc	timing.inc	prop.inc

Subroutines:

LOCTBL	DNRM2	NRMMX	PROB	GETSEC
SETREA	GENGRP	ICOPY	FORMEQ	interf
listbl	DAXPY	GENGRP		

File Name: ns2d.f

Usage: call NS2D (a , belm , condns, constr, coord , diag , disdiv, div , divh , eps , frstel, grad , groupe, ide , idp , intgp , intgv , invmp , isolid, lm , matelm, mbetat, mcond , meldof, mp , mpdof , mprop , mspht , mvisc , nbritr, nbul , ncbett, nccond, ncoord, ncvisc, ndep , ndof , ndp , neldof, nelem , nelgrp, neq , nfixbv, nintgp, nintgv, nmatcf, nodes , nshapg, nshapp, nshapt, nshapv, nshp , ntype , numnp , pe , presur, prop , pspace, resid , solver, u , vel , trures)

Function: driver for computing element matrices for 2-d flows : 2-D cartesian case.

Parameters:

prop(i).......... array of element group fluid properties

mxgaus............. max number of gauss points for any terms in the element matrix

mxshpv............. max number of velocity shape functions for any 2-d elements

mxshpp............. max number of pressure shape functions for any 2-d elements

maxndp............. max number of of nodes per element

rgu(i),sgu(i): local coordinates of gauss points for momentum matrices

wgu(i)......... weigth for gaussian integration of momentum

rgp(i),sgp(i): local coordinates of gauss points for pressure/continuity terms

wgp(i)......... weigth for gaussian integration of pressure

rgf(i), sgf(i), wgf(i): local coordinates of gauss points for blade force/tangency terms

wgf(i)......... weigth for gaussian integration of blade force

gshapu(i,j).... geometrical shape functions at gauss points

ushapu(i,j).... u-shape functions at momentum gauss points

pshapu(i,j).... p-shape functions at momentum gauss points

tshapu(i,j).... t-shape functions at momentum gauss points

dsudru(i,j),dsudsu(i,j):local derivatives of shape functions for u evaluated at momentum integration points dsudrp(i,j),dsudsp(i,j): local derivatives of shape functions for u evaluated at pressure integration points

Include files:

blank.inc	skyline.inc	eqsize.inc	disks.inc	problm.inc
option.inc	timvol.inc	timing.inc	parm2d.inc	

Subroutines:

OFFSET	ADDTBL	LOCTBL	PROB	GAUSS
SHAPG2	SHAPU2	SHAPS2	SHAPG2	SETREA
LMREDO	VELREC	EGAUS2	SHAPP2	DERIV2
MASS	DIV2D	EXMOM2	INVMAT	EGAUS2
EVAL2	EVSUPG	SHAPSU	DIFF2D	CONV2D
SORC2D	FORC2D	GRAD2D	PRE2D	RESIDU
LIN2D	LINSU	ULN2D	REDUI2	UZAWA
ASSRES	PENAL	DSCDIV	ASSEMB	SP_ASMB

File Name: paktbl.f

Usage: call paktbl

Function : Perform garbage collection on the array that all tables are managed. This allows deletion of arrays in arbitrary order in the memory manager.

Parameters:

Include files:

blank.inc   disks.inc   tables.inc

Subroutines:

File Name: print_b.f
Usage: call print_b(n, b, idump)
Function: Write vector b of length n to a I/O unit idump.
Parameters:

n: length of vector b.

b: double precision vector of length n.

idump: I/O unit to write out vector b.

File Name: prtmt_blocks.f

Usage: call prtmt_blocks(output)

Function: Write out the matrix $\begin{bmatrix} A & B \\ C & 0 \end{bmatrix}$ into three files: matrix.A, matrix.B and matrix.C. It also writes out the right-hand-side vector $\begin{bmatrix} f \\ g \end{bmatrix}$ into two file: rhs.f and rhs.g. This is the input format requested by STOKES stand-alone version.

Parameters: output: I/O unit number for write.

Include files:

blank.inc	intf.inc	tables.inc	disks.inc	execut.inc
neqns.inc				

Subroutines:

mprtmt	print_b

File Name: prtmt_global.f

Usage: call prtmt_global(output)

Function: Write out the global matrix, right-hand-side vector and solution vector into the file: matrix.G.

Parameters: output: I/O unit number for write.

Include files:

blank.inc	intf.inc	tables.inc	disks.inc	execut.inc
neqns.inc				

Subroutines:

mprtmt

File Name: reset_block_A.f

Usage: call reset_block_A(n, rptr_G, cind_G, G, np, perm, invp, nerror)

Function: Reset the value of matrix A, the sub matrix of matrix of $G = \begin{bmatrix} A & B \\ C & 0 \end{bmatrix}$. The input matrix G is in CSR format and the output matrices A is also in CSR format. matrix A is stored in tables rptr_A, cind_A and val_A .

Parameters:

integer n: size of matrix

integer nerror: error code. = 0, no error detected = 1, error is found

rptr_G, cind_G, G: matrix G in CSR format.

nv: number of velocity unknown.

np: number of presure unknown.

perm(n): permutation vector. A(i,j) will be permuted to A(perm(i),perm(j)).

invp(n): inverse of perm(n).

Include files:

blank.inc tables.inc disks.inc intf.inc

Subroutines:

dcopy

File Name: retbl.f

Usage: call RSTBL (namein, length, typein , from)

Function: reset the size of the table. The size of a table can be changed to be larger or smaller. If the table becomes larger, this routine may shift the contents of the memory array to the right for space. This in turn may change pointer values, meaning your code had better use loctbl to find them again.

Parameters:

namein .... name of the table to create

length .... length of the array

typein .... type of array to allocate
= 'integer' integer array
= 'real' real array

from .... name of routine requesting allocation of array 'name'

Include files:

itwo.inc	blank.inc	disks.inc	execut.inc	tables.inc

Subroutines:

UPCASE

File Name: set_blocks_csr.f

Usage: call set_blocks_csr(n, rptr_G, cind_G, G, np, perm, invp, nerror)

Function: Set up the matrix of $G = \begin{bmatrix} A & B \\ C & 0 \end{bmatrix}$ form. The input matrix G is in CSR format and the output matrices A, B, C are also in CSR format. Matrix A is stored in tables rptr_A, cind_A and val_A respectively. B and C are in the similar format.

Parameters:

n: size of matrix

rptr_G, cind_G, G: matrix G in CSR format.

np: number of presure unknown. A(n-np, n-np) B(n-np, np) C(np, n-np) $\begin{bmatrix} A & B \\ C & 0 \end{bmatrix}$ is the permuted global matrix

perm(n): permutation vector. A(i,j) will be permuted to A(perm(i),perm(j)).

invp(n): inverse of perm(n).

nerror: error code. = 0, no error detected = 1, error is found

Include files:

blank.inc tables.inc disks.inc intf.inc

Subroutines:

get_perm blkperm

File Name: skl2coord1.f

Usage: call skl2coord1(n, nnz, diag, A, aa, irn, icn)

Function: Set up the coordinatewise data structure for MA28. The sizes of the arrays that store the values and row and column index is greater than number of nonzero elements of the matrix because they need to include some working space. Here the size is determined so that these can be set as large as possible.

Parameters:

n: size of the matrix.

nnz : number of non zeros in matrix in skyline format. It is greater than the actual number of nonzero elements.

diag, A: matrix A in skyline format.

aa, irn, icn : pointers to the tables that contains the coordinatewise data structure of the matrix.

Include files:

blank.inc tables.inc disks.inc execut.inc ma28.inc

Subroutines:

addtbl dcopy

File Name: skl2coord2.f

Usage: call skl2coord2(n, nnz, diag, A, aa, ivect, jvect)

Function: Set up the coordinate wise data structure for MA28. This subroutine is different from the routine skl2coord1 in the size of the tables for the storage. The size of the tables for the coordinate data structure is EQUAL to the number of nonzero elements of the matrix. No extra working space is needed. This is used after the pivot sequence has been set previously.

Parameters:

n: size of the matrix.

nnz: number of non zero elements of the matrix.

diag, A: the matrix in skyline format.

aa, ivect, jvect: pointers to the tables of coordinate wise matrix A.

Include files:

blank.inc tables.inc disks.inc execut.inc ma28.inc

Subroutines:

loctbl addtbl dcopy

File Name: skl2csr.f

Usage: call skl2csr(n, nnz, diag, A, rhs)

Function: Convert the problem in skyline format to CSR format.

Parameters:

n: size of matrix

nnz: number of non-zeros in the matrix of skyline format. It could be changed if there are explicit zeros in the original matrix form.

n, nnz, diag, A: matrix A in skyline format.

rhs: right-hand-side vector of the problem.

Include files:

blank.inc	tables.inc	disks.inc	intf.inc

Subroutines:

addtbl	dcopy	convert_csr	rstbl	paktbl

File Name: skl2spaks.f

Usage: call skl2spaks(n, diag, A, spcon, spmap, work, isize)

Function: Read in the structure of the matrix to SPARSPAK from the input matrix which is in skyline format. Reorder the matrix and set up the internal data structure.

Parameters:

n: size of the matrix.

diag, A: matrix in skyline format.

spcon, spmap: array for control parameter passing in SPARSPAK.

work: double precision work array for runing SPARSPAK.

isize: The size of the array work.

Include files:

blank.inc	intf.inc	tables.inc	disks.inc	execut.inc
neqns.inc	sparspak.inc	spksys.inc		

Subroutines:

addtbl	loctbl	sprspk	ijbegn	inij
ijend	ordrb6	ordra2	ordra4	ordrb4
ordra6				

File Name: skl2spakv.f

Usage: skl2spakv(n, diag, A, spcon, spmap, work, isize)

Function: Read in the numerical values of the matrix to SPARSPAK from the input matrix which is in skyline format.

Parameters:

n: size of the matrix.

diag, A: matrix in skyline format.

spcon, spmap: array for control parameter passing in SPARSPAK.

work: double precision work array for runing SPARSPAK.

isize: The size of the array work.

Include files:

blank.inc	intf.inc	tables.inc	disks.inc	execut.inc
neqns.inc	sparspak.inc	spksys.inc		

Subroutines:

inaij6	inaij2	inaij4

File Name: sp_asmb.f

Usage: call SP_ASMB (a_elem , nrow , ncol , lmrow , lmcol , a_glob , eqsize , eq_prof, neq , eq_set, r_permut, c_permut, mrow , mcol)

Fonction: assemblage d'une matrice elementaire dans une matrice globale creuse stockee en format 'SPARSE COMPRESSED ROW' (stockage morse par rangees comprimees) Note: this file is different from the version v6. There is a bug in that version that being fixed in this version. Note: we warned you about Franglais.

Parameters:

a_elem (i,j) .... matrice elementaire

nrow, ncol .... nombre de rangees et colonnes de a_elem(,)

lmrow (i) .... vecteur d'assemblage des rangees

lmcol (i) .... vecteur d'assemblage des colonnes

a_glob(i) .... matrice globale en stockage morse

eqsize .... longueur de a_glob()

eq_prof(i) .... pointeur sur le premier coefficient des des rangees dnas a_glob()

neq .... nombre d'equations

eq_set(i) .... position des coefficients non-nuls des rangees de a_glob()

r_permut() .... vercteur de permutation des rangees pour l'assemblage trie

c_permut() .... vercteur de permutation des colones pour l'assemblage trie

Include files:

Subroutines:

QSORTI

File Name: splib.f

Usage: call splib(totalt, a, colind, rwptr, n, x, b, tol, reltv, maxits, pcmeth, solmeth, iparms, rparms, redo, errflg, iunit, instlvl)

Function: Main subroutine of SPLIB, sparse matrix iterative library. This subroutine is from SPLIB and special changes have been make to be able to run in connection with CADYF. The changes are: 1. All the needed space is allocated from a single array managed by a memory menagement routines from CADYF instead of from the independent package. Therefore memory.inc is changed. 2. There will no initialization for tables (CADYF does that). 3. The call to rmvtbl is changed to have only two parameters (standalone version of SPLIB uses three parameters).

Parameters:

n : Order of the matrix A.

a : real*8 array for storing nonzero entries of A

colind : integer array of column indices for entries in a; must be of length nzmax.

rwptr : integer array of pointers to beginning position of rows in array a. Must be of length nmax+1

x : real*8 vector of unknowns to be solved for.

b : real*8 right hand side vector of system to be solved: Ax = b

see lrhs and lsol for options on this vector.

tol : tolerance on norm used in stopping test.

maxits : Maximum number of iterations allowed. On return, contains actual number performed.

solmeth : Solution method to use, in an integer m. If

m = 1, use BiCG
m = 2, use CG on normal equations AA'y = b, x = A'y.
m = 3, use CG on normal equations A'A x = A'b
m = 4, use CGS
m = 5, use CGSTAB
m = 6, use GMRES(k)
m = 7, use transpose-free QMR
m = 8, use Bi-CG stabilized (Templates version of CG-Stab).

m = 9, use Templates version of GMRES

pcmeth : Preconditioning method to use, in form of an integer m. If

m = 0, use no preconditioner.

m = 1, use ILU(s)

m = 2, use MILU(s,rpcprm)

m = 3, use ILUT(s,rpcprm)

m = 4, use SSOR(rpcprm)

m = 5, use TRID(s), where s is the block size.

m = 6, use ILU(-1), the space saver version of ILU(0)

m = 7, use ECIMGS.

rparms : real preconditioner parameter, with interpretation depending on preconditioning method to be used.

- tolerance for numerical dropping with ILUT;

- omega parameter for SSOR(omega);

- relaxation constant for MILU.

- sparsity pattern indicator for ECIMGS

iparms : integer parameters, with interpretation depending on methods used.

- levels of fill for ILU(s), MILU(s), ILUT

- block size for TRID.

- iparms(2) is maximum Kyrlov subspace size for GMRES(k).

instlvl : Level of instrumentation. The level is cummulative, so specifying output of Oetli/Prager norms implies residual norms, etc

instlvl< 0 means no output from splib (but memory manager might)

instlvl = 0 means only summary data

instlvl = 1 means residual norm data

instlvl = 2 means preconditioned residual norm

instlvl = 3 means relative residual norm

instlvl = 4 means relative preconditioned residual norm

instlvl = 5 means Oetli/Prager norms

instlvl = 6 means error norms

iunit : vector of I/O unit numbers to use for output. The I/O unit numbers are provided in the vector iunit(16). The units are optional, depending on the error norm(s) you wish to plot/use. However, if you specify in the instrumentation common block printing of graphs for error norms, the following units must be opened for writing:

iunit(1): File for output of summary data

iunit(2-3): Not used here

iunit(4): $\| Ax - b \|$ vs. iter

iunit(5): $\| inv(M)(Ax - b) \|$ vs. iter

iunit(6): $\| Ax - b \| / \| r_0 \|$vs. iter

iunit(7): $\| inv(M)(Ax - b) \| / \| inv(M)r_0 \|$ vs. iter

iunit(8): $\| Ax - b \|_{inf} /(\| A \|_{inf} \| x \|_1 + \| b \|_{inf})$vs. iter

iunit(9): $\| x - xstar \|$ vs. iter

iunit(10): Not used.

iunit(11): $\| Ax - b \|$ vs. time

iunit(12): $\| inv(M)(Ax - b) \|$ vs. time

iunit(13): $\| Ax - b \| / \| r_0 \|$ vs. time

iunit(14): $\| inv(M)(Ax - b) \| / \| inv(M)r_0 \|$ vs. time

iunit(15): $\| Ax - b \|_{inf} /(\| A \|_{inf} \| x \|_1 + \| b \|_{inf})$ vs. time

iunit(16): $\| x - xstar \|$ vs. time

work : Array of work storage, for preconditioners, solvers, and. instrumentation routines. The matrices and vectors used are all indexed from this integer array, using pointers.

space : maximum number of integer words allowed for all problem dependent arrays. This includes the preconditioners and the additional work vectors needed in the solvers.

redo : Logical variable; recompute preconditioner if redo = .true.

Include files:

memory.inc tables.inc blank.inc

Subroutines:

addtbl	memleft	getnext	rmvtbl	istbl
ilus	ilut	ssor	tridiag	ilu0

ecimgs	bicg	cgne	cgnr	cgs
cgstab	gmres	qmrtf	bicgs	gmrest
jacobi	gauss	sor	orthomin	bmux
daxpy	dnrm2	listbl	prtres	

File Name: tma28.f

Usage: call tma28(totalt)

Function: Interface call to MA28 linear solver package.

(1) set up the metrix into the coordinate-wise format and set up the vectors and other work space before call MA28 subroutines.

(2) Call MA28 subroutine to solve the system.

(3) adjust the parameters if necessary

(4) Print out the performance information.

Parameters:

totalt: total time on solving linear system.

Include files:

blank.inc	intf.inc	tables.inc	disks.inc	execut.inc
neqns.inc	ma28.inc	problm.inc		

Subroutines:

loctbl	addtbl	skl2coord1	skl2coord2	csr2coord1
csr2coord2	ma28ad	ma28bd	ma28cd	instr_direct_skl
instr_direct_csr				

File Name: tskl.f
Usage: call tskl(totalt)
Functions: Interface call to skyline solver to solve the problem.

(1) Set up matrix and right-hand-side vector.

(2) Call factlu to do the LU factorization.

(3) Call solvlu to compute the forward/backward substitution to solve the systems.

(4) Print out the performance information.

Parameters:

totalt: total time on linear system solver.

Include files:

blank.inc	intf.inc	tables.inc	disks.inc	execut.inc
neqns.inc				

Subroutines:

loctbl	addtbl	dcopy	getsec	instr_direct_skl
factlu	solvlu	instr_direct_csr		

File Name: tsplib.f
Usage: call tsplib(totalt)
Functions: Interface call to SPLIB solver.

(1) Set up the metrix, right-hand-side vector and other parameters.

(2) Call SPLIB to solve the problem.

(3) Adjust the parameters. Rerun SPLIB if necessary.

Parameters:

totalt: total time on linear solver.

Include files:

blank.inc	intf.inc	tables.inc	disks.inc	execut.inc
neqns.inc	splib.inc			

Subroutines:

loctbl	addtbl	splib

File Name: tsppak.f

Usage: call tsppak(totalt)

Functions: Interface call to SPARSPAK solver.

(1) Read in matrix structure and setup internal data structures after preordering. Then read in numerical values of right- hand-side vector and the velues of matrix elements.

(2) Apply solver to solve the problem.

(3) Adjust the parameters if necessary

(4) Print out the performance data.

Parameters:

totalt: total time on linear system solving. It does not include the time on data structure read in and set up.

Include file:

blank.inc	intf.inc	tables.inc	disks.inc	execut.inc
neqns.inc	sparspak.inc	spksys.inc		

Subroutines:

loctbl	addtbl	skl2spaks	skl2spakv	csr2spaks
csr2spakv	inrhs	solve6	solve2	instr_direct_skl
dcopy	solve4	statsa	instr_direct_csr	

File Name: tstokes.f

Usage: call tstokes(totalt)

Function: Interface call to STOKES package.

(1) Set up the matrix of $\begin{bmatrix} A & B \\ C & 0 \end{bmatrix}$ and right-hand-side vector $\begin{bmatrix} f \\ g \end{bmatrix}$.

(2) Call STOKES to solve the problem.

(3) Permute the solution and right hand side vector to the original order so that CADYF will recognize.

(4) Adjust the parameters if necessary

Parameters:

totalt: total time on linear system solver. It does not include the time on instrumentation.

Include files:

blank.inc	intf.inc	tables.inc	disks.inc	execut.inc
neqns.inc	splib.inc	crdtyp.inc		

Subroutines:

loctbl	addtbl	set_blocks_csr	reset_block_A	dvperm
stokes				

References

[1] R. Bramley, *An orthogonal projection algorithm for linear Stokes problems*, Tech. Rep. 1190, Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois, 1992.

[2] R. Bramley, X. Wang, and D. Pelletier, *Orthogonalization based iterative methods for generalized stokes problems*, in Solution Techniques for Large–Scale CFD Problems, W. G. Habashi, ed., Centre de Recherce en Calcul Applique, 1994.

[3] E. Chu, A. George, J. Liu, and E. Ng, *SPARSPAK: Waterloo sparse matrix package user's guide for SPARSPAK-A*, Tech. Rep. CS-84-36, Department of Computer Science, University of Waterloo, 1984.

[4] I. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices*, Oxford Science Publications, Oxford, 1 ed., 1986.

[5] I. Duff, R. Grimes, and J. Lewis, *User's guide for the Harwell–Boeing sparse matrix collection*, Tech. Rep. TR/PA/92/86, CERFACS, Toulouse France, 1986.