Definition and Implementation of a SLICOT Standard Interface and the associated MATLAB Gateway for the Solution of Nonlinear Control Systems by using ODE and DAE Packages [1]

Enrique Arias [2], Ignacio Blanquer [2], Vicente Hernández [2], Pedro Ruiz [2].

March, 2000

[1]This document presents research results of the European Community BRITE-EURAM III Thematic Networks Programme NICONET (contract number BRRT-CT97-5040) and is distributed by the Working Group on Software WGS. *WGS secretariat*: Mrs. Ida Tassens, ESAT - Katholieke Universiteit Leuven, K. Mercierlaan 94, 3001-Leuven-Heverlee, BELGIUM. This report is also available by anonymous ftp from *wgs.esat.kuleuven.ac.be* in the directory *pub/WGS/REPORTS/SLWN2000-3.ps.Z*

[2]Departamento de Sistemas Informáticos y Computación (DSIC). Universidad Politécnica de Valencia (UPV). Valencia. Spain. {*earias, iblanque, vhernand, pruiz*}*@dsic.upv.es.*

Contents

1 Introduction

This paper presents the **SLICOT** (Subroutine Library in Control and Systems Theory) **Implementation of the Nonlinear Control Systems**. Here, a common interface to several ODE and DAE libraries is prepared. This interface will be the entry point of the SLICOT nonlinear solvers and will enable the users to test the advantages of using one or other approaches. The packages that are being addressed are:

- ODEPACK[1] (LSODE, LSODA, LSODES, LSODAR, LSODI, LSOIBT)
- DASSL[2]
- DASPK[3, 4]
- GELDA[6]
- RADAU5[5]

In order to increase the usability of the interface, an implementation of a MATLAB **Gateway to the Nonlinear Control System Simulation Interface** has been developed. This gateway enables the user to define the problems using MATLAB code, including the definition of the system functions and jacobians. The user-friendliness on changing the solver integrator is guaranteed, thus it is provided by the SLICOT interface.

The interface defines a common layout that can be open to other libraries, just adding internal interface routines to translate the argument formats. The gateway will be improved as new packages are available in the SLICOT library.

The paper is divided in 5 sections, including this introduction. Second section contains a brief description of the integrator packages that are already included in the interface. Section number 3 presents the interface and describes the syntax and capabilities. The fourth section describes the steps that are required for using the interface, along with an extensive description of all the parameters and arguments, including one example coded in both Fortran and MATLAB source code. Finally, section 5 describes the details of the implementation of both the interface library and the MATLAB Gateway.

2 Description of the Packages

2.1 ODEPACK

This package is very simple and offers a very limited interface. Output vector is the same as the state vector and no parameters or input vectors are provided to the routines, so parameters must be specified using common blocks and input vectors have to be computed in the user-defined subroutines.

The interface differs very little from each one of the ODE solvers (LSODE, LSODA, LSODES).

LSODE

This is the basic solver of the package. The interface is

```
      subroutine lsode (f, neq, x, t, tout, itol, rtol, atol, itask,
     1            istate, iopt, rwork, lrw, iwork, liw, jac, mf)
```

and the type of problems that the system is able to cope is

$$\frac{dx(t)}{dt} = f(t, x).$$

The user must specify the 'f' routine. This routine has the syntax

```
      subroutine f (neq, t, x, xdot),
```

which supplies the vector function f by loading `xdot(i)` with $f(i)$.

LSODA

This is a variant of the basic `LSODE` solver which switches automatically between stiff and nonstiff methods. This means that the user does not have to determine whether the problem is stiff or not, and the solver will automatically choose the appropriate method. It always starts with the nonstiff method.

The syntax is the same as `LSODE`

```
      subroutine lsoda (f, neq, x, t, tout, itol, rtol, atol, itask,
     1            istate, iopt, rwork, lrw, iwork, liw, jac, jt)
```

and works with the same kind of problems. The syntax of the user-supplied routine has the same syntax and problems.

LSODES

This is a variant of the basic `LSODE` solver which is intended for problems in which the Jacobian matrix $\frac{\delta f}{\delta y}$ has an arbitrary sparse structure (when the problem is stiff).

The syntax is the same as `LSODE`

```
      subroutine lsodes (f, neq, x, t, tout, itol, rtol, atol, itask,
     1            istate, iopt, rwork, lrw, iwork, liw, jac, jt)
```

and works with the same kind of problems. The syntax of the user-supplied routine has the same syntax and problems.

The DAE problems that can be solved with ODEPACK are limited to the expression

$$F(x(t), t)\dot{x}(t) = A(x(t), t).$$

So, the output vector is directly the state vector and the input vector cannot be indicated as an argument. Moreover, no parameters can be indicated in the functions, which can be specified via common blocks.

There are two solvers in ODEPACK for DAEs, LSODI and LSOIBT. The difference between them is that the latter is for block-tridiagonal matrices. The approach followed is very similar in both of them.

LSODI and LSOIBT require two subroutines for computing the contribution of A and the residual factor.

LSODI

The syntax for LSODI is

```
  subroutine lsodi (res, adda, jac, neq, x, xdoti, t, tout, itol,
 1  rtol, atol, itask, istate, iopt, rwork, lrw, iwork, liw, mf )
```

Most of the arguments are common to all the routines and particularly to the standard definition. The key points are located in the definition of the *res* and *adda* routines.

The 'res' routine has the following syntax

```
       subroutine res (neq, t, x, s, r, ires)
```

and computes the residual factor

$$r = A(x(t), t) - F(x(t), t)\frac{dx(t)}{dt},$$

in which the approximation of the derivative $\frac{dx(t)}{dt}$ is provided by the solver routine as an argument (s).

The `adda` subroutine has the following syntax

```
    subroutine adda ( neq, t, x, ml, mu, L, nrowp )
```

and computes the expression

$$F(x(t), t) = F(x(t), t) + L,$$

with L an internal matrix factor computed by ODEPACK in each stage.

LSOIBT

The syntax for LSOIBT is

```
  subroutine lsoibt (res, adda, jac, neq, x, xdoti, t, tout, itol,
 1  rtol, atol, itask, istate, iopt, rwork, lrw, iwork, liw, mf )
```

The behaviour of the routine is the same as LSODI, with the only difference that it is designed for working with block-tridiagonal systems. The user must supply the routines DAEDE and DAEDA oriented to deal with block-tridiagonal matrices.

2.2 RADAU5

This package computes the numerical solution of a differential algebraic system of first order ordinary diferential equations in the form

$$M\dot{y} = F(x,y)$$

The system can be implicit or explicit. The method used is an implicit Runge-Kutta Method of order 5 with step size control and continuous output. It offers a simple interface. Output vector is the same as the state vector and no parameters or input vectors are provided to the routines, so parameters must be specified using common blocks and input vectors have to be computed in the user-defined subroutines. The interface is similar to ODE solvers (LSODE, LSODA, LSODES or LSODAR). The syntax is the following

```
      SUBROUTINE RADAU5(N,FCN,X,Y,XEND,H,RTOL,ATOL,ITOL,
     &                  JAC ,IJAC,MLJAC,MUJAC,
     &                  MAS ,IMAS,MLMAS,MUMAS,
     &                  SOLOUT,IOUT,
     &                  WORK,LWORK,IWORK,LIWORK,RPAR,IPAR,IDID)
```

User has to supply the subroutines `'FCN'`, `'JAC'`, `'MAS'` and `'SOLOUT'`. The first one computes the value of $F(x,y)$, and the syntax of this routine is

```
 SUBROUTINE FCN(N,X,Y,F,RPAR,IPAR)
```

`'JAC'` routine, which is only called if `IJAC` parameter of RADAU5 is equal to 1, computes the partial derivatives of $F(x,y)$ with respect to y. This subroutine has the following syntax:

```
SUBROUTINE JAC(N,X,Y,DFY,LDFY,RPAR,IPAR)
```

where `'LDFY'` parameter stands for the column length of the array and it is provided by the calling program. The Jacobian can be expressed in a full or sparse storage. If `'MLJAC'` parameter is equal to the dimension of the system, the Jacobian is supposed to be a full matrix and the partial derivatives are stored in `'DFY'` as

$$DFY(I,J) = \frac{\partial F(I)}{\partial Y(J)}.$$

Otherwise, the Jacobian is taken as a banded matrix and the partial derivatives are stored diagonal-wise as

$$DFY(I-J+MUJAC+1,J) = \frac{\partial F(I)}{\partial Y(J)}.$$

If 'IJAC=0' the 'JAC' subroutine is never called, and Jacobian is computed internally by finite differences.

'MAS' subroutine computes the mass matrix 'M'. As Jacobian case, the 'IMAS' parameter determines if mass matrix is assumed to be the identity matrix and needs not to be defined ('IMAS=0'), or the 'MAS' subroutine has to be called ('IMAS=1'). In the first case, a dummy subroutine has to be supplied. In the last one the syntax of 'MAS' subroutine is the following:

```
SUBROUTINE MAS(N,AM,LMAS,RPAR,IPAR)
```

If 'MLMAS=N' the mas-matrix is stored as a full matrix, otherwise the matrix is expected to be stored diagonal-wise with $AM(I-J+MUMAS+1,J)=M(I,J)$.

Finally, 'IOUT' parameter determine if the numerical solution during integration has to be provided by 'SOLOUT' subroutine ('IOUT=1') or not ('IOUT=0'). The syntax of 'SOLOUT' routine is:

```
SUBROUTINE SOLOUT (NR,XOLD,X,Y,CONT,LRC,N,RPAR,IPAR,IRTRN)
```

SOLOUT furnishes the solution 'Y' at the NR-th grid point 'X' (thereby the initial values is the first grid-point). 'XOLD' is the preceeding grid-point. 'IRTRN' serves to interrupt the integration. If 'IRTRN' is lower than 0, RADAU5 returns to the calling program.

2.3 DASSL

DASSL package solves a system of differential/algrebraic equations of the form

$$G(t,y,\dot{y})=0, \tag{1}$$

from T (initial time) to TOUT (final time).

DASSL uses the backward differentiation formulas of orders one through five to solve this type of system. Values for y and $\dot{y}$ at the initial time must be provided as input. These values have to satisfy the expression (1), so they must be consistent.

The syntax of DASSL is the following

```
SUBROUTINE DDASSL (RES, NEQ, T, Y, YPRIME, TOUT, INFO, RTOL, ATOL,
     1   IDID, RWORK, LRW, IWORK, LIW, RPAR, IPAR, JAC)
```

The arguments of the subroutine are very similar to the arguments of the previous packages. Two parameters are emphasized: subroutines 'RES' and 'JAC'.

'RES' subroutine defines the differential/algebraic system. This subroutine has to be supplied by the user in the form

```
SUBROUTINE RES(T,Y,YPRIME,DELTA,IRES,RPAR,IPAR)
```

For the given values of (t) ('T'), y ('Y') and $\dot{y}$('YPRIME'), the subroutine should return the residual of the defferential/algebraic system

$$\Delta = G(t, y, \dot{y}).$$

'INFO(5)' can be set to ignored 'JAC' parameter. Otherwise, subroutine 'JAC' defines the matrix of partial derivatives. This subroutine has the following form

```
SUBROUTINE JAC(T,Y,YPRIME,PD,CJ,RPAR,IPAR)
```

where

$$PD = \frac{\partial G}{\partial y} + CJ\frac{\partial G}{\partial \dot{y}}$$

and CJ is a scalar which is input to JAC.

For the given values of t, y and $\dot{y}$, the subroutine must evaluate the non-zero partial derivatives for each equation and each solution component, and store these values in the matrix PD. The elements of PD are set to zero before each call to JAC so only non-zero elements need to be defined.

As in RADAU5, Jacobian matrix can be stored in full or sparse (banded) format.

2.4 DASPK

DASPK solves large-scale systems of differential/algebraic equations of the form

$$F(t, y, \dot{y}) = 0,$$

using a combination of Backward Differentiation Formula (BDF) methods and a choice of two linear system solution methods: direct (dense or band) or Krylov (iterative).

As in DASSL, initial values must be given as inputs and they should be consistent. Integration over the specified range of t is usually accomplished in a series of steps. The algorithm for computing an integration step involves replacing the derivatives with difference approximations and using a predictor-corrector method. In the predictor-corrector method an initial guess for the new solution is developed by evaluating the predictor polynomial, which interpoles solution values at previous time steps. The predictor and corrector polynomials are specified using BDF formulas of orders one through five. In each corrector step a sequence of nonlinear systems are solved by a Newton-type iteration. Each of these iterations requires the solution of a linear system. These linear systems are approximately solved by the preconditioned generalized minimum residual (GMRES) method.

The syntax of DASPK package is

```
SUBROUTINE DDASPK (RES, NEQ, T, Y, YPRIME, TOUT, INFO, RTOL, ATOL,
     1   IDID, RWORK, LRW, IWORK, LIW, RPAR, IPAR, JAC, PSOL)
```

The syntax is very similar to the DASSL syntax, as exception of 'PSOL' argument. 'PSOL' is the name of a subroutine to be provided by the user for the preconditioning of the iterative Krylov linear solver. The purpose of 'PSOL' is to solve the linear system associated to the preconditioning matrix 'P'.

2.5 DGELDA

DGELDA solves linear differential/algebraic equations with variable coefficients of the form

$$
\begin{aligned}
E(t)\dot{x}(t) &= A(t)x(t) + f(t) \\
x(t_0) &= x_0
\end{aligned}
$$

for x in a specified range of the independent variable t. The syntax of DGELDA is the following

```
      SUBROUTINE DGELDA (EDIF, ADIF, FDIF, N, T, TOUT, X, XPRIME,
     $                   CVAL, IPAR, RPAR, IWORK, LIW, RWORK, LRW,
     $                   RTOL, ATOL, METHOD, INFO, IWARN, IERR)
```

Four parameters can be emphasized:

- `EDIF`. This is a subroutine which the user provides to define the matrix $E(t)$ and its derivatives. It has the form

  ```
  SUBROUTINE EDIF(N,T,IDIF,E,LDE,IPAR,RPAR,IERR)
  ```

  The subroutine takes as input the number of equations `N`, the time `T` and the integer parameter `IDIF`. `EDIF` must not alter these input parameters or the leading dimension `LDE` of the array `E`. As output, the subroutine produces the `IDIF`-th derivative of $E(t)$ at time `T` in the leading `N` by `N` part of the array `E`. The integer flag `IERR` is always zero on input and `EDIF` should alter `IERR` only if `IDIF` is larger than the highest derivative of $E(t)$ the subroutine provides (set IERR $= -2$) or if another problem occurs (set IERR $= -1$). `IPAR` and `RPAR` are integer and real arrays which can be used for the communication between the user's calling program and the subroutines `EDIF`, `ADIF` and `FDIF`.

- `ADIF`. This is a subroutine which the user provides to define the matrix $A(t)$ and its derivatives. It is of the form

  ```
  SUBROUTINE ADIF(N,T,IDIF,A,LDA,IPAR,RPAR,IERR)
  ```

- `FDIF`. This is a subroutine which the user provides to define the vector $f(t)$ and its derivatives. It is of the form

  ```
  SUBROUTINE FDIF(N,T,IDIF,F,IPAR,RPAR,IERR)
  ```

  and the input and output parameters are similar to these of `EDIF`, except that the first `N` elements of the 1-dimensional array `F` contain the `IDIF`-th derivative of $f(t)$ at time `T`. Note further, since `F` is a 1-dimensional array no leading dimension is needed.

- `METHOD`. This parameter indicates which integration method should be used as follows:

  `METHOD`=1 the code uses a BDF solver,

  `METHOD`=2 the code uses a Runge-Kutta solver.

The most important invariant in the analysis of linear DAE's is the so called *strangeness index*, which generalizes the differential index for systems with undetermined components.

The implementation of DGELDA is based on the construction of the discretization scheme introduced in [12], which first determines all the local invariants and then transforms the system into a strangeness-free DAE with the same solution set.

The strangeness-free DAE is solved by either BDF methods, which were adapted from DASSL, or a Runge-Kutta method, which was adapted from RADAU5.

3 Design of the SLICOT Standard Interface

It is not easy to standarise currently available libraries in non-linear systems. Neither the algebraic expression, nor the parameters involved are compatible in every system.

Initially, a DSblock-like interface [7, 8, 9, 14] was proposed. This interface deals with non-linear control systems expressed in terms of ODEs

$$\left.\begin{aligned}\dot{x}(t) &= f(x(t), u(t), p, t)\\ y(t) &= g(x(t), u(t), p, t)\end{aligned}\right\}$$

or DAEs,

$$\left.\begin{aligned}f(\dot{x}(t), x(t), u(t), p, t) &= 0\\ y(t) &= g(\dot{x}(t), x(t), u(t), p, t)\end{aligned}\right\}$$

where $x(t) \in \mathbb{R}^{\mathrm{n}}$ is the state vector, $u(t) \in \mathbb{R}^{\mathrm{m}}$ is the input vector, $y(t) \in \mathbb{R}^{\mathrm{r}}$ is the output vector, $p \in \mathbb{R}^{\mathrm{q}}$ is the parameter vector.

Although this interface is very clear and appropriate for the definition of nonlinear systems, in the DAE case, it makes it impossible to link with solvers such as ODEPACK, RADAU5 or GELDA, since these packages require to define the system in different ways

- ODEPACK: $A\dot{x}(t) = G(x(t), t)$.
- RADAU5: $M\dot{x}(t) = F(x(t), t)$.
- GELDA: $E(t)\frac{dx(t)}{dt} = A(t)x(t) + F(t)$.

In order to compute the effect of the different functions/factors in the cases above described, the packages require to provide two or three subroutines for each one of the elements of the differential equation. This prevents from using a single routine from the top level.

The objective of the interface is not only to ease to change the solver package. However, and for the sake of reusability, it has been consider to provide with three different interfaces for DAEs, one for each model. This will allow to adapt quickly codes that are already running with any of the packages, and although different packages cannot be merged automatically, parameters and options will be specified with the same syntax.

The constraints of this approach lie on the compliance with the DSblock standard is not obtained and the set of systems that can be modelled is also a sub-set (although enough large) of the systems addressed by the DSblock model. On the other side, the interface is open to a wider variety of nonlinear solvers.

Taking into account the above constraints, a standard using three routines is proposed, but able to work with the whole set of DAE problems. This is achieved by changing the meaning of each one of the user-supplied functions depending on the DAE package selected.

Following this approach, four interfaces will be proposed, one for solving ODEs and three for solving DAES.

3.1 The ODE Solver

The expression of an ODE nonlinear system is

$$\left.\begin{array}{r}\frac{dx(t)}{dt} = f(x(t), u(t), p, t)\\ y(t) = g(x(t), u(t), p, t)\end{array}\right\}$$

The interface for the ODE has been implemented following the standards for the production of SLICOT software [11, 13], and it is given by

```
      SUBROUTINE ODESolver(ISOLVER, CODEDER_, CODEOUT_,
     $                     CJACFX_, CJACFU_, CJACFP_,
     $                     NX, NY, NU, TINI, TOUT, X, U, Y,
     $                     IPAR, DPAR, RTOL, ATOL,
     $                     IWORK, LIWORK, DWORK, LDWORK,
     $                     IWARN, INFO)
```

In the case of the MATLAB interface, the structure is the same except for the dimension arguments which are automatically computed. Therefore, the MATLAB interface is:

```
[x,y,ipar,dpar,iwarn,info] = ODESolver(isolver, odeder, odeout,
                                       jacfx, jacfu, jacfp,
                                       x, u, p, tini, tout,
                                       ipar, rpar, rtol, atol,
                                       iwork, dwork);
```

in which the arguments are defined in the following:

Mode Arguments

`ISOLVER` is an integer argument which indicates the nonlinear model solver to be used. Each solver has been given an index (and a constant parameter) which is specified in the subroutine documentation. The values of this index are:

- LSODE : Set `ISOLVER = 1` or equal to `LSODE_`.
- LSODA : Set `ISOLVER = 2` or equal to `LSODA_`.

- LSODES : Set `ISOLVER = 3` or equal to `LSODES_`.
- LSODI : Set `ISOLVER = 4` or equal to `LSODI_`.
- LSOIBT : Set `ISOLVER = 5` or equal to `LSOIBT_`.
- RADAU5 : Set `ISOLVER = 6` or equal to `RADAU5_`.
- DASSL : Set `ISOLVER = 7` or equal to `DASSL_`.
- DASPK : Set `ISOLVER = 8` or equal to `DASPK_`.
- DGELDA : Set `ISOLVER = 9` or equal to `DGELDA_`.

External Subroutine Call Arguments

An external subroutine must be provided by the user, with the syntax:

```
SUBROUTINE ODEDER(NX, NU, T, X, U, F, IPAR, DPAR, INFO)
```

or in the case of MATLAB

```
function [f,ipar,dpar,info]=odeder(t,x,u,p,ipar,dpar)
```

which returns the expression $f(x(t), u(t), p, t)$. In the case of MATLAB, the name of the module is specified in the call. In the FORTRAN case, the name is fixed.

For computing the output, the user must supply:

```
SUBROUTINE ODEOUT(NX, NU, NP, NY, T, X, U, P, Y, IPAR, DPAR, INFO)
```

or in the case of MATLAB

```
function [y,ipar,dpar,info]=odeout(t,x,u,p,ipar,dpar)
```

which returns the expression $g(\dot{x}(t), x(t), u(t), p, t)$.

If required, the expressions regarding the Jacobians are

```
SUBROUTINE JACFX(NX, LDFX, T, X, FX, IPAR, DPAR, INFO)
```

and

```
function [fx,ipar,dpar,info]=jacfx(t,x,ipar,dpar)
```

for the MATLAB case, which returns the Jacobian $\frac{\partial f}{\partial x}$.

```
SUBROUTINE JACFU(NX, NU, LDFU, T, X, U, FU, IPAR, DPAR, INFO)
```

and

```
function [fu,ipar,dpar,info]=jacfu(t,x,u,ipar,dpar)
```

for the MATLAB case, which returns the Jacobian $\frac{\partial f}{\partial u}$.

```
SUBROUTINE JACFP(NX, NP, LDFP, T, X, FP, IPAR, DPAR, INFO)
```

and

```
function [fp,ipar,dpar,info]=jacfp(t,y,ipar,dpar)
```

for the MATLAB case, which returns the Jacobian $\frac{\partial f}{\partial p}$.

Each one of the packages will include stub routines that will be automatically called by the solver package and will translate the arguments to the standard format, and thus providing an homogeneous definition for all of them. All the features of each package are described in the implementation section.

Size Arguments

- `NX`: Stores the dimension of the state vector.
- `NU`: Stores the dimension of the input vector.
- `NY`: Stores the dimension of the output vector.
- `LDWORK`: Stores the dimension of the real work area vector.
- `LIWORK`: Stores the dimension of the integer work area vector.

Array Arguments

- `X`: Array of dimension `NX` containing the state vector.
- `U`: Array of dimension `NU` containing the input vector.
- `Y`: Array of dimension `NY` containing the output vector.
- `RPAR`: Array of dimension `MAXNRP` containing the real parameter vector.
- `IPAR`: Array of dimension `MAXNIP` containing the integer parameter vector.
- `DWORK`: Array of dimension `LDWORK` containing the real work area vector. The size depends on the package called.

- IWORK: Array of dimension LIWORK containing the integer work area vector. The size depends on the package called.

Scalar Arguments

- TINI: Initial time.
- TOUT: Stopping time.
- RTOL: Scalar containing the relative tolerance of the whole X array.
- ATOL: Scalar or array (see parameters), containing the absolute tolerance of either the whole X array, or each one of the elements.
- INFO: Error return code.

3.2 The DAE Solver

The DAE standard will be able to cope with three different models of DAEs. The most general ones (case I), which can only be solved by DASSL, for the dense case or, DASPK, for the sparse case, have the expression

$$\left.\begin{array}{r} F(\dot{x}(t), x(t), u(t), p, t) = 0 \\ y(t) = g(\dot{x}(t), x(t), u(t), p, t) \end{array}\right\}$$

A restricted case (case II) can be solved also with RADAU5, LSODI or LSOIBT, if the system can be expressed as

$$\left.\begin{array}{r} F(x(t), u(t), p, t)\dot{x}(t) = A(x(t), u(t), p, t) \\ y(t) = g(\dot{x}(t), x(t), u(t), p, t). \end{array}\right\}$$

And finally, the GELDA package is able to solve DAEs with the expression (case III):

$$F(u(t), p, t)\dot{x}(t)dt = A(u(t), p, t)x(t) + E(u(t), p, t).$$

The interface for the DAE solver in any case will be

```
  SUBROUTINE DAESolver(ISOLVER, DAEDF, DAEDA, DAEDE, DAEOUT,
 &                     JACFX, JACFU, JACFP,
 &                     NX, NY, NU, TINI, TOUT, X, U, Y,
 &                     IPAR, DPAR, RTOL, ATOL,
 &                     IWORK, LIWORK, DWORK, LDWORK, IWARN, INFO)
```

and in the case of MATLAB

```
function [X,Y,IPAR,DPAR,IWARN,INFO] = DAESolver(ISOLVER,
                                          DAEDF, DAEDA, DAEDE, DAEOUT,
                                          JACFX, JACFU, JACFP,
                                          TINI, TOUT,
                                          X, XDOTI, U, P, IPAR, DPAR,
                                          RTOL, ATOL, DWORK, IWORK,
                                          IWARN, INFO);
```

In this subroutine most of the parameters have a similar meaning as the ODE case. Only the new arguments are described here

```
      SUBROUTINE DAEDF(NX, NU, NP, LDF, T, X, XPRIME, U, P, F,
     &                 IPAR, DPAR, IWARN, INFO)
```

and for the MATLAB case

```
function [F,IWARN,INFO]=DAEDF(LDF, T, X, XDOT, U, P, IPAR, DPAR)
```

This user-supplied subroutine have different meanings depending on the solver selected.

- Expression as case I, DAEDF must supply $F(\dot{x}(t), x(t), u(t), p, t)$.
- Expression as case II, DAEDF must supply $F(x(t), u(t), p, t)$.
- Expression as case III, DAEDF must supply $F(t, u(t), p, t)$.

```
      SUBROUTINE DAEDA(NX, NP, LDA, T, X, U, P. A, IPAR, DPAR, IWARN, INFO)
```

and for the MATLAB case

```
function [A, IWARN, INFO]=DAEDA(LDA, T, X, XDOT, U, P, IPAR, DPAR)
```

This subroutine is required only when dealing with expressions of type case II or case III. Otherwise is ignored. When dealing with case B, it must return the expression $A(x(t), u(t), p, t)$, and in the case III it must provide the $A(t)$ matrix.

```
      SUBROUTINE DAEDE(NU, NP, LDF, T, U, P, F, IPAR, DPAR, IWARN, INFO)
```

and for the MATLAB case

```
function [F, IWARN, INFO]=DAEDE(LDF, T, U, P, IPAR, DPAR)
```

This subroutine is required only for the case III expressions, and should return $F(u(t), p, t)$.

```
   SUBROUTINE DAEOUT(NX, NP, NU, T, X, XDOT, U, P, Y, IPAR, DPAR,
  &                  IWARN, INFO)
```

and for the MATLAB case

```
   function [F,IWARN,INFO]=DAEOUT(T, X, XDOT, U, P, IPAR, DPAR)
```

which returns the expression $y(t) = g(\dot{x}(t), x(t), u(t), p, t)$.
If required, the expressions regarding the Jacobians are

```
   SUBROUTINE JACFX(NX, LDFX, T, X, XDOT, FX, IPAR,
  &                 DPAR, IWARN, INFO)
```

and

```
  function [FX,IWARN,INFO]=JACFX(LDFX, T, X, XDOT, FX, IPAR, DPAR)
```

for the MATLAB case, which returns the Jacobian $\frac{\partial f}{\partial x}$.

```
   SUBROUTINE JACFXDOT(NX, LDFX, T, X, XDOT, FX, IPAR,
  &                 DPAR, IWARN, INFO)
```

and

```
  function [FX,IWARN,INFO]=JACFXDOT(LDFX, T, X, XDOT, FX, IPAR, DPAR)
```

for the MATLAB case, which returns the Jacobian $\frac{\partial f}{\partial \dot{x}}$.

```
   SUBROUTINE JACFU(NX, NU, LDFU, T, X, XDOT, U, FU, IPAR, DPAR,
  &                 IWARN, INFO)
```

and

```
  function [FU, IWARN, INFO]=JACFU(LDFU, T, X, XDOT, U, IPAR, DPAR)
```

for the MATLAB case, which returns the Jacobian $\frac{\partial f}{\partial u}$.

```
   SUBROUTINE JACFP(NX, NU, LDFP, T, X, XDOT, U, FP, IPAR, DPAR,
  &                 IWARN, INFO)
```

and

```
function [FP, IWARN, INFO]=JACFP(LDFP, T, X, XDOT, U, IPAR, DPAR)
```

for the MATLAB case, which returns the Jacobian $\frac{\partial f}{\partial p}$.

All the arguments have the same meaning as in the rest of the interface functions, and `xdot` stands for the time derivative of `x`.

Each one of the packages will include stub routines that will be automatically called by the solver package and will translate the arguments to the standard format, and thus providing an homogeneous definition for all of them. All the features of each package are described in the implementation section.

4 Using the Interface

4.1 Overview

A program using the interface will have a part common to all possible packages and a part that will depend on the package used. Most of the arguments should take the defaults values, but some of the packages need more information.

Before calling the solver, some parameters must be adjusted, such as matrix format, tolerance mode and Jacobians computation. Several of them are common to every package, and their position is normalised to ease the interface. It is important to note that not all the packages have the same location for all the common parameters. Moreover, its meaning can vary. In these cases the interface normalises also the value, offering a uniform entry point.

The parameters that are specific of each package must also be indicated before calling the solver.

After the parameters are specified, including the auxiliary routines, the solver can be called through the unique interface. After its execution, the output can be collected standarised through the proper output arguments.

4.2 The Initialisation Process

All the initialisation parameters must be provided through the `IPAR` and `DPAR` arrays. The parameters which are specific of each packages are directly passed to the solver, and no extra comment is required. The range of parameters of the `IPAR` and `DPAR` arrays are the following:

`IPAR` and `DPAR` Arrays:

In-Out Mode	Range	Owner
INPUT	1,..,15	General
	16,..,25	ODEPACK
	26,..,35	RADAU5
	36,..,50	DASSL/PK
	51,..,60	GELDA
	61,..,100	*Reserved*
OUTPUT	101,..,110	General
	111,..,125	ODEPACK
	126,..,135	RADAU5
	136,..,145	DASSL/PK
	146,..,155	GELDA
	156,..,200	*Reserved*
Any Mode	201,..	User Available

The parameters that are used in more than one package are normalised and the syntax for `DPAR` is as follows

- `DPAR(1)`: Initial step size guess (optional in ODEPACK).
- `DPAR(2)`: Maximum absolute step size allowed.

The rest of the parameters are specified using the rest of the IPAR values. Each one of the packages have a reserved area, which ranges

- ODEPACK Package:
  - `DPAR(16)`: Critical value of t which the solver is not overshoot.
  - `DPAR(17)`: Minimum absolute step size allowed.
  - `DPAR(18)`: Tolerance scale factor, greater than 1.0.
- LSODES:
  - `DPAR(19)`: The element threshhold for sparsity determination when moss = 1 or 2.
- RADAU5:
  - `DPAR(26)`: The rounding unit, default 1E-16.
  - `DPAR(27)`: The safety factor in step size prediction, default 0.9D0.
  - `DPAR(28)`: Decides whether the Jacobian should be recomputed, default 0.001D0.
  - `DPAR(29)`: Stopping criterion for Newton's method, default MIN(0.03D0, RTOL(1)**0.5D0).
  - `DPAR(30), DPAR(31)`: This saves, together with a large DPAR(28), LU-decompositions and computing time for large systems.
  - `DPAR(32), DPAR(33)`: Parameters for step size selection.
- DASSL and DASPK:

– `DPAR(36)`: Stopping point (Tstop).

- DASPK:
  – `DPAR(37)`: convergence test constant in SPIGMR algorithm (0 .LT. DPAR(14) .LT. 1.0).
  – `DPAR(38)`: minimum scaled step in linesearch algorithm. The default is equal to (unit roundoff)**(2/3). (> 0)
  – `DPAR(39)`: swing factor in the Newton iteration convergence test (default 0.1) (> 0).
  – `DPAR(40)`: safety factor used in step size prediction.
  – `DPAR(41), DPAR(42)` : restric the relation between the new and old stepsize in step size selection (1/DPAR(41) .LE. Hnew/Hold .LE. 1/DPAR(42)).
  – `DPAR(43), DPAR(44)` : QUOT1 and QUOT2 repectively. If QUOT1 < Hnew/Hold < QUOT2 and A and E are constants, the work can be saved by setting Hnew=Hold and using the system matrix of the previous step.

The syntax for `IPAR` is as follows

- `IPAR(1)`: Tolerance error dimensions.
  – 0 : Default value, both `RTOL` and `ATOL` are scalars.
  – 1 : `RTOL` is scalar and `ATOL` is a vector.
  – 2 : `RTOL` and `ATOL` are vectors.
- `IPAR(2)`: Compute Output Values
  – 1 : Yes.
  – 0 : No.
- `IPAR(3)`: Jacobian and method indicator.
  – 0 : No Jacobian used (non-stiff method), (10 in ODEPACK).
  – 1 : User supplied full Jacobian (stiff).
  – 2 : User supplied banded Jacobian (stiff).
  – 3 : User supplied sparse Jacobian (stiff).
  – 10 : internally generated full Jacobian (stiff).
  – 11 : internally generated banded Jacobian (stiff).
  – 12 : internally generated sparse Jacobian (stiff).
- `IPAR(4)`: Dense / Sparse factor.
  – 0 : Dense storage (LSODE, LSODA, RADAU5, DASSL).
  – 1 : Sparse storage (LSODES, DASPK).
- `IPAR(5)`: Maximum number of steps allowed during one call to the solver.

- `IPAR(6)`: ml, lower half-bandwithds of the banded Jacobian, excluding tne main diagonal.
- `IPAR(7)`: mu, upper half-bandwithds of the banded Jacobian, excluding tne main diagonal.
- `IPAR(8)`: Extra printing.
  - 0 : no printing.
  - 1 : for minimal printing.
  - 2 : for full printing.
- `IPAR(9)`: mfmass (method flag for mass-matrix).
  - 0 : No mass-matrix used (non-stiff method).
  - 1 : User supplied full mass-matrix (stiff).
  - 2 : User supplied banded mass-matrix (stiff).
  - 10 : Identity mass-matrix is used (stiff).
- `IPAR(10)`: mlmass, lower half-bandwithds of the banded mass matrix, excluding the main diagonal.
- `IPAR(11)`: mumass, upper half-bandwithds of the banded mass matrix, excluding the main diagonal.
- `IPAR(12)`: Maximum number of steps allowed during one call to the solver.
- `IPAR(13)`: Maximum order to be allowed (default values : 12 if meth = 1, 5 if meth = 2). If exceds the default value, it will be reduced to the default value. In DASSL, DASPK and DGELDA : (1 .LE. MAXORD .LE. 5).

The rest of the parameters are specified using the rest of the IPAR values. Each one of the packages have an area reserved, which ranges

- ODEPACK Package:
  - `IPAR(16)`: Status flag.
  - `IPAR(17)`: Optional inputs, must be 0.
  - `IPAR(18)`: Maximum number of messages printed, default value is 10.
- LSODE/S:
  - `IPAR(19)`: Maximum order to be allowed.
- LSODA:
  - `IPAR(20)` : jt, Jacobian type indicator, set equal to 2
  - `IPAR(21)`: Flag to generate extra printing at method switches:
  - `IPAR(22)`: Maximum order to be allowed for the nonstiff method, default value is 12. If exceds the default value, it will be reduced to this value.

- `IPAR(23)`: Maximum order to be allowed for the stiff method, default value is 5. If exceds the default value, it will be reduced to this value.

- LSOIBIT:
  - `IPAR(24)`: mb, block size ((mb .GE. 1) and mb*IPAR(25) = NX).
  - `IPAR(25)`: nb, number of blocks in the main diagonal ((nb .ge. 4) and nb*IPAR(24) = NX).

- RADAU5:
  - `IPAR(26)`: Transforms the Jacobian matrix to Hessenberg form.
  - `IPAR(27)`: Maximum number of Newton iterations.
  - `IPAR(28)`: Starting values for Newton's method
  - `IPAR(29)`: Dimension of the index 1 variables.
  - `IPAR(30)`: Dimension of the index 2 variables.
  - `IPAR(31)`: Dimension of the index 3 variables.
  - `IPAR(32)`: Switch for step size strategy.
  - `IPAR(33)`: Value of M1.
  - `IPAR(34)`: Value of M2.

- DASSL and DASPK:
  - `IPAR(36)`: Initialization. Must set to 0 to indicate the start of every new problem.
    - 0 : Yes (on each new problem).
    - 1 : No (allows 500 new steps).
  - `IPAR(37)`: The code solve the problem without invoking any special non negativity contraints:
    - 0 : Yes.
    - 1 : To have constraint checking only in the initial condition calculation.
    - 2 : To enforce nonnegativity in X during the integration.
    - 3 : To enforce both options 1 and 2.
  - `IPAR(38)`: Solver try to compute the initial T, X and XPRIME:
    - 0 : The initial T, X and XPRIME are consistent.
    - 1 : Given X_d calculate X_a and X_d' (X_d differential variables in X, X_a algebraic variables in X).
    - 2 : Given $\dot{X}$ calculate X. (X_d differential variables in X, X_a algebraic variables in X).

- DASPK:
  - `IPAR(39)`: DASPK use:
    - 0: Direct methods (compatible with DASSL).

    - 1: Krylov method
  - `IPAR(40)`: DASPK uses scalars MAXLm KMP, NRMAX and EPLI when uses Krylov method.
    - 0: Uses default values.
    - 1: Uses user values.
  - `IPAR(41)`: Integration after the initial condition calculation is done. Used when `INFO(11) > 0`.
    - 0: Yes.
    - 1: No.
  - `IPAR(42)`: Errors are controled localy on all the variables.
    - 0: Yes.
    - 1: No .
  - `IPAR(43)`: Use default values for initial condition heuristic controls.
    - 0: Yes.
    - 1: No and provide MXNIT, MXNJ, MXNH, LSOFF, STPTOL, EPINIT.
  - `IPAR(44)`: Maximum number of iterations in the SPIGMR algorithm (.LE. NX).
  - `IPAR(45)`: Number of vectors on which orthogonalization is done in the SPIGMR algorithm (.LE. IPAR(44)).
  - `IPAR(46)`: Maximum number of restarts of the SPIGMR algorithm per nonlinear iteration (.GE. 0).
  - `IPAR(47)`: Maximum number of Newton iterations per Jacobian or preconditioner evaluation (> 0).
  - `IPAR(48)`: Maximum number of Jacobian or preconditioner evaluations (> 0).
  - `IPAR(49)`: Maximum number of values of the artificial stepsize parameter H to be tried if IPAR(38) = 1 (> 0).
  - `IPAR(50)`: Flag to turn off the linesearch algorithm.
    - 0: On.
    - 1: Off (default).

- DGELDA:
  - `IPAR(51)`: Contains the strangeness index.
  - `IPAR(52)`: Number of differential components.
  - `IPAR(53)`: Number of algebraic components.
  - `IPAR(54)`: Number of undetermined components.
  - `IPAR(55)`: Method used:
    - 1: Uses the BDF solver.
    - 2: Uses the Runge-Kutta solver.
  - `IPAR(56)`: E(t) and A(t) are:

    * 0: Constants.
    * 1: Time Dependent.
  - IPAR(57): Maximum index of the problem (.GE. 0).
  - IPAR(58): Step size strategy:
    * 0: Mod. predictive controlled of Gustafsson(safer).
    * 1: Classical step size control(faster).

4.3 The Auxiliar Routines

The system model is specified through the ODEDER_ (or DAEDF_, DAEDA_ and DAEDE_ in the case of DAEs) subroutine arguments. In the Fortran case, the value of this parameters is not relevant. It was initially planned to specify the subroutine names but since FORTRAN-77 does not allow function pointer stored as common variables, the function names could not be accessed by the packages. It would have been possible if the integrator packages were modified, but this could bring errors when updating the integrator version. In this case, the names ODEDER_ for ODEs and DAEDF_, DAEDA_ and DAEDE_ in the case of DAEs must be used. A similar work has been performed with the Jacobian routines JACX_, JACY_ and JACU_.

These user-defined subroutines are finally called by the stub routines. The stub routines have been designed to transform the common format into the package-dependant format.

In the case of MATLAB the functions can be specified through the arguments. In this case, and since MATLAB does not provide an easy way to specify routines as arguments, the name of the MATLAB source code ('.m') file is specified. Thus, the user has to write one module for each one of the functions of the system.

These functions will be stored in the gateway system. When this information is required, each one of the packages makes a call to the stub routines included in the gateway. These routines make the argument translation and call the MATLAB user defined files (see Implementation section for more details).

4.4 Calling the Solver

Once the solver parameters have been set-up, the solver driver subroutines ODESolver abd DAESolver can be called. The only parameter that must be modified when changing the package is the solver index ISOLVER. The syntax of each parameter can be found in section 1.

```
      SUBROUTINE ODESolver(ISOLVER, CODEDER_, CODEOUT_, CJACFX_,
     $                     CJACFU_, CJACFP_,
     $                     NX, NY, NU, TINI, TOUT, X, U, Y,
     $                     IPAR, DPAR, RTOL, ATOL,
     $                     IWORK, LIWORK, DWORK, LDWORK,
     $                     IWARN, INFO)
```

and for the MATLAB case

```
[X,Y,IPAR,DPAR,IWARN,INFO] = ODESolver(ISOLVER, ODEDER, ODEOUT, JACFX,
                                       JACFU, JACFP,
                                       X, U, P, TINI, TOUT,
                                       IPAR, DPAR, RTOL, ATOL,
                                       IWORK, DWORK);
```

Or in the case of DAEs:

```
 CALL DAESolver(ISOLVER, CDAEDF_,CDAEDA_, CDAEDE_, CDAEOUT_,
&                      CJACFX_, CJACFU_, CJACFP_, CJACFXDOT_,
&                      NX, NY, NU, NP, TINI, TOUT,
&                      X, XDOTI, Y, U, P,
&                      IPAR, DPAR, RTOL, ATOL,
&                      IWORK, LIWORK, DWORK, LDWORK, IWARN, INFO)
```

and for the MATLAB case

```
[X,Y,IPAR,DPAR,IWARN,IERR] = DAESolver(ISOLVER, DAEDF,DAEDA,DAEDE,DAEOUT
                                       JACFX, JACFU, JACFP, JACFXDOT,
                                       X, XDOT, U, P, TINI, TOUT,
                                       IPAR, DPAR )
```

4.5 The Output

The different packages use different locations for the output data. ODEPACK and RADAU5 use the `IWORK` array to show the results. `IWORK` array format has not been normalised and should not be used as output, since in some cases values can be undefined. Parameter arrays will be used instead. These arrays will be automatically filled in with the proper values in the proper positions, thus giving a coherent format. The output values included in the parameter arrays `IPAR` and `DPAR` are:

- ODEPACK:
  - `DPAR(111)`: Step size in t last used (successfully).
  - `DPAR(112)`: Step size to be attempted on the next step.
  - `DPAR(113)`: Current value of the independent variable which the solver has actually reached.
  - `DPAR(114)`: Tolerance scale factor, greater than 1.0.
- LSODA:
  - `DPAR(115`: Value of t at the time of the last method switch, if any.
- DASPK:

- `DPAR(111)`: Step size in t last used (successfully).
- `DPAR(136)`: Current value of the independent variable which the solver has actually reached.
- `DPAR(137)`: Stopping point (Tstop).

And the output values included in the parameter array `IPAR` are:

- All Packages:
  - `IPAR(101)`: Number of steps taken for the problem.
  - `IPAR(102)`: Number of f evaluations.
  - `IPAR(103)`: Number of Jacobian evaluations.
- ODEPACK:
  - `IPAR(111)`: The method order last used(successfully).
  - `IPAR(112)`: The order to be attempted on the next step.
  - `IPAR(113)`: Index of the component of largest in the weighted local error vector ($e(i)/ewt(i)$).
  - `IPAR(114)`: Length of rwork actually required.
  - `IPAR(115)`: Length of iwork actually required.
- LSODA:
  - `IPAR(116)`: Method indicator for the last successful step.
  - `IPAR(117)`: Current method indicator.
- LSODES:
  - `IPAR(118)`: Number of nonzero elements in the Jacobian matrix, including the diagonal (miter = 1 or 2).
  - `IPAR(119)`: Number of groups of column indices, used in difference quotient Jacobian aproximations if miter = 2.
  - `IPAR(120)`: Number of sparse LU decompositions.
  - `IPAR(121)`: Base address in rwork of the history array.
  - `IPAR(122)`: Base address of the structure descriptor array ian.
  - `IPAR(123)`: Base address of the structure descriptor array jan.
  - `IPAR(124)`: Number of nonzero elements in the strict lower triangle of the LU factorization.
  - `IPAR(125)`: Number of nonzero elements in the strict upper triangle of the LU factorization.
- RADAU5:

- `IPAR(126)`: Number of accepted steps.
- `IPAR(127)`: Number of rejected steps.
- `IPAR(128)`: Number of LU decompositions of both matrices.
- `IPAR(129)`: Number of forward-backward substitutions, of both systems.

- DASSL and DASPK:
  - `IPAR(136)`: Total number of error test failures so far.
  - `IPAR(137)`: Total number of convergence test failures.

- DASPK:
  - `IPAR(138)`: Number of convergence failures of the linear iteration.
  - `IPAR(139)`: Length of IWORK actually required.
  - `IPAR(140)`: Length of RWORK actually required.
  - `IPAR(141)`: Total number of nonlinear iterations.
  - `IPAR(142)`: Total number of linear (Krylov) iterations
  - `IPAR(143)`: Number of PSOL calls.

- GELDA:
  - `IPAR(146)`: The derivative degree of the function expected by the solver (IDIF parameter).

4.6 An Example

The following is an example of how to call the standard. The example problem has been obtained from LSODE ODEPACK routine. Two versions (the former in Fortran and the latter in MATLAB are provided.

4.7 Fortran-77 Version

```
c example problem.
c
c the following is a simple example problem, with the coding
c needed for its solution by lsode.  the problem is from chemical
c kinetics, and consists of the following three rate equations..
c     dy1/dt = -.04*y1 + 1.e4*y2*y3
c     dy2/dt = .04*y1 - 1.e4*y2*y3 - 3.e7*y2**2
c     dy3/dt = 3.e7*y2**2
c on the interval from t = 0.0 to t = 4.e10, with initial conditions
c y1 = 1.0, y2 = y3 = 0.  the problem is stiff.
c
c the following coding solves this problem with lsode, using mf = 21
```

```
c and printing results at t = .4, 4., ..., 4.e10.  it uses
c itol = 2 and atol much smaller for y2 than y1 or y3 because
c y2 has much smaller values.
c at the end of the run, statistical quantities of interest are
c printed (see optional outputs in the full description below).
c
c the output of this program (on a cdc-7600 in single precision)
c is as follows..
c
c   at t =  4.0000e-01   y =  9.851726e-01  3.386406e-05  1.479357e-02
c   at t =  4.0000e+00   y =  9.055142e-01  2.240418e-05  9.446344e-02
c   at t =  4.0000e+01   y =  7.158050e-01  9.184616e-06  2.841858e-01
c   at t =  4.0000e+02   y =  4.504846e-01  3.222434e-06  5.495122e-01
c   at t =  4.0000e+03   y =  1.831701e-01  8.940379e-07  8.168290e-01
c   at t =  4.0000e+04   y =  3.897016e-02  1.621193e-07  9.610297e-01
c   at t =  4.0000e+05   y =  4.935213e-03  1.983756e-08  9.950648e-01
c   at t =  4.0000e+06   y =  5.159269e-04  2.064759e-09  9.994841e-01
c   at t =  4.0000e+07   y =  5.306413e-05  2.122677e-10  9.999469e-01
c   at t =  4.0000e+08   y =  5.494529e-06  2.197824e-11  9.999945e-01
c   at t =  4.0000e+09   y =  5.129458e-07  2.051784e-12  9.999995e-01
c   at t =  4.0000e+10   y = -7.170586e-08 -2.868234e-13  1.000000e+00
c
c   no. steps = 330  no. f-s = 405  no. j-s =  69
C
      program ex1
      external fex, jex
C
      parameter(nd=400,lwork=4*nd*nd+12*nd+20,liwork=nd*nd+20)
      integer liwork, lwork, nd
C
      double precision atol, rtol, rwork, t, tout, y
      dimension y(3), atol(3), rwork(lwork), iwork(liwork)
      integer numarg
      INTEGER iSolverID
      character*80 arg
C
      integer ipar(400)
      double precision rpar(400)
      integer idid, iwarn
C
C      ... The code starts here
C
C      ... The LSODE solver is selected
C
      iSolverID = 1
```

```
C      iSolverID = 2
C      iSolverID = 5
C      iSolverID = 6
C      iSolverID = 7
C      iSolverID = 8
C      iSolverID = 9
C
C      ... The example starts here...
C
      neq = 3
      y(1) = 1.d0
      y(2) = 0.d0
      y(3) = 0.d0
      t = 0.d0
      tout = .4d0
      rtol = 1.d-4
      atol(1) = 1.d-6
      atol(2) = 1.d-10
      atol(3) = 1.d-6
C
C         .. Initial time step size ..
C
      rpar(1) = 1E-7
C
C         .. itol, Indicates if atol is array or escalar, 0=scalar ..
C
      ipar(1) = 1
C
C         .. mf, method flag, Stiff, User supplied full jacobian
C
      ipar(5) = 1
C
C         .. dense storage ..
C
      ipar(6) = 0
C
      do 40 iout = 1,10

      SUBROUTINE ODESolver(ISOLVER, fex, 0, jex, 0, 0,
     $                     neq, 0, 0, t, tout, y, 0, 0,
     $                     ipar, dpar, rtol, atol,
     $                     iwork, liwork, rwork, ldwork,
     $                     iwarn, idid)
C
        write(6,20)t,y(1),y(2),y(3)
```

```
  20    format(7h at t =,e12.4,6h   y =,3e14.6)
        if (idid .lt. 0) go to 80
  40    tout = tout*10.d0
      write(6,60) ipar(27),ipar(28),ipar(29)
  60  format(/12h no. steps =,i4,11h  no. f-s =,i4,11h  no. j-s =,i4)
      stop
  80  write(6,90)idid
  90  format(///22h error halt.. istate =,i3)
      stop
C *** Last line of PROGRAM ***
      END
C
C *** SUBROUTINE ODEDER starts ***
C
      SUBROUTINE ODEDER( NX, NU, T, X, U, DPAR, IPAR, F, IERR)
C
C     .. Parameters ..
C
C     .. Scalar Arguments ..
      INTEGER NX, NU
      INTEGER IERR
      DOUBLE PRECISION T
C     .. Array Arguments ..
      DOUBLE PRECISION X(NX), U(NU), F(NX), DPAR(*)
      INTEGER IPAR(*)
C
C     .. Executable Statements
C
      F(1) = -.04d0*X(1) + 1.d4*X(2)*X(3)
      F(3) = 3.d7*X(2)*X(2)
      F(2) = -F(1) - F(3)
      return
C *** Last line of ODERER ***
      END
C
C *** SUBROUTINE JACFX starts ***
C
      SUBROUTINE JACFX(NX, NU, LDFX, T, X, U, DPAR, IPAR, FX, IERR)
C
C     .. Parameters ..
C
C     .. Scalar Arguments ..
      INTEGER NX, NU, LDFX
      DOUBLE PRECISION T
      INTEGER IERR
```

```
C     .. Array Arguments ..
      DOUBLE PRECISION X(NX), U(NU), FX(LDFX, NX), DPAR(*)
      INTEGER IPAR(*)
C
C     .. Executable Statements
C
      FX(1,1) = -.04d0
      FX(1,2) = 1.d4*X(3)
      FX(1,3) = 1.d4*X(2)
      FX(2,1) = .04d0
      FX(2,3) = -FX(1,3)
      FX(3,2) = 6.d7*X(2)
      FX(2,2) = -FX(1,2) - FX(3,2)
      return
C *** Last line of JACFX ***
      END
```

4.8 MATLAB

```
ODEDER='fchemakzo';
JACFX='jchemakzo';
X=[ 0.437 0.00123 0 0 0 0.367 ]';

N=10;
TOUT= 0.4;
tol = 1E-4;

ODEOUT='';
JACFU='';
JACFP='';

U=[];
P=[];

TINI=0;

RTOL=tol;
ATOL=tol;

DWORK=zeros(10000,1);
IWORK=zeros(10000,1);

for I=1:N
```

```
    DPAR=zeros(400,1);
    IPAR=zeros(400,1);

    IPAR(2) = 1;

  [X,Y,IPAR,DPAR,IWARN,INFO] = ODESolver(ISOLVER, ODEDER, ODEOUT, JACFX,
   JACFU, JACFP,  TINI, TOUT, X, U, P, IPAR, DPAR, RTOL, ATOL, IWORK, DWORK);

  end


function [ydot,ipar,dpar,info]=fex(t,y,u,p,ipar,dpar)

  ydot(1) = -.04d0*y(1) + 1.d4*y(2)*y(3);
  ydot(3) = 3.d7*y(2)*y(2);
  ydot(2) = -ydot(1) - ydot(3);

  return

function [pd,ipar,dpar,info]=jex(t,y,ipar,dpar)

  pd(1,1) = -.04d0;
  pd(1,2) = 1.d4*y(3);
  pd(1,3) = 1.d4*y(2);
  pd(2,1) = .04d0;
  pd(2,3) = -pd(1,3);
  pd(3,2) = 6.d7*y(2);
  pd(2,2) = -pd(1,2) - pd(3,2);

  return;
```

5 Implementation

5.1 General Rules

The implementation of the interfaces is mainly the same for every package. First, the call to the general solver package is performed, by updating the sintax of the call and providing the workspace needs or local information variables.

All the information specific to each package is embedded in the parameter vectors, thus resulting in a minimum impact on the users caller code.

The most difficult part arises when dealing with the function arguments. In several cases, they get a fixed value (such as identity in the case of the mass matrix for RADAU5 and ODEs), other cases require a transformation routine to change the parameters specification and finally in other cases special work (such as algebraic operations and calls to several routines) is needed.

When the package does not make use of more detailed information, such as the parameter or input vectors, a check is performed in order to ensure that a null vector (zero-sized) has been chosen as argument. Otherwise, a warning will be issued.

It is important to note that no reference to parameters or input values can be provided through the function arguments. If required, the caller always can make use of COMMON blocks. So, for example in LSODE case, a stub routine (`FLSODE`) is provided to interface between the LSODE F specification and the general `ODERER` subroutine.

5.2 MATLAB Gateway

The most difficult part arises when dealing with the function arguments. The user defines the function subroutines in MATLAB code. The name of each module is indicated in the argument list. Since the SLICOT interface deals with user routines in binary code (compiled FORTRAN, C, ..), a gateway routine is automatically provided. This subroutine is called by each one of the packages and prepares the arguments to call the MATLAB interpreter engine. Through a external call, the MATLAB code is executed and its result is returned back to the gateway. This gateway adapts again the output arguments of the MATLAB user function and return those to the SLICOT solver package.

The result of the package is returned to the MATLAB system also through the gateway. This four-way interface is graphically depicted in figure 1.

Both Fortran-77 and MATLAB libraries are written in a single module. Since the integrator package should call either the Fortran-77 or MATLAB versions of the user-supplied routines, Different versions of the library must be generated. Link to MATLAB routines is produced when compiling the library with the `LINKMATLAB` definition. In such case, instead of calling `ODEDER_`, `ODEOUT` and the like, calls to `MATODEDER`, `MATODEOUT` and similar are produced. These internal subroutines automatically bind to the user-supplied functions through `MEXCALLMATLAB` MATLAB API. Input and Output arguments are tailored to fit the API sintax.

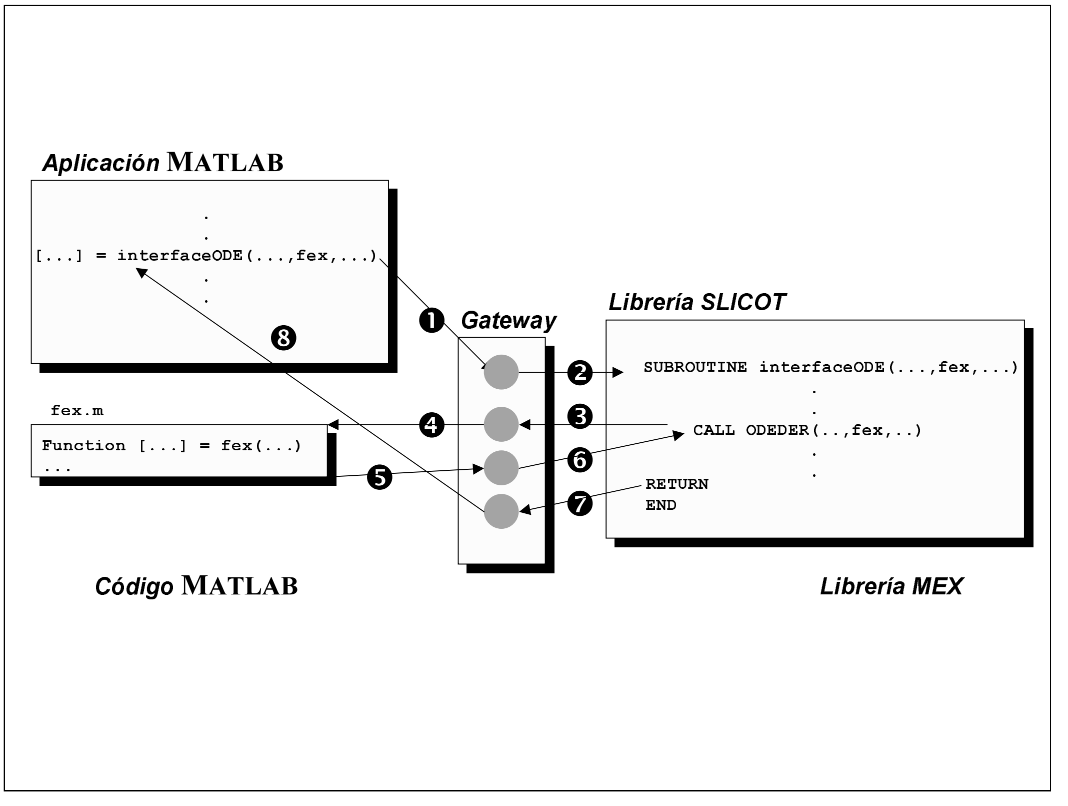
Checking is also performed with the parameters coming from MATLAB before they are used in the SLICOT call. Working arrays must also be supplied by the caller. MATLAB arrays will be used as auxiliar workspace by the SLICOT Fortran-77 packages.

5.3 ODEPACK

Stub routines are provided for calling the user-defined routines from the special sintax of ODEPACK. In ODE cases (LSODE, LSODA and LSODES) the interface is quite straightforward, since only the argument sintax must be altered.

However, this is not the case of the DAE solvers, which also the function meaning, that is the output of the function, must be changed.

LSODE, LSODA, LSODES



32

.

Figure 1: MATLAB Gateway Structure.

The user must specify a routine that could provide the value of the f function

$$\dot{x}(t) = f(t, x).$$

Stub routines called `FLSODE`, `FLSODA` and `FLSODES` are provided to interface between the LSODE, LSODA and LSODES f specification and the general `ODERER` subroutine. The same is for the stub routines called `JLSODE`, `JLSODA` and `JLSODES` which interface between the Jacobian specification LSODE, LSODA and LSODES and the standarised `JACFX` subroutine.

LSODI, LSOIBT

The problems that can be solved with this package are limited to the expression:

$$F(x(t), t)\dot{x}(t) = A(x(t), t).$$

LSODI and LSOIBT require two subroutines for computing the contribution of $A(x(t), t)$ and the residual factor.

The approximation of the derivative provided by the user-defined function through the residual factor is

$$r = A(x(t), t) - F(x(t), t)\dot{x}(t),$$

and the mass matrix is provided through the `adda` subroutine which computes the expression

$$F(x(t), t) = F(x(t), t) + L,$$

with L an internal matrix factor computed by ODEPACK in each stage.

Actually, the expected output of these routines is not the same as the subroutines of the standard (`DAEDF` and `DAEDA`) which define the contribution of the $F(x(t), t)$ and $A(x(t), t)$ matrices. However, it can be automatically computed by calling the standard subroutines which converts the results to the expected output. Subroutines `RLSODI` and `RLSOIBT` compute this transformation. The expression of the mass matrix is automatically computed by the `ALSODI` and `ALSOIBT` stub routines, which call the user-supplied function `DAEDA`.

The behaviour of the routine is the same as LSODI, with the only difference that it is designed for working with block-tridiagonal systems. The same stub routines are provided (`RLSOIBT` and `ALSOIBT`) oriented to work with block-tridiagonal systems. The user must supply the routines `DAEDE` and `DAEDA` oriented to deal with block-tridiagonal matrices.

5.4 RADAU5

This package computes the numerical solution of a differential algebraic system of first order ordinary diferential equations in the form

$$M\dot{y} = F(x, y)$$

The subroutine which serves as interface between RADAU5 $F(x,y)$ specification and the general ODERER subroutine is called FRADAU5. In the case of the Jacobian specification, JRADAU5 stub routine is used to computes the Jacobian of F with respect to x for the

$$\dot{x} = F(x(t), u(t), p, t)$$

5.5 DASSL

DASSL package solves a system of differential/algrebraic equations of the form

$$G(t, y, \dot{y}) = 0 \tag{2}$$

from T (initial time) to TOUT (final time).

Due to the consistency of x, $\dot{x}$ and t a previous called to ODEDER routine must be done to compute $\dot{x}$. Then a call to DASSL subroutine can be done. FDASSL routine should return the residual of the defferential/algebraic system. So, first a call to ODEDER routine is done to compute the value of the function, and then to rest this value with the value of the derivatives:

```
CALL ODEDER_( NX_, 0, T, X, 0.0D0, RPAR, IPAR, DELTA, IERR)
DO 100 I=1, NX_
       DELTA(I) = -DELTA(I)+XPRIME(I)
 100  CONTINUE
```

JDASSL routine must evaluate the non-zero partial derivates for each equation and each solution component, and store these values in the matrix PD. The elements of PD are set to zero before each call to JAC, so only non-zero elements need to be defined. To carry out this task, firstly a call to JACFX routine is done to compute the Jacobian matrix.

Once the first term of the above expression is computed, there exist two different situations: If the pacakge is dealing with an ODE system, the derivative of the independent variable is explicit, and so, the Jacobian of the left side of the equation with respect $x^{\cdot}$ is the identity. In this case the value CJ is substracted from the diagonal elements of PD. On the other hand, if the system dealing with is a DAE, the derivative is not explicit and the Jacobian is the mass matrix. The DAEDA function is evaluated and weights CJ before they are substracted from each element.

5.6 DASPK

DASPK solves large-scale systems of differential algebraic equations of the form

$$F(t, y, \dot{y}) = 0.$$

All the comments regarding the user-defined functions in the DASSL section can be applied to DASPK. In addition, the implementation had to deal with the preconditioner solvers.

If an iterative solver is selected, a routine for computing and applying the preconditioner should be supplied. If not supplied, the convergence could be very slow or even the algorithm could diverge. DASPK provides two pairs of subroutines to deal with Banded Jacobian and Incomplete LU Factorization preconditioners. An argument in the interface enables to use any of them. In order to avoid particularities, the interface is not open to user-defined preconditioner routines.

5.7 DGELDA

DGELDA solves linear differential algebraic equations with variable coefficients of the form

$$\begin{aligned} E(t)\dot{x}(t) &= A(t)x(t) + f(t) \\ x(t_0) &= x_0 \end{aligned}$$

The potential of GELDA is oriented to differential equations with time-varing coefficients. Since it requires linear equations, the interface for ODE is not valid. The interface for DAE is also particular for the case of GELDA. Since a new variation in the interface was required, and the DAEs include ODEs, GELDA is only available from the DAE interface.

The changes are minimum, since a special interface was dedicated to it. It was only require to adapt the parameter array and to include the `IDIF` argument as a component of the `IPAR` array.

References

[1] A.C. Hindmarsh, *Brief Description of ODEPACK – A Systematized Collection of ODE Solvers*, http://www.netlib.org/odepack/doc

[2] L.R. Petzold *DASSL Library Documentation*, http://www.netlib.org/ode/

[3] P.N. Brown, A.C. Hindmarsh, L.R. Petzold, *DASPK Package – 1995 Revision*

[4] R.S. Maier, *Using DASPK on the TMC CM5. Experiences with Two Programming Models*, Minesota Supercomputer Center, Technical Report.

[5] E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems.*, Springer Seried in Computational Mathermatics 14, Springer-Verlag 1991, Second Edition 1996.

[6] P. Kunkel, V. Mehrmann, W. Rath und J. Weickert, 'GELDA: A Software Package for the Solution of General Linear Differential Algebraic equations', *SIAM Journal Scientific Computing*, Vol. 18, 1997, pp. 115 - 138.

[7] M. Otter, *DSblock: A neutral description of dynamic systems. Version 3.3*, http://www.netlib.org/odepack/doc

[8] M. Otter, H. Elmqvist, *The DSblock model interface for exchanging model components*, Proceedings of EUROSIM 95, ed. F.Brenenecker, Vienna, Sep. 11-15, 1995

[9] M. Otter, *The DSblock model interface, version 4.0*, Incomplete Draft, http://dv.op.dlr.de/õtter7dsblock/dsblock4.0a.html

[10] Ch. Lubich, U. Novak, U. Pohle, Ch. Engstler, *MEXX - Numerical Software for the Integration of Constrained Mechanical Multibody Systems*, http://www.netlib.org/odepack/doc

[11] Working Group on Software (WGS), *SLICOT Implementation and Documentation Standards (version 1.0)*, WGS-Report 90-1, Eindhoven University of Technology, May 1990.

[12] P. Kunkel and V. Mehrmann, *Canonical forms for linear differential-algebraic equations with variable coefficients.*, J. Comput. Appl. Math., 56:225–259, 1994.

[13] Working Group on Software (WGS), *SLICOT Implementation and Documentation Standards*, WGS-Report 96-1, Eindhoven University of Technology, [11] updated: February 1998, ftp://wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/rep96-1.ps.Z.

[14] A. Varga, *Standarization of Interface for Nonlinear Systems Software in SLICOT*, Deutsches Zentrum für Luft un Raumfahrt, DLR. SLICOT-Working Note 1998-4, 1998, Available at ftp://wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/SLWN1998-4.ps.Z.