Nonlinear Control Systems Simulation Toolbox in SLICOT [1]

Vicente Hernández [2], Ignacio Blanquer [2], Enrique Arias [3], Victor M. García [2], Lourdes Peñalver [4], Pedro Ruiz [2].

August, 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/SLWN2000-5.ps.Z*

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

[3]Departamento de Informática (DI). Universidad de Castilla La Mancha (UCLM). Albacete. Spain. {*earias*}*@info-ab.uclm.es.*

[4]Departamento de Informática de Sistemas y Computadores (DISCA). Universidad Politécnica de Valencia (UPV). Valencia. Spain. {*lourdes*}*@disca.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 different approaches.

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 objective of the interface is to facilitate the change of the solver package. It is not easy to standardise currently available libraries in non-linear systems. Neither the algebraic expression, nor the parameters involved are compatible in every system.

However, and for the sake of reusability, it has been considered to provide 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.

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

This section describes the structure of the SLICOT interface for ODEs and DAEs, including the syntax and the main details about how to use the interface. Section 2 covers the packages that are included in the interface, with a brief description and references to further information. Section number 3 describes the integration of the SLICOT interface in a user-friendly package. Section 4 presents a test battery that has been used for the validation of the interface, including several test cases from different areas of engineering. Finally, section 5 describes a test example coming from the robot manipulator industry in which the package is used for the optimal control of two Puma robot manipulators.

1.1 The ODE Solver

The expression of an ODE nonlinear control system is

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

where $x(t)$ is the state vector, $u(t)$ is the input vector, $y(t)$ is the output vector, p is the vector of parameters and t is the time.

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 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)
```

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)
```

which returns the expression $g(x(t), u(t), p, t)$.

If required, the expressions regarding the Jacobians are:

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

which returns the Jacobian $\frac{\partial f}{\partial x}$.

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

which returns the Jacobian $\frac{\partial f}{\partial u}$.

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

which returns the Jacobian $\frac{\partial f}{\partial p}$.

Each one of the packages includes stub routines that are automatically called by the solver package and 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 [18].

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.
- `DPAR`: 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.

1.2 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

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

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

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

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

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

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)
```

In this subroutine most of the parameters have a similar meaning as the ODE case.

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

This user-supplied subroutine has 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 $F(u(t), p, t)$.

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

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

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

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

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

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)
```

which returns the Jacobian $\frac{\partial f}{\partial x}$.

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

which returns the Jacobian $\frac{\partial f}{\partial \dot{x}}$.

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

which returns the Jacobian $\frac{\partial f}{\partial u}$.

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

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`.

2 Selection and collation of subroutines for standardisation

The packages that are being addressed are:

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

Following there is a brief description of each one.

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, LSODA and LSODES

LSODE is the basic solver of the package. The type of problems that the system is able to cope is

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

The user must specify the 'f' routine.

`LSODA` 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. `LSODES` 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).

LSODI, LSOIBT

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.

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 routines for computing the residual factor and the $A(x(t), t)$ function.

The residual factor is

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

in which the approximation of the derivative $\dot{x}(t)$ is provided by the solver routine as an argument.

The second subroutine to be provided computes the expression

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

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

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 ($M = I$). 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 ODEPACK solvers (`LSODE`, `LSODA`, or `LSODES`).

The user has to supply four subroutines. The first one computes the value of $F(x, y)$. The Jacobians could be provided if specified, through a routine that computes the partial derivatives of $F(x, y)$ with respect to y. The Jacobian can be expressed in a full or sparse storage. If not full, the Jacobian is taken as a banded matrix and the partial derivatives are stored diagonal-wise. If this subroutine is not specified, the Jacobian is computed internally by finite differences. The third subroutine computes the mass matrix M. As Jacobian case, if it is not specified it is assumed to be the identity matrix and needs not to be defined. 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, the numerical solution during integration has to be provided by the output subroutine, which furnishes the solution `'Y'` at the NR-th grid point `'X'` (thereby the initial values is the first grid-point).

2.3 DASSL

DASSL package solves a system of differential/algebraic 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 (BDF) 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.

Two subroutines need to be specified. One subroutine defines the differential/algebraic system. For the given values of t, y and $\dot{y}$, the subroutine should return the residual of the differential/algebraic system

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

A second routine should provide the Jacobian in the form

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

where CJ is a scalar which is input to the routine.

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 output matrix.

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 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 interpolates 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 is very similar to the DASSL syntax, as exception of an argument which defines a subroutine to be provided by the user for the preconditioning of the iterative Krylov linear solver. The purpose of the routine is to solve the linear system associated to the preconditioning matrix.

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.

Four subroutines define the behaviour of the system

- `EDIF`. This is a subroutine which the user provides to define the matrix $E(t)$ and its derivatives. The subroutine produces the i-th derivative of $E(t)$ at the time specified
- `ADIF`. This is a subroutine which the user provides to define the matrix $A(t)$ and its derivatives.
- `FDIF`. This is a subroutine which the user provides to define the vector $f(t)$ and its derivatives.

  The package can use BDF and Runge-Kutta solvers.

  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 Integration of subroutines in user-friendly environments

3.1 General Implementation

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 syntax of the call and providing the workspace needs or local information variables.

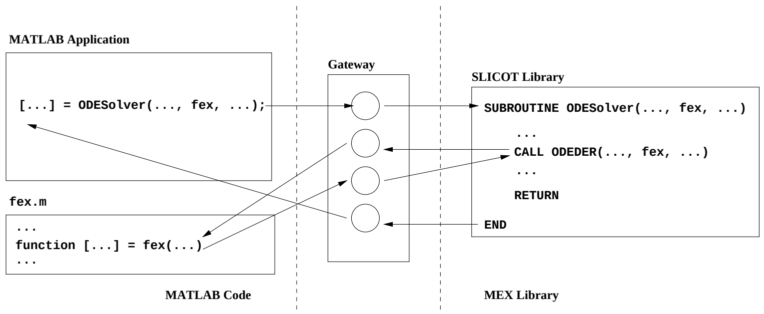

Figure 1: MATLAB Gateway Structure.

All the information specific to each package is embedded in the parameter vectors, thus resulting in a minimum impact on the user's calling 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 `ODEDER` subroutine.

3.2 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 an 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 syntax.

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 auxiliary workspace by the SLICOT Fortran-77 packages.

The syntax for the routines in MATLAB in the case of ODEs is

```
[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

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

More details can be found in [18].

4 Selection of benchmarks problems, testing and performance comparisons

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 default 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 Jacobian computations. Several of them are common to every package, and their position is normalised to standardise 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 standardised 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. The complete syntax for all parameters can be found in [18].

4.3 The Auxiliary 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 these 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 [18]).

4.4 Calling the Solver

Once the solver parameters have been set-up, the solver driver subroutines `ODESolver` and `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 [18].

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 extensively described in [18].

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 scalar, 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

      CALL ODESolver(ISOLVER, fex, 0, jex, 0, 0,
     $                     neq, 0, 0, t, tout, y, 0, 0,
     $                     ipar, dpar, rtol, atol,
     $                     iwork, liwork, rwork, lwork,
     $                     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, F, IPAR, DPAR, 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 ODEDER ***
      END
C
C *** SUBROUTINE JACFX starts ***
C
      SUBROUTINE JACFX(NX, NU, LDFX, T, X, U, FX, IPAR, DPAR, 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='fchemcin1';
  JACFX='jchemcin1';
  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, X, U, P, TINI, TOUT, 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;
```

4.9 Benchmarks

The test battery comprises 6 cases for the Ordinary Differential Equations (ODE) solver and 5 cases for the Differential Algebraic Equations (DAE) solver.

In the following subsections, a description of each one of the examples is given.

4.9.1 ODE Examples

Chemical Akzo Nobel Problem (CHEMAKZO)

This case has been obtained from the IVPTestSet[17] collection. It is a stiff system of 6 non-linear Ordinary Differential Equations.

The problem is of the form

$$\dot{y} = f(y), y(0) = y_0,$$

with

$$y \in \mathbb{R}^6, 0 \leq \mathrm{t} \leq 180.$$

The function f is defined by

$$f(y) = \begin{pmatrix} -2r_1 & +r_2 & -r_3 & -r_4 & & \\ -\frac{1}{2}r_1 & & & -r_4 & -\frac{1}{2}r_5 & +F_{in} \\ r_1 & -r_2 & +r_3 & & & \\ & -r_2 & +r_3 & -2r_4 & & \\ & r_2 & -r_3 & & +r_5 & \\ & & & & -r_5 & \end{pmatrix},$$

where r_i and F_{in} are auxiliary variables, given by

$$\begin{aligned} r_1 &= k_1 \cdot y_1^4 \cdot y_2^{\frac{1}{2}}, \\ r_2 &= k_2 \cdot y_3 \cdot y_4, \\ r_3 &= \frac{k_2}{K} \cdot y_1 \cdot y_5, \\ r_4 &= k_3 \cdot y_1 \cdot y_4^2, \\ r_5 &= k_4 \cdot y_6^2 \cdot y_2^{\frac{1}{2}}, \\ F_{in} &= klA \cdot (\frac{p(CO_2)}{H} - y_2). \end{aligned}$$

The values of the parameters $k_1, k_2, k_3, k_4, K, klA, p(CO_2)$ and H are

$$\begin{aligned} k_1 &= 18.7, & K &= 34.4, \\ k_2 &= 0.58, & klA &= 3.3, \\ k_3 &= 0.09, & p(CO_2) &= 0.9, \\ k_4 &= 0.42, & H &= 737. \end{aligned}$$

Finally, the initial vector y_0 is given by $y_0 = (0.437, 0.00123, 0, 0, 0, 0.367)^T$.

Table 1 presents the reference solution at the end of the integration interval. The reference solution was computed by RADAU5 on a Cray C90, using double precision, $work(1) = uround = 1.01 \cdot 10^{-19}, rtol = atol = h0 = 1.1 \cdot 10^{-18}$.

y_1	0.1161602274780192	y_4	$0.3396981299297459 \cdot 10^{-2}$
y_2	$0.1119418166040848 \cdot 10^{-2}$	y_5	0.1646185108335055
y_3	0.1621261719785814	y_6	0.1989533275954281

Table 1: Reference solution of the CHEMAKZO Problem at the end of the integration interval using RADAU5.

Chemical Kinetics 1 Problem (CHEMKI1)

This case has been obtained from the ODEPACK test battery[1]. It is a 3-equations stiff dense problem

$$\begin{aligned} \dot{x}_1 &= -0.04x_1 + 1.e4x_2x_3 \\ \dot{x}_2 &= 0.04x_1 - 1.e4x_2x_3 - 3.e7x_2^2 \\ \dot{x}_3 &= 3.e7x_2^2 \end{aligned}$$

on the interval from $t = 0.0$ to $t = 4.e10$, with initial conditions $x_1 = 1.0$, $x_2 = x_3 = 0$.

The conditions set-up for the problem are:

- Relative Tolerance : 1.e-4
- Absolute Tolerance : 1.e-6

Table 2 presents the reference solution at the end of the integration interval using LSODE.

x_1	-7.170586e-08
x_2	-2.868234e-13
x_3	1.000000e+00

Table 2: Reference solution of the CHEMKI1 Problem at the end of the integration interval using LSODE.

Medical Akzo Nobel Problem (MEDAKZO)

This case has been obtained from the IVPTestSet[17] collection. The problem consists of 2 partial differential equations. Semi-discretization of this system yields a stiff ODE.

The problem is of the form

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

with

$$y \in \mathrm{I\!R}^{2\mathrm{N}}, 0 \leq \mathrm{t} \leq 20.$$

Here, the integer N is a user-supplied parameter. The function f is given by

$$\begin{aligned} f_{2j-1} &= \alpha_j \frac{y_{2j+1} - y_{2j-3}}{2\triangle\zeta} + \beta_j \frac{y_{2j-3} - 2y_{2j-1} + y_{2j+1}}{(\triangle\zeta)^2} - ky_{2j-1}y_{2j}, \\ f_{2j} &= -ky_{2j}y_{2j-1}, \end{aligned}$$

where

$$\begin{aligned} \alpha_j &= \frac{2(j\triangle\zeta - 1)^3}{c^2}, \\ \beta_j &= \frac{(j\triangle\zeta - 1)^4}{c^2}. \end{aligned}$$

The integer j ranges from 1 to N, $\triangle\zeta = \frac{1}{N}$, $y_{-1}(t) = \phi(t)$, $y_{2N+1} = y_{2N-1}$ and $g \in \mathrm{I\!R}^{2\mathrm{N}}$ is given by

$$g = (0, v_0, 0, v_0, \ldots, 0, v_0)^T.$$

The function ϕ is given by

$$\phi(t) = \begin{cases} 2 \text{ for } t \in (0, 5], \\ 0 \text{ for } t \in (5, 20]. \end{cases}$$

which means that f undergoes a discontinuity in time at $t = 5$. Suitable values for the parameters k, v_0 and c are 100, 1 and 4, respectively.

The numerical experiments were done for the case $N = 200$. In Table 3 we give the value of some components of the reference solution at the end of the integration interval. These components correspond to the values of u and v in $x = 1, 2.4, 4.0$ and 6.0.

The reference solution was computed on the Cray C90, using PSIDE [19] with Cray double precision and $atol = rtol = 10^{-10}$.

y_{79}	$0.2339942217046434 \cdot 10^{-3}$	y_{199}	$0.11737412926802 \cdot 10^{-3}$
y_{80}	$-0.1127916494884468 \cdot 10^{-141}$	y_{200}	$0.61908071460151 \cdot 10^{-5}$
y_{149}	$0.3595616017506735 \cdot 10^{-3}$	y_{239}	$0.68600948191191 \cdot 10^{-11}$
y_{150}	$0.1649638439865233 \cdot 10^{-86}$	y_{240}	0.99999973258552

Table 3: Reference solution of the MEDAKZO Problem at the end of the integration interval using PSIDE [19].

High Irradiance Response Problem (HIRES)

This case has been obtained from the IVPTestSet[17] collection. This is a stiff system of 8 non-linear ordinary differential equations.

The problem is of the form

$$\dot{y} = f(y), y(0) = y_0,$$

with

$$y \in \mathbb{R}^8, 0 \leq \mathrm{t} \leq 321.8122.$$

The function f is defined by

$$f(y) = \begin{pmatrix} -1.71y_1 & +0.43y_2 & +8.32y_3 & +0.0007 & \\ 1.71y_1 & -8.75y_2 & & & \\ -10.03y_3 & +0.43y_4 & +0.035y_5 & & \\ 8.32y_2 & +1.71y_3 & -1.12y_4 & & \\ -1.745y_5 & +0.43y_6 & +0.43y_7 & & \\ -280y_6y_8 & +0.69y_4 & +1.71y_5 & -0.43y_6 & +0.69y_7 \\ 280y_6y_8 & -1.81y_7 & & & \\ -280y_6y_8 & +1.81y_7 & & & \end{pmatrix}.$$

The initial vector y_0 is given by $(1, 0, 0, 0, 0, 0, 0, 0.0057)^T$.

Table 4 presents the reference solution at the end of the integration interval. The reference solution was computed by RADAU5 on a Cray C90, using double precision, $work(1) = uround = 1.01 \cdot 10^{-19}, rtol = atol = h0 = 1.1 \cdot 10^{-18}$.

Air Pollution Problem (POLLU)

This case has been obtained from the IVPTestSet[17] collection. This is a stiff system of 20 non-linear ordinary differential equations. It is the chemical reaction part of the air pollution model developed at The Dutch National Institute of Public Health and Environmental Protection.

y_1	$0.7371312573325668 \cdot 10^{-3}$	y_5	$0.2386356198831331 \cdot 10^{-2}$
y_2	$0.1442485726316185 \cdot 10^{-3}$	y_6	$0.6238968252742796 \cdot 10^{-2}$
y_3	$0.5888729740967575 \cdot 10^{-4}$	y_7	$0.2849998395185769 \cdot 10^{-2}$
y_4	$0.1175651343283149 \cdot 10^{-2}$	y_8	$0.2850001604814231 \cdot 10^{-2}$

Table 4: Reference solution of the HIRES Problem at the end of the integration interval using RADAU5.

The problem is of the form

$$\dot{y} = f(y), y(0) = y_0,$$

with

$$y \in \mathbb{R}^{20}, 0 \leq \mathrm{t} \leq 60.$$

The function f is defined by

$$f = \begin{pmatrix} -\sum_{j\in\{1,10,14,23,24\}} r_j + \sum_{j\in\{2,3,9,11,12,22,25\}} r_j \\ -r_2 - r_3 - r_9 - r_{12} + r_1 + r_{21} \\ -r_{15} + r_1 + r_{17} + r_{19} + r_{22} \\ -r_2 - r_{16} - r_{17} - r_{23} + r_{15} \\ -r_3 + 2r_4 + r_6 + r_7 + r_{13} + r_{20} \\ -r_6 - r_8 - r_{14} - r_{20} + r_3 + 2r_{18} \\ -r_4 - r_5 - r_6 + r_{13} \\ r_4 + r_5 + r_6 + r_7 \\ -r_7 - r_8 \\ -r_{12} + r_7 + r_9 \\ -r_9 - r_{10} + r_8 + r_{11} \\ r_9 \\ -r_{11} + r_{10} \\ -r_{13} + r_{12} \\ r_{14} \\ -r_{18} - r_{19} + r_{16} \\ -r_{20} \\ r_{20} \\ -r_{21} - r_{22} - r_{24} + r_{23} + r_{25} \\ -r_{25} + r_{24} \end{pmatrix},$$

where the r_i are auxiliary variables, given in Table 5. The values of the parameters k_j are in Table 6. Finally, the initial vector y_0 is given by

$$y_0 = (0, 0.2, 0, 0.04, 0, 0, 0.1, 0.3, 0.01, 0, 0, 0, 0, 0, 0, 0, 0.007, 0, 0, 0)^T.$$

Table 7 presents the reference solution at the end of the integration interval. The reference solution was computed by RADAU5 on a Cray C90, using double precision, $work(1) = uround = 1.01 \cdot 10^{-19}, rtol = atol = h0 = 1.1 \cdot 10^{-18}$.

$r_1 = k_1 \cdot y_1$	$r_{10} = k_{10} \cdot y_{11} \cdot y_1$	$r_{19} = k_{19} \cdot y_{16}$
$r_2 = k_2 \cdot y_2 \cdot y_4$	$r_{11} = k_{11} \cdot y_{13}$	$r_{20} = k_{20} \cdot y_{17} \cdot y_6$
$r_3 = k_3 \cdot y_5 \cdot y_2$	$r_{12} = k_{12} \cdot y_{10} \cdot y_2$	$r_{21} = k_{21} \cdot y_{19}$
$r_4 = k_4 \cdot y_7$	$r_{13} = k_{13} \cdot y_{14}$	$r_{22} = k_{22} \cdot y_{19}$
$r_5 = k_5 \cdot y_7$	$r_{14} = k_{14} \cdot y_1 \cdot y_6$	$r_{23} = k_{23} \cdot y_1 \cdot y_4$
$r_6 = k_6 \cdot y_7 \cdot y_6$	$r_{15} = k_{15} \cdot y_3$	$r_{24} = k_{24} \cdot y_{19} \cdot y_1$
$r_7 = k_7 \cdot y_9$	$r_{16} = k_{16} \cdot y_4$	$r_{25} = k_{25} \cdot y_{20}$
$r_8 = k_8 \cdot y_9 \cdot y_6$	$r_{17} = k_{17} \cdot y_4$	
$r_9 = k_9 \cdot y_{11} \cdot y_2$	$r_{18} = k_{18} \cdot y_{16}$	

Table 5: Auxiliary variables of the POLLU Problem.

$k_1 = 0.350$	$k_{10} = 0.900 \cdot 10^4$	$k_{19} = 0.444 \cdot 10^{12}$
$k_2 = 0.266 \cdot 10^2$	$k_{11} = 0.220 \cdot 10^{-1}$	$k_{20} = 0.124 \cdot 10^4$
$k_3 = 0.123 \cdot 10^5$	$k_{12} = 0.120 \cdot 10^5$	$k_{21} = 0.210 \cdot 10$
$k_4 = 0.860 \cdot 10^{-3}$	$k_{13} = 0.188 \cdot 10$	$k_{22} = 0.578 \cdot 10$
$k_5 = 0.820 \cdot 10^{-3}$	$k_{14} = 0.163 \cdot 10^5$	$k_{23} = 0.474 \cdot 10^{-1}$
$k_6 = 0.150 \cdot 10^5$	$k_{15} = 0.480 \cdot 10^7$	$k_{24} = 0.178 \cdot 10^4$
$k_7 = 0.130 \cdot 10^{-3}$	$k_{16} = 0.350 \cdot 10^{-3}$	$k_{25} = 0.312 \cdot 10$
$k_8 = 0.240 \cdot 10^5$	$k_{17} = 0.175 \cdot 10^{-1}$	
$k_9 = 0.165 \cdot 10^5$	$k_{18} = 0.100 \cdot 10^9$	

Table 6: Parameter values of the POLLU Problem.

4.9.2 DAE Examples

Andrews' Squeezing Mechanism Problem (ANDREWS)

This case has been obtained from the IVPTestSet[17] collection. The problem is a non-stiff second order DAE of index 3, consisting of 21 differential and 6 algebraic equations. Only the RADAU5 package from the interface is able to deal with non-linear higher order problems. Therefore, this case will not work with the rest of the packages.

The problem is of the form

$$K\dot{y} = \phi(y), y(0) = y_0, \dot{y}(0) = \dot{y}_0,$$

where

$$y = \begin{pmatrix} q \\ \dot{q} \\ \ddot{q} \\ \lambda \end{pmatrix}, K = \begin{bmatrix} I & O & O & O \\ O & I & O & O \\ O & O & O & O \\ O & O & O & O \end{bmatrix}, \phi(y) = \begin{pmatrix} \dot{q} \\ \ddot{q} \\ M(q)\ddot{q} - f(q,\dot{q}) + G^T(q)\lambda \\ g(q) \end{pmatrix}.$$

Here,

$$0 \leq t \leq 0.03,$$

y_1	$0.5646255480022769 \cdot 10^{-1}$	y_{11}	$0.1135863833257075 \cdot 10^{-7}$
y_2	0.1342484130422339	y_{12}	$0.2230505975721359 \cdot 10^{-2}$
y_3	$0.4139734331099427 \cdot 10^{-8}$	y_{13}	$0.2087162882798630 \cdot 10^{-3}$
y_4	$0.5523140207484359 \cdot 10^{-2}$	y_{14}	$0.1396921016840158 \cdot 10^{-4}$
y_5	$0.2018977262302196 \cdot 10^{-6}$	y_{15}	$0.8964884856898295 \cdot 10^{-2}$
y_6	$0.1464541863493966 \cdot 10^{-6}$	y_{16}	$0.4352846369330103 \cdot 10^{-17}$
y_7	$0.7784249118997964 \cdot 10^{-1}$	y_{17}	$0.6899219696263405 \cdot 10^{-2}$
y_8	0.3245075353396018	y_{18}	$0.1007803037365946 \cdot 10^{-3}$
y_9	$0.7494013383880406 \cdot 10^{-2}$	y_{19}	$0.1772146513969984 \cdot 10^{-5}$
y_{10}	$0.1622293157301561 \cdot 10^{-7}$	y_{20}	$0.5682943292316392 \cdot 10^{-4}$

Table 7: Reference solution of the POLLU Problem at the end of the integration interval using RADAU5.

$$\begin{aligned}
q &\in \mathbb{R}^7, \\
\lambda &\in \mathbb{R}^6, \\
M &: \mathbb{R}^7 \to \mathbb{R}^{7\times 7}, \\
f &: \mathbb{R}^{14} \to \mathbb{R}^7, \\
g &: \mathbb{R}^7 \to \mathbb{R}^6, \\
G &= \frac{\partial g}{\partial q}.
\end{aligned}$$

The function $M(q) = (M_{ij}(q))$ is given by

$$\begin{aligned}
M_{11}(q) &= m_1 \cdot ra^2 + m_2(rr^2 - 2da \cdot rr \cdot \cos q_2 + da^2) + I_1 + I_2, \\
M_{21}(q) &= M_{12}(q) = m_2(da^2 - da \cdot rr \cdot \cos q_2) + I_2, \\
M_{22}(q) &= m_2 \cdot da^2 + I_2, \\
M_{33}(q) &= m_3(sa^2 + sb^2) + I_3, \\
M_{44}(q) &= m_4(e - ea)^2 + I_4, \\
M_{54}(q) &= M_{45}(q) = m_4((e - ea)^2 + zt(e - ea)\sin q_4) * I_4, \\
M_{55}(q) &= m_4(zt^2 + 2zt(e - ea)\sin q_4 + (e - ea)^2) + m_5(ta^2 + tb^2) + I_4 + I_5, \\
M_{66}(q) &= m_6(zf - fa)^2 + I_6, \\
M_{76}(q) &= M_{67}(q) = m_6((zf - fa)^2 - u(zf - fa)\sin q_6) + I_6, \\
M_{77}(q) &= m_6((zf - fa)^2 - 2u(zf - fa)\sin q_6 + u^2) + m_7(ua^2 + ub^2) + I_6 + I_7, \\
M_{ij}(q) &= 0 \text{ for all other cases.}
\end{aligned}$$

The function $f = (f_i(q, \dot{q})$ reads

$$\begin{aligned}
f_1(q, \dot{q}) &= mom - m_2 \cdot da \cdot rr \cdot \dot{q}_2(\dot{q}_2 + 2\dot{q}_1)\sin q_2, \\
f_2(q, \dot{q}) &= m_2 \cdot da \cdot rr \cdot \dot{q}_1^2 \cdot \sin q_2, \\
f_3(q, \dot{q}) &= F_x(sc \cdot \cos q_3 - sd \cdot \sin q_3) + F_y(sd \cdot \cos q_3 + sc \cdot \sin q_3),
\end{aligned}$$

$$
\begin{aligned}
f_4(q,\dot q) &= m_4 \cdot zt(e-ea)\dot q_5^2 \cdot \cos q_4,\\
f_5(q,\dot q) &= -m_4 \cdot zt(e-ea)\dot q_4(\dot q_4 + 2\dot q_5)\cos q_4,\\
f_6(q,\dot q) &= -m_6 \cdot u(zf-fa)\dot q_7^2 \cdot \cos q_6,\\
f_7(q,\dot q) &= m_6 \cdot u(zf-fa)\dot q_6(\dot q_6 + 2\dot q_7)\cos q_6.
\end{aligned}
$$

F_x and F_y are defined by

$$
\begin{aligned}
F_x &= F(xd-xc),\\
F_y &= F(yd-yc),\\
F &= -c_0(L-l_0)/L,\\
L &= \sqrt{(xd-xc)^2+(yd-yc)^2},\\
xd &= sd\cdot\cos q_3 + sc\cdot\sin q_3 + xb,\\
yd &= sd\cdot\sin q_3 - sc\cdot\cos q_3 + yb.
\end{aligned}
$$

The function $g=(g_i(q))$ is given by

$$
\begin{aligned}
g_1(q) &= rr\cdot\cos q_1 - d\cdot cos(q_1+q_2) - ss\cdot\sin q_3 - xb,\\
g_2(q) &= rr\cdot\sin q_1 - d\cdot sin(q_1+q_2) + ss\cdot\cos q_3 - yb,\\
g_3(q) &= rr\cdot\cos q_1 - d\cdot cos(q_1+q_2) - e\cdot sin(q_4+q_5) - zt\cdot\cos q_5 - xa,\\
g_4(q) &= rr\cdot\sin q_1 - d\cdot sin(q_1+q_2) + e\cdot cos(q_4+q_5) - zt\cdot\sin q_5 - ya,\\
g_5(q) &= rr\cdot\cos q_1 - d\cdot cos(q_1+q_2) - zf\cdot cos(q_6+q_7) - u\cdot\sin q_7 - xa,\\
g_6(q) &= rr\cdot\sin q_1 - d\cdot sin(q_1+q_2) - zf\cdot sin(q_6+q_7) + u\cdot\cos q_7 - ya.
\end{aligned}
$$

The constants arising in these formulas are given by

$m_1 = 0.04325$	$I_1 = 2.194\cdot10^{-6}$	$ss = 0.035$
$m_2 = 0.00365$	$I_2 = 4.410\cdot10^{-7}$	$sa = 0.01874$
$m_3 = 0.02373$	$I_3 = 5.255\cdot10^{-6}$	$sb = 0.01043$
$m_4 = 0.00706$	$I_4 = 5.667\cdot10^{-7}$	$sc = 0.018$
$m_5 = 0.07050$	$I_5 = 1.169\cdot10^{-5}$	$sd = 0.02$
$m_6 = 0.00706$	$I_6 = 5.667\cdot10^{-7}$	$ta = 0.02308$
$m_7 = 0.05498$	$I_7 = 1.912\cdot10^{-5}$	$tb = 0.00916$
$xa = -0.06934$	$d = 0.028$	$u = 0.04$
$ya = -0.00227$	$da = 0.0115$	$ua = 0.01228$
$xb = -0.03635$	$e = 0.02$	$ub = 0.00449$
$yb = 0.03273$	$ea = 0.01421$	$zf = 0.02$
$xc = 0.014$	$rr = 0.007$	$zt = 0.04$
$yc = 0.072$	$ra = 0.00092$	$fa = 0.01421$
$c_0 = 4530$	$l_0 = 0.07785$	$mom = 0.033$

Consistent initial values are

$$y_0 = (q_0, \dot q_0, \ddot q_0, \lambda_0)^T \text{ and } y_0' = (\dot q_0, \ddot q_0, \dddot q_0, \dot\lambda_0)^T,$$

where

$$
\begin{aligned}
q_0 &= \begin{pmatrix} -0.0617138900142764496358948458001 \\ 0 \\ 0.455279819163070380255912382449 \\ 0.222668390165885884674473185609 \\ 0.487364979543842550225598953530 \\ -0.222668390165885884674473185609 \\ 1.23054744454982119249735015568 \end{pmatrix}, \\
\dot{q}_0 &= \dddot{q}_0 = (0,0,0,0,0,0,0)^T, \\
\ddot{q}_0 &= \begin{pmatrix} 14222.4439199541138705911625887 \\ -10666.8329399655854029433719415 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \\
\lambda_0 &= \begin{pmatrix} 98.5668703962410896057654982170 \\ -6.12268834425566265503114393122 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \\
\dot{\lambda}_0 &= (0,0,0,0,0,0)^T.
\end{aligned}
$$

The index of the $q, \dot{q}, \ddot{q}$ and λ components in y is $1, 2, 3$ and 3, respectively.

Table 8 presents the reference solution at the end of the integration interval. In computing the values, only the first seven components were considered, since they refer to the physically important quantities. The reference solution was computed on the Cray C90, using PSIDE [19] with Cray double precision and $atol = rtol = 10^{-14}$.

y_1	$0.15810771 \cdot 10^2$	y_4	-0.53473012	y_6	0.53473012
y_2	$-0.15756371 \cdot 10^2$	y_5	0.52440997	y_7	$0.10480807 \cdot 10$
y_3	$0.40822240 \cdot 10^{-1}$				

Table 8: Reference solution (first 7 components) of the ANDREWS Problem at the end of the integration interval using PSIDE.

Car Axis Problem (CARAXIS)

This case has been obtained from the IVPTestSet[17] collection. The problem is a stiff DAE of index 3, consisting of 8 differential and 2 algebraic equations. Only the RADAU5 package from the interface is able to deal with non-linear higher order problems. Therefore, this case will not work with the rest of the packages.

The problem is of the form

$$\begin{aligned} \dot{p} &= q, \\ K\dot{q} &= f(t,p,\lambda), \quad p,q \in \mathbb{R}^4, \ \lambda \in \mathbb{R}^2, \ 0 \le \mathrm{t} \le 3, \\ 0 &= \phi(t,p), \end{aligned}$$

with initial conditions $p(0) = p_0, q(0) = q_0, \dot{p}(0) = \dot{p}_0, \dot{q}(0) = \dot{q}_0, \lambda(0) = \lambda_0$ and $\dot{\lambda}(0) = \dot{\lambda}_0$.

The matrix K reads $\varepsilon^2 \frac{M}{2} I_4$, where I_4 is the 4×4 identity matrix. The function $f : \mathbb{R}^7 \to \mathbb{R}^4$ is given by

$$f(t,p,\lambda) = \left(\begin{array}{l} (l_0 - l_l)\frac{x_l}{l_l} + \lambda_1 x_b + 2\lambda_2(x_l - x_r) \\ (l_0 - l_l)\frac{y_l}{l_l} + \lambda_1 y_b + 2\lambda_2(y_l - y_r) - \varepsilon^2\frac{M}{2} \\ (l_0 - l_r)\frac{x_r - x_b}{l_r} - 2\lambda_2(x_l - x_r) \\ (l_0 - l_r)\frac{y_r - y_b}{l_r} - 2\lambda_2(y_l - y_r) - \varepsilon^2\frac{M}{2} \end{array} \right).$$

Here, $(x_l, y_l, x_r, y_r)^T := p$, and l_l and l_r are given by

$$\sqrt{x_l^2 + y_l^2} \text{ and } \sqrt{(x_r - x_b)^2 + (y_r - y_b)^2}.$$

Furthermore, the functions $x_b(t)$ and $y_b(t)$ are defined by

$$\begin{aligned} x_b(t) &= \sqrt{l^2 - y_b^2(t)}, \\ y_b(t) &= r sin(\omega t). \end{aligned}$$

The function $\phi : \mathbb{R}^5 \to \mathbb{R}^2$ reads

$$\phi(t,p) = \left(\begin{array}{l} x_l x_b + y_l y_b \\ (x_l - x_r)^2 + (y_l - y_r)^2 - l^2 \end{array} \right).$$

The constants are listed below.

$l = 1$	$\varepsilon = 10^{-2}$	$h = 1/5$	$\omega = 10$
$l_0 = 1/2$	$M = 10$	$\tau = \pi/5$	

Consistent initial values are

$$p_0 = \begin{pmatrix} 0 \\ 1/2 \\ 1 \\ 1/2 \end{pmatrix}, q_0 = \begin{pmatrix} -1/2 \\ 0 \\ -1/2 \\ 0 \end{pmatrix}, \dot{q}_0 = \frac{2}{M\varepsilon^2} f(0, p_0, \lambda_0), \lambda_0 = \dot{\lambda}_0 = (0,0)^T.$$

The index of the variables p, q and λ is $1, 2$ and 3, respectively.

Table 9 presents the reference solution at the end of the integration interval. The reference solution was computed on the Cray C90, using PSIDE with Cray double precision and $atol = rtol = 10^{-16}$.

Transistor Amplifier Problem (TRANSAMP)

This case has been obtained from the IVPTestSet[17] collection. The problem is a stiff DAE of index 1 consisting of 8 equations.

y_1	$0.4934557842755629 \cdot 10^{-1}$	y_6	$0.7446866596327776 \cdot 10^{-2}$
y_2	0.4969894602303324	y_7	$0.1755681574942899 \cdot 10^{-1}$
y_3	$0.1041742524885400 \cdot 10$	y_8	0.7703410437794031
y_4	0.3739110272652214	y_9	$-0.4736886750784630 \cdot 10^{-2}$
y_5	$-0.7705836840321485 \cdot 10^{-1}$	y_{10}	$-0.1104680411345730 \cdot 10^{-2}$

Table 9: Reference solution of the CARAXIS Problem at the end of the integration interval using PSIDE.

The problem is of the form

$$M\dot{y} = f(y), \quad y(0) = y_0, \quad \dot{y}(0) = \dot{y}_0,$$

with

$$y \in \mathbb{R}^8, \quad 0 \le \mathrm{t} \le 0.2.$$

The matrix M is of rank 5 and given by

$$M = \begin{pmatrix} -C_1 & C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ C_1 & -C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -C_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -C_3 & C_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & C_3 & -C_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -C_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -C_5 & C_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & C_5 & -C_5 \end{pmatrix},$$

and the function f by

$$f(y) = \begin{pmatrix} -\frac{U_e(t)}{R_0} + \frac{y_1}{R_0} \\ -\frac{U_b}{R_2} + y_2(\frac{1}{R_1} + \frac{1}{R_2}) - (\alpha - 1)g(y_2 - y_3) \\ -g(y_2 - y_3) + \frac{y_3}{R_3} \\ -\frac{U_b}{R_4} + \frac{y_4}{R_4} + \alpha g(y_2 - y_3) \\ -\frac{U_b}{R_6} + y_5(\frac{1}{R_5} + \frac{1}{R_6}) - (\alpha - 1)g(y_5 - y_6) \\ -g(y_5 - y_6) + \frac{y_6}{R_7} \\ -\frac{U_b}{R_8} + \frac{y_7}{R_8} + \alpha g(y_5 - y_6) \\ \frac{y_8}{R_9} \end{pmatrix},$$

where g and U_e are auxiliary functions given by

$$g(x) = \beta(e^{\frac{x}{U_F}} - 1) \text{ and } U_e(t) = 0.1 sin(200\pi t).$$

The values of the technical parameters are

$U_b = 6$,	$R_0 = 1000$,	
$U_F = 0.026$,	$R_k = 9000$	for $k = 1, \ldots, 9$,
$\alpha = 0.99$,	$C_k = k \cdot 10^{-6}$	for $k = 1, \ldots, 5$.
$\beta = 10^{-6}$,		

Consistent initial values at $t = 0$ are

$$y_0 = \begin{pmatrix} 0 \\ U_b/(\frac{R_2}{R_1}+1) \\ U_b/(\frac{R_2}{R_1}+1) \\ U_b \\ U_b/(\frac{R_6}{R_5}+1) \\ U_b/(\frac{R_6}{R_5}+1) \\ U_b \\ 0 \end{pmatrix}, \quad \dot{y}_0 = \begin{pmatrix} 51.338775 \\ 51.338775 \\ -U_b/((\frac{R_2}{R_1}+1)(C_2 \cdot R_3)) \\ -24.9757667 \\ -24.9757667 \\ -U_b/((\frac{R_6}{R_5}+1)(C_4 \cdot R_7)) \\ -10.00564453 \\ -10.00564453 \end{pmatrix}.$$

The first, fourth and seventh component of y_0 were determined numerically. All components of y are of index 1.

Table 10 presents the reference solution at the end of the integration interval. The reference solution was computed on the Cray C90, using PSIDE [19] with Cray double precision and $atol = rtol = 10^{-14}$.

y_1	$-0.5562145012262709 \cdot 10^{-2}$	y_5	$0.2704617865010554 \cdot 10$
y_2	$0.3006522471903042 \cdot 10$	y_6	$0.2761837778393145 \cdot 10$
y_3	$0.2849958788608128 \cdot 10$	y_7	$0.4770927631616772 \cdot 10$
y_4	$0.2926422536206241 \cdot 10$	y_8	$0.1236995868091548 \cdot 10$

Table 10: Reference solution of the TRANSAMP Problem at the end of the integration interval using PSIDE.

Chemical Kinetics 2 Problem (CHEMKI2)

This problem has been obtained from the ODEPACK test battery[1]. This is a simple example problem, with the coding needed for its solution by lsoda. The problem is from chemical kinetics, and consists of the following three rate equations

$$\begin{aligned} \dot{y}_1 &= -0.04y_1 + 1e4y_2y_3 \\ \dot{y}_2 &= 0.04y_1 - 1e4y_2y_3 - 3e7y_2^2 \\ 0 &= y_1 + y_2 + y_3 - 1 \end{aligned}$$

on the interval from $t = 0.0$ to $t = 4.e10$, with initial conditions $x1 = 1.0$, $x2 = x3 = 0$. The problem is stiff.

The conditions set-up for the problem are:

- Relative Tolerance : 1.e-4
- Absolute Tolerance : 1.e-6

Table 11 presents the reference solution at the end of the integration interval.

Gelda Problem (GELDA)

y_1	1.404280e-08
y_2	5.617126e-14
y_3	1.000000e+00

Table 11: Reference solution of the CHEMKI2 Problem at the end of the integration interval.

This case has been obtained from the GELDA[6] benchmark test

$$\begin{bmatrix} 0 & 0 \\ 1 & -T \end{bmatrix} \dot{x} = -\begin{bmatrix} 1 & T \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} e^{-T} \\ 0 \end{bmatrix}$$

with t defined in $[0, 1]$. The solution is

$$x(t) = \begin{bmatrix} (1+t)e^{-t} \\ e^{-t} \end{bmatrix}.$$

4.9.3 Results

In the following subsections, the experimental results obtained for each one of the test performed are given. A comparison between the expected and obtained results is shown. Table 12 summarizes the test performed. Combinations out of the table are not available due firstly to format incompatibilities (i.e. combining a sparse problem with a non sparse method).

ODE Examples	LSODE	LSODA	LSODES	RADAU5	DASSL	DASPK
CHEMAKZO	Yes	Yes	Yes	Yes	Yes	Yes
CHEMKI1	Yes	Yes	Yes	Yes	Yes	No
MEDAKZO	Yes	Yes	Yes	Yes	Yes	Yes
HIRES	Yes	Yes	Yes	Yes	Yes	Yes
POLLU	Yes	Yes	Yes	Yes	Yes	Yes
DAE Examples	**LSODI**	**LSOIBT**	**RADAU5**	**DASSL**	**DASPK**	**GELDA**
ANDREWS	No	No	Yes	No	No	No
CARAXIS	No	No	Yes	No	No	No
TRANSAMP	No	No	Yes	Yes	No	No
CHEMKI2	Yes	No	Yes	Yes	Yes	No
GELDA	No	No	No	No	No	Yes

Table 12: Test cases performed

Several problems (DAE Examples and CHEMKI1) cannot be executed with all the packages. Obviously, the interface does not provide a solution for the packages that cannot deal with problems of higher order, so the behaviour of the packages without the interface is the same as through the interface.

The following tables show the relative errors of the different cases.

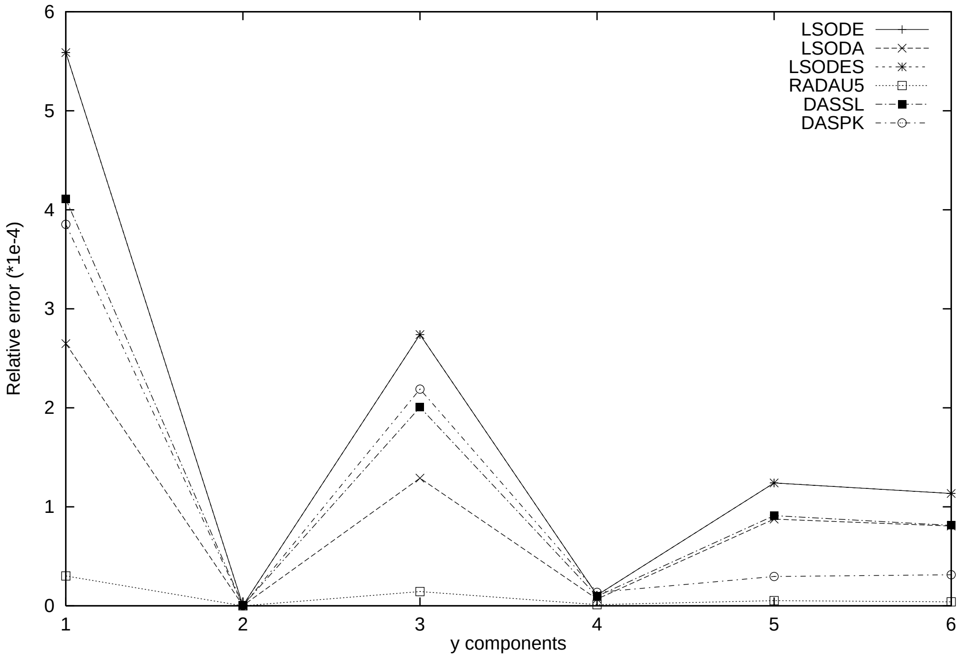

Figure 2: Relative Errors in the CHEMAKZO Problem

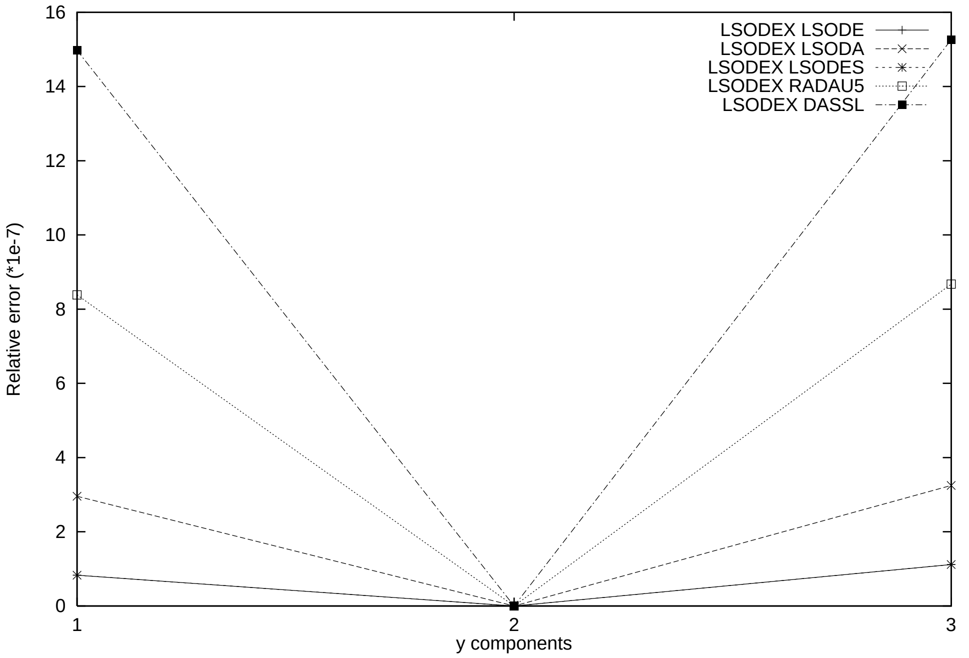

Figure 3: Relative Errors in the CHEMKI1 Problem

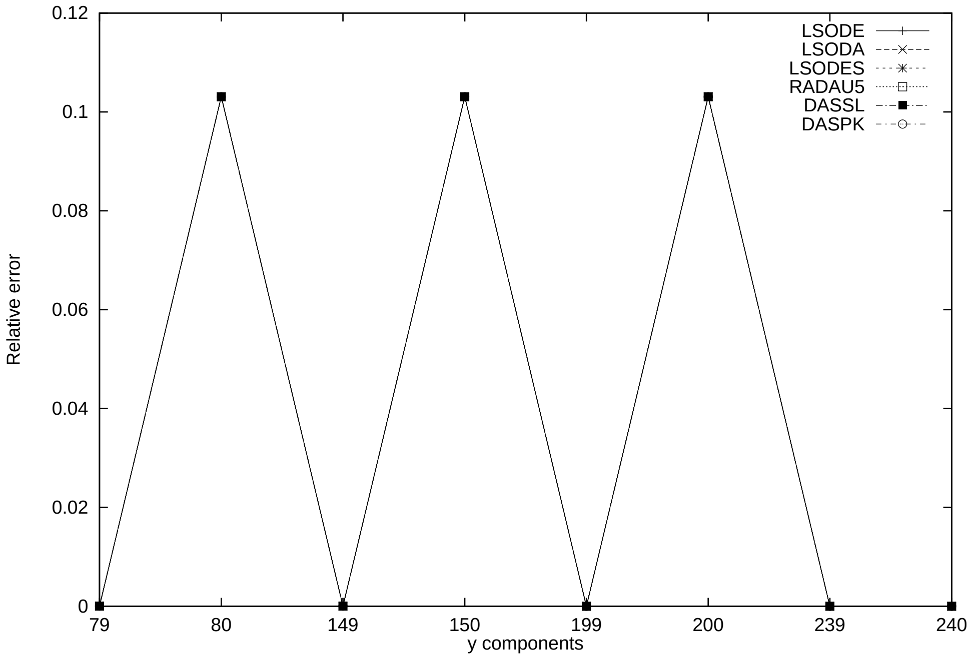

Figure 4: Relative Errors in the MEDAKZO Problem

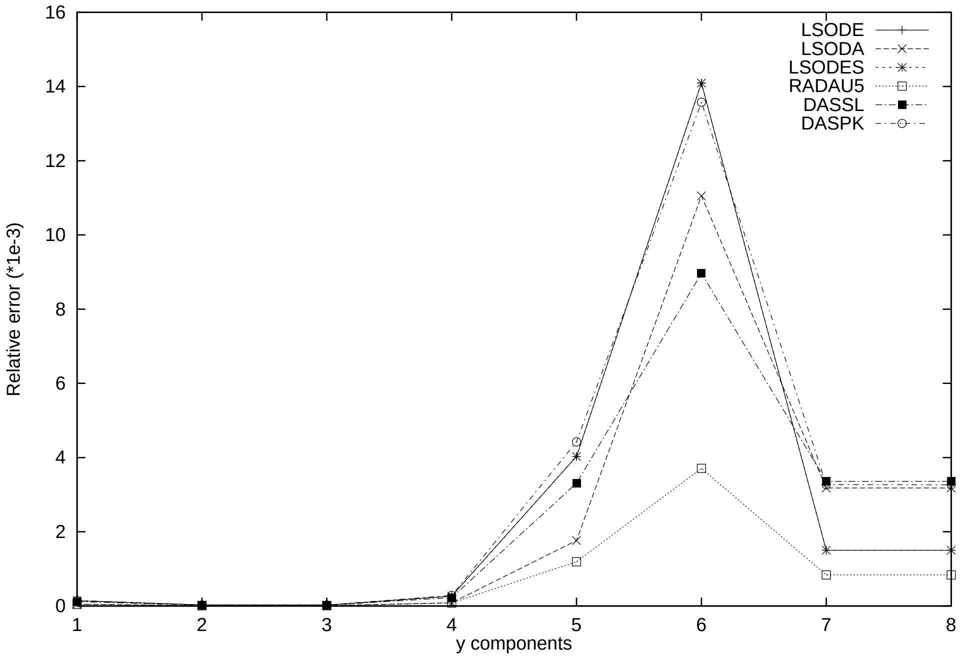

Figure 5: Relative Errors in the HIRES Problem

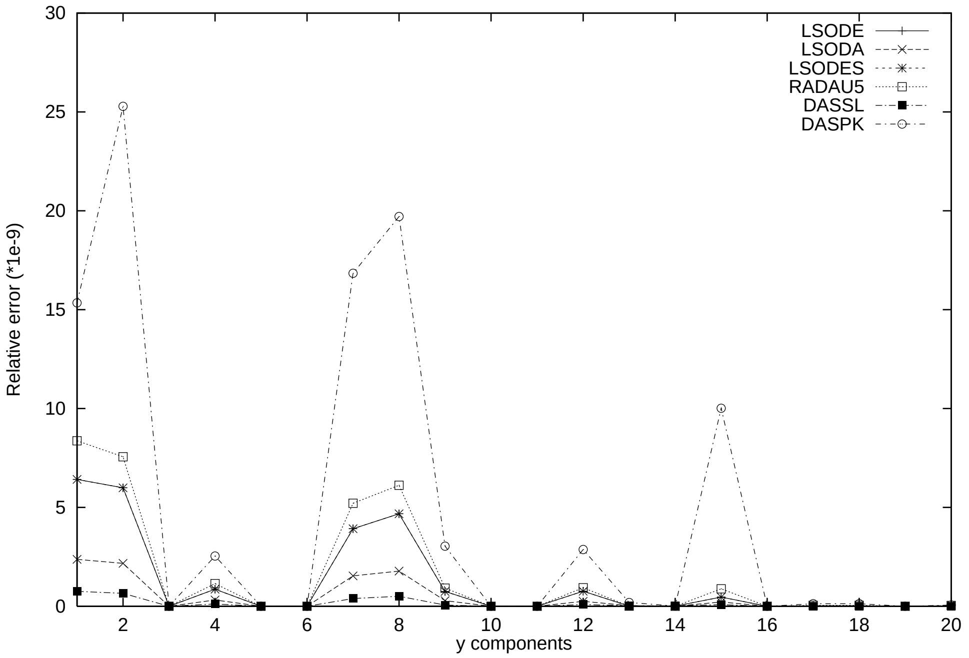

Figure 6: Relative Errors in the POLLU Problem

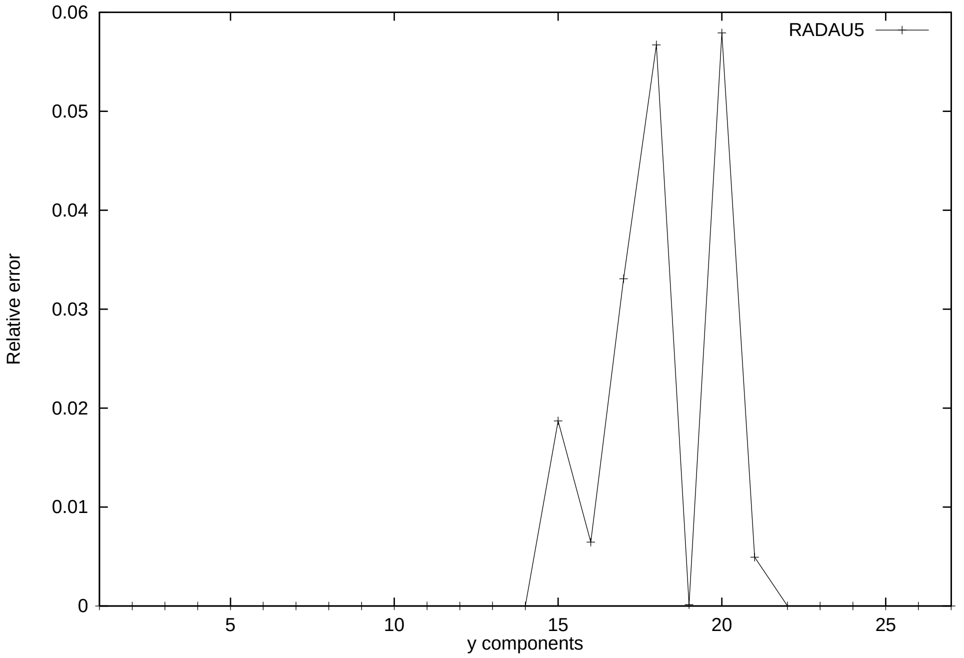

Figure 7: Relative Errors in the ANDREWS Problem

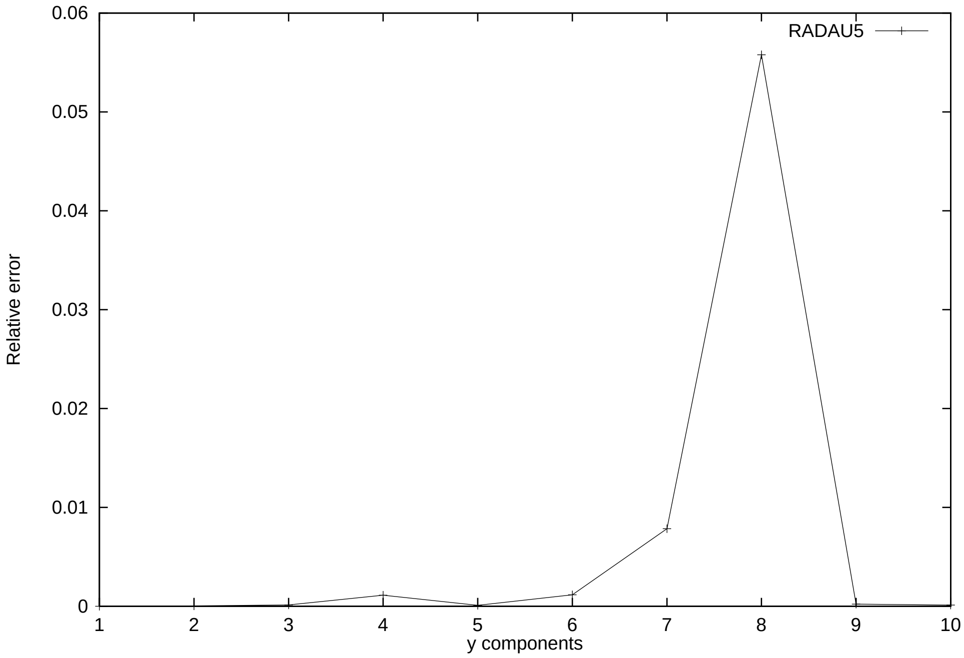

Figure 8: Relative Error in the CARAXIS Problem

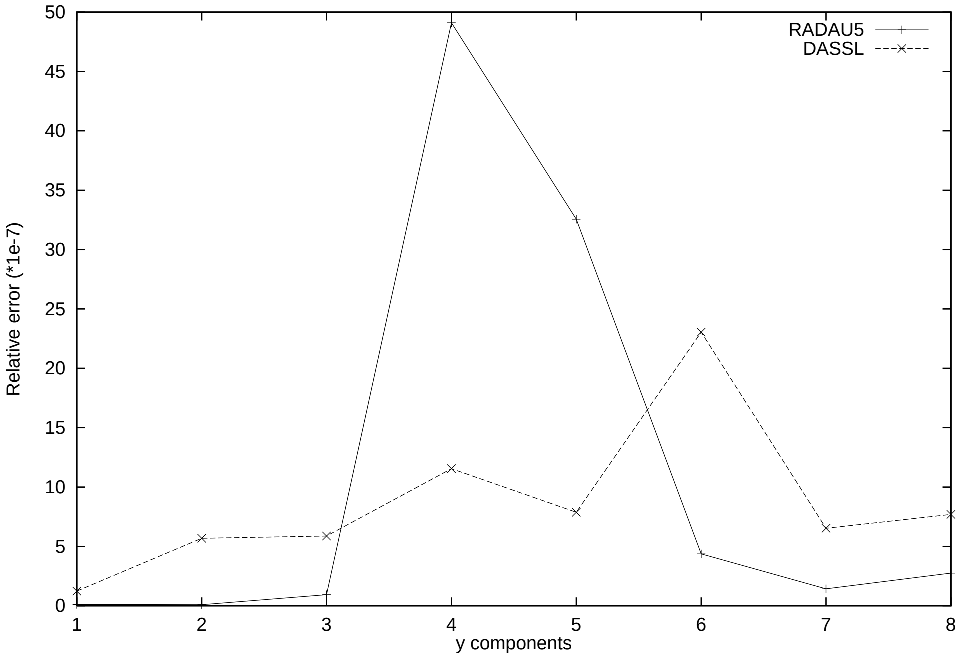

Figure 9: Relative Error in the TRANSAMP Problem

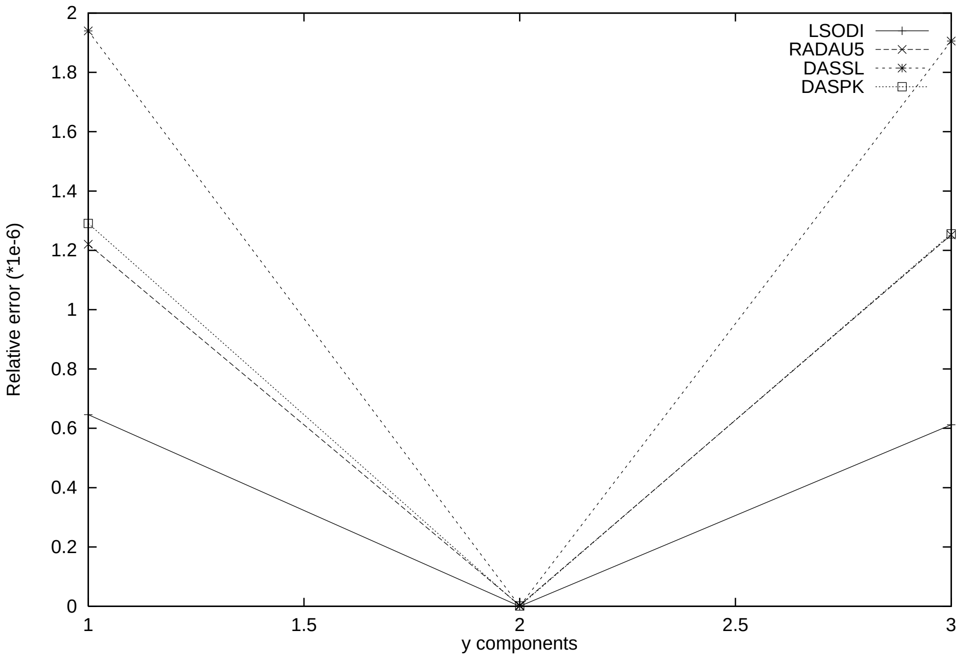

Figure 10: Relative Error in the CHEMKI2 Problem

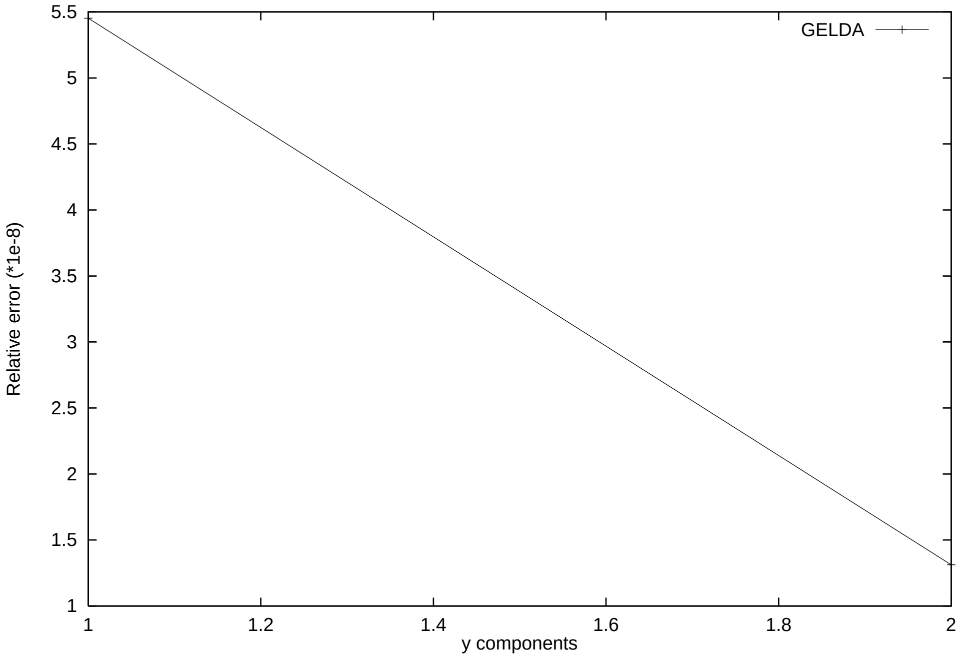

Figure 11: Relative Error in the GELDA Problem

5 Testing on industrial benchmark problems and comparisons

5.1 Robot Models

The validation of task V.A. has been focused on the simulation of Robot Manipulators. Two industrial cases coming from two real robots are used to demonstrate the validity and suitability of the package [21].

From both robots, the SLICOT toolbox for the simulation of nonlinear control systems will solve the dynamic equation of the robot for different time steps.

The motion of the robots is controlled by an optimal control [15] [16] [20]. The Lagrange-Euler formulation will be used since this formulation provides explicit state equations for the robot dynamics that can be used to analyze and design advanced control schemes.

5.2 Dynamic Equations of Rigid Manipulators

The dynamic equation for a robot manipulator with n links can be written using the Lagrange-Euler formulation as the matrix equation

$$D(q)\ddot{q} + h(q, \dot{q}) + g(q) = \tau, \tag{2}$$

where,

- τ is a $n \times 1$ vector which represents the nominal driving torques,
- q is a $n \times 1$ vector containing the generalized coordinates, and $\dot{q}$, $\ddot{q}$ stand for the first and second derivatives of vector q,
- $D(q)$ is the $n \times n$ inertia matrix of the dynamic system,
- $h(q, \dot{q})$ is the $n \times 1$ vector of centrifugal and Coriolis forces,
- $g(q)$ is the $n \times 1$ vector of gravitational forces.

The dynamic equation (2) can be expressed explicitly as

$$\begin{bmatrix} \tau_1 \\ \vdots \\ \tau_n \end{bmatrix} = \begin{bmatrix} d_{11} & \dots & d_{1n} \\ \vdots & \ddots & \vdots \\ d_{n1} & \dots & d_{nn} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \vdots \\ \ddot{q}_n \end{bmatrix} + \begin{bmatrix} h_1(q, \dot{q}) \\ \vdots \\ h_n(q, \dot{q}) \end{bmatrix} + \begin{bmatrix} c_1(q) \\ \vdots \\ c_n(q) \end{bmatrix}. \tag{3}$$

The equation requires the computation of different coefficients, such as d_{ij}, $h_i(q, \dot{q})$ and $c_i(q)$. $D(q)$ is a symmetric and positive definite matrix, whose elements are computed as

$$d_{ij} = \sum_{k=max\{i,j\}}^{n} tr(U_{jk} J_k U_{ik}^T), \tag{4}$$

where $J_k, k = 1 : n$, is the tensor of inertia of link k, and $tr()$ denotes the trace of a matrix. Using the rotational radius, the tensor of inertia of link i are defined as

$$J_i = m_i \begin{bmatrix} \frac{-k_{ixx}^2 + k_{iyy}^2 + k_{izz}^2}{2} & k_{ixy}^2 & k_{ixz}^2 & \bar{x_i} \\ k_{ixy}^2 & \frac{k_{ixx}^2 - k_{iyy}^2 + k_{izz}^2}{2} & k_{iyz}^2 & \bar{y_i} \\ k_{ixz}^2 & k_{iyz}^2 & \frac{k_{ixx}^2 + k_{iyy}^2 - k_{izz}^2}{2} & \bar{z_i} \\ \bar{x_i} & \bar{y_i} & \bar{z_i} & 1 \end{bmatrix}, \tag{5}$$

where k_{ijl} is the rotation radius of element i with respect the jl axis, and $(\bar{x_i}, \bar{y_i}, \bar{z_i})$ is the mass centrum of element i.

The U_{ij} terms are computed by

$$U_{ij} = \frac{\partial^0 A_j}{\partial q_i} = \begin{cases} {}^0A_{i-1} Q_i \; {}^{i-1}A_j, i \leq j \\ 0, i > j \end{cases} \tag{6}$$

where Q_i is a constant matrix that enables to compute the partial derivative with respect to link i. Depending on the link type, Q_i is given by

$$Q_i = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{7}$$

for revolution links, and

$$Q_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8}$$

for prismatic links.

A block upper triangular matrix structure U is defined, where each block is one of the matrices U_{ij}

$$U = \begin{bmatrix} U_{11} & U_{12} & \cdots & U_{1n} \\ & U_{22} & \cdots & U_{2n} \\ & & \ddots & \vdots \\ & & & U_{nn} \end{bmatrix}. \tag{9}$$

To obtain the matrix U it is necessary to compute transformation matrices ${}^jA_i \in R^{4x4}$.

The ${}^jA_i$ terms are matrices that transform the point coordinates with respect to link i to coordinates with respect to link j. Using Denavit-Hartenberg convention, ${}^jA_i$ satisfies the relation

$${}^jA_i = ^j A_{i-1}\; {}^{i-1}A_j, \tag{10}$$

where ${}^{i-1}A_i$ evaluates to

$${}^{i-1}A_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

for revolution links and

$${}^{i-1}A_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & 0 \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{12}$$

for prismatic links. The parameters α_i, a_i represent the kinematic structure of link i and θ_i, d_i are the mobility parameters of link i. Then, the following block upper triangular matrix can be defined

$$A = \begin{bmatrix} {}^0A_1 & {}^0A_2 & \cdots & {}^0A_n \\ & {}^1A_2 & \cdots & {}^1A_n \\ & & \ddots & \vdots \\ & & & {}^{n-1}A_n \end{bmatrix}. \tag{13}$$

The centrifugal and Coriolis forces vector $h_i(q, \dot{q})$ can be computed as $h_i(q, \dot{q}) = \dot{q}^T H_i \dot{q}, i = 1 : n$, where $H_i = [h_{ijk}]$ is an $n \times n$ matrix given by

$$h_{ijk} = \sum_{l=max\{i,j,k\}}^{n} tr(U_{kjl} J_l U_{il}^T). \tag{14}$$

These matrices define the matrix structure

$$H = [H_1, H_2, \ldots, H_n]^T. \tag{15}$$

To obtain H_i, it is necessary to compute matrices $U_{kjl} \in R^{4x4}$, which represent the second derivative terms. U_{kjl} is given by

$$U_{kjl} \equiv \frac{\partial U_{jl}}{\partial q_k} = \begin{cases} {}^0A_{j-1} Q_j \, {}^{j-1}A_{k-1} Q_k \, {}^{k-1}A_l, & j \leq k \leq l \\ {}^0A_{k-1} Q_k \, {}^{k-1}A_{j-1} Q_j \, {}^{j-1}A_l, & k \leq j \leq l \\ 0, & j > l \text{ or } k > l. \end{cases} \tag{16}$$

Note that $U_{kjl} = U_{jkl}$. As a consequence matrices H_i are symmetric.

From matrices U_{kjl}, the following matrix structure is defined

$$DU = [DU_1, DU_2, \cdots, DU_n]^T. \tag{17}$$

Finally, the gravitatory forces vector are computed as

$$c_i = \sum_{j=i}^{n} (-m_j g_j^T U_{ij} \, {}^j r_j), \tag{18}$$

where ${}^j r_j$ is the position of the mass centrum of link j with respect to its coordinate system and m_j is the mass of link j.

Rewriting the centrifugal and Coriolis term in the form

$$h(q, \dot{q}) = H(q, \dot{q}) \dot{q} \tag{19}$$

where

$$H(q, \dot{q}) = [H_1 \dot{q}, H_2 \dot{q}, \ldots, H_n \dot{q}]^T \tag{20}$$

equation (2) can be written as

$$D(q)\ddot{q} = H(q, \dot{q})\dot{q} - D^{-1}(q) g(q). \tag{21}$$

Introducing the state variable

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \tag{22}$$

equation (21) can be expressed as

$$\begin{bmatrix} I & 0 \\ 0 & D(q) \end{bmatrix} \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & -H(q,\dot{q}) \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} \tau + \begin{bmatrix} 0 \\ -g(q) \end{bmatrix} \tag{23}$$

which corresponds to a first order DAE or

$$\begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & -D^{-1}q^T H(q,\dot{q}) \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ D^{-1}(q) \end{bmatrix} \tau + \begin{bmatrix} 0 \\ -D^{-1}(q)g(q) \end{bmatrix} \tag{24}$$

which corresponds to a first order ODE.

The first model considered in this paper describes the kinematic and dynamic characteristics of a 6-links Unimation Puma 560 manipulator. All quantities are in standard SI units.

The following figure shows the details of coordinate frames used for the model in its zero angle pose.

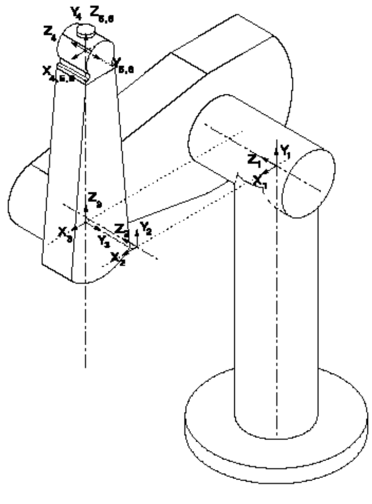

Figure 12: Puma 560 in its zero angle pose

The values for the parameters of the robot are the following:

$$
J = \begin{bmatrix} J_1 \\ J_2 \\ J_3 \\ J_4 \\ J_5 \\ J_6 \end{bmatrix} = \left[\begin{array}{rrrr}
0.1750 & 0 & 0 & 0 \\
0 & -0.1750 & 0 & 0 \\
0 & 0 & 0.1750 & 0 \\
0 & 0 & 0 & 0 \\
\hline
0.4665 & 0 & 0 & -6.3301 \\
0 & 0.0725 & 0 & 0.1044 \\
0 & 0 & 0.0575 & 3.9585 \\
-6.3301 & 0.1044 & 3.9585 & 17.4000 \\
\hline
0.0162 & 0 & 0 & -0.0974 \\
0 & -0.0037 & 0 & -0.0677 \\
0 & 0 & 0.0697 & 0.3360 \\
-0.0974 & -0.0677 & 0.3360 & 4.8000 \\
\hline
0.0006 & 0 & 0 & 0 \\
0 & 0.0012 & 0 & 0.0156 \\
0 & 0 & 0.0006 & 0 \\
0 & 0.0156 & 0 & 0.8200 \\
\hline
0.0002 & 0 & 0 & 0 \\
0 & 0.0001 & 0 & 0 \\
0 & 0 & 0.0002 & 0 \\
0 & 0 & 0 & 0.3400 \\
\hline
0.0000 & 0 & 0 & 0 \\
0 & 0.0000 & 0 & 0 \\
0 & 0 & 0.0001 & 0.0029 \\
0 & 0 & 0.0029 & 0.0900
\end{array}\right]
$$

$$
c = \begin{bmatrix} -62.6111 \\ 107.8150 \\ -53.7063 \\ 76.0364 \\ 71.9230 \\ 76.6860 \end{bmatrix}
$$

$$
E = \begin{bmatrix}
\theta_1 & a_1 & \alpha_1 & d_1 \\
\theta_2 & a_2 & \alpha_2 & d_2 \\
\theta_3 & a_3 & \alpha_3 & d_3 \\
\theta_4 & a_4 & \alpha_4 & d_4 \\
\theta_5 & a_5 & \alpha_5 & d_5 \\
\theta_6 & a_6 & \alpha_6 & d_6
\end{bmatrix} = \left[\begin{array}{rrrr}
0 & 0 & 1.5708 & 0 \\
0 & 0.4318 & 0 & 0 \\
0 & 0.0203 & -1.5708 & 0.1500 \\
0 & 0 & 1.5708 & 0.4318 \\
0 & 0 & -1.5708 & 0 \\
0 & 0 & 0 & 0
\end{array}\right]
$$

The second model is a 6-links Puma 500 robot with parameters values

$$
J = [J_1, J_2, J_3, J_4, J_5, J_6]^T
$$

$$J_i = \begin{bmatrix} 1.3333 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1.0000 & 0 & 0 & 1 \end{bmatrix}, i = 1:6.$$

$$c = \begin{bmatrix} 0 \\ 0 \\ 0.00033801568002016 \\ 0.00019978562557102 \\ 0.0000000000006843 \\ -0.00003599741001280 \end{bmatrix}$$

$$E = \begin{bmatrix} \theta_1 & a_1 & \alpha_1 & d_1 \\ \theta_2 & a_2 & \alpha_2 & d_2 \\ \theta_3 & a_3 & \alpha_3 & d_3 \\ \theta_4 & a_4 & \alpha_4 & d_4 \\ \theta_5 & a_5 & \alpha_5 & d_5 \\ \theta_6 & a_6 & \alpha_6 & d_6 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1.57079632679490 & 0 \\ 0 & 0.71 & 0 & 0 \\ 0 & 0.85 & -1.57079632679490 & 0.125 \\ 0 & 0 & -1.57079632679490 & 0 \\ 0 & 0 & -1.57079632679490 & 0 \end{bmatrix}$$

5.3 Optimal Control of Robot Manipulators

Given a desired manipulator trajectory $q_d(t) \in R^n$, the tracking error is defined by

$$e(t) = q_d(t) - q(t) \tag{25}$$

and the instantaneous performance measure is defined as

$$r(t) = \dot{e}(t) + \Gamma e(t) \tag{26}$$

where Γ is a constant gain matrix which needs to be dermined from the user-specified performance index [20].

The robot dynamics (2) may be rewritten in terms of the instantaneous performance measure as

$$D(q)\dot{r}(t) = -h(q, \dot{q})r(t) - \tau(t) + \phi(x) \tag{27}$$

where the robot nonlinear function ϕ is

$$\phi(x) = D(q)(\ddot{q}_d + \Gamma\dot{e}) + h(q, \dot{q})(\dot{q}_d + \Gamma e) + c(q). \tag{28}$$

This function $\phi(x)$ captures all the unknown dynamics of the robot manipulator. Let the control input torque be defined as

$$\tau(t) = \phi(x) - u(t) \tag{29}$$

with $u(t)$ an auxiliary control input to be optimized [20]. The closed-loop system becomes

$$D(q)\dot{r}(t) = -h(q,\dot{q})r(t) + u(t). \tag{30}$$

This is the error system wherein the instantaneous performance mesaure dynamics is driven by an auxiliary control input.

From the robot dynamic equation described in (2), the control aim is to keep the dynamic response of the robot according to some established operation criteria. The control problem can be considered as a tracking problem. A feedback control is presented in [20] to make the robot track a reference trajectory.

Thus, the motion control objective is to follow a desired trajectory $q_d(t)$ and $\dot{q}_d(t)$ with suitable small position errors $e(t)$ and velocity errors $\dot{e}(t)$. The position error $e(t)$ is expressed in terms of the instantaneous performance index as

$$\dot{e}(t) = -\Gamma e(t) + r(t). \tag{31}$$

Combining (31) and (21), the following system is obtained

$$\begin{bmatrix} \dot{e} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} -\Gamma & I \\ 0 & -D^{-1}(q)H(q,\dot{q}) \end{bmatrix} \begin{bmatrix} e \\ r \end{bmatrix} +$$

$$+ \begin{bmatrix} 0 \\ D^{-1}(q) \end{bmatrix} u \tag{32}$$

or using an equivalent shorter notation

$$\dot{z} = A(q,\dot{q})z(t) + B(q)u(t) \tag{33}$$

where

$$A(q,\dot{q}) = \begin{bmatrix} -\Gamma & I \\ 0 & -D^{-1}(q)H(q,\dot{q}) \end{bmatrix}, \tag{34}$$

$$B(q) = \begin{bmatrix} 0 \\ D^{-1}(q) \end{bmatrix}, \tag{35}$$

$$z = \begin{bmatrix} e \\ r \end{bmatrix}. \tag{36}$$

If the robot dynamic is known exactly, then a quadratic performance index $J(u)$ can be formulated as follows

$$J(u) = \int_{t_0}^{\infty} L(z,u)dt \tag{37}$$

with the Lagrangian

$$L(z,u) = \frac{1}{2}z^T(t)Qz(t) + \frac{1}{2}u^T(t)Ru(t) =$$

$$= \frac{1}{2}\left[e^T r^T\right]\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{12} & Q_{22} \end{bmatrix}\begin{bmatrix} e \\ r \end{bmatrix} + \frac{1}{2}u^T Ru, \tag{38}$$

where $R = R^T > 0 \in R^{n\times n}$ is the control weight matrix and $Q = Q^T > 0 \in R^{2n\times 2n}$ is the state weight matrix.

An optimal control input $u^*(t)$ minimizes the quadratic performance index (37) if there exists a function $V = V(z,t)$ satisfying the Hamilton-Jacobi-Belman (H-J-B) equation

$$\frac{\partial V(z,t)}{\partial t} + \min_u H(z,u,\frac{\partial V(z,t)}{\partial z},t) = 0\ , \tag{39}$$

where the Hamiltonian of optimization is

$$H(z,u,\frac{\partial V(z,t)}{\partial z},t) = L(z,u) + \frac{\partial V(z,t)}{\partial z}\dot{z}. \tag{40}$$

In this case $V = \frac{1}{2}z^T P(q)z$, where

$$P(q) = \begin{bmatrix} K & 0 \\ 0 & D(q) \end{bmatrix},\ K = K^T \in R^{n\times n}. \tag{41}$$

The optimal control input $u^*(t)$ that minimizes (37) under the constraints in (32) is given by

$$u^*(t) = -R^{-1}B^T P(q)z =$$

$$= -R^{-1}r(t) = -R^{-1}\{\dot{e}(t) - \Gamma e(t)\}, \tag{42}$$

where P(q) is the solution of the Riccati differential equation

$$P(q)A + A^T P(q)^T - P(q)BR^{-1}B^T P(q) + \dot{P}(q) + Q = 0. \tag{43}$$

However, a suitable choice of the weight matrices allows to obtain the matrices K and Γ [20] without solving any Riccati differential equation. If weight matrices Q and R are chosen such that

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{12}^T & Q_{22} \end{bmatrix} > 0,\quad R^{-1} = Q_{22}, \tag{44}$$

with $Q_{12} + Q_{12}^T < 0$, then K and Γ can be determined by

$$K = K^T = -\frac{1}{2}(Q_{12} + Q_{12}^T) > 0 \tag{45}$$

$$\Gamma^T K + K\Gamma = Q_{11}. \tag{46}$$

Note that (46) represents a non-symmetric Lyapunov equation.

The following graphs show the trajectories and the references for each one of the links of both robot models, using four different packages from the SLICOT nonlinear interface solver. In these graphs time in seconds is represented on the horizontal axis and position in radians on the vertical axis.

The optimal trajectory control will be applied to the two Puma robot models described above where the reference trajectory is given by

$$q_d(t) = q_d = [1.5, 1.0, 1.0, 2.0, 1.0, 2.0]^T,$$

where the values are given in radians, and the initial position is $q(0) = \dot{q}(0) = 0$.

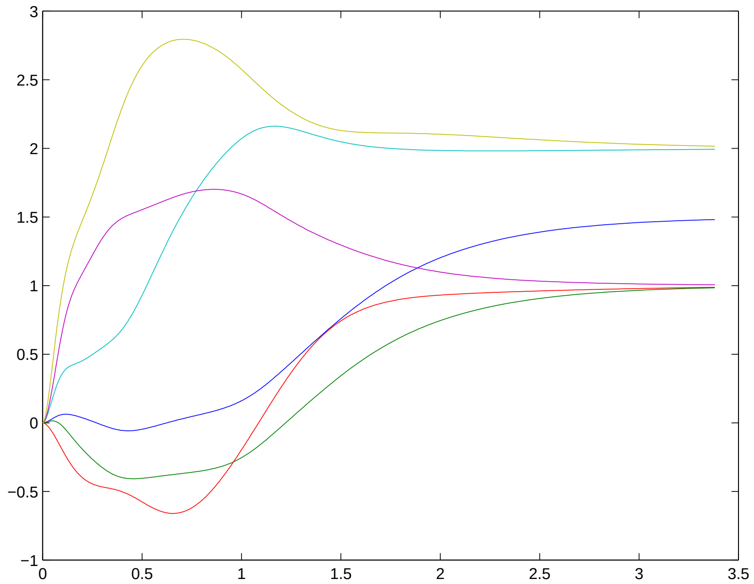

Figure 13: Simulation of the Puma 500 Robot Manipulator Control with LSODE.

The code for the simulation (in Matlab) for robot Puma 500 is shown below. Code for Puma 560 robot only differs in the value of the parameters J, c and E. The complete set of routines to run the tests can be downloaded from the NICONET webpage.

```
%
%
%  x= [q;qdot];
%
```

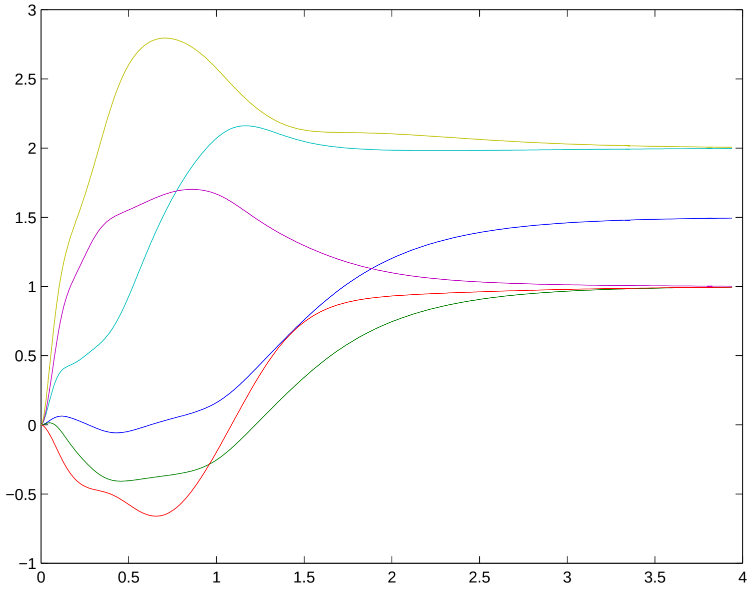

Figure 14: Simulation of the Puma 500 Robot Manipulator Control with LSODA.

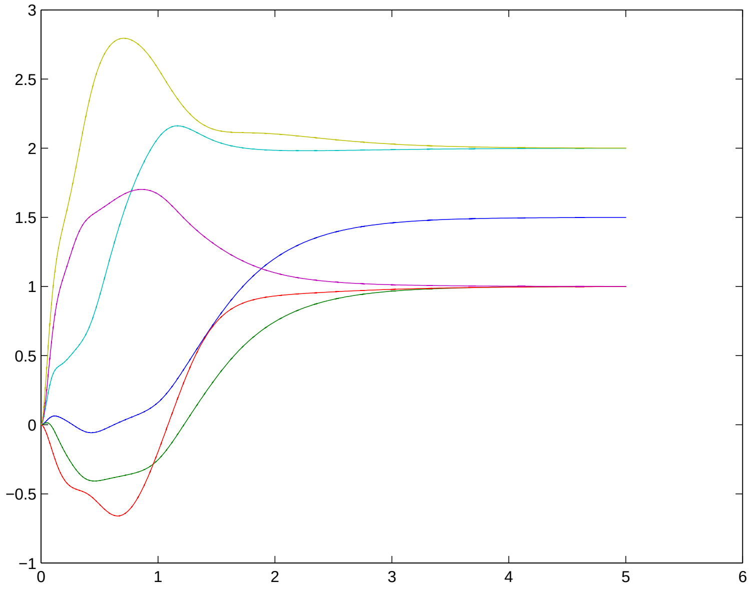

Figure 15: Simulation of the Puma 500 Robot Manipulator Control with LSODI.

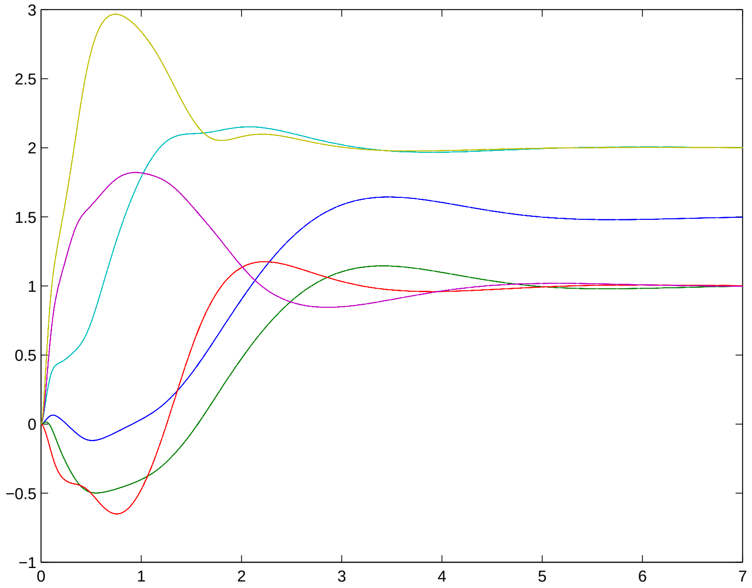

Figure 16: Simulation of the Puma 500 Robot Manipulator Control with RADAU5.

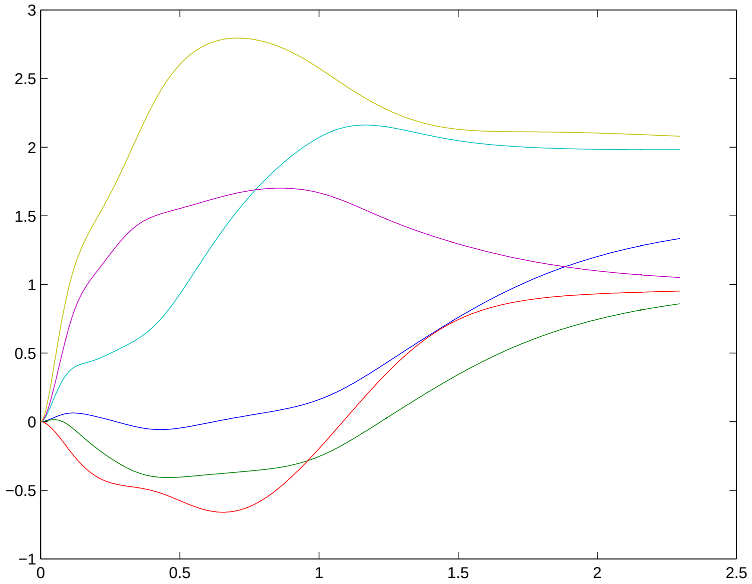

Figure 17: Simulation of the Puma 500 Robot Manipulator Control with DASSL.

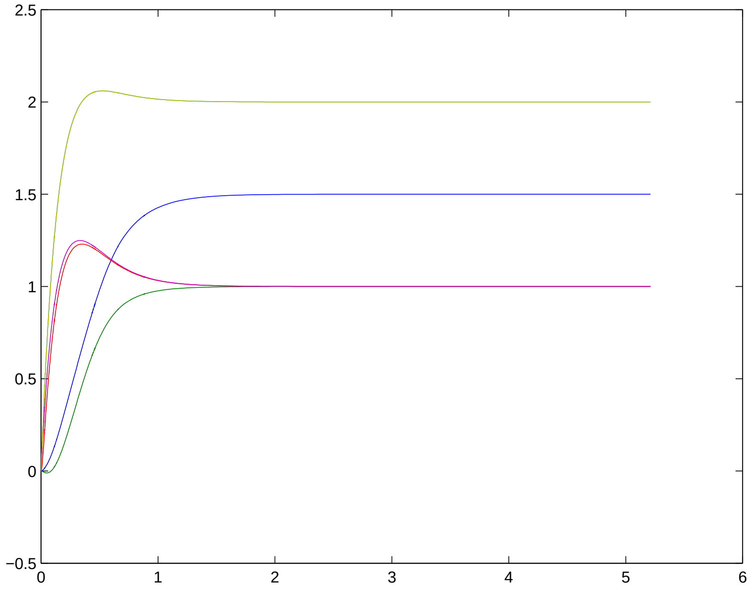

Figure 18: Simulation of the Puma 560 Robot Manipulator Control with LSODE.

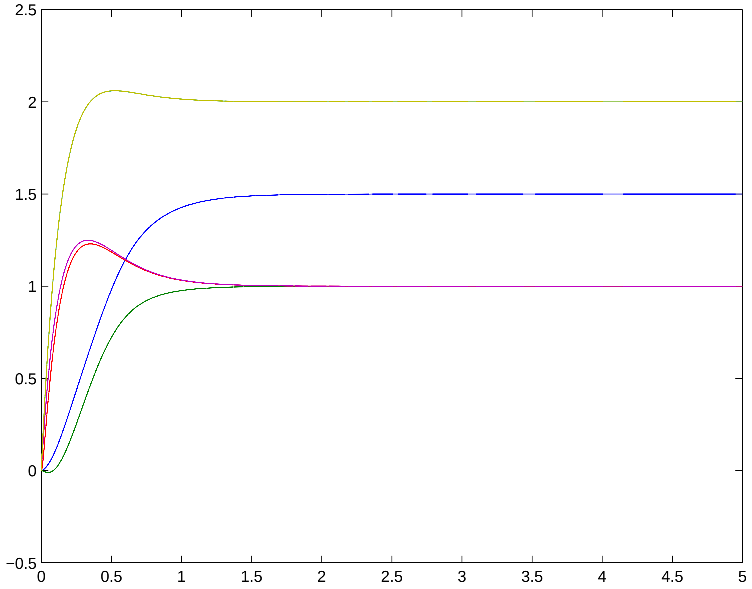

Figure 19: Simulation of the Puma 560 Robot Manipulator Control with RADAU5.

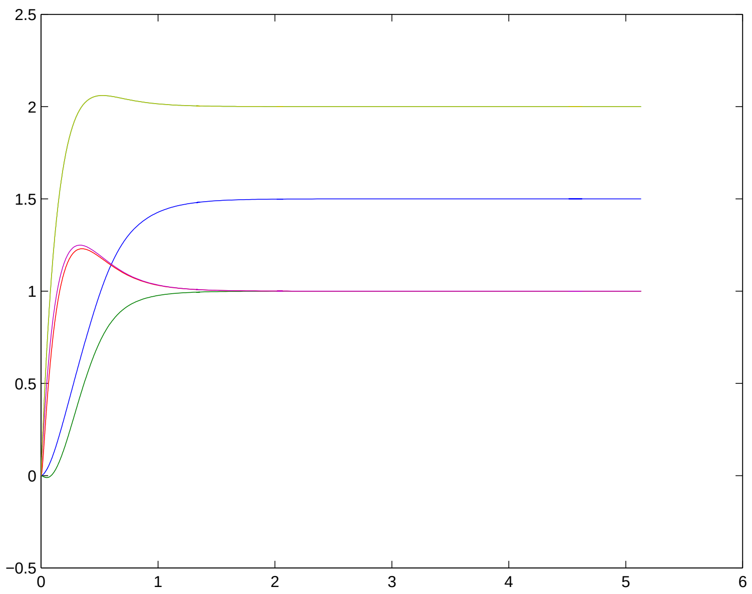

Figure 20: Simulation of the Puma 560 Robot Manipulator Control with DASSL.

```
%  xdot = [ 0               I      ]       [    0    ]          [       0       ]
%         [ 0 -D(q)^-1 H(q,qdot)) ] * x + [ D^-1(q) ] * tau + [ -D^-1(q)c(q) ]
%
%

function [xdot, ierr] = fdif(rpar, ipar, x, uu, p, t)

  nn = size(x,1);

  n = nn/2;

% ... Set the Constants ...

  g  = [0 0 -9.8 0]';
  E  = [         0                  0                  0                  0
                 0                  0  -1.57079632679490                  0
                 0   0.71000000000000                  0                  0
                 0   0.85000000000000  -1.57079632679490   0.12500000000000
                 0                  0  -1.57079632679490                  0
                 0                  0  -1.57079632679490                  0];

  Tl = zeros(6,1);
```

```
  r = [ 1     1     1     1     1     1
        0     0     0     0     0     0
        0     0     0     0     0     0
        1     1     1     1     1     1];

  turn = [ 0         0         0    1.1547         0    1.1547
           0         0         0    1.1547         0    1.1547
           0         0         0    1.1547         0    1.1547
           0         0         0    1.1547         0    1.1547
           0         0         0    1.1547         0    1.1547
           0         0         0    1.1547         0    1.1547];

  m = ones(6,1);

  u = zeros(6,1);

% Start of computations

  q  = x(1:n);
  qp = x(n+1:nn);

% Intertial Matrix

  J1 = [1.3333      0      0      1
             0      0      0      0
             0      0      0      0
        1.3333      0      0      1];

  J = [ J1; J1; J1; J1; J1; J1];

%  Iteration dependent values

  for i=1:n
     if (Tl(i)==0)
        E(i,1)=q(i);
     else
        E(i,4)=q(i);
     end
  end

  A    = computeA(n, Tl, E);
  U    = computeU(A, Tl, n);
  DU   = computeDU(A, U, Tl, n);
  Y    = computeY(U, J, Tl, n);
  H    = computeH( DU, Y, Tl, n);
```

```
  D    = computeD( U,  Y, Tl, n);
  Dinv = inv(D);
  c    = computec(m, g, r, U, Tl, n);

  for i=1:n
      Vm(i,:)=qp'*H((i-1)*n+1:i*n,:);
  end

  F1 = [ zeros(n)    eye(n)
         zeros(n)  -Dinv*(Vm)];

  F2 = [ zeros(n)
            Dinv    ];

  F3 = [ zeros(n,1)
           -Dinv*G   ];


% Optimal Control

% Q=[Q11 Q12; Q21 Q22]
  Q = [ ones(6)*2+eye(6)*8 eye(6)* -1 ; eye(6)* -1 eye(6)*30 ];
  Q11=Q(1:n,1:n);
  Q12=Q(1:n,n+1:2*n);
  Q22=Q(n+1:2*n,n+1:2*n);


  qd = [1.5000    1.0000    1.0000    2.0000    1.0000    2.0000]';
  qpd = zeros(6,1);
  qppd = zeros(6,1);

% Inverse of weigth matrix R
  iR=Q22;


% Gamma matrix

  Gamma = ones(6) + eye(6)*4;

  % e

    e=qd-q;

    % ep
```

```
  ep=qpd-qp;

u=-iR*(ep+Gamma*e);

phi=D*(qppd+Gamma*ep)+Vm*(qpd+Gamma*e)+G;

% tau

tau=h-u;

xdot = F1*x + F2*tau + F3;
```

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.

[15] D. Kirk, *Optimal Control Theory: An Introduction*, Prentice-Hall. Englewood Cliffs, NJ, 1970.

[16] F.L. Lewis and V.L. Syrmos, *Optimal Control*, Addison-Wesley. New York, 1995.

[17] W.M. Lioen, J.J.B de Swart, *Test Set for Initial Value Problem Solvers*, Technical Report NM-R9615, CWI, Amsterdam, 1996. http://www.cwi.nl/cwi/projects/IVPTestset/.

[18] V. Hernandez, I. Blanquer, E. Arias, and P. Ruiz, *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*, Universidad Politecnica de Valencia, DSIC. SLICOT Working Note 2000-3: July 2000. Available at ftp://wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/SLWN2000-3.ps.Z.

[19] J.J.B. de Swart, W.M. Lioen, W.A. van der Veen, *PSIDE, November 25, 1998. Available at http://www.cwi.nl/cwi/projects/PSIDE/.*

[20] Kim, H. Young, F.L. Lewis, D.M. Dawson, *Intelligent optimal control of robotic manipulators using neural networks.*

[21] J.C. Fernandez, E. Arias, V. Hernández, L. Peñalver, *High Performance Algorithm for Tracking Trajectories of Robot Manipulators*, Preprints of the Proceedings of the 6^{th} IFAC International Workshop on Algorithms and Architectures for Real-Time Control (AARTC-2000), pages 127-134.