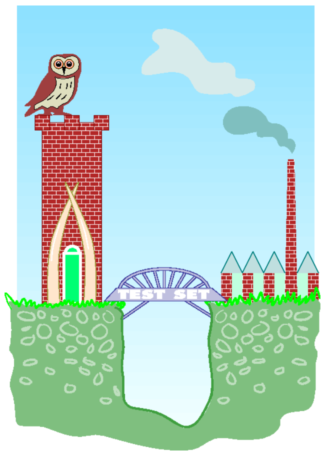
Test Set for Initial Value Problem Solvers

Francesca Mazzia[*], Cecilia Magherini[**]
[*] *Dipartimento di Matematica, Università degli studi di Bari, Italy*
[**] *Dipartimento di Matematica Applicata, Università di Pisa*
(mazzia@dm.uniba.it,cecilia.magherini@ing.unipi.it)

Release 2.4 February 2008

Abstract

The test set for IVP solvers presents a collection of Initial Value Problems to test solvers for implicit differential equations. This test set can both decrease the effort for the code developer to test his software in a reliable way, and cross the bridge between the application field and numerical mathematics, by helping people working in several branches of scientific disciplines in choosing the code most suitable for their problems. This document contains the descriptive part of the test set. It describes the solvers used in the comparisons, the test problems and their origin, and reports on the behavior of the solvers on these problems. The latest version of this document and the software part of the test set is available via the world wide web at http://www.dm.uniba.it/~testset. The software part serves as a platform on which one can test the performance of a solver on a particular test problem oneself. Instructions how to use this software are in this paper as well. The idea to develop this test set was discussed at the workshop ODE to NODE, held in Geiranger, Norway, 19–22 June 1995 and was developed by the CWI group. After the workshop ANODE01, held in Auckland, New Zealand, 2001, the testset moved to the University of Bari.



1991 Mathematics Subject Classification: Primary: 65Y20, Secondary: 65-04, 65C20, 65L05.
Computing Reviews Classification System: G.1.7, G.4.
*Keywords and Phrases:*test problems, software, IVP, IDE, ODE, DAE.

Acknowledgements: This work has been supported by the project "Metodi numerici e software per equazioni differenziali ordinarie: problemi a valori iniziali ed al contorno", sponsored by the University of Bari (2006-2008) and by the PRIN 2004-2006 project "Metodi numerici e software matematico per le applicazioni". We thanks the GNCS-INDAM that supported financially the project form 2001 to 2005 and made possible the use of supercomputer facilities in 2002. We thanks the CWI group, that formerly maintained the testset, for providing useful software. We thank all contributors to this test set, without whom it would not be possible to collect problems from such a wide variety of application fields.

Contents

Part II - Problems

I Introduction

I.1 The idea behind this test set

Both engineers and computational scientists alike will benefit greatly from having a standard test set for Initial Value Problems (IVPs) which includes documentation of the test problems, experimental results from a number of proven solvers, and Fortran subroutines providing a common interface to the defining problem functions. Engineers will be able to see at a glance which methods will be most effective for their class of problems. Researchers will be able to compare their new methods with the results of existing ones without incurring additional programming workload; they will have a reference with which their colleagues are familiar. This test set tries to fulfill these demands and tries to set a standard for IVP solver testing. We hope that the following features of this test set will enable the achievement of this goal:

- uniform presentation of the problems,
- ample description of the origin of the problems,
- robust interfaces between problem and drivers,
- portability among different platforms,
- contributions by people from several application fields,
- presence of real-life problems,
- being used, tested and debugged by a large, international group of researchers,
- comparisons of the performance of well-known solvers,
- interpretation of the numerical solution in terms of the application field,
- ease of access and use.

There exist other test sets, e.g., NSDTST and STDTST by Enright & Pryce [EP87], PADETEST by Bellen [Bel92], the Geneva test set by Hairer & Wanner [HW] and the Test Frame for Ordinary Differential Equations by Nowak and Gebauer [NG97], which all have their own qualities.

I.2 Structure of this test set

The test set consists of a descriptive part and a software part. The first part describes solvers and test problems and reports on the behavior of the solvers when applied to these problems. Section II explains how this information is presented. The software serves as a platform to test the performance of a solver on a particular test problem by a user of the test set. In Section IV we specify the format of the Fortran subroutines and explain how to run test problems with the help of drivers that make these codes suitable for runs with a number of solvers. Currently, BIMD, DASSL, GAMD, MEBDFDAE, MEBDFI, PSIDE, RADAU, RADAU5 and VODE are supported.

I.3 How to submit new test problems

We invite people to contribute new test problems to this test set. To restrict the amount of time required for the maintainers of the test set to incorporate new problems, it is important that the submissions are in the prescribed format. Firstly, every problem should have a description containing the 4 sections mentioned in Section II, preferably as a LaTeX-file. Secondly, a set of Fortran subroutines that is necessary for the implementation has to be supplied in the format specified in Section IV

Submissions can be sent by e-mail to `testset@dm.uniba.it`

I.4 How to obtain this test set

The latest release of this test set can be obtained via the WWW page with URL

http://www.dm.uniba.it/~testset ,

The first release of this test set appeared in [LSV96], the second release in [LS98], the other releases in [MI03], [MMI06].

I.5 Acknowledgements

We gratefully acknowledge Jacek Kierzenka for his help in defining the interface that that allow the use of the IVP test set problems in the MATLAB environment, the CWI group that set up the first two versions of the testset: P.J. van der Houwen , W. Hoffmann, B.P. Sommeijer, W.M. Lioen, W.A. van der Veen, J.J.B. de Swart, J.E. Frank. In particular we wish to thank P.J. van der Houwen and Walter Lioen, who helped us during the installation procedure.

I.6 People involved

This test set is maintained by the INdAM Bari unit project group Codes and test problems for Differential Equations (coordinator F. Mazzia). The revision 2.4 has been sponsored by the project ”Metodi numerici e software per equazioni differenziali ordinarie: problemi a valori iniziali ed al contorno” of the University of Bari (2006). The revision 2.3 has been sponsored by the project PRIN 2004 ”Metodi numerici e software matematico per le applicazioni” (coordinator L. Brugnano, local coordinator F. Mazzia) and by the project ”Metodi Numerici per equazioni differenziali” (coordinator P. Amodio), sponsored by the University of Bari. In January 2002 a steering committee of A. Bellen (Università di Trieste, Italy) , J. R. Cash (Imperial College, London, U.K.), E. Hairer (Université de Genève, Switzerland), F. Krogh (Math à la Carte, Tujiunga, California, U.S.A), L. Petzold (University of California, Snata Barbara, U.S.A), B. Simeon, G. Soderlind (Lund University,Sweden), D. Trigiante (Università di Firenze, Italy) and P.J. van der Houwen (formerly at CWI, Amsterdam, The Netherlands) has been set up to oversee this project.

References

[Bel92] A. Bellen. PADETEST: a set of real-life test differential equations for parallel computing. Technical Report 103, Dipartimento di Scienze Matematiche, Università di Trieste, 1992.

[EP87] W.H. Enright and J.D. Pryce. Two Fortran packages for assessing initial value methods. *ACM Transactions on Mathematical Software*, 13-I:1–27, 1987.

[HW] E. Hairer and G. Wanner. *Testset of Stiff ODEs*. Geneva. Available at http://www.unige.ch/math/folks/hairer/testset/testset.html.

[LS98] W.M. Lioen and J.J.B. de Swart. *Test Set for Initial Value Problem Solvers*, dec 1998. Available at http://www.dm.uniba.it/~testset.

[LSV96] W.M. Lioen, J.J.B. de Swart, and W.A. van der Veen. Test set for IVP solvers. Technical Report NM-R9615, CWI, Amsterdam, 1996.

[MI03] F. Mazzia and F. Iavernaro. Test set for initial value problem solvers. Technical Report 40, Department of Mathematics, University of Bari, 2003. Available at http://www.dm.uniba.it/~testset.

[MMI06] F. Mazzia, C. Magherini, and F. Iavernaro. Test set for initial value problem solvers. Technical Report 43, Department of Mathematics, University of Bari, 2006. Available at `http://www.dm.uniba.it/~testset`.

[NG97] Ulrich Nowak and Susanna Gebauer. A new test frame for ordinary differential equations. Technical Report SC 97-68, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, 1997.

II Format of the problem descriptions

Every problem description contains the four sections described below.

II.1 General information

The problem identification is given: the type of problem (IDE, ODE or DAE), its dimension and index. The contributor and any further relevant information are listed too. What is meant here by IDE, ODE, DAE and index, is explained in §IV.

II.2 Mathematical description of the problem

All ingredients that are necessary for implementation are given in mathematical formulas.

II.3 Origin of the problem

A brief description of the origin of the problem, in order to give its physical interpretation. References to the literature are given for further details.

II.4 Numerical solution of the problem

This section contains:

1. **Reference solution at the end of the integration interval.** The values of (some of) the components of a reference solution at the end of the integration interval are listed.

2. **Run characteristics.** Integration statistics, if applicable, of runs with BIMD, DASSL, GAMD, MEBDFDAE, MEBDFI, PSIDE, RADAU, RADAU5, and VODE serve to give insight in the numerical difficulty of the problem.

   The experiments were done on an Alphaserver DS20E, with a 667 MHz EV67 processor. We used the Fortran 90 compiler with optimization: `f90 -O5 <source code>`. If a run did not produce correct results then we report what went wrong.

   The characteristics are in the following format:

   - *solver*
     The name of the numerical solver with which the run was performed.
   - *rtol*
     The user supplied relative error tolerance.
   - *atol*
     The user supplied absolute error tolerance.
   - *h0*
     The user supplied initial step size (if relevant).
   - *scd*
     The scd values denote the minimum number of significant correct digits in the numerical solution at the end of the integration interval, i.e.

     $$\text{scd} := -\log_{10}(\|\text{ relative error at the end of the integration interval }\|_\infty). \qquad (.\text{II}.1)$$

     If some components of the solution vector are not taken into account for the computation of the scd value, or if the absolute error is computed instead of the relative error, then this is specified locally.

- *mescd*

$$\text{mescd} := -\log_{10}(\|\ \text{absolute error} ./ (\text{atol}./\text{rtol} + |\ \text{ytrue}\ |)\ \|_\infty). \qquad (.II.2)$$

  where the absolute error is computed at the end of the integration interval, atol and rtol are the input tolerances, ytrue is the exact solution at the end of the integration interval and ./ and .∗ are element by element operators. In this case all the components of the solution are taken into account.

- *steps*
  Total number of steps taken by the solver (including rejected steps due to error test failures and/or convergence test failures).
- *accept*
  The number of accepted steps.
- *# f* and *# Jac*
  The number of evaluations of the derivative function and its Jacobians, respectively.
- *# LU*
  The number of LU-decompositions (not for DASSL). The codes, except for RADAU and RADAU5, count the LU-decompositions of systems of dimension d, where d is the dimension of the test problem.

  RADAU and RADAU5 use an s-stage Radau IIA method. For RADAU5, $s = 3$ and for RADAU, $s = 3$, 5 or 7. Every iteration of the inexact Newton process, used for solving systems of non-linear equations, requires the solution of a linear system of dimension sd. By means of transformations, this linear system is reduced to $(s + 1)/2$ linear systems of dimension d. Of these systems, one system is real, and $(s - 1)/2$ systems are complex. The decompositions of all $(s + 1)/2$ linear systems are counted by RADAU and RADAU5 as 1 LU-decomposition.
- *CPU*
  The CPU time in seconds to perform the run on the aforementioned computer. Since timings may depend on other processes (like e.g. daemons), we perform 10 runs, discard the maximum and minimum values and list the medium of the CPU times.

PSIDE – Parallel Software for Implicit Differential Equations – is a Fortran 77 code for solving IDE problems. It is developed for parallel, shared memory computers. The integration characteristics in the tables refer to a one-processor computer. Since PSIDE can do four function evaluations and four linear system solves concurrently on a computer with four processors, one may divide the number of function evaluations, decompositions and solves in the tables by four to obtain the analogous *effective* characteristics for four-processor machines.

3. **Behavior of the numerical solution.** Plots of (some of) the solution components over (part of) the integration interval are presented.

4. **Work-precision diagrams.** For every relevant solver, a range of input tolerances and, if necessary, a range of initial stepsizes, were used to produce plots of the resulting scd or mescd values, defined in Formulas (.II.1) and (.II.2), against the number of CPU seconds needed for the run on the aforementioned computer, with the setting as described before. Here we took again the medium of the CPU times of 10 runs, after discarding the maximum and minimum values. The format of these diagrams is as in Hairer & Wanner [HW96, pp. 166–167, 324–325]. The range of input tolerances and initial stepsizes is problem dependent and specified locally. The input parameters for the runs in the tables with run characteristics are such that these runs appear in the work-precision diagrams as well. The code PSIDE has been performed only on one processor.

We want to emphasize that the reader should be careful with using these diagrams for a mutual comparison of the solvers. The diagrams just show the result of runs with the prescribed input on the specified computer. A more sophisticated setting of the input parameters, another computer or compiler, as well as another range of tolerances might change the diagrams considerably.

References

[HW96] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-algebraic Problems.* Springer-Verlag, second revised edition, 1996.

IV The software part of the test set

IV.1 Classification of test problems

We have categorized the test problems in three classes: IDEs, ODEs and DAEs.

In this test set, we call a problem an **IDE** (system of Implicit Differential Equations) if it is of the form

$$\begin{array}{ll} f(t,y,y') = 0, & t_0 \le t \le t_{\mathrm{end}}, \\ y, f \in \mathbf{R}^d, & \\ y(t_0) \text{ and } y'(t_0) \text{ are given.} & \end{array}$$

A problem is named an **ODE** (system of Ordinary Differential Equations), if it has the form

$$\begin{array}{ll} y' = f(t,y), & t_0 \le t \le t_{\mathrm{end}}, \\ y, f \in \mathbf{R}^d, & \\ y(t_0) \text{ is given,} & \end{array}$$

whereas the label **DAE** is given to problems which can be cast in the form

$$\begin{array}{ll} My' = f(t,y), & t_0 \le t \le t_{\mathrm{end}}, \\ y, f \in \mathbf{R}^d, & M \in \mathbf{R}^{d\times d}, \\ y(t_0) \text{ and } y'(t_0) \text{ are given,} & \end{array}$$

where M is a constant, possibly singular matrix. Note that ODEs and DAEs are subclasses of IDEs.

IV.2 How to perform tests

You can perform one of the following types of tests:

- solve test set problems with solvers that are supported in the test set,
- solve test set problems with your own solver,
- solve your own problem with solvers that are supported in the test set,
- solve a test set problem using the web facility,
- solve your own problem using the web facility,
- solve test set problems using a MATLAB solver,
- solve you own problem in the test set format using a MATLAB solver.

For the first five types of tests, four types of codes are involved: a solver, a driver, a problem code and auxiliary routines, for the last two types of tests the matlab interface of the problem is generated using two axiliary routines. The solvers available are described in §I-1-1–I-9-1. Currently, there are 9 solvers available:

1. BIMD for ODEs and DAEs of index less than or equal to 3,
2. DASSL for ODEs and IDEs/DAEs of index less than or equal to 1,
3. GAMD for ODEs and DAEs of index less than or equal to 3,
4. MEBDFDAE for ODEs and DAEs of index less than or equal to 3,
5. MEBDFI for ODEs and IDEs/DAEs of index less than or equal to 3,

6. PSIDE for ODEs and IDEs/DAEs of index upto at least 3,
7. RADAU for ODEs and DAEs of index less than or equal to 3,
8. RADAU5 for ODEs and DAEs of index less than or equal to 3, and
9. VODE for ODEs.

These solvers can be obtained via [MM08] in the files `bimd.f`, `ddassl.f`, `gamd.f90`, `mebdfd.f`, `mebdfi.f`, `pside.f`, `radau.f`, `radau5.f` and `vode.f`. These files contain versions of the solvers with which the numerical experiments were conducted. The official links to the solvers, which possibly direct to more recent versions, can be found at [MM08] too.

The drivers `bimdd.f`, `dassld.f`, `gamdd.f`, `mebdfdaed.f`, `mebdfid.f`, `psided.f`, `radaud.f`, `radau5d.f` and `voded.f`, which are available at [MM08], are such that runs can be performed that solve the problem numerically with the aforementioned solvers.

For every test problem, the file `problem.f` contains a set of nine Fortran 77 subroutines defining the problem. Although the format of the subroutines is the same for all three classes, the meaning of the arguments may depend on the problem class. Section IV.3 describes the format of the problem codes.

The auxiliary linear algebra routines for the solvers are in `bimda.f`, `dassla.f`, `gamda.f90`, `psidea.f`, `radaua.f` (for both RADAU and RADAU5) and `vodea.f`. For MEBDFDAE/MEBFI, the linear algebra routines are included in `mebdfdae.f/mebdfi.f`. The auxiliary file `report.f` contains a user interface. All these files are available at [MM08] as well.

IV.2.1 How to solve test problems with available solvers

Compiling

```
f77  -o dotest  solverd.f  problem.f  solvera.f  solver.f  report.f,
```

for the solvers written in Fortran 77, will yield an executable `dotest` that solves the problem, of which the Fortran routines in the format described in Section IV.3 are in the file `problem.f`. A complete description of each solver together with some examples are reported in the SOLVERS sections §I-2-1–I-9-1. A `makefile` is also available in the [MM08] to help in the compilation steps.

IV.2.2 How to solve test problems with your own solver

The following guidelines serve to test your own solver with the test set problems.

- Write your own solver in a format similar to existing solvers in the file `own.f`.
- (Optional) You may like to put the linear algebra subroutines in a separate file `owna.f`. In this way you can, for example, use the linear algebra of an existing solver.
- Write driver subroutines in the file `ownd.f`. If the format of your solver is similar to that of a solver that is already available in the test set, then this will only require minor modifications of the driver routines of that solver.
- Adjust the file `report.f` as indicated in the comment lines of this file. This will only be a minor modification.
- Compiling

  ```
  f77  -o dotest  ownd.f  problem.f  own.f  owna.f  report.f,
  ```

  will yield an executable `dotest` that solves the problem, of which the Fortran routines are in the file `problem.f`

IV.2.3 How to solve your own problem with available solvers

The following guidelines serve to solve your own problem with the solvers that are supported in the test set.

- Write your own problem in a format similar to that of the test set problems in the file `newprob.f`. This format is described precisely in Section IV.3.
- Adjust the file `report.f` as indicated in the comment lines of this file. This will only be a minor modification.
- To solve your problem with, for example, DASSL, compiling

  ```
  f77  -o dotest  dassld.f  newprob.f  ddassl.f  dassla.f  report.f,
  ```

  will give you the desired executable `dotest`.

IV.2.4 How to solve a test set problem using the web facility

In [MM08], following the link “compile and run on line” it is possible to solve a test set problem on-line, using the supported solvers. The user input are the relative tolerance *rtol*, the absolute tolerance *atol* and the initial stepsize *h0* for the solvers that need it. As a results the solution computed in the last point, the *scd* and *mescd* and some integration characteristics, as described in §II.4, are displayed. The plots of some component of the solution are also visualized.

IV.2.5 How to solve your own problem using the web facility

In [MM08], following the link “compile and run on line” it is possible to upload a file containing the subroutines describing the problem written using the format described precisely in §IV.3. Then it is possible to choose one of the supported solver for the solution of the problem. The user input are the relative tolerance *rtol*, the absolute tolerance *atol* and the initial stepsize *h0* for the solvers that need it. As a results the solution computed in the last point, the *scd* and *mescd* if the reference solution is available and some integration characteristics, as described in §II.4, are displayed. The plots of the components of the solution defined in the subroutine `setoutput` are also visualized.

IV.2.6 How to solve test set problems using a MATLAB solver

The MATLAB [Mat] function `minterface.m` together with the fortran function `matlab_interface.F`, allow to construct the mex files to run problems in the MATLAB environment. The only restriction is that you need to put the problems and the auxiliary routines in the correct directory. We suggest to download the complete distribution tree of the IVP test set in [MM08] if you want to use the matlab interface.

The MATLAB instruction:

```
MPROB = minterface(problem)
```

returns a function handle to a MEX-Function interface to problem `problem`. If needed, the Fortran MEX-Function interface is automatically generated and compiled, you need a Fortran compiler compatible with the MATLAB environment to complete the compilation steps. Moreover, before using, for the first time, this utilities, at the MATLAB prompt type

```
mex -setup
```

and select the Fortran compiler you want to use.

The interface `mprob` supports the following calling sequences:

```
    PROB = MPROB('Prob')
    [Y0,YP0,CONSIST] = MPROB('Init',NEQ,T0)
    [ATOL,RTOL]      =  MPROB('Tolerances',NEQN,ATOL,RTOL)
    [F,IERR,RPAR,IPAR] = MPROB('Feval',NEQ,T,Y,YP,RPAR,IPAR)
    [J,IERR,RPAR,IPAR] = MPROB('Jeval',LDIM,NEQ,T,Y,YP,RPAR,IPAR)
    [M,IERR,RPAR,IPAR] = MPROB('Meval',LDIM,NEQ,T,Y,YP,RPAR,IPAR)
    Y = MPROB('Solut',NEQ,TFINAL)
    [MESCD,SCD] = MPROB('Report',NEQ,YREF,Y,PROBNM,TOLVEC,ATOL,RTOL)
    MPROB('Help')
```

The input parameters are the same defined in IV.3 for the fortran functions defining the problem.

The function `odetest.m` contains a user interface to run and compile the problems in the MATLAB environment. As an example the instruction:

```
>> [sol,stats] = odetest(problem,'ode15s',1e-5,1e-4,1)
```

solves the problem using the matlab solver 'ode15s', with absolute tolerance equal to 1e-5, relative tolerance equal to 1e-4, the first component of the solution is plotted using the MATLAB function 'odeplot'. The output variable sol contains information about the solution.

Use the MATLAB help to have information about the input/output parameters of the functions.

IV.2.7 How to solve you own problem in the test set format using a MATLAB solver

Write your own problem in a format similar to that of the test set problems, as described in Section IV.3 in the file newprob.f. Then put the file in the correct directory in the testset distribution. The instruction:

```
  MPROB = minterface(newprob)
```

returns a function handle to a MEX-Function interface to problem `newprob`.

The instruction `odetest`

```
>> [sol,stats] = odetest(newprob,'ode15s',1e-5,1e-4,1)
```

will automatically generate the function handle and solve the problem with the MATLAB solver `ode15s`.

IV.3 Format of the problem codes

The eight subroutines that define the problem are called `PROB`, `INIT`, `SETTOLERANCES`, `SETOUTPUT`, `FEVAL`, `JEVAL`, `MEVAL`, and `SOLUT`. The following subsections describe the format of these subroutines in full detail. An additional function `PIDATE` allows to check the problem interface date, for the current release this function should be equal to:

```
      integer function pidate()
      pidate = 20060828
      return
      end
```

In the sequel, the variables listed under `INTENT(IN)`, `INTENT(INOUT)`, and `INTENT(OUT)` are input, update and output variables, respectively.

IV.3.1 Subroutine `PROB`

This routine gives some general information about the test problem.

```
      SUBROUTINE PROB(FULLNM,PROBLM,TYPE,
     +                NEQN,NDISC,T,
     +                NUMJAC,MLJAC,MUJAC,
     +                NUMMAS,MLMAS,MUMAS,
     +                IND)
      CHARACTER*(*) FULLNM, PROBLM, TYPE
      INTEGER NEQN,NDISC,MLJAC,MUJAC,MLMAS,MUMAS,IND(*)
      DOUBLE PRECISION T(0:*)
      LOGICAL NUMJAC, NUMMAS
C     INTENT(OUT) FULLNM,PROBLM,TYPE,NEQN,NDISC,T,NUMJAC,MLJAC,
C    +            MUJAC,NUMMAS,MLMAS,MUMAS,IND
```

Meaning of the arguments:

`FULLNM`
: This character string contains the long name of the problem, e.g. `Chemical Akzo Nobel problem`.

`PROBLM`
: This character string contains the short name of the problem, e.g. `chemakzo`, and corresponds to the name of the Fortran source file.

`TYPE`
: This character string takes the value `IDE`, `ODE` or `DAE`, depending on the type of problem.

`NEQN`
: The dimension d of the problem, which is the number of equations to be solved.

`NDISC`
: The number of discontinuities in time of the function f or its derivative. The solver is restarted at every such discontinuity by the driver.

`T`
: An array containing time points.

  - If `NDISC .EQ. 0`, then `T(0)` contains t_0 and `T(1)` contains t_{end}.
  - If `NDISC .GT. 0`, then `T(0)` contains t_0, `T(NDISC+1)` contains t_{end} and `T(1)` ... `T(NDISC)` are the time points where the function f or its derivative has a discontinuity in time.

`NUMJAC`
: To solve the problem numerically, it is necessary to use the partial derivative $J := \partial f / \partial y$. If J is available analytically, then `NUMJAC = .FALSE.` and J is provided via subroutine `JEVAL`. If J is not available, then `NUMJAC = .TRUE.` and `JEVAL` is a dummy subroutine. In this case, the solvers approximate J by numerical differencing.

`MLJAC` and `MUJAC`
: These integers contain information about the structure of $J := \partial f / \partial y$. If J is a full matrix, then `MLJAC = NEQN`, otherwise `MLJAC` and `MUJAC` equal the number of nonzero lower co-diagonals and the number of nonzero upper co-diagonals of J, respectively.

`NUMMAS`
: Only relevant for IDEs.

- For IDEs, it is necessary to use the partial derivative $M := \partial f/\partial y'$. If M is available analytically, then `NUMMAS = .FALSE.` and M is provided via subroutine `MEVAL`. If M is not available, then `NUMMAS = .TRUE.` and `MEVAL` is a dummy subroutine. In this case, the solvers have to approximate M by numerical differencing.
- For DAEs and ODEs, `NUMMAS` is not referenced.

`MLMAS` and `MUMAS`

These integers contain information about the structure of the constant matrix M (for DAEs) or the matrix $M := \partial f/\partial y'$ (for IDEs).

- For IDEs and DAEs: If M is a full matrix, then `MLMAS = NEQN`, otherwise `MLMAS` and `MUMAS` equal the number of nonzero lower co-diagonals and the number of nonzero upper co-diagonals of M, respectively.
- For ODEs, `MLMAS` and `MUMAS` are not referenced.

`IND`

Connected to IDEs and DAEs is the concept of index.

- For ODEs, `IND` is not referenced.
- For IDEs and DAEs, `IND` is an array of length `NEQN` and `IND(I)` specifies the index of variable `I`.

IV.3.2 Subroutine `INIT`

This routine contains the initial values $y(t_0)$ and $y'(t_0)$.

```
      SUBROUTINE INIT(NEQN,T,Y,YPRIME,CONSIS)
      INTEGER NEQN
      DOUBLE PRECISION T,Y(NEQN),YPRIME(NEQN)
      LOGICAL CONSIS
C     INTENT(IN)  NEQN,T
C     INTENT(OUT) Y,YPRIME,CONSIS
```

Meaning of the arguments:

`NEQN`

The dimension of the problem.

`Y(NEQN)`

Contains the initial value $y(t_0)$.

`YPRIME(NEQN)`

Only relevant for IDEs and DAEs.

- For IDEs and DAEs, `YPRIME` contains the initial value $y'(t_0)$.
- For ODEs, `YPRIME` is not set. If needed by the solver, it is computed in the driver as $y'(t_0) = f(t_0, y_0)$.

`CONSIS`

Only relevant for IDEs and DAEs.

- For IDEs and DAEs, `CONSIS` is a switch for the consistency of the initial values. If `CONSIS .EQ. .TRUE.`, then $y(t_0)$ and $y'(t_0)$ are assumed to be consistent. If `CONSIS .EQ. .FALSE.`, then $y(t_0)$ and $y'(t_0)$ are possibly inconsistent. Solvers with a facility to compute consistent initial values internally, will try to do so in this case. Currently, all problems in the test set have consistent initial values.
- For ODEs, `CONSIS` is not referenced.

IV.3.3 Subroutine SETTOLERANCES

This routine defines the input tolerances `RTOL` and `ATOL`.

```
      SUBROUTINE SETTOLERANCES(NEQN,RTOL,ATOL,TOLVEC)
      INTEGER NEQN
      LOGICAL TOLVEC
      DOUBLE PRECISION RTOL(NEQN), ATOL(NEQN)
C     INTENT(IN)    NEQN
C     INTENT(INOUT) RTOL, ATOL
C     INTENT(OUT)   TOLVEC
```

Meaning of the arguments:

`NEQN`
The dimension of the problem.

`RTOL`
Contains the relative tolerances.

- In input contains the value `RTOL(1)`.
- In output could contain a vector valued `RTOL`, with different values for the relative tolerances in each component.

`ATOL`
Contains the absolute tolerances.

- In input contains the value `ATOL(1)`.
- In output could contain a vector valued `ATOL`, with different values for the absolute tolerances in each component.

`TOLVEC`
Logical output variable.

- `TOLVEC = .TRUE.` if all the component of `RTOL` and `ATOL` are initialized.
- `TOLVEC = .FALSE.` if only the first component of `RTOL` and `ATOL` is initialized.

IV.3.4 Subroutine SETOUTPUT

This routine contains information about the required output.

```
      SUBROUTINE SETOUTPUT(NEQN,SOLREF,PRINTSOLOUT,
     +                     NINDSOL,INDSOL)

      LOGICAL SOLREF, PRINTSOLOUT
      INTEGER NEQN, NINDSOL
      INTEGER INDSOL(NEQN)
C     INTENT(IN)    NEQN
C     INTENT(OUT)   NINDSOL, INDSOL(NEQN), PRINTSOLOUT, SOLREF
```

Meaning of the arguments:

`NEQN`
The dimension of the problem.

`SOLREF`
Contains information about the reference solution.

- `SOLREF = .TRUE.` means that the reference solution is available in the function solut.
- `SOLREF = .FALSE.` means that the reference solution is not available, the subroutine `SOLOUT` must be a dummy subroutine.

`PRINTSOLOUT`
Contains information about the required output.

- `PRINTSOLOUT=.TRUE.` means that some components of the intermediate computed values of the solution are printed in the output file called problemSOLVER.txt.
- This option is not activated for the code `pside`. Moreover a MATLAB file called problemSOLVER.m and a SCILAB file called problemSOLVER.sci are generated as utilities to generate the plots of the printed components of the solution.
- `PRINTSOLOUT=.FALSE.` means that no intermediate values are printed.

`NINDSOL`
If `PRINTSOLOUT=.TRUE.`, `NINDSOL` contains the number of components to be printed.

`INDSOL`
If `PRINTSOLOUT=.TRUE.`, `INDSOL(1:NINDSOL)` contains the index of the `NINDSOL` components to be printed.

IV.3.5 Subroutine `FEVAL`

This subroutine evaluates the function f.

```
      SUBROUTINE FEVAL(NEQN,T,Y,YPRIME,F,IERR,RPAR,IPAR)
      INTEGER NEQN,IERR,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),YPRIME(NEQN),F(NEQN),RPAR(*)
C     INTENT(IN)    NEQN,T,Y,YPRIME
C     INTENT(INOUT) RPAR,IPAR
C     INTENT(OUT)   F,IERR
```

Meaning of the arguments:

`NEQN`
The dimension of the problem.

`T`
The time point where the function is evaluated.

`Y(NEQN)`
The value of y in which the function is evaluated.

`YPRIME(NEQN)`
: Only relevant for IDEs.

  - For IDEs, this is the value of y' in which the function f is evaluated.
  - For ODEs and DAEs, `YPRIME` is not referenced.

`F(NEQN)`
: The resulting function value $f(\mathtt{T},\mathtt{Y})$ (for ODEs and DAEs), or $f(\mathtt{T},\mathtt{Y},\mathtt{YPRIME})$ (for IDEs).

`IERR`
: `IERR` is an integer flag which is always equal to zero on input. Subroutine `FEVAL` sets `IERR = -1` if `FEVAL` can not be evaluated for the current values of `T`, `Y` and `YPRIME`. Some solvers have the facility to attempt to prevent the occurrence of `IERR = -1`, or return to the driver in that case.

  `IERR` has an analogous meaning in subroutines `JEVAL` and `MEVAL`.

`RPAR` and `IPAR`
: `RPAR` and `IPAR` are double precision and integer arrays, respectively, which can be used for communication between the driver and the subroutines `FEVAL`, `JEVAL` and `MEVAL`. If `RPAR` and `IPAR` are not needed, then these parameters are ignored by treating them as dummy arguments.

  `RPAR` and `IPAR` have the same meaning in subroutines `JEVAL` and `MEVAL`.

IV.3.6 Subroutine `JEVAL`

This subroutine evaluates the derivative (or Jacobian) of the function f with respect to y.

```
      SUBROUTINE JEVAL(LDIM,NEQN,T,Y,YPRIME,DFDY,IERR,RPAR,IPAR)
      INTEGER LDIM,NEQN,IERR,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),YPRIME(NEQN),DFDY(LDIM,NEQN),RPAR(*)
C     INTENT(IN)    LDIM,NEQN,T,Y,YPRIME
C     INTENT(INOUT) RPAR,IPAR
C     INTENT(OUT)   DFDY,IERR
```

Meaning of the arguments:

`LDIM`
: The leading dimension of the array `DFDY`.

`NEQN`
: The dimension of the problem.

`T`
: The time point where the derivative is evaluated.

`Y(NEQN)`
: The value of y in which the derivative is evaluated.

`YPRIME(NEQN)`
: Only relevant for IDEs.

  - For IDEs, this is the value of y' in which the derivative $\partial f(t,y,y')/\partial y$ is evaluated.
  - For ODEs and DAEs, `YPRIME` is not referenced.

`DFDY(LDIM,NEQN)`
: The array with the resulting Jacobian matrix.

- If $\partial f/\partial y$ is a full matrix (`MLJAC` = `NEQN`), then `DFDY(I,J)` contains $\partial f_{\mathtt{I}}/\partial y_{\mathtt{J}}$.
- If $\partial f/\partial y$ is a band matrix ($0 \leq$ `MLJAC` $<$ `NEQN`), then `DFDY(I-J+MUJAC+1,J)` contains $\partial f_{\mathtt{I}}/\partial y_{\mathtt{J}}$ (LAPACK / LINPACK / BLAS storage).

`IERR`, `RPAR` and `IPAR`

See the description of subroutine `FEVAL`.

IV.3.7 Subroutine `MEVAL`

For ODEs, `MEVAL` is not called and a dummy subroutine is supplied. For DAEs, it supplies the constant matrix M. For IDEs, it evaluates the matrix $M := \partial f/\partial y'$.

```
      SUBROUTINE MEVAL(LDIM,NEQN,T,Y,YPRIME,DFDDY,IERR,RPAR,IPAR)
      INTEGER LDIM,NEQN,IERR,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),YPRIME(NEQN),DFDDY(LDIM,NEQN),RPAR(*)
C     INTENT(IN)    LDIM,NEQN,T,Y,YPRIME
C     INTENT(INOUT) RPAR,IPAR
C     INTENT(OUT)   DFDDY,IERR
```

Meaning of the arguments:

`LDIM`

The leading dimension of the matrix M.

`NEQN`

The dimension of the problem.

`T`

The time point where M is evaluated. (For DAEs, `T` is not referenced.)

`Y(NEQN)`

The value of y in which M is evaluated. (For DAEs, `Y` is not referenced.)

`YPRIME(NEQN)`

The value of y' in which M is evaluated. (For DAEs, `YPRIME` is not referenced.)

`DFDDY(LDIM,NEQN)`

This array contains the constant matrix M (for DAEs) or $M := \partial f/\partial y'$ (for IDEs).

- If M is a full matrix (`MLMAS` = `NEQN`), then `DFDDY(I,J)` contains $M_{\mathtt{I},\mathtt{J}}$ for DAEs and $\partial f_{\mathtt{I}}/\partial y'_{\mathtt{J}}$ for IDEs.
- If M is a band matrix ($0 \leq$ `MLMAS` $<$ `NEQN`), then `DFDDY(I-J+MUMAS+1,J)` contains $M_{\mathtt{I},\mathtt{J}}$ for DAEs and $\partial f_{\mathtt{I}}/\partial y'_{\mathtt{J}}$ for IDEs. (LAPACK / LINPACK / BLAS storage).

`IERR`, `RPAR` and `IPAR`

See the description of subroutine `FEVAL`.

IV.3.8 Subroutine `SOLUT`

This routine contains the reference solution.

```
      SUBROUTINE SOLUT(NEQN,T,Y)
      INTEGER NEQN
      DOUBLE PRECISION T,Y(NEQN)
C     INTENT(IN)  NEQN,T
C     INTENT(OUT) Y
```

Meaning of the arguments:

`NEQN`
: The dimension of the problem.

`T`
: The value of t, in which the reference solution is given (normally t_{end}).

`Y(NEQN)`
: This array contains the reference solution in $t = \texttt{T}$.

IV.4 Format of the solver codes

The following guidelines serve to write a solver that could be easily inserted in the test set.

- Write your own solver in a format similar to existing solvers in the file `own.f`.
- Put the linear algebra subroutines in a separate file `owna.f`.
- Write driver subroutines in the file `ownd.f`. If the format of your solver is similar to that of a solver that is already available in the test set, then this will only require minor modifications of the driver routines of that solver.
- Adjust the file `report.f` as indicated in the comment lines of this file. This will only be a minor modification.

References

[Mat] The Mathworks. Matlab. http://www.mathworks.com/.

[MM08] F. Mazzia and C. Magherini. *Test Set for Initial Value Problem Solvers, release 2.4*. Department of Mathematics, University of Bari and INdAM, Research Unit of Bari, February 2008. Available at http://www.dm.uniba.it/~testset.