# PSIDE Users' Guide

Walter M. Lioen
*CWI, PO Box 94079, 1090 GB Amsterdam, The Netherlands (Walter.Lioen@cwi.nl)*

Jacques J.B. de Swart
*CWI, PO Box 94079, 1090 GB Amsterdam, The Netherlands (Jacques.de.Swart@cwi.nl)* &
*Paragon Decision Technology, PO Box 3277, 2001 DG Haarlem, The Netherlands (jacques@paragon.nl)*

Wolter A. van der Veen
*formerly at CWI, PO Box 94079, 1090 GB Amsterdam, The Netherlands*
*MacNeal-Schwendler (E.D.C.) B.V., Groningenweg 6, 2803 PV Gouda, The Netherlands (wolter@macsch.com)*

edition December 15, 1998 for version 1.3

ABSTRACT
PSIDE – Parallel Software for Implicit Differential Equations – is a code for solving implicit differential equations on shared memory parallel computers. In this paper we describe the user interface.

*1991 Mathematics Subject Classification:* Primary: 65-04, Secondary: 65L05, 65Y05.
*1991 Computing Reviews Classification System:* G.1.7, G.4.
*Keywords and Phrases:* numerical software, parallel computers, IVP, IDE, ODE, DAE.
*Note:* The maintenance of PSIDE belongs to the project MAS2.2: 'Parallel Software for Implicit Differential Equations'.
*Acknowledgements:* This work is supported financially by the 'Technologiestichting STW' (Dutch Foundation for Technical Sciences), grants no. CWI.2703, CWI.4533. The use of supercomputer facilities was made possible by the 'Stichting Nationale Computerfaciliteiten' (National Computing Facilities Foundation, NCF), with financial support from the 'Nederlandse Organisatie voor Wetenschappelijk Onderzoek' (Netherlands Organization for Scientific Research, NWO).
*Note:* The latest version of PSIDE and this document can always be found at [SLV98a].

## 1. Introduction

PSIDE solves Implicit Differential Equations (IDEs) of the form

$$\begin{aligned} &g(t, y, y') = 0\,, \quad g, y \in \mathbb{R}^d, \\ &t_0 \le t \le t_{\text{end}}, \quad y(t_0) = y_0\,, \quad y'(t_0) = y'_0\,, \end{aligned} \tag{1.1}$$

were $y_0$ and $y'_0$ are such that $g(t_0, y_0, y'_0) = 0$ (for higher-index problems the initial values have to satisfy more conditions; see §4). It uses the four-stage Radau IIA method. The nonlinear systems are solved by a modified Newton process, in which every Newton iterate itself is computed by means of the Parallel Iterative Linear system Solver for Runge–Kutta (PILSRK) proposed in [HS97]. This process is constructed such that the four stage values can be computed simultaneously, thereby making PSIDE suitable for execution on four processors; see §7 for installation instructions. Full details about the algorithmic choices and the implementation of PSIDE can be found in [SLV98b].

## 2. Subroutine heading of PSIDE

PSIDE is a Fortran 77 routine, whose heading reads

```
      SUBROUTINE PSIDE(NEQN,Y,DY,GEVAL,
     +                 JNUM,NLJ,NUJ,JEVAL,
     +                 MNUM,NLM,NUM,MEVAL,
     +                 T,TEND,RTOL,ATOL,IND,
     +                 LRWORK,RWORK,LIWORK,IWORK,
     +                 RPAR,IPAR,IDID)
      INTEGER NEQN,NLJ,NUJ,NLM,NUM,IND(*),LRWORK,LIWORK,
     +                 IWORK(LIWORK),IPAR(*),IDID
      DOUBLE PRECISION Y(NEQN),DY(NEQN),T,TEND,RTOL(*),ATOL(*),
     +                 RWORK(LRWORK),RPAR(*)
      LOGICAL JNUM,MNUM
      EXTERNAL GEVAL,JEVAL,MEVAL
C
C     INTENT(IN)    NEQN,JNUM,NLJ,NUJ,MNUM,NLM,NUM,TEND,RTOL,ATOL,IND,
C    +              LRWORK,LIWORK
C     INTENT(INOUT) Y,DY,T,RWORK,IWORK,RPAR,IPAR
C     INTENT(OUT)   IDID
```

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

## 3. Arguments

`NEQN`
: On entry, this is the dimension $d$ of the IDE (1.1), the number of equations to be solved.

`Y(NEQN)`
: On entry, this array contains the initial value $y_0$.
On exit, `Y` contains $y($`T`$)$, the computed solution approximation at `T`.
(After successful return, `T = TEND`.)

`DY(NEQN)`
: On entry, this array contains the initial value $y'_0$.
On exit, `DY` contains $y'($`T`$)$, the computed derivative approximation at `T`.
(After successful return, `T = TEND`.)

`GEVAL`
: This is the subroutine which you provide to define the IDE

```
      SUBROUTINE GEVAL(NEQN,T,Y,DY,G,IERR,RPAR,IPAR)
      INTEGER NEQN,IERR,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),G(NEQN),RPAR(*)
C     INTENT(IN)    NEQN,T,Y,DY
C     INTENT(INOUT) IERR,RPAR,IPAR
C     INTENT(OUT)   G
```

For the given values of `T`, `Y`, and `DY` the subroutine should return the residual of the IDE

$$\texttt{G} = g(\texttt{T}, \texttt{Y}, \texttt{DY}).$$

You must declare the name `GEVAL` in an external statement in your program that calls PSIDE.

`IERR` is an integer flag which is always equal to zero on input. Subroutine `GEVAL` should set `IERR = -1` if `GEVAL` can not be evaluated for the current values of `Y` and `DY`. PSIDE will then try to prevent `IERR = -1` by using a smaller stepsize.

All other parameters have the same meaning as within subroutine PSIDE.

`JNUM`

To solve the IDE it is necessary to use the partial derivatives $J = \partial g / \partial y$. The solution will be more reliable if you provide $J$ via the subroutine `JEVAL`, in this case set `JNUM = .FALSE.`. If you do not provide a subroutine to evaluate $J$, provide a dummy `JEVAL`, set `JNUM = .TRUE.` and PSIDE will approximate $J$ by numerical differencing.

`NLJ` and `NUJ`

If $J$ is a full matrix, set `NLJ = NEQN`, otherwise set `NLJ` and `NUJ` equal to the lower bandwidth and upper bandwidth of $J$, respectively.

`JEVAL`

This is the subroutine which you provide to define $J$ (if `JNUM .EQ. .FALSE.`)

```
      SUBROUTINE JEVAL(LDJ,NEQN,NLJ,NUJ,T,Y,DY,DGDY,RPAR,IPAR)
      INTEGER LDJ,NEQN,NLJ,NUJ,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),DGDY(LDJ,NEQN),RPAR(*)
C     INTENT(IN)    LDJ,NEQN,NLJ,NUJ,T,Y,DY
C     INTENT(INOUT) RPAR,IPAR
C     INTENT(OUT)   DGDY
```

For the given values of `T`, `Y`, and `DY` the subroutine should return the partial derivatives, such that

- `DGDY(I,J)` contains $\partial g_{\mathtt{I}}(\mathtt{T},\mathtt{Y},\mathtt{DY})/\partial y_{\mathtt{J}}$ if $J$ is a full matrix ($\mathtt{NLJ} = \mathtt{NEQN}$);
- `DGDY(I-J+NUJ+1,J)` contains $\partial g_{\mathtt{I}}(\mathtt{T},\mathtt{Y},\mathtt{DY})/\partial y_{\mathtt{J}}$ if $J$ is a band matrix ($0 \leq \mathtt{NLJ} < \mathtt{NEQN}$) (LAPACK / LINPACK / BLAS storage).

You must declare the name `JEVAL` in an external statement in your program that calls PSIDE.

`LDJ` denotes the leading dimension of $J$.

All other parameters have the same meaning as within subroutine PSIDE.

`MNUM`

To solve the IDE it is necessary to use the partial derivatives $M = \partial g / \partial y'$. The solution will be more reliable if you provide $M$ via `MEVAL`, in this case set `MNUM = .FALSE.`. If you do not provide a subroutine to evaluate $M$, provide a dummy `MEVAL`, set `MNUM = .TRUE.` and PSIDE will approximate $M$ by numerical differencing.

`NLM` and `NUM`

If $M$ is a full matrix, set `NLM = NEQN`, otherwise set `NLM` and `NUM` equal to the lower bandwidth and upper bandwidth of $M$, respectively. It is supposed that `NLM .LE. NLJ` and `NUM .LE. NUJ`.

`MEVAL`

This is the subroutine which you provide to define $M$ (if `MNUM .EQ. .FALSE.`)

```
      SUBROUTINE MEVAL(LDM,NEQN,NLM,NUM,T,Y,DY,DGDDY,RPAR,IPAR)
      INTEGER LDM,NEQN,NLM,NUM,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),DGDDY(LDM,NEQN),RPAR(*)
C     INTENT(IN)    LDM,NEQN,NLM,NUM,T,Y,DY
C     INTENT(INOUT) RPAR,IPAR
C     INTENT(OUT)   DGDDY
```

For the given values of `T`, `Y`, and `DY` the subroutine should return the partial derivatives, such that

- `DGDDY(I,J)` contains $\partial g_{\mathtt{I}}(\mathtt{T},\mathtt{Y},\mathtt{DY})/\partial y'_{\mathtt{J}}$ if $M$ is a full matrix ($\mathtt{NLM} = \mathtt{NEQN}$);
- `DGDDY(I-J+NUM+1,J)` contains $\partial g_{\mathtt{I}}(\mathtt{T},\mathtt{Y},\mathtt{DY})/\partial y'_{\mathtt{J}}$ if $M$ is a band matrix ($0 \le \mathtt{NLM} < \mathtt{NEQN}$) (LAPACK / LINPACK / BLAS storage).

You must declare the name `MEVAL` in an external statement in your program that calls PSIDE.

`LDM` denotes the leading dimension of $M$.

All other parameters have the same meaning as within subroutine PSIDE.

`T`

On entry, `T` must specify $t_0$, the initial value of the independent variable.
On successful exit (`IDID .EQ. 1`), `T` contains `TEND`.
On an error return, `T` is the point reached.

`TEND`

On entry, `TEND` must specify the value of the independent variable at which the solution is desired.

`RTOL` and `ATOL`

You must assign relative `RTOL` and absolute `ATOL` error tolerances to tell the code how small you want the local errors to be. You have two choices

- both `RTOL` and `ATOL` are scalars (set `IWORK(1) = 0`): the code keeps, roughly, the local error of `Y(I)` below `RTOL*ABS(Y(I))+ATOL`;
- both `RTOL` and `ATOL` are vectors (set `IWORK(1) = 1`): the code keeps the local error of `Y(I)` below `RTOL(I)*ABS(Y(I))+ATOL(I)`.

In either case all components must be non-negative.

`IND`

If `IWORK(2) .EQ. 1` , then `IND` should be declared of length `NEQN` and `IND(I)` must specify the index of variable `I`. If `IWORK(2) .EQ. 0` , then `IND` is not referenced and the problem is assumed to be of index 1.
See §4 for information how to determine the index of variables of certain problem classes.

`LRWORK`

On entry `LRWORK` must specify the length of the `RWORK` array. You must have for the full partial derivatives case (when $\mathtt{NLJ} = \mathtt{NEQN}$)

```
LRWORK .GE. 20 + 27*NEQN + 6*NEQN**2,
```

for the case where $M$ is banded and $J$ is full (when $\mathtt{NLJ} = \mathtt{NEQN}$ and $\mathtt{NLM} < \mathtt{NEQN}$)

```
LRWORK .GE. 20 + (27 + NLM+NUM+1 + 5*NEQN)*NEQN,
```

and for the case where both partial derivatives are banded (when `NLJ` < `NEQN`)

```
LRWORK .GE. 20 + (27 + NLJ+NUJ+NLM+NUM+2 + 4*(2*NLJ+NUJ+1))*NEQN.
```

`RWORK`

Real work array of length `LRWORK`. `RWORK(1)`, ... ,`RWORK(20)` serve as parameters for the code. For standard use, set `RWORK(1)`, ... ,`RWORK(20)` to zero before calling.

On entry:

– if `RWORK(1) .GT. 0D0` then PSIDE will use `RWORK(1)` as initial stepsize instead of determining it internally.

On exit:

– `RWORK(1)` contains the stepsize used on the last successful step.

`LIWORK`

On entry `LIWORK` must specify the length of the `IWORK` array. You must have

```
LIWORK .GE. 20 + 4*NEQN.
```

`IWORK`

Integer work array of length `LIWORK`. `IWORK(1)`, ... ,`IWORK(20)` serve as parameters for the code. For standard use, set `IWORK(1)`, ... ,`IWORK(20)` to zero before calling.

On entry:

– if `IWORK(1) .EQ. 1` then `RTOL` and `ATOL` are vectors instead of scalars,
– if `IWORK(2) .EQ. 1` then `IND` is a vector,
– set `IWORK`(10) = 0 if PSIDE is called for the first time; for subsequent calls of PSIDE do not reinitialize the parameters `IWORK(10)`, ... ,`IWORK(19)` to zero.

On exit:

– `IWORK(10)` contains the number of successive PSIDE calls,
– `IWORK(11)` contains the number of $g$ evaluations,
– `IWORK(12)` contains the number of $J$ and $M$ evaluations ($J$ and $M$ are computed in tandem and count as 1),
– `IWORK(13)` contains the number of LU-decompositions.
– `IWORK(14)` contains the number of forward/backward solves,
– `IWORK(15)` contains the total number of steps (including rejected steps),
– `IWORK(16)` contains the number of rejected steps due to error control,
– `IWORK(17)` contains the number of rejected steps due to Newton failure,
– `IWORK(18)` contains the number of rejected steps due to excessive growth of the solution,
– `IWORK(19)` contains the number of rejected steps due to `IERR .EQ. -1` return of `GEVAL`.

The integration characteristics in `IWORK(11)`, ... ,`IWORK(14)` refer to an implementation on a one-processor computer. When implemented on a parallel computer with four processors, one may divide these numbers by four to obtain the number of sequential evaluations, decompositions and solves.

`RPAR` and `IPAR`

`RPAR` and `IPAR` are double precision and integer arrays which you can use for communication between your calling program and the subroutines `GEVAL`, and/or `JEVAL`, `MEVAL`. They are not altered by PSIDE. If you do not need `RPAR` and `IPAR`, ignore these parameters by treating them as dummy arguments. If you choose to use them, dimension them in `GEVAL` and/or `JEVAL`, `MEVAL` as arrays of appropriate length. Because of the parallel implementation of PSIDE, `GEVAL` must not alter `RPAR` and `IPAR` to prevent concurrent updating. `JEVAL` and `MEVAL` may alter them.

`IDID`

On exit:

- if `IDID .EQ. 1` then the integration was successful,
- if `IDID .EQ. -1` then PSIDE could not reach `TEND` because the stepsize became too small,
- if `IDID .EQ. -2` then something else went wrong. For example this happens when the input was invalid. An error message will be printed.

## 4. Index determination

As mentioned before, it is important for higher-index problems to set the index of the variables in the vector `IND`. In this section we specify for certain problem classes, which can easily be written in the form (1.1), how this should be done. The results were taken from [HLR89]. For higher-index problems in these classes we also list the additional conditions that have to be fulfilled by the initial values. We refer to [SLV98b] for information on how PSIDE uses `IND`. If $\phi$ is a function of $q$, then we will denote the (partial) derivative of $\phi$ with respect to $q$ by $\phi_q$.

### 4.1 ODEs

First of all, Ordinary Differential Equations (ODEs), which are of the form

$$\begin{array}{ll} y' = f(t,y)\,, & y, f \in \mathbb{R}^d, \\ t_0 \le t \le t_{\text{end}}, & y(t_0) = y_0\,, \end{array}$$

are of index 1, i.e. we can set `IWORK(2)` $= 0$.

### 4.2 DAEs of index 1

The class of Differential–Algebraic Equations (DAEs) takes the form

$$\begin{array}{lclll} y' & = & f(t,y,z)\,, & y, f \in \mathbb{R}^{d_1}, & \\ 0 & = & g(t,y,z)\,, & z, g \in \mathbb{R}^{d_2}, & \\ \multicolumn{3}{l}{t_0 \le t \le t_{\text{end}},} & y(t_0) = y_0\,, & z(t_0) = z_0\,, \end{array} \tag{4.1}$$

where $y_0$ and $z_0$ are such that $g(t_0, y_0, z_0) = 0$. If $g_z$ is invertible in the neighborhood of the solution, then (4.1) is of index 1 and `IWORK(2)` $= 0$ is the right setting.

### 4.3 IDEs with invertible mass matrix

Also of index 1 are problems of the form

$$\begin{array}{ll} M(y)y' = f(t,y)\,, & y, f \in \mathbb{R}^d, \\ t_0 \le t \le t_{\text{end}}, & y(t_0) = y_0\,, \end{array}$$

where $M(y)$ (often called the *mass matrix*) is invertible in the neighborhood of the solution. Again, set `IWORK(2)` $= 0$.

### 4.4 DAEs of index 2

An often arising subclass of (4.1) where $g_y$ is *not* invertible is

$$\begin{array}{lll} y' &=& f(t,y,z)\,, \quad y,f \in \mathbb{R}^{d_1}, \\ 0 &=& g(t,y)\,, \quad\;\; z,g \in \mathbb{R}^{d_2}, \\ t_0 \le t \le t_{\mathrm{end}}, && y(t_0)=y_0\,, \quad z(t_0)=z_0\,, \end{array} \tag{4.2}$$

where $y_0$ and $z_0$ are such that $g(t_0,y_0)=0$ and $g_y(t_0,y_0)f(t_0,y_0,z_0)=0$. If $g_y f_z$ is invertible in the neighborhood of the solution, then (4.2) is of index 2. The variables $y$ and $z$ are of index 1 and 2, respectively, so set `IND(I)` $=1$ if `I` corresponds to a $y$-component, and `IND(I)` $=2$ if `I` corresponds to a $z$-component.

### 4.5 IDEs of index 3

If the problem is of the form

$$\begin{array}{lll} y' &=& f(t,y,z)\,, \qquad\quad y,f \in \mathbb{R}^{d_1}, \\ z' &=& k(t,y,z,u)\,, \quad z,k \in \mathbb{R}^{d_2}, \\ 0 &=& g(t,y)\,, \qquad\quad\; u,g \in \mathbb{R}^{d_3}, \\ t_0 \le t \le t_{\mathrm{end}}, && y(t_0)=y_0\,, \quad z(t_0)=z_0\,, \quad u(t_0)=u_0\,, \end{array} \tag{4.3}$$

where $g_y f_z k_u$ is invertible in the neighborhood of the solution and $y_0$, $z_0$ and $u_0$ satisfy the conditions

$$\begin{aligned} & g(t_0,y_0)=0\,, \\ & g_y(t_0,y_0)f(t_0,y_0,z_0)=0\,, \\ & g_{yy}(t_0,y_0)(f(t_0,y_0,z_0),f(t_0,y_0,z_0))+ \\ & \qquad g_y(t_0,y_0)(f_y(t_0,y_0,z_0)f(t_0,y_0,z_0)+f_z(t_0,y_0,z_0)k(t_0,y_0,z_0))=0\,, \end{aligned}$$

then (4.3) is an IDE of index 3. The variables $y$, $z$ and $u$ are of index 1, 2 and 3, respectively, so set `IND(I)` $=1$ if `I` corresponds to a $y$-component, `IND(I)` $=2$ if `I` corresponds to a $z$-component, and `IND(I)` $=3$ if `I` corresponds to a $u$-component.

### 4.6 Multibody systems of index 3

In mechanics one often encounters the problem

$$\begin{array}{lll} q' &=& u\,, \qquad\qquad\qquad\quad q,u \in \mathbb{R}^{d_1}, \\ M(q)u' &=& f(t,q,u)+G^{\mathrm{T}}(q)\lambda\,, \quad f \in \mathbb{R}^{d_2}, \\ 0 &=& g(t,q)\,, \qquad\qquad\quad \lambda,g \in \mathbb{R}^{d_3}, \\ t_0 \le t \le t_{\mathrm{end}}, && q(t_0)=q_0\,,\; u(t_0)=u_0\,,\; \lambda(t_0)=\lambda_0\,, \end{array} \tag{4.4}$$

where $G(q)=g_q$, the matrix $M(q)$ non-singular in the neighborhood of the solution and $q_0$, $u_0$ and $\lambda_0$ are such that they satisfy

$$\begin{aligned} & g(t_0,q_0)=0\,, \\ & G(t_0,q_0)u_0=0\,, \\ & g_{qq}(t_0,q_0)(u_0,u_0)+G(t_0,q_0)M^{-1}(q_0)(f(t_0,q_0,u_0)+G^{\mathrm{T}}(q_0)\lambda_0)=0\,. \end{aligned}$$

We could rewrite the system to the form (4.3) by premultiplying both sides of the $u'$-equation by $M^{-1}(q)$. Consequently, (4.4) is of index 3 and the variables $q$, $u$ and $\lambda$ are of index 1, 2 and 3, respectively, so set `IND(I)` $=1$ if `I` corresponds to a $q$-component, `IND(I)` $=2$ if `I` corresponds to a $u$-component, and `IND(I)` $=3$ if `I` corresponds to a $\lambda$-component.

## 5. Examples

Machine readable versions of the following two example drivers are available from [SLV98a].

### 5.1 Van der Pol problem

Here we give a simple example, solving the Van der Pol problem, an ODE of dimension 2.

*5.1.1 Driver for Van der Pol problem*

```
      PROGRAM VDPOL
C
C PSIDE example: Van der Pol problem
C
C  - ODE of dimension 2                               y' = f
C  - formulated as general IDE                        g = f - y' = 0
C  - analytical partial derivative J (full 2x2 matrix)  dg/dy = df/dy
C  - analytical partial derivative M (band matrix)      dg/dy' = -I
C
      INTEGER NEQN,NLJ,NUJ,NLM,NUM
      LOGICAL JNUM,MNUM
      PARAMETER (NEQN=2,NLJ=NEQN,NUJ=NEQN,NLM=0,NUM=0)
      PARAMETER (JNUM=.FALSE., MNUM=.FALSE.)
      INTEGER LRWORK, LIWORK
      PARAMETER (LRWORK = 20+27*NEQN+6*NEQN**2, LIWORK = 20+4*NEQN)

      INTEGER IND,IWORK(LIWORK),IPAR,IDID
      DOUBLE PRECISION Y(NEQN),DY(NEQN),T,TEND,RTOL,ATOL,
     +                 RWORK(LRWORK),RPAR

      EXTERNAL VDPOLG,VDPOLJ,VDPOLM

      INTEGER I

C initialize PSIDE

      DO 10 I=1,20
         IWORK(I) = 0
         RWORK(I) = 0D0
   10 CONTINUE

C consistent initial values

      T     =  0D0
      Y(1)  =  2D0
      Y(2)  =  0D0
      DY(1) =  0D0
      DY(2) = -2D0

      TEND  = 41.5D0

C set scalar tolerances

      RTOL = 1D-4
      ATOL = 1D-4

      WRITE(*,'(1X,A,/)') 'PSIDE example solving Van der Pol problem'

      CALL PSIDE(NEQN,Y,DY,VDPOLG,
     +           JNUM,NLJ,NUJ,VDPOLJ,
     +           MNUM,NLM,NUM,VDPOLM,
     +           T,TEND,RTOL,ATOL,IND,
     +           LRWORK,RWORK,LIWORK,IWORK,
     +           RPAR,IPAR,IDID)

      IF (IDID.EQ.1) THEN
         WRITE(*,'(1X,A,F5.1)') 'solution at t = ', TEND
         WRITE(*,*)
         DO 20 I=1,NEQN
            WRITE(*,'(4X,''y('',I1,'') ='',E11.3)') I,Y(I)
   20    CONTINUE
         WRITE(*,*)
         WRITE(*,'(1X,A,I4)') 'number of steps =', IWORK(15)
         WRITE(*,'(1X,A,I4)') 'number of f-s   =', IWORK(11)
         WRITE(*,'(1X,A,I4)') 'number of J-s   =', IWORK(12)
         WRITE(*,'(1X,A,I4)') 'number of LU-s  =', IWORK(13)
      ELSE
         WRITE(*,'(1X,A,I4)') 'PSIDE failed: IDID =', IDID
      ENDIF

      END

      SUBROUTINE VDPOLG(NEQN,T,Y,DY,G,IERR,RPAR,IPAR)
      INTEGER NEQN,IERR,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),G(NEQN),RPAR(*)
      G(1) = Y(2)-DY(1)
      G(2) = 500D0*(1D0-Y(1)*Y(1))*Y(2)-Y(1)-DY(2)
      RETURN
      END

      SUBROUTINE VDPOLJ(LDJ,NEQN,NLJ,NUJ,T,Y,DY,DGDY,RPAR,IPAR)
      INTEGER LDJ,NEQN,NLJ,NUJ,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),DGDY(LDJ,NEQN),RPAR(*)
      DGDY(1,1) = 0D0
      DGDY(1,2) = 1D0
      DGDY(2,1) = -1000D0*Y(1)*Y(2)-1D0
      DGDY(2,2) = 500D0*(1D0-Y(1)*Y(1))
      RETURN
      END

      SUBROUTINE VDPOLM(LDM,NEQN,NLM,NUM,T,Y,DY,DGDDY,RPAR,IPAR)
      INTEGER LDM,NEQN,NLM,NUM,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),DGDDY(LDM,NEQN),RPAR(*)
      DGDDY(1,1) = -1D0
      DGDDY(1,2) = -1D0
      RETURN
      END
```

*5.1.2 Output for Van der Pol problem*

This is the output of the example given in the previous subsection.

```
 PSIDE example solving Van der Pol problem

 solution at t =  41.5

    y(1) =  0.194E+01
    y(2) = -0.140E-02

 number of steps =  22
 number of f-s   = 214
 number of J-s   =   2
 number of LU-s  =  88
```

### 5.2 Pendulum problem

Here we give a simple example, solving the Pendulum problem, an IDE of dimension 5 and index 3.

*5.2.1 Driver for Pendulum problem*

```
      PROGRAM PENDUL
C
C PSIDE example: Pendulum problem
C
C  - IDE of dimension 5
C
C                                p' = q
C                              M q' = f - G^T lambda
C                                0  = g(p)
C
C  - index of p = 1; index of q = 2; index of lambda = 3
C  - formulated as general IDE:  g(t,y,y') = 0
C  - analytical partial derivative J (full 5x5 matrix)
C  - analytical partial derivative M (band matrix)
C
      INTEGER NEQN,NLJ,NUJ,NLM,NUM
      LOGICAL JNUM,MNUM
      PARAMETER (NEQN=5,NLJ=NEQN,NUJ=NEQN,NLM=0,NUM=0)
      PARAMETER (JNUM=.TRUE., MNUM=.TRUE.)
      INTEGER LRWORK, LIWORK
      PARAMETER (LRWORK = 20 + (27 + NLM+NUM+1 + 5*NEQN)*NEQN,
     +           LIWORK = 20 + 4*NEQN)

      INTEGER IND(NEQN),IWORK(LIWORK),IPAR,IDID
      DOUBLE PRECISION Y(NEQN),DY(NEQN),T,TEND,RTOL,ATOL,
     +                 RWORK(LRWORK),RPAR

      DOUBLE PRECISION GRAV,MASS,LEN
      PARAMETER(GRAV=1D0,MASS=1D0,LEN=1D0)

      EXTERNAL PENG,PENJ,PENM

      INTEGER I

C initialize PSIDE

      DO 10 I=1,20
         IWORK(I) = 0
         RWORK(I) = 0D0
   10 CONTINUE

C consistent initial values

      DO 20 I=1,NEQN
         Y(I)  = 0D0
         DY(I) = 0D0
   20 CONTINUE
      T      =  0D0
      Y(1)   =  LEN
      DY(4)  = -GRAV/MASS

      TEND   = 10D0

C set index of variables

      IWORK(2) = 1
      IND(1) = 1
      IND(2) = 1
      IND(3) = 2
      IND(4) = 2
      IND(5) = 3

C set scalar tolerances

      RTOL = 1D-4
      ATOL = 1D-4

      WRITE(*,'(1X,A,/)') 'PSIDE example solving Pendulum problem'

      CALL PSIDE(NEQN,Y,DY,PENG,
     +           JNUM,NLJ,NUJ,PENJ,
     +           MNUM,NLM,NUM,PENM,
     +           T,TEND,RTOL,ATOL,IND,
     +           LRWORK,RWORK,LIWORK,IWORK,
     +           RPAR,IPAR,IDID)

      IF (IDID.EQ.1) THEN
         WRITE(*,'(1X,A,F5.1)') 'solution at t = ', TEND
         WRITE(*,*)
         DO 30 I=1,NEQN
            WRITE(*,'(4X,''y('',I1,'') ='',E11.3)') I,Y(i)
   30    CONTINUE
         WRITE(*,*)
         WRITE(*,'(1X,A,I5)') 'number of steps =', IWORK(15)
         WRITE(*,'(1X,A,I5)') 'number of f-s   =', IWORK(11)
         WRITE(*,'(1X,A,I5)') 'number of J-s   =', IWORK(12)
         WRITE(*,'(1X,A,I5)') 'number of LU-s  =', IWORK(13)
      ELSE
         WRITE(*,'(1X,A,I4)') 'PSIDE failed: IDID =', IDID
      ENDIF

      END

      SUBROUTINE PENG(NEQN,T,Y,DY,G,IERR,RPAR,IPAR)
      INTEGER NEQN,IERR,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),G(NEQN),RPAR(*)
      DOUBLE PRECISION GRAV,MASS,LEN
      PARAMETER(GRAV=1D0,MASS=1D0,LEN=1D0)
      G(1) = DY(1)-Y(3)
      G(2) = DY(2)-Y(4)
      G(3) = MASS*DY(3)+Y(1)*Y(5)
      G(4) = MASS*DY(4)+Y(2)*Y(5)+GRAV
      G(5) = Y(1)*Y(1)+Y(2)*Y(2)-LEN*LEN
      RETURN
      END

      SUBROUTINE PENJ(LDJ,NEQN,NLJ,NUJ,T,Y,DY,DGDY,RPAR,IPAR)
      INTEGER LDJ,NEQN,NLJ,NUJ,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),DGDY(LDJ,NEQN),RPAR(*)
      INTEGER I,J
      DO 20 J=1,NEQN
         DO 10 I=1,NEQN
            DGDY(I,J) = 0D0
   10    CONTINUE
   20 CONTINUE
      DGDY(1,3) = -1D0
      DGDY(2,4) = -1D0
      DGDY(3,1) = Y(5)
      DGDY(3,5) = Y(1)
      DGDY(4,2) = Y(5)
      DGDY(4,5) = Y(2)
      DGDY(5,1) = 2D0*Y(1)
      DGDY(5,2) = 2D0*Y(2)
      RETURN
      END

      SUBROUTINE PENM(LDM,NEQN,NLM,NUM,T,Y,DY,DGDDY,RPAR,IPAR)
      INTEGER LDM,NEQN,NLM,NUM,IPAR(*)
      DOUBLE PRECISION T,Y(NEQN),DY(NEQN),DGDDY(LDM,NEQN),RPAR(*)
      DOUBLE PRECISION GRAV,MASS,LEN
      PARAMETER(GRAV=1D0,MASS=1D0,LEN=1D0)
      DGDDY(1,1) = 1D0
      DGDDY(1,2) = 1D0
      DGDDY(1,3) = MASS
      DGDDY(1,4) = MASS
      DGDDY(1,5) = 0D0
      RETURN
      END
```

*5.2.2 Output for Pendulum problem*

This is the output of the example given in the previous subsection.

```
 PSIDE example solving Pendulum problem

 solution at t =  10.0

    y(1) = -0.812E+00
    y(2) = -0.584E+00
    y(3) = -0.631E+00
    y(4) =  0.877E+00
    y(5) =  0.175E+01

 number of steps =  142
 number of f-s   = 2880
 number of J-s   =   82
 number of LU-s  =  564
```

## 6. PSIDE and the ‘Test set for Initial Value Problem solvers’

A more involved driver for PSIDE is supplied with the ‘Test set for Initial Value Problem solvers’, which is available via the World Wide Web [LS98]. This test platform not only contains the Fortran 77 routines for many test problems, but also drivers for the solvers DASSL [Pet91], MEBDFDAE [Cas98], RADAU [HW98], RADAU5 [HW96], VODE [BHB97], and PSIDE. This means that if one wants to solve a particular set of differential equations with PSIDE, it suffices to write the code that defines the problem in the test set format and link it with PSIDE and the test set driver. An additional benefit is that it is easy to compare solvers mutually. To give an impression of the performance of PSIDE in relation to that of the other solvers, we give in Figure 1–2 work-precision diagrams for two problems from [LS98]. They correspond to the Medical Akzo Nobel problem, a set of semi-discretized partial differential equations of dimension 400 which describe the injection of a medicine in a tumorous tissue, and the NAND gate, a set of 14 implicit differential equations of index 1 which model a electrical circuit performing the logical `NOT(AND)` operation. To produce these diagrams, we used for every solver a range of input tolerances, measured the accuracy delivered by the solver in number of correct digits and plotted these numbers against the CPU times needed for the runs on a logarithmic scale. The PSIDE-1 curves correspond to timings on a one-processor machine, the PSIDE-4 curves were obtained by dividing the one-processor timings by the speed-up factors on four processors. For an explanation how these factors were obtained, we refer to [LS98]. Results of MEBDFDAE, RADAU, RADAU5 and VODE are not included in Figure 2, because these solvers can not handle implicit differential equations directly. From Figure 1 we see that for the Medical Akzo Nobel problem PSIDE on one processor is about as efficient as DASSL, MEBDFDAE and VODE and less efficient than RADAU5, whereas PSIDE using four processors is the most efficient solver. Figure 2 reveals that for the NAND gate

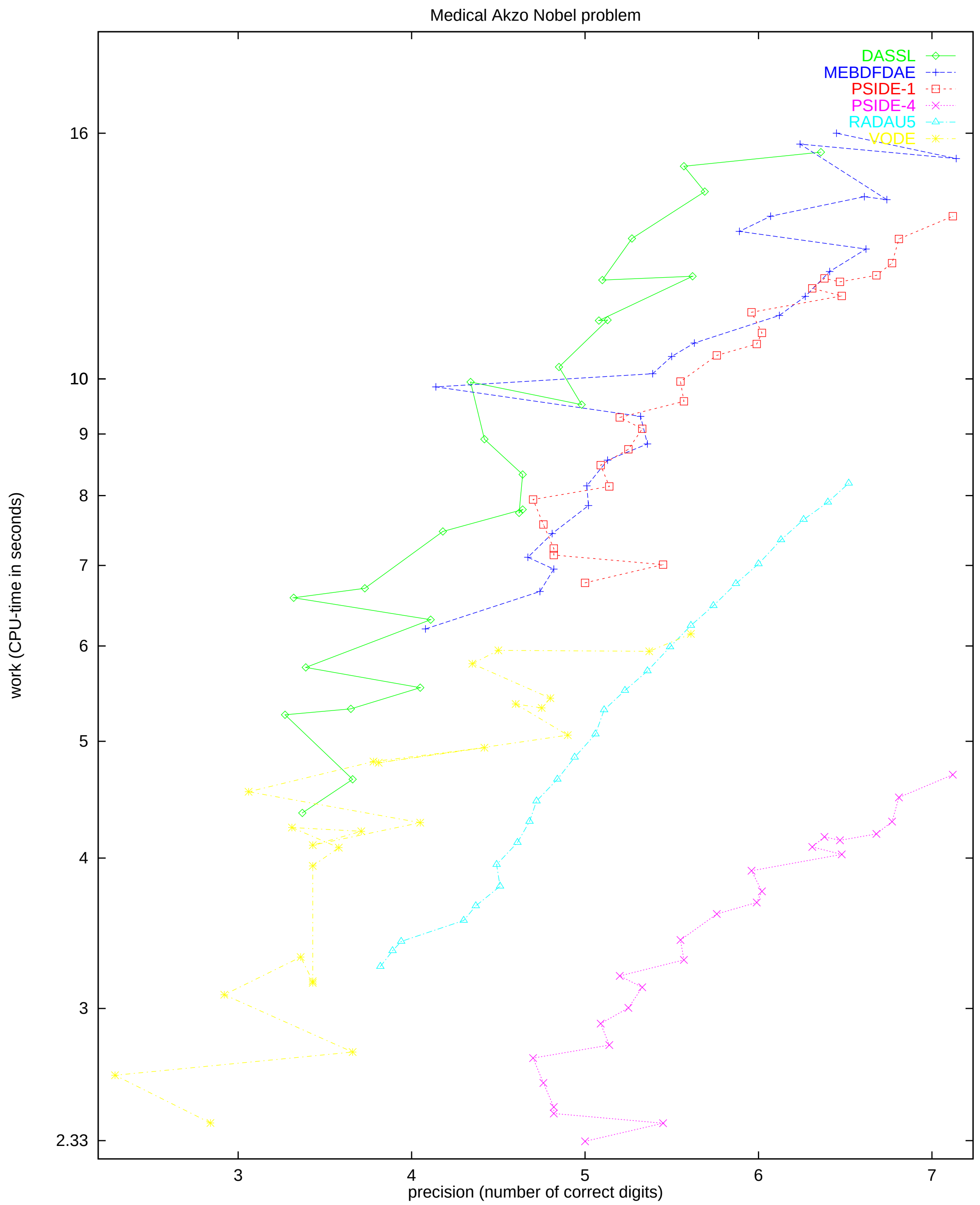

FIGURE 1: *Work-precision diagram for Medical Akzo Nobel problem*

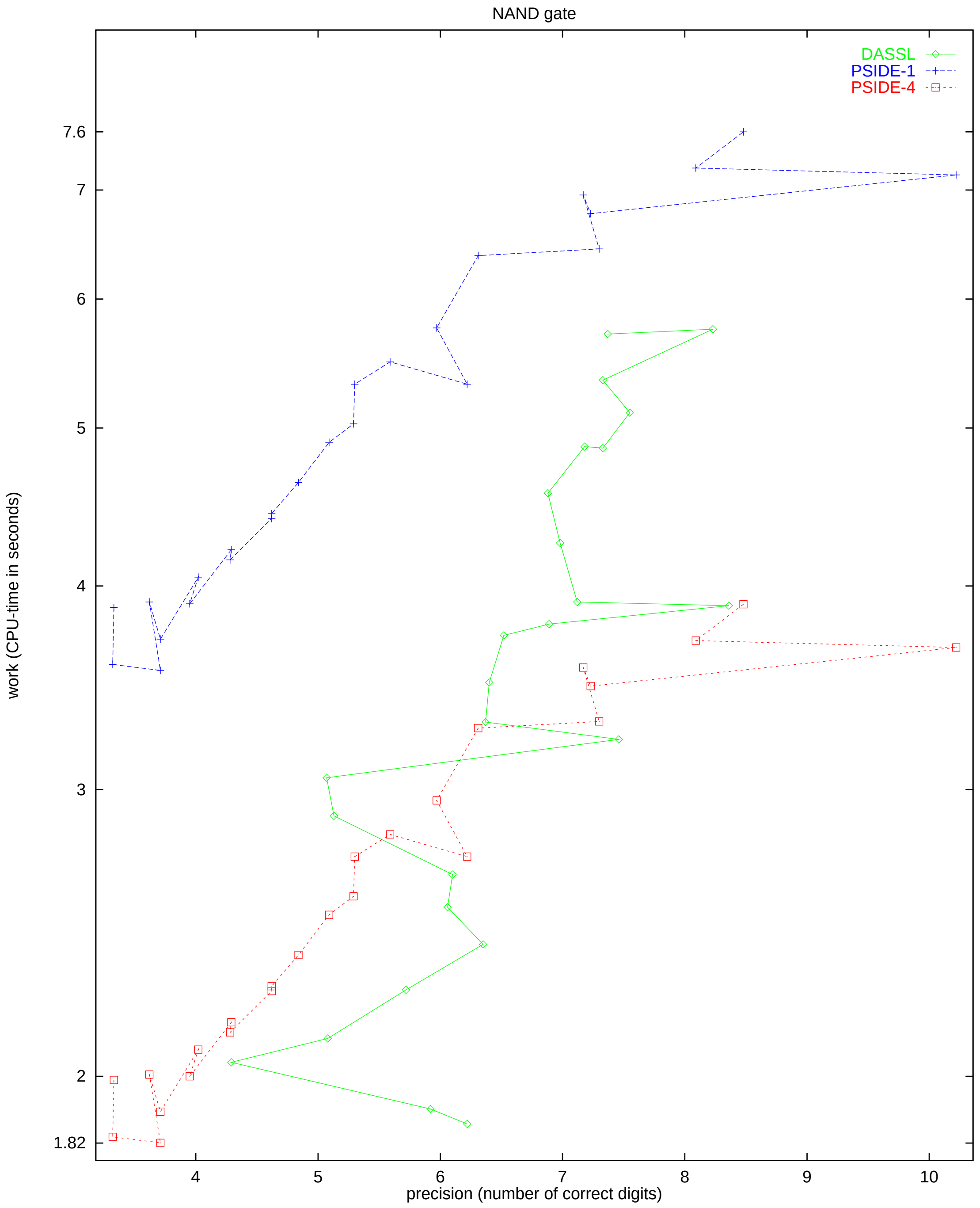

FIGURE 2: *Work-precision diagram for NAND gate*

DASSL performs better than PSIDE in one-processor mode, but for higher tolerances worse if four processors are used. These figures are quite representative for the numerous comparisons in [LS98], which show that the speed-up factor of PSIDE with respect tot the other solvers is between 1.14 and 3.28, depending on problem and solver.

Integration characteristics, complete descriptions of the test problems and the test set format, full details about the work-precision diagrams, as well as comparisons for other test problems, can be found in [LS98].

## 7. Installing PSIDE

### 7.1 Parallelism

The Fortran 77 source of PSIDE [SLV98a] contains Cray autotasking directives [Cra94]. Because of the high-level parallelism – the four stage values can be computed in parallel (cf. [SLV98b, Section 2]) – we explicitly autotask the loops that should be computed in parallel. If you are not working on a Cray, it should be almost trivial to change the given autotasking directives to facilitate your compiler. The autotasking directives appear in the source code as `CMIC$ DOALL PRIVATE(VAR1,...) SHARED(VAR1,...)`, beginning at column 1. The `DOALL` directive indicates that the `DO` loop that begins on the next line may be executed in parallel. `PRIVATE(...)` specifies that each processor will have its own private copy of these variables. `SHARED(...)` identifies those variables that are shared between processors.

### 7.2 Linear algebra

For PSIDE's linear algebra we chose to use LAPACK. However, if you do *not* have available on your system both

- a machine tuned LAPACK [ABB+95], and
- a machine optimized BLAS

we suggest you use the linear algebra routines available at [SLV98a]. Of course the latter will work, however, it may give much worse performance.

## References

[ABB+95] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide, Second Edition*. SIAM, Philadelphia, 1995.

[BHB97] Peter N. Brown, Alan C. Hindmarsh, and George D. Byrne. *VODE: A variable coefficient ODE solver*, May 15, 1997. Bug fix release November 12, 1998. Available at `http://www.netlib.org/ode/vode.f`.

[Cas98] J. Cash. *MEBDFDAE*, November 6, 1998. Available at `http://www.ma.ic.ac.uk/~jcash/IVP_software/finaldae/readme.html`.

[Cra94] Cray Research, Inc. *CF77 Commands and Directives*, SR-3771 6.0 edition, 1994.

[HLR89] E. Hairer, C. Lubich, and M. Roche. *The Numerical Solution of Differential-Algebraic Systems by Runge–Kutta Methods*. Lecture Notes in Mathematics 1409. Springer-Verlag, 1989.

[HS97] P.J. van der Houwen and J.J.B. de Swart. Parallel linear system solvers for Runge–Kutta methods. *Advances in Computational Mathematics*, 7:157–181, 1997.

[HW96] E. Hairer and G. Wanner. *RADAU5*, July 9, 1996. Available at `ftp://ftp.unige.ch/pub/doc/math/stiff/radau5.f`.

[HW98] E. Hairer and G. Wanner. *RADAU*, September 18, 1998. Available at `ftp://ftp.unige.ch/pub/doc/math/stiff/radau.f`.

[LS98] W.M. Lioen and J.J.B. de Swart. *Test Set for Initial Value Problem Solvers*, December 1998. Available at `http://www.cwi.nl/cwi/projects/IVPtestset/`.

[Pet91] L.R. Petzold. *DASSL: A Differential/Algebraic System Solver*, June 24, 1991. Available at `http://www.netlib.org/ode/ddassl.f`.

[SLV98a] J.J.B. de Swart, W.M. Lioen, and W.A. van der Veen. *PSIDE*, November 25, 1998. Available at `http://www.cwi.nl/cwi/projects/PSIDE/`.

[SLV98b] J.J.B. de Swart, W.M. Lioen, and W.A. van der Veen. *Specification of PSIDE.* CWI, 1998. Available at `http://www.cwi.nl/cwi/projects/PSIDE/`.