

Users' Guide for the code RADAR5

Nicola Guglielmi *

*Dip. di Matematica Pura e Applicata, Università dell'Aquila,
via Vetoio (Coppito), I-67010 L'Aquila, Italy,
e-mail: guglielm@univaq.it*

and

Ernst Hairer

*Section de Mathématiques, Université de Genève,
CH-1211 Genève 24, Switzerland,
e-mail: Ernst.Hairer@math.unige.ch*

October 2000

©2000 N. Guglielmi, E. Hairer

Technical Report

1 The problem

We consider initial value problems for delay differential equations

$$\begin{aligned} M y'(t) &= f(t, y(t), y(\alpha_1(t, y(t))), \dots, y(\alpha_m(t, y(t))))), \\ y(t_0) &= y_0, \quad y(t) = g(t) \quad \text{for } t < t_0, \end{aligned} \tag{1}$$

where M is a constant $d \times d$ matrix and $\alpha_i(t, y(t)) \leq t$ for all $t \geq t_0$ and for all i . The value $g(t_0)$ may be different from y_0 , allowing for a discontinuity at t_0 .

The presence of the matrix M in the problem formulation has several reasons. If a partial delay differential equation is discretized in space by finite elements, we obtain an equation of the form (1) where M is the mass matrix. A multiplication of the equation by M^{-1} would destroy the sparsity pattern of the problem and is not recommended.

Since we allow the matrix M to be singular, the above formulation includes all kinds of differential-algebraic delay equations. For $M = \text{diag}(I, \varepsilon I)$ with a very small $\varepsilon > 0$, we get singularly perturbed problems, which form an important class of stiff problems. Moreover, neutral problems

$$y'(t) = f(t, y(t), y(\alpha(t, y(t))), y'(t), y'(\alpha(t, y(t))))$$

can be written in the form (1), if we introduce a new variable $z(t) = y'(t)$ for the derivative. In fact, this problem becomes equivalent to

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y'(t) \\ z'(t) \end{pmatrix} = \begin{pmatrix} z(t) \\ -z(t) + f(t, y(t), y(\alpha(t, y(t))), z(t), z(\alpha(t, y(t)))) \end{pmatrix}.$$

*Partially supported by the Italian M.U.R.S.T. (project: "Numerical methods for evolutionary problems") and I.N.D.A.M.-G.N.I.M. (project: "Numerical methods for ordinary differential equations and applications").

In this implementation, the special structure of the right-hand side is exploited on the linear algebra level during the solution of arising nonlinear systems.

We assume throughout that a solution exists on the considered interval of integration (possibly with discontinuities).

The aim of next section is to show how implicit Runge-Kutta methods, in particular collocation methods based on Radau nodes, can be applied to solve problems of type (1). Many aspects of the implementation are a straightforward extension of ideas implemented in the code RADAU5 for ordinary differential equations, and described in Hairer & Wanner [9, Sect. IV.8].

Here we restrict our discussion to new difficulties that are due to the presence of small delays, and to large elements in the derivative of f with respect to the retarded arguments. Some example problems with their drivers for RADAR5 are provided at the end of the report.

2 Numerical Method

Our main interest is the numerical solution of stiff delay differential equations. Since collocation methods based on Radau nodes have been successfully applied to stiff ordinary differential equations (see the code RADAU5 of [9]), and since these methods have excellent stability properties also for delay equations (see for example Zennaro [14] and Guglielmi & Hairer [6]), it is quite natural to take them as a basis for a code solving stiff problems of the type (1).

For ease of presentation, we assume that only one lag term is present ($m = 1$) and that the problem is autonomous:

$$M y'(t) = f(y(t), y(\alpha(t, y(t)))) \quad (2)$$

where $\alpha(t, y(t)) \leq t$, $y(t_0) = y_0$, and $y(t) = g(t)$ for $t < t_0$. Needless to say that the presented code is written for the general situation.

Radau IIA methods are implicit Runge-Kutta methods, whose coefficient matrix $A = (a_{ij})$ is invertible, and whose weights satisfy $b_i = a_{si}$ ("stiff accuracy"). We denote $c_i = \sum_j a_{ij}$ (see [9] for more information on these methods). For an implementation we consider a grid $t_0 < t_1 < t_2 < \dots$, and we denote the stepsize by $h_n = t_{n+1} - t_n$.

An application of the Radau IIA methods to the problem (2) yields approximations $y_n \approx y(t_n)$ by solving the nonlinear system

$$M(Y_i^{(n)} - y_n) = h_n \sum_{j=1}^s a_{ij} f(Y_j^{(n)}, Z_j^{(n)}), \quad y_{n+1} = Y_s^{(n)}, \quad (3)$$

where $Z_i^{(n)}$ is a suitable approximation to $y(\alpha_i^{(n)})$ with

$$\alpha_i^{(n)} = \alpha(t_n + c_i h_n, Y_i^{(n)})$$

We put

$$Z_i^{(n)} = \begin{cases} g(\alpha_i^{(n)}) & \text{if } \alpha_i^{(n)} < t_0 \\ u_m(\alpha_i^{(n)}) & \text{if } \alpha_i^{(n)} \in [t_m, t_{m+1}], \end{cases} \quad (4)$$

where $u_m(t)$ is a polynomial approximation of the solution $y(t)$ on the interval $[t_m, t_{m+1}]$. A natural choice for u_m is the collocation polynomial, which is of degree s and passes through the values y_m , and $Y_i^{(m)}$ for $i = 1, \dots, s$. Using the Lagrange interpolation formula it is seen to be of the form

$$u_m(t_m + \tau h_m) = \ell_0(\tau)y_m + \sum_{i=1}^s \ell_i(\tau)Y_i^{(m)}, \quad (5)$$

where $\ell_i(\tau)$ is the polynomial of degree s satisfying $\ell_i(c_i) = 1$ and $\ell_i(c_j) = 0$ for $j \neq i$ (here we add $c_0 = 0$ to the nodes c_1, \dots, c_s of the method).

Order of Convergence. If the matrix M is invertible, and if the delay is larger than the step size (i.e., $t - \alpha(t, y(t)) \geq h$), it follows from the standard theory for ordinary differential equations that the local error at grid points is $\mathcal{O}(h^{2s})$, and that the internal stages $Y_i^{(n)}$ approximate the local solution at $t_n + c_i h_n$ with an error of size $\mathcal{O}(h^{s+1})$. In the case of a state-dependent delay this gives an additional $\mathcal{O}(h^{s+2})$ contribution to the local error (due to the multiplication by h_n in (3)). Therefore, on bounded intervals, the global error is of size $\mathcal{O}(h^{s+1})$. For stiff problems, a further order reduction to $\mathcal{O}(h^s)$ is possible, which is in complete analogy to stiff ordinary differential equations (see e.g., [9, Chap. VI]).

If the delay is smaller than the step size, the theory for ordinary differential equations can no longer be applied, and a more involved analysis is necessary. Such a study is beyond the scope of this report. However, we want to emphasize that a general purpose code for stiff delay equations should be able to allow for step sizes larger than the delay, because it is known that stiff solvers are efficient only if ‘large’ step sizes can be used. Difficulties in an implementation that arise due to the presence of large step sizes, are discussed in [7].

2.1 Solving the Nonlinear Equations

An efficient solution of the nonlinear equations (3) is the most demanding part of an implementation of implicit Runge-Kutta methods. For stiff problems (or when M is singular), this system cannot be solved by fixed point iteration, and one is obliged to use some kind of simplified Newton iterations.

As common in the implementation of implicit Runge-Kutta methods, we pre-multiply the system (3) by $A^{-1} = (\omega_{ij})$ and so obtain the nonlinear system $F(Y) = 0$, where $Y = (Y_1, \dots, Y_s)^T$ and the i th component of $F(Y)$ is

$$F_i(Y) = \sum_{j=1}^s \omega_{ij} M(Y_j - y_n) - hf(Y_i, Z_i) \quad (6)$$

(here, we suppress the subscript n when it does not give rise to confusion). In view of an application of simplified Newton iterations we compute (denoting as δ_{ij} the Kronecker symbol

$$\frac{\partial F_i}{\partial Y_j} = \omega_{ij} M - h \delta_{ij} \left(f_y(Y_i, Z_i) + f_z(Y_i, Z_i) u'_m(\alpha_i) \alpha_y(t_n + c_i h, Y_i) \right), \quad (7)$$

where we have to add the term

$$-h f_z(Y_i, Z_i) \ell_j(\sigma_i) \quad \text{if } \sigma_i := (\alpha(t_n + c_i h, Y_i) - t_n)/h > 0. \quad (8)$$

For an efficient implementation, we replace the exact Jacobian of the nonlinear system with an approximation which, written in tensor notation, is given by

$$A^{-1} \otimes M - hI \otimes (f_y + f_z u'_m(\alpha) \alpha_y) - hL \otimes f_z. \quad (9)$$

The arguments of f_y , f_z , α , and α_y are chosen independent of i . The $s \times s$ matrix L has elements given by

$$l_{ij} = \begin{cases} \ell_j(\sigma_i) & \text{if } \alpha(t_n + c_i h, Y_i) > t_n \\ 0 & \text{else.} \end{cases}$$

For an implementation we distinguish the two situations:

Step size is smaller than the delay or, more precisely, if $\sigma_i \leq 0$ for $i = 1, \dots, s$. For a constant delay (i.e., $\alpha(t, y) = t - \tau$), this happens if and only if $h \leq \tau$ (because $c_i \leq c_s = 1$). In this situation, the matrix L vanishes identically, and the matrix (9) has exactly the same structure as for ordinary differential equations $My' = f(y)$. Transforming the matrix A^{-1} to diagonal form, the linear system with matrix (9) can be solved efficiently as described in [9, Sect. IV.8]. The user of our code can either provide a subroutine with the analytic expression of $f_y(y, z) + f_z(y, z)y'(\alpha(t, y))\alpha_y(t, y)$ (where $z \approx y(\alpha(t, y(t)))$), or he can choose the option of computing it internally by numerical differentiation. Observe that the nasty second term of this matrix is only present for state-dependent delays.

Step size is larger than the delay. In this case, the matrix L in (9) is non-zero. Since, in general, the matrices A^{-1} and L are not simultaneously diagonalizable, the tensor product structure cannot easily be exploited for an efficient solution of the linear systems with matrix (9). However, when the delay is very small compared to the step size (i.e., $\alpha(t_n + c_i h, Y_i) \approx t_n + c_i h$), we get $\sigma_i \approx c_i$, and the matrix L becomes close to the identity. Consequently, the second and third terms in (9) can be considered together, and the idea of diagonalizing the matrix A^{-1} can again be applied.

In conclusion, we adopt the following strategy for approximating the Jacobian of the nonlinear Runge-Kutta equations: if $\alpha(t_n + c_i h, Y_i) \leq t_n$ for at least one i , we let $L = 0$, otherwise we put $L = I$ (identity) in the Jacobian approximation of (9). In both cases standard techniques for stiff ordinary differential equations can be applied, and the tensor product structure of the linear system can be exploited. In case of difficulties in the convergence of the simplified Newton iterations we use the correct L . This, however, requires the LR decomposition of the full matrix (9), which is about 5 times as expensive for the 3-stage method ($s = 3$).

3 The code

The code RADAR5 is written in ANSI Fortran-90 and is made of the following routines, which constitute the kernel of the program.

3.1 The main routine RADAR5

The typical CALL to the main subroutine is as follows.

```

      CALL RADAR5(N,FCN,PHI,ARGLAG,X,Y,XEND,H,
&
&          RTOL,ATOL,ITOL,
&          JAC,IJAC,MLJAC,MUJAC,
&          JACLAG,NLAGS,NJACL,
&          IMAS,SOLOUT,IOUT,
&          WORK,IWORK,RPAR,IPAR,IDID,
&          GRID,IPAST,MAS,MLMAS,MUMAS)

```

List of arguments

N Denotes the dimension of the system.

FCN Name (external) of the subroutine computing the right hand-side of the system of delay differential equations. This has to be provided by the user. For a description see Sect. 5.1 and for examples see Sect. 8.

PHI Name (external) of the function providing the initial functions for the dependent variables of the system of delay differential equations. This has to be provided by the user. For a description see Sect. 5.2 and for examples see Sect. 8.

ARGLAG Name (external) of the function providing the deviating arguments. This has to be provided by the user. For a description see Sect. 5.3 and for examples see Sect. 8.

X At the input denotes the initial value of the independent variable. At the output denotes the last value for which the solution has been computed.

Y At the input denotes the initial values for the vector of dependent variables (it may be different from the value of PHI at X; in this case it is highly recommended to set IWORK(13) and GRID(1) (see below). At the output denotes the numerical solution at X.

XEND Final value of X (XEND-X has to be positive).

H Initial guess for the stepsize; for stiff equations with initial transient, $H=1.D0/(\text{norm of } F')$; usually 1.D-3 or 1.D-5 is good. If $H=0$, the code puts $H=1.D - 6$. At output denotes the predicted stepsize of the last accepted step.

RTOL Relative tolerance; can be both a scalar or a vector.

ATOL Absolute tolerance; can be both a scalar or a vector.

ITOL Switch for RTOL and ATOL. If ITOL=0 both RTOL and ATOL are scalars. The code keeps (roughly) the local error as

$$\frac{Y(I)}{RTOL * ABS(Y(I)) + ATOL}.$$

If ITOL=1 both RTOL and ATOL are vectors. The code keeps (roughly) the local error as

$$\frac{Y(I)}{RTOL(I) * ABS(Y(I)) + ATOL(I)}.$$

JAC Name (external) of the subroutine which computes the partial derivatives of $F(X,Y,Z)$ (where Z denotes the retarded variables) with respect to Y . This routine is only called if $IJAC=1$; otherwise (if $IJAC=0$) a numerical approximation is provided by the code. In the case $IJAC=0$ the user has to supply a dummy subroutine. For a description see Sect. 5.4 and for examples see Sect. 8.

IJAC Switch for the computation of the Jacobian. If $IJAC=0$ the Jacobian is computed internally by finite differences; the subroutine **JAC** is not necessary. If $IJAC=1$ the Jacobian is supplied by the subroutine **JAC**.

MLJAC Switch for the banded structure of the Jacobian: If $MLJAC = N$ the Jacobian is a full matrix. If $0 \leq MLJAC < N$ $MLJAC$ is the lower bandwidth of the Jacobian matrix.

MUJAC The upper bandwidth of the Jacobian matrix. Does not need to be set if $MLJAC = N$.

JACLAG Name (external) of the subroutine which computes the partial derivatives of $F(X,Y,Z)$ with respect to Z (where Z denote the delayed variables). This has to be provided by the user only if he sets $NLAGS > 0$ (in such case the user has to provide the derivative entries with respect to the $NLAGS$ deviating arguments). In the case $NLAGS = 0$ the user has to supply a dummy subroutine. For a description see Sect. 5.5 and for examples see Sect. 8.

NLAGS Denotes the number of delay arguments which the user wants to take into account when computing the partial derivatives of $F(X,Y,Z)$ with respect to Z (see (8)). It is of interest for the computation of the Jacobian of the nonlinear Runge–Kutta iteration. If it is set to 0 no derivatives with respect to delayed variables are considered; if set to $P (> 0)$, $NJACL$ derivative entries with respect to P delayed variables are computed. For a detailed explanation see Sect. 5.5 and for examples see Sect. 8.

NJACL Total number of derivative entries with respect to delayed variables which are intended to be computed by the routine **JACLAG**.

IMAS Gives information on the mass matrix. If $IMAS=0$ then M is taken as the identity matrix. If $IMAS=1$ the mass matrix is supplied by the user through the subroutine **MAS**.

SOLOUT Name (external) of the subroutine providing the numerical solutions during the integration. If $IOUT=1$ it is called after every successful step; if $IOUT=0$ it is never called; in such a case the user should provide a dummy routine. It has also access to the continuous output of the approximate solutions. For a description see Sect. 5.6 and for examples see Sect. 8.

IOUT Flag for calling the subroutine **SOLOUT**. If $IOUT=1$ it is called after every accepted step. If $IOUT=0$ it is never called. This option is recommended when doing tests on performances.

WORK Array of state parameters of real kind for controlling and tuning the execution. $WORK(1)$, $WORK(2)$, ..., $WORK(20)$ serve as parameters for the code; for a standard use of the code (default values) they can be set to 0 by the user before calling **RADAR5**. For a description see Sect. 3.2.2 and for examples see Sect. 8.

IWORK Array of state parameters of integer kind for controlling and tuning the execution. $IWORK(1)$, $IWORK(2)$, ..., $IWORK(20)$ serve as parameters for the code; for a standard use of the code

(default values) they can be set to 0 by the user before calling RADAR5. For a description see Sect. 3.2.1 and for examples see Sect. 8.

RPAR Array of real parameters which can be used for communication between your calling program and the routines/functions FCN, JAC, MAS, SOLOUT, ARGLAG, PHI, JACLAG. It is useful to pass the variables which parametrize the differential system. For further details see the examples in Sect. 8.

IPAR Array of integer parameters which can be used for communication between your calling program and the routines/functions FCN, JAC, MAS, SOLOUT, ARGLAG, PHI, JACLAG.

IDID Reports on succesfullness of the integration upon return:

IDID= 1: Succesful computation;

IDID= 2: Succesful computation interrupted by routine SOLOUT;

IDID=-1: Non-consistent input values;

IDID=-2: Too many stepsizes required ($> NMAX$)

IDID=-3: Stepssize becomes too small;

IDID=-4: Jacobian matrix repeteadly singular;

IDID=-5: Computation interrupted by routine YLAGR5;

IDID=-6: The equation makes use of advanced arguments.

GRID Array of length LGRID (which must be at least IWORK(13)+1). Contains prescribed mesh points, which the integration method has to take as grid points. For a correct use it is necessary that $X < GRID(1) < GRID(2) < \dots < GRID(NGRID) \leq XEND$.

IPAST Integer array of dimension IWORK(15) (number of components for which dense output is required). For $0 < IWORK(15) < N$ the components for which dense output is required have to be specified in IPAST(1),...,IPAST(IWORK(15)) (for NRDENS=N this is done automatically).

MAS Name (external) of the subroutine computing the mass matrix M. If IMAS=0 the subroutine is not needed and the mass matrix is assumed to be the identity; in such a case a dummy routine has to be provided by the user. If IMAS=1 the matrix has to be provided. For a description see Sect. 5.7 and for examples see Sect. 8.

MLMAS Switch for the banded structure of the mass matrix. If MLMAS=N, M is a full matrix; if $0 \leq MLMAS < N$, MLMAS is the lower bandwidth of the matrix M.

Constraint: MLMAS \leq MLJAC.

MUMAS Upper bandwith of the mass matrix. Does not need to be set if MLMAS = N.

Constraint: MUMAS \leq MUJAC.

3.2 Input work parameters to be specified by the user

Several parameters of the code are tuned to make it work well. They may be defined by setting WORK(1),...,WORK(10) as well as IWORK(1),...,IWORK(15) different from zero. For zero input the code automatically choses default values.

3.2.1 The IWORK array (integer parameters)

IWORK(1) If IWORK(1) $\neq 0$, the code transforms the Jacobian matrix to Hessenberg form. This is particularly advantageous for large systems with full Jacobian.

It does not work for banded Jacobian and for implicit systems (IMAS=1).

IWORK(2) This is the maximal number of allowed steps. The default value is 100000.

IWORK(3) This is the maximum number of Newton iterations for the solution of the algebraic system of equations in each step. The default value is 7.

IWORK(4) If IWORK(4) = 0 the extrapolated collocation polynomial is taken to provide starting values for Newton iteration. If IWORK(4) $\neq 0$ no extrapolation is done but the initial solution value is used. The latter is recommended if Newton's method has difficulties with convergence. The default value is 0.

IWORK(5), IWORK(6), IWORK(7) Are relevant to delay differential algebraic systems of index > 1 (see explanations inside the code).

IWORK(8) Switch for stepsize strategy If IWORK(8) = 1 the modified predictive control (developed by Gustafsson) is used. If IWORK(8) = 2 the classical stepsize control is used (for a detailed description see [9]) The first choice often produce safer results while the second is generally faster. The default value is 1.

IWORK(9), IWORK(10) Parameters relevant to systems with special structure; see the explanation inside the code.

IWORK(11) Step size selection strategies for stiff ordinary differential equations are usually based on error estimations at grid points. For delay equations, where the accuracy of the dense output strongly influences the performance, such an approach often is not sufficient (see [7]). For this reason the code also controls the error in the dense output.

IWORK(11) switches to different types of error controls.

-1: pure control of the dense output (makes use of a quadratic and a linear polynomials interpolating the stage values);

0: mixed control of dense output and discrete output;

1: simpler mixed control;

2: pure control of the discrete output (is provided by the routine ESTRAD).

Default value is 0. When the solution is definitely smooth 2 is recommended.

IWORK(12) Maximum number of steps stored in the dense output array (PAST). It has to be declared in the calling driver program. Has to be sufficiently large if the delays can be large so that many back steps has to be stored.

IWORK(13) Number of prescribed grid points in the integration mesh. In the integration, typically, at these points the solution or one of its derivative may have a discontinuity. These points have to be set by the user as GRID(1),...,GRID(NGRID). Default value is 0.

IWORK(14) An efficient solution of the nonlinear equations (3) is the most demanding part of an implementation of implicit Runge-Kutta methods. We consider two possible iterations (see [7] for a detailed explanation).

IWORK(14) is the selector for the iteration. If is set to 1 it forces the code to a simplified iteration (always preserving the block tensor structure of the Jacobian). If set to 2 it possibly executes a full iteration (when the selected stepsize is larger than some delays and the simplified iteration does not converge). Default value is 1.

IWORK(15) Number of solution components for which the dense output is required in the computation of the right-hand side (often denoted as NRDENS).

3.2.2 The WORK array (real parameters)

WORK(1) The rounding unit. Default value is 10^{-16} .

WORK(2) The safety factor in stepsize prediction. Default value is 0.9.

WORK(3) Determines whether the Jacobian should be recomputed. If negative forces the code to compute the Jacobian after every accepted step. Suggested values: 0.1 if Jacobian evaluations are expensive, 0.001 if Jacobian evaluations are not expensive. Default value is 0.001.

WORK(4) Stopping criterion for Newton iteration (usually chosen < 1). Smaller values make the code slower but safer. Default value: $\min(0.03, \sqrt{\text{RTOL}(1)})$.

WORK(5), WORK(6) If $\text{WORK}(5) < \text{HNEW}/\text{HOLD} < \text{WORK}(6)$, then the stepsize is not changed. This saves LU-decompositions and computing time for large systems. Suggested values: $\text{WORK}(5)=1.0$, $\text{WORK}(6)=1.2$ for small systems, $\text{WORK}(5)=0.99$, $\text{WORK}(6)=2.0$ for large full systems. Default values: $\text{WORK}(5)=1.0$, $\text{WORK}(6)=1.2$.

WORK(7) Maximal allowed stepsize. Default value: XEND-X.

WORK(8), WORK(9) Parameters controlling the stepsize selection. The new stepsize is chosen subject to the restriction:

$$\text{WORK}(8) \leq \text{HNEW}/\text{HOLD} \leq \text{WORK}(9)$$

Default values: $\text{WORK}(8)=0.2$, $\text{WORK}(9)=8.0$.

WORK(10) Parameter for tuning the error control of dense output (it is active if $\text{IWORK}(11) = 0$). Range of admissible values $0 \leq \text{WORK}(10) \leq 1$. For smaller values the control is stronger. Suggested values: 0.0 for problems with almost discontinuous solutions (like shocks), 1.0 for problems with fairly smooth solutions, intermediate values for intermediate problems. Default value: 0.D0.

3.3 Output parameters for statistics

IWORK(13) Number of full Newton iterations.

IWORK(14) Number of function evaluations (those for numerical evaluation of the Jacobian are not taken into account).

IWORK(15) Number of Jacobian evaluations (either analytically or numerically).

IWORK(16) Total number of computed steps.

IWORK(17) Total number of accepted steps.

IWORK(18) Total number of rejected steps due to error test (step rejections in the first step are not taken into account).

IWORK(19) Number of LU-decompositions.

IWORK(20) Number of forward-backward substitutions of both systems; the NSTEP forward backward substitutions needed for stepsize selection are not taken into account.

3.4 The routine LAGR5

It provides the position in the dense output array (PAST) for every delayed component which is present in the problem.

The typical call is as follows.

```
CALL LAGR5(IL,X,Y,ARGLAG,PAST,THETA,IPOS,RPAR,IPAR)
```

List of arguments

IL The index addressing the required delayed argument.

X,Y The argument for the delay term α_{IL} .

ARGLAG The name of the function computing the deviating arguments.

PAST The array storing the dense output.

IPOS Captures the position (in the dense output array) of the interval where the deviating arguments falls.

THETA Relative position (its value is between 0 and 1) of the deviating argument within the interval addressed by IPOS.

RPAR,IPAR User-defined arrays containing optional parameters for the delay equation.

3.5 The routine YLAGR5

It provides an approximation to the IC-th component of the solution at any abscissa of the integration interval. Such an approximation is determined by means of the computed piecewise polynomial continuous extension of the discrete numerical solution. Its call has to be preceded by a call of LAGR5.

The typical call is as follows.

```
CALL YLAGR5(IC,THETA,IPOS,PHI,RPAR,IPAR,PAST,IPAST,NRDS)
```

List of arguments

IC The index addressing the required component.

THETA, IPOS As in LAGR5. The required component (addressed by IC) has to be approximated at the abscissa addressed by means of IPOS and THETA.

PHI The initial function.

PAST The array storing the dense output.

IPAST Integer array specifying the indexes of the components whose continuous approximation are stored into PAST during integration.

NRDS Total number of components for which dense output is required.

RPAR, IPAR As in RADAR5.

4 Modules common to all the routines

The module `IP_ARRAY` at the beginning of the main file `radar5.f` makes common (to all routines which declare it) the vector `IPOSV` which is a useful multiple pointer to the dense output for different delays. Its dimension is set to 10. If the equation to be solved makes use of more than 10 delays, the user has to modify the dimension of the vector `IPOSV` and set it to the desired value and finally has to recompile (or to remake if using the make command).

4.1 Workspace

The module has to be as follows:

```
MODULE IP_ARRAY
  INTEGER, dimension(#NL) :: IPOSV
END MODULE IP_ARRAY
```

where `#NL` is larger than the number of considered (different) delays.

5 Routines to be provided by the user

Each user-defined subroutine is described in this section. The argument lists of each subroutine contain arguments in one of three classifications:

Input The argument is read, but not written to.

Modified The argument is both read and written.

Output The argument is only written.

5.1 Subroutine FCN

The FCN subroutine computes the right hand side of the considered system. It has to be declared external in the calling program and its name may be chosen by the user.

5.1.1 Argument list

```
SUBROUTINE FCN(N,X,Y,F,ARGLAG,PHI,RPAR,IPAR,PAST,IPAST,NRDS)

IMPLICIT REAL*8 (A-H,K,O-Z)
INTEGER, PARAMETER :: DP=kind(1D0)
REAL(kind=DP), dimension(N) :: Y
REAL(kind=DP), dimension(N) :: F
REAL(kind=DP), dimension(1) :: PAST
INTEGER, dimension(1) :: IPAST
REAL(kind=DP), dimension(1) :: RPAR
EXTERNAL PHI
```

5.1.2 Input

N,ARGLAG,PHI,RPAR,IPAR,IPAST As in the main routine RADAR5

X,Y Values of the independent and dependent variables at which the function has to be evaluated.

PAST Array containing the required dense output.

NRDS Number of components for which dense output is required.

5.1.3 Output

F Vector of values assumed by the right hand side.

5.2 Function PHI

The PHI function provides initial functions for the components which appear in the system with delays. It has to be declared external in the calling program and its name may be chosen by the user.

5.2.1 Argument list

```
FUNCTION PHI(I,X,RPAR,IPAR)
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER, PARAMETER :: DP=kind(1D0)
REAL(kind=DP), dimension(1) :: RPAR
```

5.2.2 Input

I Index of the component.

X Value of the independent variable at which the function has to be evaluated.

RPAR, IPAR As in the main routine RADAR5

5.2.3 Output

PHI Required value of the initial function.

5.3 Subroutine ARGLAG

The ARGLAG function provides the deviating arguments $\alpha_i(t, y(t))$, $i = 1, 2, \dots$, in the system. It has to be declared external in the calling program and its name may be chosen by the user.

5.3.1 Argument list

```
FUNCTION ARGLAG(IL,X,Y,RPAR,IPAR)
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER, PARAMETER :: DP=kind(1D0)
REAL(kind=DP), dimension(1) :: Y
INTEGER, dimension(1) :: IPAR
REAL(kind=DP), dimension(1) :: RPAR
```

5.3.2 Input

IL Index of the deviating argument.

X,Y Values of the independent and dependent variables at which the function has to be evaluated.

RPAR, IPAR As in the main routine RADAR5

5.3.3 Output

ARGLAG Required value of the deviating argument.

5.4 Subroutine JAC

The JAC subroutine provides the Jacobian of the function $f(t, y(t), y(\alpha_1(t, y(t))), \dots, y(\alpha_m(t, y(t))))$ (FCN) with respect to $y(t)$ (Y variables). It is optional (it is activated if IJAC=1). If used, it has to be declared external in the calling program and its name may be chosen by the user. If not provided set IJAC=0 and put a dummy name in the call statement to the main routine RADAR5.

5.4.1 Argument list

```
SUBROUTINE JAC(N,X,Y,DFY,LDFY,ARGLAG,PHI,RPAR,IPAR,PAST,IPAST,NRDS)
IMPLICIT REAL*8 (A-H,K,O-Z)
INTEGER, PARAMETER :: DP=kind(1D0)
REAL(kind=DP), dimension(N) :: Y
REAL(kind=DP), dimension(LDFY,N) :: DFY
REAL(kind=DP), dimension(1) :: PAST
INTEGER, dimension(1) :: IPAST
REAL(kind=DP), dimension(1) :: RPAR
EXTERNAL PHI
```

5.4.2 Input

N,ARGLAG,PHI,RPAR,IPAR,IPAST As in the main routine RADAR5

X,Y Values of the dependent and independent variable at which the function has to be evaluated.

RPAR, IPAR As in the main routine RADAR5

PAST, NRDS As in the routine FCN

LDFY Leading dimension of the Jacobian.

5.4.3 Output

DFY Matrix containing the standard Jacobian evaluated at X,Y.

5.5 Subroutine JACLAG

The JACLAG subroutine provides the derivatives of $f(t, y(t), y(\alpha_1(t, y(t))), \dots, y(\alpha_m(t, y(t))))$ (FCN) with respect to $y(\alpha_1(t, y(t))), \dots, y(\alpha_m(t, y(t)))$ (delayed variables). It may be very important when the code uses stepsizes larger than delays and aims to perform *exact* Newton iterations. It is suggested but optional (if the user does not want to use this option he has to set IWORK(14)=1 and NLAGS=0). If used, it has to be declared external in the calling program and its name may be chosen by the user. If not provided set NLAGS=0 and put a dummy name in the call statement to RADAR5.

5.5.1 Argument list

```
SUBROUTINE JACLAG(N,X,Y,DFYL,ARGLAG,PHI,IVE,IVC,IVL,
&                RPAR,IPAR,PAST,IPAST,NRDS)

IMPLICIT REAL*8 (A-H,O-Z)
INTEGER, PARAMETER :: DP=kind(1D0)
REAL(kind=DP), dimension(N) :: Y
REAL(kind=DP), dimension(1) :: DFYL
REAL(kind=DP), dimension(1) :: PAST
```

```

INTEGER, dimension(1) :: IPAST
REAL(kind=DP), dimension(1) :: RPAR
INTEGER, dimension(1) :: IVE,IVC,IVL
EXTERNAL PHI

```

5.5.2 Input

N,ARGLAG,PHI,RPAR,IPAR,IPAST As in the main routine RADAR5

X,Y Values of the dependent and independent variable at which the function has to be evaluated.

RPAR, IPAR As in the main routine RADAR5

PAST, NRDS As in the routine FCN

5.5.3 Output

IVE,IVC,IVL Vector of indeces which address the derivative entries with respect to delayed terms in the right hand side.

For the K-th entry,

IVE(K) has to indicate the number of the relevant equation;

IVC(K) has to indicate the number of the relevant component;

IVL(K) has to indicate the number of the relevant delay.

DFYL Array containing the values of the derivative entries with respect to delayed components. For the K-th entry,

DFYL(K) has to store the value of the derivative.

This array will be used by the code – when required – to construct the full Jacobian matrix.

5.6 Subroutine SOLOUT

The SOLOUT subroutine provides the numerical approximation of the solution during the integration.

5.6.1 Argument list

```

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

```

```

IMPLICIT REAL*8 (A-H,O-Z)
INTEGER, PARAMETER :: DP=kind(1D0)
REAL(kind=DP), PARAMETER :: XSTEP=0.01D0
REAL(kind=DP), dimension(N) :: Y
REAL(kind=DP), dimension(LRC) :: CONT
REAL(kind=DP), dimension(1) :: RPAR
EXTERNAL PHI

```

5.6.2 Input

X,Y Values of the dependent and independent variable at which the function has to be evaluated.

N,RPAR,IPAR As in the main routine RADAR5

LRC Dimension of the array **CONT**.

5.6.3 Modified

CONT Vector of dense output for the current stepsize and for all components (used for output aims).

5.6.4 Output

NR Number of the mesh point at which the solution is furnished.

XOLD The preceeding mesh point.

HSOL The value of the stepsize for the current step.

IRTRN If set < 0 , serves to interrupt the integration.

5.7 Subroutine MAS

The **MAS** subroutine computes the mass matrix **M** of the implicit system. It has to be declared external in the calling program and its name may be chosen by the user. If is not provided the system is considered explicit, that is M is taken as the identity matrix.

5.7.1 Argument list

```
SUBROUTINE MAS(N,Q,LQ,RPAR,IPAR)
```

```
    INTEGER, PARAMETER :: DP=kind(1D0)  
    REAL(kind=DP), dimension(1) :: RPAR  
    REAL(kind=DP), dimension(LQ,N) :: Q
```

5.7.2 Input

N,RPAR,IPAR As in the main routine RADAR5

LQ Leading dimension of the mass matrix.

5.7.3 Output

Q The $N \times N$ mass matrix of the system.

6 How to install RADAR5

This section describes how to obtain and install RADAR5.

Table 1: Source files in the RADAR5 package

files	description
<code>radar5.f</code>	main routines implementing the integration scheme
<code>dc_decde1.f</code>	routines involving error estimation
<code>decsol.f</code>	rotines providing the necessary linear algebra
<code>contr5.f</code>	routine providing continuous output

6.1 Getting the files and installing the code

The code, together with seven driver programs, is available at the address ‘<http://www.unige.ch/math/folks/haier/>’ under item ‘software’ and is mirrored at the address

‘<http://www.mathsun1.univ.trieste.it/math/>’ .

The user can download the tar-file archive and generate the directory of the program and the directories including the examples in its own computer by making use of the Unix command `tar -xvf radar5.f`

People who do not use UNIX operative systems may contact directly the authors and will be provided of the package in the desired format.

RADAR5 Versions 1.0 consists of 4 ANSI Fortran-90 files plus one documentation/installation postscript file. Also included in the distribution are seven drivers for the test programs. Makefiles for automatic compilation are also provided.

7 Examples provided

RADAR5 comes with seven examples:

- a) ENZYME, kinetics with an inhibitor molecule, dimension 4, one constant delay;
- b) HAYASHI, an almost singular state dependent neutral problem, dimension 2 as differential-algebraic problem, one vanishing delay;
- c) HEPATITIS, acute hepatitis B virus infection, dimension 10, 5 constant delays;
- d) OREGONATOR, chemical kinetics, dimension 2, one constant delay;
- e) ROBERTSON, chemical reaction with steady state solution, dimension 3, one constant delay, very large time interval (step sizes larger than the delay);
- f) SDISC, non smooth artificial problem, dimension 1, 1 constant delay;
- g) WALTMAN, threshold model for antibody production, dimension 6, 2 state-dependent delays tending to zero, discontinuities in the right-hand side of the equations.

8 Example of use

8.1 The Oregonator model.

This problem, taken from [5], consists of two equations and one constant delay. They are given by

$$\begin{aligned}y_1'(t) &= kM1 A y_2(t) - kM2 y_1(t) y_2(t - \tau) + kM3 B y_1(t) - 2 kM4 y_1(t)^2 \\y_2'(t) &= -kM1 A y_2(t) - kM2 y_1(t) y_2(t - \tau) + fr kM3 B y_1(t)\end{aligned}$$

with

$$\begin{aligned}kM1 &= 1.34 \\kM2 &= 1.6 \cdot 10^9 \\kM3 &= 8.0 \cdot 10^3 \\kM4 &= 4.0 \cdot 10^7 \\kM5 &= 1.0 \\fr &= 1.0 \\A &= 6.0 \cdot 10^{-2} \\B &= 6.0 \cdot 10^{-2} \\\tau &= 0.15\end{aligned}$$

Initial values and functions are $y_1(t) = 10^{-10}$ and $y(t) = 10^{-5}$ for $t \leq 0$. We consider the integration interval $[0, 100.5]$ and $Atol = 10^{-9} \cdot Rtol$.

The driver program

```
C
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*4 TARRAY(2)
      INTEGER, PARAMETER :: DP=kind(1D0)
      INTEGER, PARAMETER :: ND=2
      INTEGER, PARAMETER :: NRDENS=1
      INTEGER, PARAMETER :: NGRID=11
      INTEGER, PARAMETER :: NLAGS=1
      INTEGER, PARAMETER :: NJACL=2
      INTEGER, PARAMETER :: MXST=4000
      INTEGER, PARAMETER :: LWORK=30
      INTEGER, PARAMETER :: LIWORK=30
      REAL(kind=DP), dimension(ND) :: Y
      REAL(kind=DP), dimension(NGRID+1) :: GRID
      REAL(kind=DP), dimension(LWORK) :: WORK
      INTEGER, dimension(LIWORK) :: IWORK
      INTEGER, dimension(NRDENS+1) :: IPAST
      REAL(kind=DP), dimension(10) :: RPAR
      INTEGER, dimension(22) :: ISTAT
      EXTERNAL FCN, PHI, ARGLAG, JFCN, JACLAG, SOLOUT
C ----- FILE TO OPEN -----
      OPEN(9, FILE='sol.out')
      OPEN(10, FILE='cont.out')

C      PARAMETERS IN THE DIFFERENTIAL EQUATION
```

```

C --- kM1
      RPAR(1)=1.34
C --- kM2
      RPAR(2)=1.6D9
C --- kM3
      RPAR(3)=8.0D3
C --- kM4
      RPAR(4)=4.0D7
C --- kM5
      RPAR(5)=1.D0
C --- fr
      RPAR(6)=1.D0
C --- A
      RPAR(7)=6.D-2
C --- B
      RPAR(8)=6.D-2
C --- Tau
      RPAR(9)=15.D-2
C
C --- DIMENSION OF THE SYSTEM
      N=ND
C --- COMPUTE THE JACOBIAN ANALYTICALLY
      IJAC=1
C --- JACOBIAN IS A FULL MATRIX
      MLJAC=N
C --- DIFFERENTIAL EQUATION IS IN EXPLICIT FORM
      IMAS=0
C --- OUTPUT ROUTINE IS USED DURING INTEGRATION
      IOUT=1
C --- INITIAL VALUES (CONSISTENT WITH INITIAL FUNCTIONS)
      X=0.0D0
      Y(1)= 1.D-10
      Y(2)= 1.D-5
C --- DELAY
      TAU=RPAR(9)
C --- ENDPOINT OF INTEGRATION
      XEND=100.5D0
C --- REQUIRED (RELATIVE AND ABSOLUTE) TOLERANCE
      ITOL=0
      RTOL=1.D-9
      ATOL=RTOL*1.D-9
C --- INITIAL STEP SIZE
      H=1.0D-6
C --- DEFAULT VALUES FOR PARAMETERS
      IWORK(1:20)=0
      WORK(1:20)=0.0D0
C --- PARAMETERS FOR ERROR ESTIMATION
      WORK(10)=1.0
      IWORK(11)=2
C --- MAX NUMBER OF STEPS ALLOWED
      IWORK(2)=100000

```

```

C --- WORKSPACE FOR THE ARRAY PAST
      IWORK(12)=MXST
C --- NUMBER OF DELAYED COMPONENTS
      IWORK(15)=NRDENS
C --- THE SECOND COMPONENT USES RETARDED ARGUMENT
      IPAST(1)=2
C --- WORKSPACE FOR GRID
      IWORK(13)=NGRID
C --- PRESCRIBED GRID-POINTS
      DO I=1,NGRID
        GRID(I)=I*TAU
      END DO
C --- CONTROL PARAMETER FOR SIMPLIFIED ITERATION
      IWORK(14)=1
C --- CALL OF THE SUBROUTINE RADAR5
      CALL DTIME(TARRAY)
      CALL RADAR5(N,FCN,PHI,ARGLAG,X,Y,XEND,H,
&               RTOL,ATOL,ITOL,
&               JFCN,IJAC,MLJAC,MUJAC,
&               JACLAG,NLAGS,NJACL,
&               IMAS,SOLOUT,IOUT,
&               WORK,IWORK,RPAR,IPAR,IDID,
&               GRID,IPAST,DUMMY,MLMAS,MUMAS)
      CALL DTIME(TARRAY)
C --- PRINT FINAL SOLUTION SOLUTION
      WRITE (6,90) X,Y(1),Y(2)
C --- PRINT STATISTICS
      DO J=14,20
        ISTAT(J)=IWORK(J)
      END DO
      WRITE(6,*) ' ***** TOL=',RTOL,' ELAPSED TIME=',TARRAY(1),' *****'
      WRITE (6,91) (ISTAT(J),J=14,20)
      WRITE (6,92) ISTAT(23)
90    FORMAT(1X,'X =',F8.2,'      Y =',2E18.10)
91    FORMAT(' fcn=',I7,' jac=',I6,' step=',I6,
&          ' accpt=',I6,' reject=',I6,' dec=',I6,
&          ' sol=',I7)
92    FORMAT(' full Newt. its =',I7)
      WRITE(6,*) 'SOLUTION IS TABULATED IN FILES: sol.out & cont.out'
      STOP
      END

```

The subroutine SOLOUT

Determines the updating of the files `sol.out` and `cont.out` where the solution is stored. The format of such files has to be chosen according to the graphics packages one intends to use for drawing the solution. The file `sol.out` contains the values of the approximate solution components at the (adaptively) computed mesh-points. The file `cont.out` contains the values of the approximate solution components at points prescribed by the user (in the present case a uniform grid has been chosen for the output representation).

```

      SUBROUTINE SOLOUT (NR,XOLD,X,HSOL,Y,CONT,LRC,N,
&                      RPAR,IPAR,IRTRN)
C ----- PRINTS THE DISCRETE OUTPUT AND THE CONTINUOUS OUTPUT
C      AT EQUIDISTANT OUTPUT-POINTS
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER, PARAMETER :: DP=kind(1D0)
      REAL(kind=DP), PARAMETER :: XSTEP=0.01D0
      REAL(kind=DP), dimension(N) :: Y
      REAL(kind=DP), dimension(LRC) :: CONT
      REAL(kind=DP), dimension(1) :: RPAR
      EXTERNAL PHI
C      XOUT IS USED FOR THE DENSE OUTPUT
      COMMON /INTERN/XOUT

      WRITE (9,99) X,Y(1),Y(2)
      IF (NR.EQ.1) THEN
        WRITE (10,99) X,Y(1),Y(2)
        XOUT=XSTEP
      ELSE
10        CONTINUE
        IF (X.GE.XOUT) THEN
          WRITE (10,99) XOUT,CONTR5(1,N,XOUT,CONT,X,HSOL),
&              CONTR5(2,N,XOUT,CONT,X,HSOL)
          XOUT=XOUT+XSTEP
          GOTO 10
        END IF
      END IF
99      FORMAT(1X,'X =',F12.8,'      Y =',2E18.10)
      RETURN
      END

```

The function ARGLAG

Provides the deviating argument α_1 .

```

      FUNCTION ARGLAG(IL,X,Y,RPAR,IPAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER, PARAMETER :: DP=kind(1D0)
      REAL(kind=DP), dimension(1) :: Y
      INTEGER, dimension(1) :: IPAR
      REAL(kind=DP), dimension(1) :: RPAR

      ARGLAG=X-RPAR(9)
      RETURN
      END

```

The subroutine FCN

Provides the right hand side of the differential system, that is

$$\begin{aligned}
 f_1 &= kM1 A y_2 - kM2 y_1 z_2 + kM3 B y_1 - 2 kM4 y_1^2 \\
 f_2 &= -kM1 A y_2 - kM2 y_1 z_2 + fr kM3 B y_1
 \end{aligned}$$

where $z_2 := y_2(\alpha_1(t))$ and $\alpha_1(t)$ denotes the deviating argument ($\alpha_1(t) = t - \tau$ in the present case).

```

      SUBROUTINE FCN(N,X,Y,F,ARGLAG,PHI,RPAR,IPAR,
&
      PAST,IPAST,NRDS)
      IMPLICIT REAL*8 (A-H,K,O-Z)
      INTEGER, PARAMETER :: DP=kind(1D0)
      REAL(kind=DP), dimension(N) :: Y
      REAL(kind=DP), dimension(N) :: F
      REAL(kind=DP), dimension(1) :: PAST
      INTEGER, dimension(1) :: IPAST
      REAL(kind=DP), dimension(1) :: RPAR
      EXTERNAL PHI
C      Parameters in the differential equation
      kM1=RPAR(1)
      kM2=RPAR(2)
      kM3=RPAR(3)
      kM4=RPAR(4)
      kM5=RPAR(5)
      fr =RPAR(6)
      A  =RPAR(7)
      B  =RPAR(8)

C      Evaluates the unique deviating arguments (IL=1) and sets
C      its value into THETA1 and the position of the relative interval
C      into IPOS1
      CALL LAGR5(1,X,Y,ARGLAG,PAST,THETA1,IPOS1,RPAR,IPAR)
C      Evaluates the (only) component requiring dense output (which is Y(2))
C      and puts the result into Z2
      Z2=YLAGR5(2,THETA1,IPOS1,PHI,RPAR,IPAR,PAST,IPAST,NRDS)
C      Computes F
      F(1)= kM1*A*Y(2) - kM2*Y(1)*Z2 + kM3*B*Y(1)-2.D0*kM4*Y(1)**2
      F(2)= -kM1*A*Y(2) - kM2*Y(1)*Z2 + fr*kM3*B*Y(1)
      RETURN
      END

```

The subroutine JFCN

Provides the Jacobian (with respect to the variables y_1 and y_2) of the right hand side analytically. We remark that this function could be provided numerically. In this case we have

$$\frac{\partial f}{\partial y} = \begin{pmatrix} -kM2 z_2 + kM3 B - 4kM4 y_1 & kM1 A \\ -kM2 z_2 + fr kM3 B & -kM1 A \end{pmatrix}$$

where $\alpha_1(t) = t - \tau$ and $z_2 = y_2(\alpha_1(t))$. Hence the routine is written as follows.

```

      SUBROUTINE JFCN(N,X,Y,DFY,LDFY,ARGLAG,PHI,RPAR,IPAR,
&
      PAST,IPAST,NRDS)
C ----- STANDARD JACOBIAN OF THE EQUATION
      IMPLICIT REAL*8 (A-H,K,O-Z)
      INTEGER, PARAMETER :: DP=kind(1D0)
      REAL(kind=DP), dimension(N) :: Y

```

```

REAL(kind=DP), dimension(LDFY,N) :: DFY
REAL(kind=DP), dimension(1) :: PAST
INTEGER, dimension(1) :: IPAST
REAL(kind=DP), dimension(1) :: RPAR
EXTERNAL PHI
kM1=RPAR(1)
kM2=RPAR(2)
kM3=RPAR(3)
kM4=RPAR(4)
kM5=RPAR(5)
fr =RPAR(6)
A  =RPAR(7)
B  =RPAR(8)

C      Determination of the deviating argument
      CALL LAGR5(1,X,Y,ARGLAG,PAST,THETA1,IPOS1,RPAR,IPAR)
      Z2=YLAGR5(2,THETA1,IPOS1,PHI,RPAR,IPAR,PAST,IPAST,NRDS)
C      Jacobian matrix (2x2) with respect to Y(1) and Y(2)
      DFY(1,1)= -kM2*Z2 + kM3*B - 4.D0*kM4*Y(1)
      DFY(1,2)=  kM1*A
      DFY(2,1)= -kM2*Z2 + fr*kM3*B
      DFY(2,2)= -kM1*A
      RETURN
      END

```

The subroutine JACLAG

Provides the Jacobian (with respect to the retarded variable $z_2(t) = y_2(\alpha_1(t))$) of the right hand side analytically. This is stored in a one dimensional array as described in Sect. 5.5. We remark that this function could not be provided numerically (at present time). In this case we have

$$\frac{\partial f}{\partial z} = \begin{pmatrix} 0 & -kM2 y_1 \\ 0 & -kM2 y_1 \end{pmatrix}$$

where $\alpha_1(t) = t - \tau$ and $z_i(t) = y_i(\alpha_1(t))$, $i = 1, 2$. Since we store such matrix in a vector (because we assume it is sparse), the routine is written as follows.

```

      SUBROUTINE JACLAG(N,X,Y,DFYL,ARGLAG,PHI,IVE,IVC,IVL,
&                      RPAR,IPAR,PAST,IPAST,NRDS)
C ----- JACOBIAN OF DELAY TERMS IN THE EQUATION
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER, PARAMETER :: DP=kind(1D0)
      REAL(kind=DP), dimension(N) :: Y
      REAL(kind=DP), dimension(1) :: DFYL
      REAL(kind=DP), dimension(1) :: PAST
      INTEGER, dimension(1) :: IPAST
      REAL(kind=DP), dimension(1) :: kM1,kM2
      REAL(kind=DP), dimension(1) :: RPAR
      INTEGER, dimension(1) :: IVE,IVC,IVL
      EXTERNAL PHI
      kM1=RPAR(1)

```

```

      kM2=RPAR(2)
      A  =RPAR(7)
      B  =RPAR(8)
C --- The Jacobian has two entries;
C      the first entry ((1)) is relevant to
C      - the first (and only) delay, whose index is hence equal to 1 (IVL(1)=1)
C      - the first equation (IVE(1)=1)
C      - the second component of the solution (IVC(1)=2)
      IVL(1)=1
      IVE(1)=1
      IVC(1)=2
C      The value of the derivative with respect to the delayed second
C      component in the first equation is finally given by
      DFYL(1)=-kM2*Y(1)

C      the second entry ((2)) is relevant to
C      - the first (and only) delay, whose index is hence equal to 1 (IVL(2)=1)
C      - the second equation (IVE(2)=2)
C      - the second component of the solution (IVC(2)=2)
      IVL(2)=1
      IVE(2)=2
      IVC(2)=2
C      The value of the derivative with respect to the delayed second
C      component in the second equation is finally given by
      DFYL(2)=-kM2*Y(1)
      RETURN
      END

```

The initial function PHI

Provides the initial functions.

```

      FUNCTION PHI(I,X,RPAR,IPAR)
      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER, PARAMETER :: DP=kind(1D0)
      REAL(kind=DP), dimension(1) :: RPAR
      SELECT CASE (I)
      CASE (2)
        PHI= 1.D-5
      END SELECT
      RETURN
      END

```

Compilation

We create a file, called “dr-oregon.f”, including all previous routines, and put it in a subdirectory (called OREGON) of the main directory RADAR5, where the source and the object files of the main routines of the code are installed. The following makefile is also created in the directory OREGON,

```

FC = f90
FCFLAGS = -c

```

```

DEBUGFLAGS = -g
LIB = /lib /usr/lib
.f.o:
    $(FC) $(FCFLAGS) $(DEBUGFLAGS) $*.f

OBJ2 =    ../radar5.o ../dc_decdel.o ../decsol.o
OBJ1 =    dr-oregon.o
prog :  $(OBJ1)
    $(FC) $(DEBUGFLAGS) -o driver $(OBJ1) $(OBJ2)

```

The simple use of the Unix command **make** generates the compilation and the linking phase for the program. The executable file is called “driver”. In case of troubles check whether the fortran libraries are stored in the directories `/lib` and `/usr/lib`.

Execution

Executing the program driver generates the following output.

```

STARTING INTEGRATION...
NUMBER OF PRESCRIBED GRID POINTS: 11
NUMBER OF DELAYED COMPONENTS: 1
X = 100.50    Y = 0.2749861866E-09  0.3559046555E-06
***** TOL=1.0000000000000013E-9  ELAPSED TIME=2.11727977 *****
fcn= 70386 jac= 5071 step= 9283 accpt= 9074 rejt= 183 dec= 7608
sol= 20435
full Newt. its =      0
SOLUTION IS TABULATED IN FILES: sol.out & cont.out
STOP

```

The program also generates two output files, `sol.out` and `cont.out`, which contain - respectively - the discrete computed solution and an approximation of the solution on a uniform mesh with density chosen by the user.

8.2 Artificial neutral problem.

Our second example, which is a modification by Enright & Hayashi [4] of a problem considered originally by Castleton & Grimm [3], is

$$\begin{aligned}
 v'(t) = & \cos(t) \left(1 + v(t) v^2(t) \right) + c v(t) v'(t v^2(t)) \\
 & + (1 - c) \sin t \cos(t \sin^2 t) - \sin(t + t \sin^2 t)
 \end{aligned} \tag{10}$$

with initial value $v(0) = 0$. For every choice of the parameter c , it has $v(t) = \sin t$ as exact solution. It has a vanishing delay at $t = 0, \pi/2, 3\pi/2, \dots$, and for $c = 1$ it has a singularity at $t = \pi/2$ (i.e., $y'(\pi/2)$ is not well defined by the equation (10)). For this problem, the numerical solution of the nonlinear Runge-Kutta equations causes some difficulties.

We have rewritten the neutral equation (10) in the form (1) by setting $y_1(t) = v(t)$ and introducing the new variable $y_2(t) = v'(t)$ and $z_1(t) = y_1(\alpha_1(t, y))$, $z_2(t) = y_2(\alpha_1(t, y))$, with

Table 2: Statistics for the problem (10) with $Rtol = Atol = 10^{-8}$

c	nr. of steps	nr. of full Jac	error at $t = \pi$
-1.0	127	12	$0.18 \cdot 10^{-8}$
-0.7	111	6	$0.42 \cdot 10^{-8}$
-0.3	99	0	$0.17 \cdot 10^{-9}$
0.3	120	2	$0.10 \cdot 10^{-8}$
0.7	144	14	$0.53 \cdot 10^{-9}$
1.0	164	19	$0.43 \cdot 10^{-7}$

$\alpha_1(t, y) = ty_1^2(t)$, as explained in the first section,

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y_1'(t) \\ y_2'(t) \end{pmatrix} = \begin{pmatrix} f_1(t, y, z) \\ f_2(t, y, z) \end{pmatrix}. \quad (11)$$

with

$$\begin{aligned} f_1(t, y_1, y_2, z_1, z_2) &= y_2(t) \\ f_2(t, y_1, y_2, z_1, z_2) &= -y_2(t) + \cos(t)(1 + z_1(t)) + c y_1(t) z_2(t) + \\ &\quad (1 - c) \sin t \cos(t \sin^2 t) - \sin(t + t \sin^2 t). \end{aligned}$$

Then we have applied our code RADAR5 with $Rtol = Atol = 10^{-8}$. We display the total number of steps (accepted and rejected), the number of steps where difficulties appeared in solving the nonlinear system (so that the correct matrix L had to be used in (9)), and the global error at the endpoint of integration. We observe that such difficulties appear only for values close to ± 1 . It is somewhat surprising that our code solves the problem correctly even for $c = 1$ (the singular case).

For conciseness we do not report the driver program and present only part of the user-defined routines.

The function ARGLAG

Provides the deviating argument, $\alpha_1(t, y) = ty_1^2$.

```

FUNCTION ARGLAG(IL,X,Y,RPAR,IPAR)
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER, PARAMETER :: DP=kind(1D0)
REAL(kind=DP), dimension(1) :: Y
INTEGER, dimension(1) :: IPAR
REAL(kind=DP), dimension(1) :: RPAR
ARGLAG=X*Y(1)**2
RETURN
END
```

The subroutine FCN

Provides the right hand side of the differential system (11).

```

SUBROUTINE FCN(N,X,Y,F,ARGLAG,PHI,RPAR,IPAR,PAST,IPAST,NRDS)
IMPLICIT REAL*8 (A-H,K,O-Z)
INTEGER, PARAMETER :: DP=kind(1D0)
REAL(kind=DP), dimension(N) :: Y
REAL(kind=DP), dimension(N) :: F
REAL(kind=DP), dimension(1) :: PAST
INTEGER, dimension(1) :: IPAST
REAL(kind=DP), dimension(1) :: RPAR
EXTERNAL PHI
C      Contains the parameter c in the equation
      C=RPAR(1)

CALL LAGR5(1,X,Y,ARGLAG,PAST,THETA,IPOS,RPAR,IPAR)
Z1=YLAGR5(1,THETA,IPOS,PHI,RPAR,IPAR,PAST,IPAST,NRDS)
Z2=YLAGR5(2,THETA,IPOS,PHI,RPAR,IPAR,PAST,IPAST,NRDS)
F(1)= Y(2)
F(2)=-Y(2)+COS(X)*(1.D0+Z1)+C*Y(1)*Z2+(1.D0-C)*SIN(X)*COS(X*SIN(X)**2)
&      -SIN(X+X*SIN(X)**2)
RETURN
END

```

The subroutine JACLAG

Provides the Jacobian (with respect to the variables z_1 and z_2) of the right hand side analytically. In this case we have

$$\frac{\partial f}{\partial z} = \begin{pmatrix} 0 & 0 \\ \cos t & c y_1 \end{pmatrix}$$

Hence both entries are relevant to the second equation and the routine is written as follows.

```

SUBROUTINE JACLAG(N,X,Y,DFYL,ARGLAG,PHI,IVE,IVC,IVL,
&      RPAR,IPAR,PAST,IPAST,NRDS)
C ----- JACOBIAN OF DELAY TERMS IN THE EQUATION
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER, PARAMETER :: DP=kind(1D0)
REAL(kind=DP), dimension(N) :: Y
REAL(kind=DP), dimension(1) :: DFYL
REAL(kind=DP), dimension(1) :: PAST
INTEGER, dimension(1) :: IPAST
REAL(kind=DP), dimension(1) :: RPAR
INTEGER, dimension(1) :: IVE,IVC,IVL
EXTERNAL PHI

C=RPAR(1)

C      the first entry ((1)) is relevant to
C      - the first (and only) delay, whose index is hence equal to 1 (IVL(1)=1)

```

```

C      - the second equation (IVE(1)=2)
C      - the first component of the solution (IVC(1)=1)
          IVL(1)=1
          IVE(1)=2
          IVC(1)=1
C      The value of the derivative with respect to the delayed second
C      component in the first equation is finally given by
          DFYL(1)=COS(X)

C      the second entry ((2)) is relevant to
C      - the first (and only) delay, whose index is hence equal to 1 (IVL(2)=1)
C      - the second equation (IVE(2)=1)
C      - the second component of the solution (IVC(2)=2)
          IVL(2)=1
          IVE(2)=2
          IVC(2)=2
          DFYL(2)=C*Y(1)

          RETURN
          END

```

The subroutine JAC

Is not provided. In this case the standard Jacobian is approximated numerically (see the released driver program).

The initial function PHI

Provides the necessary initial functions, if required.

```

FUNCTION PHI(I,X,RPAR,IPAR)
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER, PARAMETER :: DP=kind(1D0)
REAL(kind=DP), dimension(1) :: RPAR
SELECT CASE (I)
CASE (1)
    PHI=SIN(X)
CASE (2)
    PHI=COS(X)
END SELECT
RETURN
END

```

The mass matrix subroutine

Provides the mass matrix for the implicit DDE.

```

SUBROUTINE QFUN(N,Q,LQ,RPAR,IPAR)
C ---- MATRIX "M" FOR THE NEUTRAL PROBLEM
INTEGER, PARAMETER :: DP=kind(1D0)
REAL(kind=DP), dimension(1) :: RPAR

```

```

REAL(kind=DP), dimension(LQ,N) :: Q

Q(1,1)=1.D0
Q(1,2)=0.D0
Q(2,1)=0.D0
Q(2,2)=0.D0
RETURN
END

```

9 Copyright Notice

The RADAR5 Package, Version 1.0, double-precision, Copyright (C) 2000, Nicola Guglielmi and Ernst Hairer,

COPYRIGHT NOTICE. The RADAR5 Package is a set of subroutines developed by Nicola Guglielmi and Ernst Hairer (the Authors) at the University of Geneva. The package has been made available to you (the User) under the following terms and conditions. Your use of RADAR5 is an implicit agreement to these conditions.

1. RADAR5 may only be used for educational and research purposes by the person or organization to whom they are supplied (the “User”).
2. Code that uses RADAR5 (a code that calls subroutines in RADAR5) does not fall under this Copyright Notice. However, code derived from RADAR5 does fall under this Copyright Notice.
3. RADAR5 is provided without warranty of any kind, either expressed or implied. Neither the University of Geneva nor the Authors shall be liable for any direct or consequential loss or damage whatsoever arising out of the use of RADAR5 by the User.
4. Any use of RADAR5 in any commercial application shall be subject to prior written agreement between the Authors and the User on suitable terms and conditions.

10 Final comments

As soon as you receive a copy of RADAR5, please send email to us at guglielm@univaq.it or Ernst.Hairer@unige.ch, so that we can put you on a mailing list for news and updates. Please include your postal address as well.

While we would appreciate hearing any bug reports and comments, we cannot promise that we can fix any specific bugs. We would also appreciate receiving copies of publications that refer to the package.

If you find this software to have a different performance than the software you were previously using, we would be very interested in getting copies of your delay equations. Of particular interest are large stiff problems coming from real-life applications.

Acknowledgments

We thank Alfredo Bellen, Marino Zennaro and Gerhard Wanner for fruitful discussions concerning the development of the code.

References

- [1] C.T.H. Baker, J.C. Butcher, C.A.H. Paul, *Experience of STRIDE applied to delay differential equations*, Technical Report 208, Univ. Manchester, 1992.
- [2] G.A. Bocharov, G.I. Marchuk, A.A. Romanyukha, *Numerical solution by LMMs of stiff delay differential systems modelling an immune response*, Numer. Math. 73, 131–148 (1996).
- [3] R.N. Castleton, L.J. Grimm, *A first order method for differential equations of neutral type*, Math. Comp. 27, 571–577 (1973).
- [4] W.H. Enright, H. Hayashi, *A delay differential equation solver based on a continuous Runge-Kutta method with defect control*, Numer. Algorithms 16, 349–364 (1998).
- [5] I. Epstein, Y. Luo, *Differential delay equations in chemical kinetics. Nonlinear models: the cross-shaped phase diagram and the Oregonator*, J. Chemical Physics 95, 244–254 (1991).
- [6] N. Guglielmi, E. Hairer, *Order stars and stability for delay differential equations*, Numer. Math. 83, 371–383 (1999).
- [7] N. GUGLIELMI AND E. HAIRER, *Implementing Radau II-A methods for stiff delay differential equations*, Comouting, in press (2001).
- [8] E. Hairer, S.P. Nørsett, G. Wanner, *Solving Ordinary Differential Equations I. Nonstiff Problems, 2nd edition*, Springer Series in Computational Mathematics 8, Springer-Verlag Berlin, 1993.
- [9] E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems, 2nd edition*, Springer Series in Computational Mathematics 14, Springer-Verlag Berlin, 1996.
- [10] Z. Jackiewicz, E. Lo, *The numerical solution of neutral functional-differential equations by Adams predictor-corrector methods*, Appl. Numer. Math. 8, 477–491 (1991).
- [11] M. Okamoto, K. Hayashi, *Frequency conversion mechanism in enzymatic feedback systems*, J. Theor. Biol. 108, 529–537 (1984).
- [12] P. Waltman, *A threshold model of antigen-stimulated antibody production*, Theoretical Immunology (Immunology Ser. 8), Dekker, New York, 437–453 (1978).
- [13] R. Weiner, K. Strehmel, *A type insensitive code for delay differential equations basing on adaptive and explicit Runge-Kutta interpolation methods*, Computing 40, 255–265 (1988).
- [14] M. Zennaro, *P-stability properties of Runge-Kutta methods for delay differential equations*, Numer. Math. 49, 305–318 (1986).