SLICOT Working Note 1998-4

Standardization of Interface for Nonlinear Systems Software in SLICOT [1]

Andras Varga[2]

June 1998

[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/pub/WGS/REPORTS/SLWN1998-4.ps.Z`

[2]Deutsches Zentrum für Luft und Raumfahrt, Institut für Robotik und Systemdynamik, DLR Oberpfaffenhofen, Postfach 1116, D-82230 Wessling, Germany

Contents

1 Introduction

The class of nonlinear dynamic systems models to be addressed are described either by *ordinary differential equations* (ODEs)

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t), p, t) \\ y(t) &= g(x(t), u(t), p, t), \end{aligned} \tag{1}$$

or by *differential-algebraic equations* (DAEs)

$$\begin{aligned} 0 &= f(\dot{x}(t), x(t), u(t), p, t) \\ y(t) &= g(\dot{x}(t), x(t), u(t), p, t), \end{aligned} \tag{2}$$

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

The ODE and DAE classes of nonlinear dynamics models include multibody mechanical systems, many chemical process models, dynamic aircraft models, electric circuits, and so on. An important characteristic of all the above models is the explicit parametric dependence, that is, parameters like masses, moments of inertia, heights, etc. are explicitly contained in the model definition. This allows for instance to perform parameter studies or parameter tuning on the above nonlinear models.

In this paper we discuss the development of standardized Fortran interfaces for the description of nonlinear systems to allow an easy interfacing with standard software for integration of differential equations, nonlinear programming and solving of nonlinear equations. A set of low-level interface subroutines is proposed to define parametrized, nonlinear dynamic systems in form of input-output "blocks". A nonlinear system model which is described according to this interface definition is called **DSblock** (Dynamic System block). The DSblock interface may serve as a neutral *model bus* to exchange nonlinear models between different modelling and simulation environments. The proposed DSblock interface is compatible with the automatically generated DSblock codes in modern object oriented modelling environments (like the Dymola[1] software based on the **Modelica** modelling language[2]).

The present proposal for the DSblock Fortran interface is a simplified version of the DSblock Standard, Version 3.3[3] developed by Martin Otter from DLR. Note that all computational facilities provided in ANDECS for nonlinear systems (simulation, trimming, linearization) are based on the DSblock 3.3 interface. The new Version 4.0 of this standard[4], has C++ as target language, and will be soon available. This standard will be directly supported by the code generation tools in Dymola. A mapping of our Fortran interface proposal to the C++ DSblock interface is in principle easy to be implemented (especially in Fortran 90) and can be done in a later phase of the NICONET project.

[1]http://www.dynasim.se/products.html
[2]http://www.Dynasim.se/Modelica/
[3]see http://www.Dynasim.se/Modelica/DSblock/DSblock33.txt
[4]http://www.op.dlr.de/ otter/dsblock/dsblock4.0a.html

2 Requirements for the Fortran Interface

The main requirement for the DSblock interface is the flexible interfacing with nonlinear systems software tools. The basic interface routines are those for evaluating the functions f and g. However, according to application type, (e.g. simulation, optimization), this minimal set can be extended with template routines which provide support for an easy interfacing with numerical integrators (e.g. ODEPACK[5] [3, 5]), nonlinear systems solvers or nonlinear programming tools (e.g. MINPACK [4]). Note that, the DSblock Version 3.3 has been developed mainly for simulation purposes and provides a complete set of interface routines (11 main routines, more than 10 servicing routines adapted to ANDECS needs) for the simulation of nonlinear systems. For example, a useful routine in this collection (which actually must be always present) serves for starting the simulation and for providing complete information on the models as the dimensions of variables, names of variable. This routine allows the choice of initial conditions and even of the integration method. For optimization and nonlinear system solvers similar routines are necessary. For numerical integrators and nonlinear system solvers a natural additional interface routine is to evaluate the Jacobian matrix $\partial f/\partial x$. Note that the use of analytic Jacobians can frequently lead to an increased effectiveness of the numerical integrators or of the nonlinear systems solvers. For the optimization of nonlinear systems, interfaces for the gradient of a special output y_0 (usually an integral cost function) with respect to a set of parameters $\tilde{p}$ is necessary to use more advanced, gradient based optimization techniques. The parameters $\tilde{p}$ can be the parameters of the nonlinear model, or the parameters intervening in the definition of the control inputs $u(t)$ (e.g. spline coefficients, controller parameters).

In the context of simulation it is important to provide support for state and time events. State-events can be expressed by algebraic conditions on special output variables

$$z(t) = h(x(t), u(t), p, t), \qquad (3)$$

where $z(t) \in \mathbb{R}^l$ is the vector of *crossing* variables. The zero crossing of any component of z can be used to define the occurrence of a state event.

Time events are used to stop the integration at specified time moments where for instance alternative input signals can be used or switching between different models can be performed. Step events are time events related to finishing a complete integration step.

An important aspect is the compliance with the new DSblock Version 4.0. To ensure the compatibility with the C++ targeted DSblock 4.0 **Integrator Interface** layer, standard Fortran–C++ interface routines will be implemented (either by vendors of software generating C++ DSblock codes) or by NICONET in a later phase. These routines are essential to ensure the upward compatibility of the SLICOT DSblock interface. Since the DSblock 4.0 interface covers a much broader class of systems (e.g. hybrid systems too), many interface routines to DSblock 4.0 will be simple dummy routines.

[5] http://ftp.tc.cornell.edu/UserDoc/Software/Num/odepack/odepack.manual

3 Standard Fortran Interface Routines for Nonlinear Systems

3.1 Subroutines for Nonlinear System Functions

The minimal set of routines to define a DSblock interface for a nonlinear system described by ODEs is formed from the following two routines:

Name	Function
ODEDER	evaluates the right hand side f of the ODE (1)
ODEOUT	evaluates the output signals function g in (1)

The detailed interface specifications for these routines are given in appendices A and B. Examples of interface routines for a DSblock describing a nonlinear pendulum are given in appendix H.

A similar set of routines to define a DSblock interface for a nonlinear system described by DAEs is formed from the following two routines:

Name	Function
DAEDER	evaluates the right hand side f of the DAE (2)
DAEOUT	evaluates the output signals function g in (2)

The detailed interface specifications for these routines are given in appendices C and D.

3.2 Subroutines for Nonlinear System Jacobians

The Jacobian $\partial f/\partial x$ of the nonlinear function in (1) is frequently required by numerical ODE integrators to increase their computational efficiency. Similarly, the routines to solve nonlinear systems of equations requires the Jacobian matrix with respect to indeterminates. For optimization or parameter estimation, the Jacobian $\partial f/\partial p$ is necessary to evaluate gradients.

The following routines represent standard interfaces for Jacobian matrices of nonlinear systems described by ODEs:

Name	Function
JACX	evaluates the Jacobian $\partial f/\partial x$ for the right hand side f of the ODE (1)
JACU	evaluates the Jacobian $\partial f/\partial u$ for the right hand side f of the ODE (1)
JACP	evaluates the Jacobian $\partial f/\partial p$ for the right hand side f of the ODE (1)

The detailed interface specifications for these routines are given in appendices E, F and G. Similar functions can be defined for the Jacobians of the output function g of the ODE (1). A completely analogous set of standard interfaces can be defined for DAEs as well.

3.3 Auxiliary Subroutines

The defined standard interfaces allow to construct easily interface routines to the main ODE and DAE integrators as well as to nonlinear system solvers or optimization routines. The defined set of routines can be extended by other standard routines to cover other functions in the definition of DSblock. Examples of such routines are:

- routines to generate input signals
- routines to load model parameters
- inquiry routines for the dimensions of the state, input, output and parameter vectors
- routines to set initial conditions
- routines to describe state or time events.

As the need arises, definitions of new standard interface routines will be added to the present standard proposal.

4 Example of Interfacing to ODE solvers

A thorough review of available ODE solvers has been done in two monographs dedicated to non-stiff ODEs [1] and stiff ODEs and DAEs [2]. Many versatile public domain codes to solve ODEs are available in Netlib, as for example, the well known ODE solver LSODE [5] belonging to the ODEPACK package[6] [3]. The general `ode` section of Netlib[7] also contains many powerful solvers for both ODEs and DAEs.

To use an ODE solver, as for example LSODE, it is necessary to prepare an interface routine which implements the evaluation of a nonlinear vectorial function $F(x(t), t)$ and possibly of its Jacobian. The function evaluation routine in LSODE must be of the form

```
SUBROUTINE F (NEQ, T, X, YDOT)
REAL*8 X(NEQ), XDOT(NEQ)
```

and supplies the vector function F by loading `XDOT(i)` with `F(i)`. Other solvers have slightly different interface routines, but with essentially the same functionality.

The routine for Jacobian has a more involved interface, allowing for instance to work with banded Jacobians:

```
SUBROUTINE JAC (NEQ, T, X, ML, MU, PD, NROWPD)
REAL*8 X(NEQ), PD(NROWPD,NEQ)
```

and supplies $\partial f/\partial x$ by loading the array `PD` as follows: for a *full* Jacobian, `PD(i,j)` is loaded with $\partial f_i/\partial x_j$ and the arguments `ML` and `MU` are ignored; for a *banded* Jacobian, `PD(i-j+MU+1,j)` is loaded with $\partial f_i/\partial x_j$, i.e. the diagonal lines of $\partial f/\partial x$ are loaded into the rows of `PD` from the top down. In either case, only nonzero elements need be loaded.

An example of an interface to LSODE for the DSblock of appendix H is given in appendix I. A more generic version of this routine would be to call standard routines to dynamically load

[6] http://www.netlib.org/odepack/
[7] http://www.netlib.org/ode/

the system parameters (allowing thus for an "adaptive" parameter steering) or to compute user defined input signals (as for instance, signals coming from a feedback coupling). Such a routine can serve as a template for interfacing with LSODE.

5 Conclusion

Although the proposed standard interfaces are intended mainly for an easy interfacing with popular ODE and DAE solvers, they can be used for interfacing with routines for nonlinear systems solvers or gradient based optimization as well. However, for the efficient evaluation of the gradients, special higher level interface routines are necessary, since the evaluation of each gradient component requires the integration of the ODE (1) over a time interval. Alternatively, reverse-time integrations can be occasionally used. The standardization of gradient computations raises several delicate technical aspects and will constitute an issue of further extension of the present standard.

References

[1] E. Hairer, S. P. Norsett, and G. Warnner. *Solving Ordinary Differential Equations, volume 1 : Non-Stiff Problems, Second Revised Edition.* Springer-Verlag, 1993.

[2] E. Hairer, S. P. Norsett, and G. Warnner. *Solving Ordinary Differential Equations, volume 2 : Stiff and Differential-Algebraic Problems, Second Revised Edition.* Springer-Verlag, 1993.

[3] A. C. Hindmarsh. ODEPACK, a systematized collection of ODE solvers. In R. S. Stepleman et.al., editor, *Scientific Computing*, pages 55–64. North-Holland, Amsterdam, 1983.

[4] J. J. Moré. User's Guide for MINPACK-1. Applied Mathematics Division Report ANL-80-74, Argonne National Laboratory, Argone,IL, 1980.

[5] K. Radhakrishnan and A. C. Hindmarsh. Description and use of LSODE, the Livermore solver for ordinary differential equations. Technical report, NASA Reference Publication 1327, and LLNL Report UCRL-ID-113855, 1993.

A Specification of ODEDER Interface Routine

```
SUBROUTINE ODEDER

Purpose:
Computes the right hand side F of the ODE

       dX/dT = F ( X(T), U(T), P, T )

Usage:
  CALL ODEDER (T, X, U, P,  F,  IERR)

  T     :  IN, DOUBLE
           Simulation time instant.
  X     :  IN, DOUBLE(*)
           Vector X at time T.
  U     :  IN, DOUBLE(*)
           Input signals U at time T.
  P     :  IN, DOUBLE(*)
           Double precision parameters.
  F     :  OUT, DOUBLE(*)
           Right hand side F(1), ..., F(NX)
           (NX is the dimension of the state vector)
  IERR  :  OUT, INTEGER
           Error parameter
           On return IERR = 0: No error occured.
                          > 0: An error occured during function evaluation.
```

B Specification of ODEOUT Interface Routine

```
SUBROUTINE ODEOUT

Purpose:
Computes the output signals Y for ODEs by evaluating

   Y = G ( X(T), U(T), P, T )

Usage:
  CALL ODEOUT (T, X, U, P, Y, IERR)

  T     :  IN, DOUBLE
           Simulation time instant.
  X     :  IN, DOUBLE(*)
           Vector X at time T.
  U     :  IN, DOUBLE(*)
           Input signals U at time T.
  P     :  IN, DOUBLE(*)
           Double precision parameters.
  Y     :  OUT, DOUBLE(NY)
           Output vector.
           (NY is the dimension of the output vector)
  IERR  :  OUT, INTEGER
           Error parameter
           On return IERR = 0: No error occured.
                          > 0: An error occured during function evaluation.
```

C Specification of DAEDER Interface Routine

```
SUBROUTINE DAEDER

Purpose:
Computes the right hand side F of the DAE

       0 = F ( XDOT(T), X(T), U(T), P, T )

Usage:
  CALL DAEDER (T, XDOT, X, U, P,  F,  IERR)

  T     :  IN, DOUBLE
           Simulation time instant.
  XDOT  :  IN, DOUBLE(*)
           Vector of derivatives XDOT at time T.
  X     :  IN, DOUBLE(*)
           Vector X at time T.
  U     :  IN, DOUBLE(*)
           Input signals U at time T.
  P     :  IN, DOUBLE(*)
           Double precision parameters.
  F     :  OUT, DOUBLE(*)
           Right hand side F(1), ..., F(NX)
           (NX is the dimension of the state vector)
  IERR  :  OUT, INTEGER
           Error parameter
           On return IERR = 0: No error occured.
                          > 0: An error occured during function evaluation.
```

D Specification of DAEOUT Interface Routine

```
SUBROUTINE DAEOUT

Purpose:
Computes the output signals Y for ODEs by evaluating

   Y = G ( XDOT(T),X(T), U(T), P, T )

Usage:
  CALL DAEOUT (T, XDOT, X, U, P, Y, IERR)

  T     :  IN, DOUBLE
           Simulation time instant.
  XDOT  :  IN, DOUBLE(*)
           Vector of derivatives XDOT at time T.
  X     :  IN, DOUBLE(*)
           Vector X at time T.
  U     :  IN, DOUBLE(*)
           Input signals U at time T.
  P     :  IN, DOUBLE(*)
           Double precision parameters.
  Y     :  OUT, DOUBLE(NY)
           Output vector.
           (NY is the dimension of the output vector)
  IERR  :  OUT, INTEGER
           Error parameter
           On return IERR = 0: No error occured.
                          > 0: An error occured during function evaluation.
```

E Specification of JACFX Interface Routine

```
SUBROUTINE JACFX

Purpose:
Computes the Jacobian of F with respect to X for the ODE

       dX/dT = F ( X(T), U(T), P, T )

Usage:
  CALL JACFX (T, X, U, P, FX, LDFX, IERR)

  T     :  IN, DOUBLE
           Simulation time instant.
  X     :  IN, DOUBLE(*)
           Vector X at time T.
  U     :  IN, DOUBLE(*)
           Input signals U at time T.
  P     :  IN, DOUBLE(*)
           Double precision parameters.
  FX    :  OUT, DOUBLE(LDFX,*)
           The NX by NX Jacobian of F with respect to X.
           (NX is the diemension of state vector X.)
           The element FX(i,j) is loaded with df_i/dx_j.
  LDFX  :  IN, INTEGER
           Leading dimension of FX. LDFX >= NX.
  IERR  :  OUT, INTEGER
           Error parameter
           On return IERR = 0: No error occured.
                          > 0: An error occured during function evaluation.
```

F Specification of JACFU Interface Routine

```
SUBROUTINE JACFU

Purpose:
Computes the Jacobian of F with respect to U for the ODE

       dX/dT = F ( X(T), U(T), P, T )

Usage:
  CALL JACFU (T, X, U, P, FU, LDFX, IERR)

  T     :  IN, DOUBLE
           Simulation time instant.
  X     :  IN, DOUBLE(*)
           Vector X at time T.
  U     :  IN, DOUBLE(*)
           Input signals U at time T.
  P     :  IN, DOUBLE(*)
           Double precision parameters.
  FU    :  OUT, DOUBLE(LDFX,*)
           The NX by NU Jacobian of F with respect to U.
           (NX, NU are the dimensions of state and
            input vectors X and U, respectively.)
           The element FU(i,j) is loaded with df_i/du_j.
  LDFX  :  IN, INTEGER
           Leading dimension of FX. LDFX >= NX.
  IERR  :  OUT, INTEGER
           Error parameter
           On return IERR = 0: No error occured.
                          > 0: An error occured during function evaluation.
```

G Specification of JACFP Interface Routine

```
SUBROUTINE JACFP


Purpose:
Computes the Jacobian of F with respect to P for the ODE

       dX/dT = F ( X(T), U(T), P, T )

Usage:
  CALL JACFP (T, X, U, P, FU, LDFX, IERR)

  T     :  IN, DOUBLE
           Simulation time instant.
  X     :  IN, DOUBLE(*)
           Vector X at time T.
  U     :  IN, DOUBLE(*)
           Input signals U at time T.
  P     :  IN, DOUBLE(*)
           Double precision parameters.
  FP    :  OUT, DOUBLE(LDFX,*)
           The NX by NP Jacobian of F with respect to P.
           (NX, NP are the dimensions of state and
            parameter vectors X and P, respectively.)
           The element FP(i,j) is loaded with df_i/dp_j.
  LDFX  :  IN, INTEGER
           Leading dimension of FX. LDFX >= NX.
  IERR  :  OUT, INTEGER
           Error parameter
           On return IERR = 0: No error occured.
                          > 0: An error occured during function evaluation.
```

H Pendulum Example

```
      SUBROUTINE PENDF (T, X, U, P, F, IERR)
C
C Realization of subroutine ODEDER for a simple pendulum model
C (calculates the right hand side f of the differential equation)
C
      DOUBLE PRECISION   T, X(*), U(*), P(*), F(*)
      INTEGER            IERR
C
      DOUBLE PRECISION   M, L, D, G
C
C Executable statements
C
      IERR = 0
C
C Determine system parameters
C
      M = P(1)
      L = P(2)
      D = P(3)
      G = P(4)
C
C Compute right hand side
C
      F(1) = X(2)
      F(2) = (U(1) - D*X(2))/(M*L*L) - (G/L)*SIN(X(1))
C
      RETURN
      END


      SUBROUTINE PENDG (T, X, U, P, Y, IERR)
C
C Realization of subroutine ODEOUT for a simple pendulum model
C (calculates the output signals y)
C
      DOUBLE PRECISION   T, X(*), U(*), P(*), Y(*)
      INTEGER            IERR
C
      DOUBLE PRECISION   L
C
C Executable statements
C
```

```
      IERR = 0
C
C Determine system parameters
C
      L = P(2)
C
C Compute output signals y
C
      Y (1) =  L*SIN(X(1))
      Y (2) = -L*COS(X(1))
      RETURN
      END
```

I LSODE Interface Routine Example

```
      SUBROUTINE F (NEQ, T, X, XDOT)
C
      INTEGER NEQ
      REAL*8 T, X(NEQ), XDOT(NEQ)
C
C Local variables
C
      INTEGER  IERR
      REAL*8   P(4), U(1)
      DATA P / 5.0D0, 1.0D0, 0.5D0, 9.81D0 /
C
C Compute the input function U(t)
C
      U(1) = SIN(T)
C
C Evaluate function
C
      CALL PENDF (T, X, U, P, XDOT, IERR)
C
      IF( IERR.NE.0 ) STOP
C
      RETURN
      END
```