14.3 Solve a Differential/Algebraic System

A. Purpose

This routine solves a system of differential/algebraic equations of the form $\mathbf{g}(t, \mathbf{y}, \mathbf{y}') = 0$. Backward differentiation formulas, see [1], and a projection algorithm are used to solve for $\mathbf{y}$ and $\mathbf{y}'$ as t varies from an initial **T** to **TOUT**. The system $\mathbf{g}() = 0$ consists of **NEQ** component functions $\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = 0$ of index 1 or index 0, followed by constraint functions $\mathbf{G}(\mathbf{t}, \mathbf{y}) = 0$. Details are provided in the report, [2]. The code here is a modification of Petzold's code **DASSL**.

B. Usage

Described below under B.1 through B.x are:

The code provides a number of options specified by an array INFO. The following summary gives the index in INFO and the page number, followed by very brief description of the option.

2	4	Type of error control
3	4	Intermediate output
4	4	no evaluations past TSTOP
5	4	Type of matrix
6	5	Level of debug print
7	5	Maximum step size
8	5	Initial step size
9	6	Restrict integration order
10	6	Constraints to project onto
11	6	Integrator solves for initial derivatives
12	6	Max. number steps between output points
13	6	Step size control algorithm

The example codes illustrate features added to **DASSL**. The additional features include:

- Optionally provide a linear equation solver tailored for this application
- Optionally project onto constraints after a successful integration step. This feature is primarily intended for application to constraints which are differentiated to reduce the index of the problem. Differentiating is required to reduce the index to 1 (0 is even better) and we recommend this option when this is done.
- Use step selection logic that provides smoother error control
- Provide all problem data (partials, residuals, linear solver) in one user-written evaluation routine.
- Optionally provide problem data with a "*reverse* communication" interface

The routine uses forward *or reverse* communication to obtain required values of $\mathbf{g}(t, \mathbf{y}, \mathbf{y}')$, partial derivative information, and to perform user-defined linear algebra computations. There is a default dense or banded matrix solver used – Linpack [3] – unless a replacement is provided. The projection algorithm by default uses Givens rotations to solve an under determined system.

B.1 Program Prototype for DDASLX, Double Precision

B.1.a The Calling Routine

EXTERNAL DDASF

DOUBLE PRECISION T, Y(≥NEQ), YPRIME(≥NEQ), TOUT, RTOL(≥1 or NEQ**), ATOL(**≥1 or NEQ**), RWORK(=LRW)**

INTEGER NEQ, INFO(N ≥16), IDID, LRW, IWORK(=LIW),LIW

The user must assign values to **T, Y(1:NEQ), YPRIME(1:NEQ)** and **TOUT**. Set flags in array **INFO(:)** and assign values of **ATOL(:)** and **RTOL(:)**.

```
DO
  CALL DDASLX(DDASF, NEQ, T, Y,
    YPRIME, TOUT, INFO, RTOL, ATOL,
    IDID, RWORK, LRW, IWORK, LIW)
  {Exit when finished.}
END DO
```

Computed quantities are returned in **T, Y(:)**, and **YPRIME(:).**

B.1.b Argument Definitions

DDASF [external] The name of the routine providing system information, $\mathbf{g}(t, \mathbf{y}, \mathbf{y}')$, system derivative information (optional), $D = (\partial \mathbf{g}/\partial \mathbf{y}) + c(\partial \mathbf{g}/\partial \mathbf{y}')$, and (optionally) solving systems of linear equations. The integrator-provided value c depends on the stepsize and method. A description of this interface is given below. Use this dummy name when reverse communication is used for all system information.

NEQ [in] The number of equations to be solved.

T [inout] The current value of the independent variable t. Initially this is set to the starting t.

Y(1:NEQ) [inout] This array contains the solution components $\mathbf{y}$ at t. Set to the initial values for $\mathbf{y}$ as input.

YPRIME(1:NEQ) [inout] This array contains the derivatives $\mathbf{y}'$ of the solution components at t. Set initial values for $\mathbf{y}'$, as input. It is *not necessary* to begin with $\mathbf{y}'$ such that $\mathbf{g}(t, \mathbf{y}, \mathbf{y}') = 0$. See **INFO(11)**, below.

TOUT [in] This is a point where a solution $(t, \mathbf{y}, \mathbf{y}')$ is desired. A value of **TOUT** less than the initial value of t causes the integration to go from larger to smaller values. Otherwise the integration increases from the initial t to the value **TOUT**. The integration is likely to proceed a small distance past **TOUT** unless **INFO(4)** has been used, see below.

INFO(1:N) [in] . The basic task of the code is to solve the system from the initial t to **TOUT** and return an answer at **TOUT**. The parameters **INFO(1:N)** are an integer array that communicates exactly how this task is carried out. (See page 4 below for details.) The size **N** of this array must be at least **16**.

RTOL(:), ATOL(:) You must assign relative (**RTOL**) and absolute (**ATOL**) error tolerances to tell the code how accurately you want the solution to be computed. They must be defined as variables because the code may change them. You have two choices: Both are arrays of size 1, (**INFO(2)=0**), or both are arrays of size **NEQ**, (**INFO(2)=1**). In either case all components must be non-negative. The tolerances are used by the code in a local error test at each step which requires roughly that $|\mathit{local\ error}| \leq \mathit{RTOL}|y| + \mathit{ATOL}$, for each vector component. (More specifically, a root-mean-square norm is used to measure the size of vectors, and the error test uses the magnitude of the solution at the beginning of the step.) The true (global) error is the difference between the true solution of the initial value problem and the computed approximation. This code only controls the local error at each step. Frequently, the true accuracy of the computed **Y** is comparable to the error tolerances. This code will usually, but not always, deliver a more accurate solution if you reduce the tolerances and integrate again. By comparing two such solutions you can get an idea of the true error in the solution at the bigger tolerances. Setting **ATOL=0**. results in a pure relative error test on that component. Setting **RTOL=0**. results in a pure absolute error test on that component. A mixed test with non-zero **RTOL** and **ATOL** corresponds roughly to a relative error test when the solution component is much bigger than **ATOL** and to an absolute error test when the solution component is smaller than the threshold **ATOL**. The code will not attempt to compute a solution at an unreasonable accuracy. It will advise you if you ask for too much accuracy and inform you as to the maximum accuracy it believes possible.

IDID [out] This integer reports what the code did. You must monitor this variable to decide what action to take next, see page 6.

RWORK(:) Dimension this work array of size **LRW** in your calling program.

LRW The declared size of the **RWORK** array. You must have (**MAXORD** = 5 unless changed by **INFO**(9)) **LRW** $\geq$ 45 + (**MAXORD** + 2 $*$ **INFO**(10) + 4) $*$ **NEQ**+

|**INFO(5)**| **= 1, 2, 7, 8, 11, 12:** + **NEQ**^{2}, dense Jacobian case, or

|**INFO(5)**| **= 3, 9, 13:** + (2*ML+MU+1)***NEQ**, banded user-defined Jacobian case, or

|**INFO(5)**| **= 4, 10, 14:** + (2*ML+MU+1)***NEQ** + 2*(**NEQ**/(ML+MU+1)+1), banded finite-difference-generated Jacobian case, or

|**INFO(5)**| **= 5 or 6:** + 0, the Jacobian is stored in the calling program or subroutine **DDASF**.

IWORK(:) Dimension this integer work array of size **LIW** in your calling program.

LIW Set it to the declared size of the **IWORK** array: **LIW** $\geq$ **30+NEQ.**

B.2 The user supplied subroutine for computing problem data

This user defined subroutine defines the system of differential/algebraic equations and related linear algebra operations. It has the form

```
SUBROUTINE DDASF (T, Y, YPRIME,
  DELTA, D, LDD, C, IRES, RWORK,
  IWORK)
```

For the given values of **T**, **Y** and **YPRIME**, the routine normally returns the residual of the differential/algebraic system **DELTA** = $\mathbf{g}(t, \mathbf{y}, \mathbf{y}')$. The array **DELTA(1:NEQ+INFO(10))** is a vector which contains the output for **DDASF**. The array **D(1:LDD,1:NEQ)** is normally used for providing partial derivatives of the system and constraint equations that arise from differentiation and reduction of the system index to **1** or **0**.

You must declare the name **DDASF** in an **EXTERNAL** statement in your program unit that calls **DDASLX**. If you use reverse communication, **INFO(5)** < 0, and |**INFO(5)**| < 6, just use the name **DDASF**.

This provided routine is a "stub" which is not called when using using reverse communication and will print an error message if called. Declare **Y(:)**, **YPRIME(:)** and **DELTA(:)** as arrays in routine **DDASF**. Respond to requests using the value of **IRES**. Use a **SAVE** statement for the local variables that are required to maintain their values between calls. These variables can also be stored in open locations of **RWORK, IWORK)**. If you have declared **LRW** $> lrw$, where lrw is the required lower bound for **LRW**, and similarly for **LIW**, then locations **RWORK(**lrw**+1:LRW)** and **IWORK(**liw**+1:LIW)** are available to pass data between your main program and **DDASF**.

IRES=0 Initialize subroutine parameters or any storage requirements. If this does not apply, then ignore this value and return without any further action. Do not alter **DELTA, D or C** when **IRES = 0**, but y values can be set.

IRES=1 Evaluate the residual of the differential/algebraic system. Place values in the array **DELTA(1:NEQ)** $= \mathbf{F}(t, \mathbf{y}, \mathbf{y}')$.

IRES=2 This case occurs for $|$**INFO(5)**$| = 2$, 4, 5, 6, 8, and 10. Evaluate the partial derivatives, $\mathbf{D}(i,j) = (\partial F_i/\partial y_j) + c(\partial F_i/\partial y'_j)$, The scalar c is an input argument to **DDASF**. For the given values of **T, Y, YPRIME,** the routine must evaluate the non-zero partial derivatives for each equation, and store these values in the matrix **D.** For dense or banded matrices, stored in the work space **RWORK(:)**, the elements of **D** are set to zero before each call to **DDASF** so only non-zero elements need to be defined. Do not alter **T, Y, YPRIME**, or **C**. The way you must store the elements into **D** depends on the structure of the matrix, indicated by **INFO(5)**:

- **INFO(5)=2, 8, and 12** Full (dense) matrix. When you evaluate the (non-zero) partial derivative of equation i with respect to variable j, you must store it in **D** according to $\mathbf{D}(i,j) = (\partial F_i/\partial y_j) + c(\partial F_i/\partial y'_j)$.
- **INFO(5)=4, 10, and 14** Banded Jacobian with **ML** lower and **MU** upper diagonal bands (refer to **INFO(6)** description of **ML** and **MU**). Give **D** a first dimension of **2*ML+MU+1**. When you evaluate the (non-zero) partial derivative of equation i with respect to variable j, you must store it in **D** according to $irow = i - j +$ **ML+MU+1**, $\mathbf{D}(irow, j) = (\partial F_i/\partial y_j) + c(\partial F_i/\partial y'_j)$.
- **INFO(5) = 5 and 6** The array **D** is a dummy argument and is not used. Save the partials $(\partial F_i/\partial y_j) + c(\partial F_i/\partial y'_j)$ in the subroutine **DDASF**. This case requires that you factor the matrix (if you ever do) at this time as the value **IRES**=3 is not provided in this case. As for that case you should set **IRES** $= 0$ to flag the fact that there were no problems in obtaining the factorization.

IRES=3 This case occurs for $|$**INFO(5)**$| > 6$. Factor the matrix of partials. Prepare to solve systems involving the matrix $D = (\partial F_i/\partial y_j) + c(\partial F_i/\partial y'_j)$. The solution method can be a convenient one of the user's choice. If the matrix is non-singular it is important to return **IRES=0** as a signal. Otherwise return **IRES=3**, if the system is numerically singular.

IRES=4 Solve a linear algebraic system using the matrix of partials D, *i.e.* solve $Dw = r$ for w. The vector r is input in array **DELTA(:)**. The solution w overwrites **DELTA(:)**. If for any reason the system cannot be solved, return $w = r$ as the approximate solution. This may cause the integrator to take corrective action such as reducing the step-size or the order of the formulas used. This situation may occur when iteratively solving a linear system, but requiring an excessive number of iterations.

IRES=5 This occurs only if **INFO(10)** $\neq 0$. Compute the residual and partials for projecting the solution **Y(:)** onto the constraints $\mathbf{G}(\mathbf{t}, \mathbf{y}) = 0$ after a step has been computed and the corrector equation has converged. Compute the partial derivatives $\partial G/\partial \mathbf{y}$ in **D(NEQ+1:NEQ+INFO(10),1:NEQ)** and the residual of the constraints $\mathbf{G}(\mathbf{t}, \mathbf{y})$ in **DELTA(NEQ+1:NEQ+INFO(10))**.

If you are handling the linear algebra ($|$**INFO(5)**$| \geq 5$) then you should also compute the projection and residual vector norm and store it in **DELTA(1:NEQ+INFO(10))**. The **DDASLX** code applies the projection, **Y(:)=Y(:) − DELTA(:)**. (Note that the flag is given to **DDASF** if **INFO(5)** ≥ 0, and else is provided using reverse communication. If for example you are using forward communication for derivatives and doing you own linear algebra using reverse communication, you will need to either deal with the linear algebra in **DDASF**, or call a routine from there that will do the job.)

The remaining cases **IRES=6,7,8** occur when using the routine **DDASLS** for computing starting values of $\mathbf{y}'$. Code does not have to be provided for these values of **IRES** if **DDASLS** is not being used. For systems of **index 0** the evaluation cases [**IRES=6,8**] will not occur, i.e. only code for [**IRES=7**] must be provided.

IRES=6 Evaluate the partial with respect to t of the

differential/algebraic system. Place values in the array **DELTA(1:NEQ)** $= \mathbf{F_t}(t_0, \mathbf{y_0}, \mathbf{y}')$. This case occurs if the system has **index 1** or higher.

IRES=7 This case occurs with either [**INFO(5) = 2, 4**]. Evaluate the partial derivatives, **D(i,j)** $= \partial \mathbf{F_i}/\partial \mathbf{y'_j}$. For the values of **T, Y, YPRIME,** the routine must evaluate the non-zero partial derivatives for each equation, and store these values in the array **D(1:NEQ,1:NEQ**. It is not necessary to store zero values. The way you must store the elements into **D(:,:)** depends on the structure of the matrix, indicated by **INFO(5)**. This structure is either a dense or banded representation.

IRES=8 This case occurs with either [**INFO(5) = 2, 4**]. Evaluate the partial derivatives, **D(i,j)** $= \partial \mathbf{F_i}/\partial \mathbf{y_j}$. For the values of **T, Y, YPRIME**, the routine must evaluate the non-zero partial derivatives for each equation, and store these values in the array **D(1:NEQ,1:NEQ**. The way you store the elements in **D(:,:)** depends on whether this is a dense or banded representation.

Ordinarily subroutine **DDASF** should not change the value of **IRES**. The following values can be set for special cases. Do not return these values when using the routine **DDASLS**.

0 This *must* be set if you are factoring the iteration matrix, to let **DDASLX** know that your matrix is not singular.

−1 Some kind of difficulty has been encountered. This causes **DDASLX** to reduce the stepsize or order which may cure the problem.

−2 Return immediately to the main program, for one reason or the other it is time to quit.

< −2 This has the same effect as setting **INFO(6)** to the negative of this number. It provides a way of turning on debug print at any time.

B.3 Setting options using the array INFO

Use the **INFO(:)** array to give the code more details about how you want your problem solved. This array should be dimensioned of size **16**. You must set all of the following items. The simplest use of the code corresponds to setting all values of **INFO** to 0, which gives the default actions.

INFO(1) Must be set by the user at the beginning of a new problem. Set by **DDASLX** to nonzero values as the integration proceeds. Set **INFO(1) = 0** at the first call for this problem. For continuation or reverse communication calls this will have a value **INFO(1) = 1**.

INFO(2) How much accuracy you want of your solution is specified by the error tolerances **RTOL** and **ATOL**. If **INFO(2) = 0, RTOL** and **ATOL** are scalars and these values are used for error control on all equations. If **INFO(2) = 1, RTOL(:)** and **ATOL(:)** are arrays of size **NEQ**. The i^{th} entries are used on the i^{th} equation.

INFO(3) The code integrates from **T** in the direction of **TOUT** by signed steps. If you wish, it will return the computed solution and derivative at the next intermediate step (the intermediate-output mode) or **TOUT**, whichever comes first. This is a good way to proceed if you want to see the behavior of the solution. If you must have solutions at a great many nonspecific **TOUT** points, this code will compute them efficiently.
Set **INFO(3) = 0** to return the solution only at **TOUT** (and not at the next intermediate step).
Set **INFO(3) = 1** to return the solution at the next intermediate step.

INFO(4) To efficiently handle solutions at many values **TOUT**, this code may integrate past **TOUT** and interpolate to obtain the result at **TOUT**. Sometimes it is not possible to integrate beyond some point **TSTOP** because the equation changes there or it is not defined past that point. Then you must tell the code not to go past this point.
Set **INFO(4)=0** so the integration can proceed without any restrictions on the independent variable **T.**
Set **INFO(4)=1** and define the stopping point **TSTOP** by setting **RWORK(2)=TSTOP.**

INFO(5) To solve differential/algebraic problems it is necessary to use a matrix of partial derivatives for the system of differential equations. This flag must be set to give information about your matrix. The first column in the following table gives the value to use for the cases as checked in later columns. The later columns are defined by:

A Full dense matrix.

B Banded matrix.

C Matrix with structure unknown to **DDASLX**.

D Partials computed using differences.

E Partial computed by the user's code.

F User does linear algebra in **DDASF**.

G User does linear algebra using reverse communication.

	A	B	C	D	E	F	G
1	x			x			
2	x				x		
3		x		x			
4		x			x		
5			x			x	
6			x				x
7	x			x		x	
8	x				x	x	
9		x		x		x	
10		x			x	x	
11	x			x			x
12	x				x		x
13		x		x			x
14		x			x		x

If **INFO(5)** < 0 all is as above, except instead of calling **DDASF** to compute **g** and all partials, reverse communication is used. The value 0 is treated the same as 1.

Although it is less trouble for you to have the code compute derivatives by numerical differencing, the solution will be more reliable and efficient if you provide the derivatives using formulas.

When B is checked, $D = (\partial \mathbf{F}/\partial \mathbf{y}) + c(\partial \mathbf{F}/\partial \mathbf{y}')$, is banded. Here c is a factor determined by **DDASLX**. The differential equation is said to have half-bandwidths **ML** (lower) and **MU** (upper) if D involves only some of the unknowns y_j with $i-$**ML**) $\leq$ **j**$\leq$ ($i+$**MU**) for all $i = 1, ...,$**NEQ**. Thus, **ML** and **MU** are the widths of the lower and upper parts of the band, with the main diagonal being excluded. If the matrix is stored in the **RWORK(:)** array and you do not indicate that the equation has a banded matrix of partial derivatives, the code works with a full array of **NEQ²** elements, stored in the conventional Fortran array style. Computations with banded matrices typically require less time and storage than with full matrices, if **2*ML+MU** $<$ **NEQ**. If you tell the code that the matrix of partial derivatives has a banded structure and you want to provide subroutine **DDASF** to compute the partial derivatives, then you must store the elements of the matrix in the Linpack band-matrix specification, indicated in the description of **DDASF**.
Provide the lower **ML** and upper **MU** bandwidths by setting **IWORK(1)=ML** and **IWORK(2)=MU**.

INFO(6) Set the level of debugging print. A value of 0 gives no print. Debug print can also be specified by setting negative value for **IRES** after starting. This is a 7 digit number $d_6d_5d_4d_3d_2d_1d_0$ defining what to print as follows.

d_0 Print on entry to **DDASLX**

0. No print
1. IDID, INFO(1), NEQ, T, TOUT
2. The above + **y**, **y**′
3. The above + Tolerances
4. The above + INFO(1:12)

d_1 Print on exit from **DDASLX**. Print is as for d_0.

d_2 Before a call to **DDASF** (or return for reverse communication that would ordinarily call **DDASF**).

0. No print.
1. Print T, IDID (which is IRES in this case)
2. The above + anything vectors used in the computations, except for **y**, and **y**′.
3. Print **y**, and **y**′.
4. Print matrix if used in computation.

d_3 As for the case above, except print is for what is in the locations recently changed.

d_4 Internal print inside subroutine **DDASTP**.

0. No print
1. **y**, **y**′ and corrections.
2. The above + error weights
3. The above + difference tables
4. The above + integration coefficients

d_5 Determines how much of WORK and IWORK are printed, when there is other print.

0. No print
1. Always print **IWORK(1:22)**
2. Always print **WORK(1:10)**
3. Always print both of the above.

d_6 For turning off, or limiting the amount of print.

0. No effect
1. No effect, but gives a way to specify a value of 0, 1 or 2 when passing a negative value of IRES after starting.

> 1. Print data for just this many of the first variables, and just this many of the first rows in matrices when variables or matrices are printed.

INFO(7) You can specify a maximum stepsize, so that the code will avoid passing over very large regions.
Set **INFO(7)=0** for the code to decide on its own maximum stepsize.
Set **INFO(7)=1** and define the maximum stepsize **HMAX** by setting **RWORK(3)=HMAX**.

INFO(8) Differential/algebraic problems may occasionally suffer from severe scaling difficulties on the first step. If you know the scaling of your problem,

you can help avoid this problem by specifying an initial stepsize **H0**.

Set **INFO(8)=0** for the code to define its own initial stepsize.

Set **INFO(8)=1** and define the initial stepsize **H0** by setting **RWORK(4)=H0.**

INFO(9) If lack of storage is an issue, or if the problem itself should use lower order formulas, one can restrict the maximum order of the backward differentiation formulas, **MAXORD**. The default value is 5. For each order decrease below 5, the code requires **NEQ** fewer locations. However lowering **MAXORD** may slow the code. In any case, you must have $1 \leq$ **MAXORD** ≤ 5.

Set **INFO(9)=0** for the maximum order to default to 5.

Set **INFO(9)**=k to set the maximum order to $k \leq 5$.

INFO(10) Set this to the number of constraints you wish to impose on the solution. This option should be used when the problem has an **index** > 1 and constraints are differentiated to reduce the index. The code will perform better if the index is reduced to 0. This option must not be used when [**INFO(5) = 3,4**]. The calculation of the residuals on the constraints for **G** are stored in positions [**NEQ+1:NEQ+INFO(10)**] of **DELTA(:)**. The partials $\partial \mathbf{G}/\partial \mathbf{y}$ are stored in rows [**NEQ+1:NEQ+INFO(10)**] of **D**.

INFO(11) DDASLX normally requires the initial **T**, **Y**, and **YPRIME** to be consistent. That is, you must have $\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = 0$ at the initial time. If you do not know the initial derivative or $\mathbf{y}'$, you can let **DDASLX** compute it.

Set **INFO(11) = 0** if the initial **T, Y, YPRIME** are consistent. If the initial $\mathbf{y}'$ values are not known we suggest using the provided starting routine **DDASLS** 8.

Set **INFO(11) = 1**, and set **YPRIME** to an initial approximation. If you have no idea what **YPRIME** should be, set it to zero. Note that the initial y should be such that there must exist a $\mathbf{y}'$ so that $\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = 0$.

INFO(12) DDASLX normally allows up to 500 internal steps between output points.

Set **INFO(12)=0** for the code to use up to 500 internal steps between output points.

Set **INFO(12)**=k and the code will use up to k internal steps between output points.

INFO(13) DDASLX normally uses the smoothed step control algorithm described in [4].

Set **INFO(13)=0** for the code to the step control method of Söderlind. We consider this superior to the logic used by the original Petzold code **DASSL**.

Set **INFO(13)=1** and the code will use the original step control logic.

INFO(14:16) Not used currently by the code, but must be set to 0.

B.4 Using Reverse Communication

When reverse communication is used certain arguments ordinarily passed to **DDASF**, are stored in **IWORK** and **RWORK** as follows.

IRES is in **IWORK(3)**.

DELTA() starts in **RWORK(IWORK(4))**.

D() start in **RWORK(IWORK(5))**.

C is in **RWORK(1)**

A return to the user's code with **IDID=4** is a signal that what is computed in **DDASF** when using forward communication should be computed at this time using the above replacements for the arguments to **DDASF**.

B.5 Modifications for single precision

Change names starting with **DDASLX** and **DDASF** to start with **SDASLX** and **SDASF**. Change all DOUBLE PRECISION type statements to REAL.

B.6 Optional Norm Routine

The **DDASLX** package provides a weighted norm **DDASNM** to measure the size of vectors such as the estimated error in each step. A subprogram may be exchanged for this routine: **DOUBLE PRECISION FUNCTION DDASNM (NEQ, V, WT, RWORK, IWORK)** Arrays **V(1:NEQ),WT(1:NEQ)** are used with this norm. Here, **V(:)** is the vector whose norm is to be computed, and **WT(:)** is a vector of weights. The routine **DDASNM** is included with **DDASLX,** and computes the weighted root-mean-square norm given by the formula **DDASNM** = **SQRT((1./NEQ)*SUM(V(:)/WT(:))**2).** This norm is suitable for most problems. In special cases, it may be more convenient or efficient to define your own norm by writing a replacement function subprogram.

B.7 Values of IDID on Return from DDASLX

IDID	Task Completed or Ongoing
1	A step was successfully taken in the intermediate-output mode. The code has not yet reached **TOUT**.
2	The integration to **TSTOP** was successfully completed (**T=TSTOP**) by stepping exactly to **TSTOP**.
3	The integration to **TOUT** was successfully completed (**T=TOUT**) by stepping past **TOUT**. **Y(:)** is obtained by interpolation. **YPRIME(:)** is obtained by interpolation.

IDID **Task Completed or Ongoing**

4 The integration has paused for reverse communication. Respond based on the values of IWORK(3).

IDID **Task Interrupted**

−1 IRES set to −2 by the user.

−2 Accuracy requested exceeds machine precision. RTOL and ATOL have been increased.

−3 There have been too many steps between output points.

Quit or Restart Integration

−4 No convergence due to IRES being set to −1.

−5 A weight for computing error norms is ≤ 0.

−6 The error test has failed repeatedly.

IDID **Invalid input**

−7 Repeated failure of the corrector to converge.

−8 The iteration matrix is singular.

−9 Repeated corrector convergence failures, with singular matrices flagged.

−10 Could not solve for the initial $\mathbf{y}'$.

−11 An **INFO** entry has a value not allowed for that option.

−12 The number of equations was set ≤ 0.

−13 The maximum order does not have a value from 1 to 5.

−14 The size of **RWORK** is too small.

−15 The size of **IWORK** is too small.

−16 An entry of **RTOL** is < 0.

−17 An entry of **ATOL** is < 0.

−18 All entries of **RTOL** and **ATOL** are 0.

−19 The value of **TOUT** is $>$ **TSTOP**.

−20 The maximum stepsize is set ≤ 0.

−21 The current **TOUT** is behind **T**.

−22 The initial stepsize has been set to 0.

−23 **TOUT** is too close to the starting **T**.

−24 **TSTOP** is not consistent with the current **T**.

−25 An illegal bandwidth.

−26 The current **T** and **TOUT** are equal.

−27 Constraints used with band matrices.

−28 When solving constraints for a user defined matrix, IRES was not set to 0.

−29 Constraints being projected onto appear inconsistent at the initial point.

−30 Constraints being projected onto appear inconsistent during the integration.

−31 No appropriate action was taken when code returned with **IDID** < 0.

B.8 Values of RWORK and IWORK on exit

RWORK(:), IWORK(:) Contain information which is often of no interest to the user but is necessary for subsequent calls. However, you may find use for the following.

RWORK(4) The step size **H** to be attempted on the next step;

RWORK(5) The current value of the independent variable, i.e., the farthest point integration has reached. This will be different from **T** only when interpolation has been performed (**IDID=3**).

RWORK(7) The stepsize used on the last successful step.

IWORK(7) The order of the method to be attempted on the next step.

IWORK(8) The order of the method used on the last step.

IWORK(13) The number of steps taken so far.

IWORK(14) The number of evaluations of the residual function so far.

IWORK(15) The number of evaluations of the matrix of partial derivatives needed so far.

IWORK(16) The total number of error test failures so far.

IWORK(17) The total number of convergence test failures so far. This includes singular iteration matrix or related failures.

B.9 Getting Debug Print After Starting

When debugging print is desired, it is frequently inconvenient to have lots of such output prior to the place where problems are known to occur. If you want to start debug print at a place where **INFO** is available, you can simply set **INFO(6)** to the value you would ordinarily set it too when starting. But in **DDASF**, **INFO** is not available. In this case you can set the return value of **IRES** to the negative of the value you would ordinarily set **INFO(6)** to. If this negative value is not < -2, you will need to set d_6 to get the desired result.

In addition is also possible for you to make a direct call to get debug print. This call has the form

CALL DDASDB (KASE, NEQ, T, Y, YPRIME, INFO, RWORK, IWORK, IRES, ATOL, RTOL)

KASE is used to specify the type of print desired. It must be a negative two digit number $-d_1d_0$, where d_1 is the value for one of the digits given in the description of **INFO(6)**, and d_0 gives the digit that this is supposed to correspond to. If you have not made a nonsense choice, this print will be as if **INFO(6)** had $d_{d_0} = d_1$.

If you are not getting this print in **DDASF** or in reverse communication that would ordinarily call **DDASF**, then the **IRES** in the calling sequence should be replaced by **IDID**. Also you can replace **INFO** with

IWORK when **INFO** is not available, and similarly can replace **ATOL** and **RTOL** with **RWORK**.

B.10 Computing Starting Values for Derivatives

When the integration starts it is often true that initial values for $\mathbf{y}(\mathbf{t_0}) = \mathbf{y_0}$ are known. It may be necessary to solve for initial derivative values $\mathbf{y}'(\mathbf{t_0}) = \mathbf{y'_0}$ that achieve consistent conditions $\mathbf{F}(\mathbf{t_0}, \mathbf{y_0}, \mathbf{y'_0}) = \mathbf{0}$. The routine **DDASLS** is provided for this. The integrator **DDASLX** provides an alternative method that is used when **INFO(11)=1** 6.
The assumptions for our starting algorithm are:

- The system has index 0 or index 1
- Initial values $\mathbf{t_0}, \mathbf{y_0}$ are known but maybe not all values $\mathbf{y'_0}$
- All estimated values of $\mathbf{y'_0}$ are meaningful to the problem
- The partials $\partial\mathbf{F}/\partial\mathbf{t} = \mathbf{F_t}$, $\mathbf{F_y}$, and $\mathbf{F_{y'}}$ are computable and continuous at $\mathbf{t_0}, \mathbf{y_0}, \mathbf{y'_0}$. The rank of $\mathbf{F_{y'}}$ must be positive.
- Any values for $\mathbf{y}'$ are allowed to change to achieve $\mathbf{F}(\mathbf{t_0}, \mathbf{y_0}, \mathbf{y'_0}) = \mathbf{0}$
- Any consistent set of values obtained that satisfy $\mathbf{F}(\mathbf{t_0}, \mathbf{y_0}, \mathbf{y'_0}) = \mathbf{0}$ satisfy the constraints $\mathbf{G}(\mathbf{t_0}, \mathbf{y_0}) = \mathbf{0}$.

The routine **DDASLS** is organized so that a user can add additional cases in the evaluation routine **DDASF** written for **DDASLX** that separately compute $\mathbf{F_t}$, $\mathbf{F_y}$ and $\mathbf{F_{y'}}$. The evaluation of $\mathbf{F}(\mathbf{t_0}, \mathbf{y_0}, \mathbf{y'_0})$ uses the same flag value [**IRES=1**] as **DDASLX**. Reverse communication is also supported.

B.11 Program Prototype for Starting Routine DDASLS, Double Precision

B.11.a The Calling Routine

EXTERNAL DDASF

DOUBLE PRECISION T, Y(≥NEQ), YPRIME(≥ NEQ), FTOL, RNKTOL, RWORK(=LRW)

INTEGER NEQ, INFO(N ≥16), IDID, LRW, IWORK(=LIW),LIW

The user must assign values to **T, Y(1:NEQ), and YPRIME(1:NEQ)**. Set flags in array **INFO(:)** and assign values of **FTOL** and **RNKTOL**.

```
CALL DDASLS(DDASF, NEQ, T, Y, &
   YPRIME, INFO, FTOL, RNKTOL, &
   C, LDC, LTD, IDID, &
   RWORK, LRW, IWORK, LIW)
```

Computed quantities are returned in **YPRIME(:).**

B.11.b Argument Definitions

DDASF [external] The name of the routine providing system information, described above, 2 Use this dummy name when reverse communication provides system information. When using reverse communication, a single initialization call to **DDASF** is made with **IRES=0**. One can use this to do any special setup that may be desired, or to set the initial y values.

NEQ [in] The number of equations to be solved.

T [in] The value of the independent variable t. This is set to the starting $t = t_0$.

Y(1:NEQ) [in] Set to the initial values for $\mathbf{y} = \mathbf{y_0}$ as input.

YPRIME(1:NEQ) [inout] This array contains the derivatives $\mathbf{y}'$ of the solution components at t_0. Set some approximate initial values for $\mathbf{y}'$, as input. It is *not necessary* to begin with $\mathbf{y}'$ such that $\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = 0$.

INFO(1:N) [in] . The only task of the code is to solve for initial values of $\mathbf{y}'$ that satisfy $\mathbf{F}(\mathbf{t_0}, \mathbf{y_0}, \mathbf{y'_0}) = \mathbf{0}$. The parameter **INFO(1:N)** is an integer array that communicates how this task is carried out. (See page 4 above for details.) The size **N** of this array must be at least **16**. Values of INFO(:) used have the same meanings as will occur when integrating the system using **DDASLX**. Only the entries **INFO(I), I=1,5,14** are used in the code. **INFO(1)** must be 0 when starting, and is 1 when doing reverse communication. It is set to 0 on all exits.

If the Jacobians are dense matrices set **|INFO(5)|=2**. When the Jacobians are banded matrices, set **|INFO(5)|=4**. To use reverse communication have **INFO(5)< 0**.

For banded Jacobians and index 1, a regularization parameter is required. For this use **INFO(14) = 0** to have the default parameter with value $\mathbf{macheps^{(2./3.)}}$. To use other values set **INFO(14) > 0** and place the regularization parameter in location **RWORK(INFO(14))**.

FTOL, RNKTOL [in] You must assign absolute (**FTOL**) and relative (**RNKTOL**) tolerances to tell the code how accurately you want the initial conditions to be computed and how to determine rank deficiencies. The code computes y'_0 such that $||F(t_0, y_0, y'_0)||_1 \leq$ **FTOL**. A diagonal term, j, of an intermediate upper triangular matrix, with column vector a_j, is classified as near zero if $|a_{j,j}| \leq$ **RNKTOL** $\times ||a_j||_1$.

Suggestion: **RNKTOL** = $\mathbf{macheps}^{(\mathbf{2./3.})}$.

C(1:LDC, 1:LTD) [inout] This is the working array where the partial derivative matrices $F_{y'}$ and F_y will be returned after evaluation by **DDASF** or reverse communication.

LDC [in] This is the leading dimension of the array **C**$(:,:)$. For dense Jacobian matrices it has a value **LDC** $\geq$ **NEQ** when $F_{y'}$ has full rank. This occurs with an index 0 problem. When $F_{y'}$ is rank deficient, **LDC** $\geq \mathbf{2} * \mathbf{NEQ}$.

When $F_{y'}$ is rank deficient and the Jacobian matrices are banded, **LDC** $\geq \mathbf{4} * \mathbf{ML} + \mathbf{2} * \mathbf{MU} + \mathbf{4}$. This occurs when the problem has index 1.

LTD [in] This is the second dimension of the array **C**$(:,:)$. For dense Jacobian matrices it has a value **LTD** $\geq$ **NEQ**. This occurs with an index 0 or index 1 problems.

When $F_{y'}$ is rank deficient, and the Jacobian matrices are banded, **LTD** $\geq \mathbf{2} * \mathbf{NEQ}$ is required. This occurs when the problem has index 1.

IDID [out] This integer reports what the code did or if evaluations are requested using reverse communication.

IDID=0 The system has index 0 or 1 and values for y'_0 were computed.

IDID=-1 An error or exceptional condition was noted. Details and reasons are flagged by various values in **IWORK(1)**.9

IDID $\geq$ **1** Compute requested value using reverse communication. Use the values **IRES** = **IWORK(3)** and **IR** = **IWORK(4)**:

IRES=1 Evaluate and place $F(t_0, y_0, y'_0)$ in **RWORK(IR+1:IR+NEQ)**.

IRES=6 Evaluate and place F_t in **RWORK(IR+1:IR+NEQ)**.

IRES=7 Evaluate and place $F_{y'}$ in **C(:,1:NEQ)** based on the storage mode used for dense or banded Jacobians.

IRES=8 Evaluate and place F_y in **C(:,1:NEQ)** based on the storage mode used for dense or banded Jacobians.

Re-enter the routine **DDASLS** and continue computation.

RWORK(:) [inout] Dimension this work array of size **LRW** in your calling program.

LRW [in] The declared size of the **RWORK** array. **LRW**$\geq$**2*NEQ+2** for dense Jacobians or **LRW**$\geq$**7*NEQ+2** for banded Jacobians.

IWORK(:) [out] Dimension this integer work array of size **LIW** in your calling program.

LIW [in] The declared size of the **RWORK** array: **LIW** $\geq$ **2*NEQ+8.**

B.12 Flag IDID=0,−1: IWORK(1)

0 The system has index 0 and consistent initial values for y' were computed.

1 The system has index 1 and consistent initial values for y' were computed.

2 The system has index 0 but the system $F(t, y, y')$ is not consistent using the tolerance **FTOL**. It should be consistent so **FTOL** may be too small for this problem. Iterations are done while the l_1 norm of F is $< 1/4$ the l_1 norm of F from the previous iteration.

3 The system has index 1 but the system $F(t, y, y')$ is not consistent using the tolerance **FTOL**.

4 The system appears to have an index > 1. The use of **DDASLX** does not apply to this system.

5 The system has rank $F_{y'} = 0$. This is not a DAE.

6 The value **NEQ** ≤ 0, i.e. no system.

7 The value **LDC** is too small. Must be $\geq$ **NEQ** for index 0 problems and dense Jacobians.
Must be $\geq (\mathbf{2} * \mathbf{ML} + \mathbf{MU} + \mathbf{1})$, for banded Jacobians, index 0.
Here **ML, MU** are the lower and upper band widths. Provide the lower **ML** and upper **MU** bandwidths by setting **IWORK(1)=ML** and **IWORK(2)=MU**.

8 The value **LDC** is too small. Must be $\geq \mathbf{2} * \mathbf{NEQ}$ for index 1 problems and dense Jacobians.
Must be $\geq \mathbf{4} * \mathbf{ML} + \mathbf{2} * \mathbf{MU} + \mathbf{4}$ if problem is banded with index 1.

9 The value **LRW** is too small. Must be $\geq \mathbf{2} * \mathbf{NEQ} + \mathbf{2}$ for dense systems and $\geq \mathbf{7} * \mathbf{NEQ} + \mathbf{2}$ for banded systems.

10 The value **LIW** is too small. Must be $\geq \mathbf{2} * \mathbf{NEQ} + \mathbf{8}$.

11 Must have $|\mathbf{INFO}(\mathbf{5})| = \mathbf{2}, \mathbf{4}$. Routine supports dense or banded matrices only. The user computes derivatives in forward **INFO**$(\mathbf{5}) > \mathbf{0}$ or reverse communication **INFO**$(\mathbf{5}) < \mathbf{0}$.

12 Must have **FTOL** $> \mathbf{0}$. Now ≤ 0.

13 Must have **RNKTOL** $\geq \mathbf{0}$. Now < 0.

14 Must have the number of sub- and super-bandwidth parameters **ML**, **MU** $\geq$ **0**. Now one of them is < 0.

15 Must have second dimension **LTD** of **C(:,:)** $\geq$ **NEQ** for dense or banded systems. Value is now $<$ **NEQ**.

For banded systems of index 1 **LTD** $\geq 2 * NEQ$. This is checked when the index is > 0, or $F_{y'}$ is singular.

There are calls made to **DDASLS** in Examples 1,2,3 and 5 that solve for initial values of y'.

C. Examples and Remarks

C.1 An Index 2 DAE Solved Using Constraints

Gear and Petzold, see [5], discuss the index 2 DAE system

$$y_1' + \eta t y_2' + (1+\eta) y_2 - sin(t) = 0$$
$$y_1 + \eta t y_2 - cos(t) = 0$$

with $y_1(0) = 1; y_2(0) = 0; \eta = 10; 0 \leq t \leq 10$.

Since **DDASLX** solves "index 0" or "index 1" problems, the problem must be transformed to one of "index 1". Differentiating the last equation results in an "index 1" system, so the integrator applies. The given equation must remain satisfied, so it is added as a *constraint*. Without this step the solution may seriously drift away from the given equation. Thus the example uses a program option, and provides derivative and residual information for the index 1 system and the constraint.

Differentiating the last equation twice results in an "index 0" system. The equations for the "index 2" and "index 1" systems are added as constraints and the integration is done once again. The results agree within the tolerances requested, as expected. This "index 1" problem, with the requested accuracy, takes more residual evaluations than the "index 0" problem. However, obtaining the "index 0" problem requires two derivative computations (analytic), whereas the "index 1" problem requires one derivative.

This example illustrates a mathematical change of the problem so that **DDASLX** applies. Then constraints are added to the stated problem, in two ways. The two initial derivative values y_1', y_2'; are computed using the routine **DDASLS**. Otherwise the integrator uses default values and linear solver code. The file used, **drddasl1.f** and the result of running this problem are given at the end of this chapter.

C.2 A Stiff ODE Test Problem

Problem E5 of Enright and Pryce, see [6], is an explicit stiff ODE, which we express as

$$\mathbf{g} = \begin{bmatrix} -b_1 y_1 - b_2 y_1 y_3 - y_1' \\ b_1 y_1 - b_3 y_2 y_3 - y_2' \\ b_1 y_1 - b_2 y_1 y_3 + b_4 y_4 - b_3 y_2 y_3 - y_3' \\ b_2 y_1 y_3 - b_4 y_4 - y_4' \end{bmatrix}$$

$$\mathbf{y}(0) = \begin{bmatrix} b_0 = 1.76 \times 10^{-3} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} b_1 = 7.89 \times 10^{-10} \\ b_2 = 1.1 \times 10^{7} \\ b_3 = 1.13 \times 10^{9} \\ b_4 = 1.13 \times 10^{3} \end{bmatrix}$$

The derivative matrix required by **DDASLX** is

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}} + c_j \frac{\partial \mathbf{g}}{\partial \mathbf{y'}} = \frac{\partial \mathbf{g}}{\partial \mathbf{y}} - c_j I_4 =$$

$$\begin{bmatrix} -b_1 - b_2 y_3 - c_j & 0 & -b_2 y_1 & 0 \\ b_1 & -b_3 y_3 - c_j & -b_3 y_2 & 0 \\ b_1 - b_2 y_3 & -b_3 y_3 & -b_2 y_1 - b_3 y_2 - c_j & b_4 \\ b_2 y_3 & 0 & b_2 y_1 & -b_4 - c_j \end{bmatrix}$$

We integrate between each value of a list of output points $t_j,\ j = 1, ..., k$. In Example 4 below we will show how to efficiently integrate variational equations with respect to the parameters $b_i,\ i = 0, 1, 2, 3, 4$. The subroutine **DDASSF2** defines the required system information of this problem. The main program **DRDDASL2** integrates the system over the sequence of intervals $[t_{i-1}, t_i],\ i = 1, ..., k$. The partial derivative formulas are used **(INFO(5)=1)** in the first integration, while finite differences **(INFO(5)=0)** are used instead of the formulas for the second integration. This usage of **DDASLX** illustrates a simple form for integrating a system of DAEs. Due to the fact that the upper triangle for the derivative matrix has three non-zeros, a special method is used for this problem. Three plane rotations are constructed from the right to obtain a lower triangular matrix. The solving step uses this decomposition. Both of these steps are added to the subroutine **DDASSF2**, indicated by the option **INFO(16)=2**. [The code and output listing are available in the download.]

C.3 A DAE Problem, the Swinging Pendulum

This material is found in [1, p. 154]. A ball of mass m, swinging on a thin bar, with dynamic tension y_5, of length l under the influence of gravity g, is expressed as

the first order system

$$\begin{aligned} y_1' &= y_3 \\ y_2' &= y_4 \\ m y_3' &= -y_1 y_5 \\ m y_4' &= -y_2 y_5 - mg \\ y_1^2 + y_2^2 &= l^2. \end{aligned}$$

This system cannot be solved by **DDASLX**, since it is an "index 3" problem. By differentiating the last equation twice and substituting from the other equations, one obtains two additional algebraic equations $y_1 y_3 + y_2 y_4 = 0$ and $m(y_3^2 + y_4^2) - m g y_2 - l^2 y_5 = 0$. Use of only the final equation, in place of the given constraint, results in an "index 1" problem. Thus **DDASLX** can be used to integrate the system. As noted in Petzold, page 155–156, there is a tendency for the integrated system to drift from the two alternate constraints. These residuals become large particularly after long integration times. In Section C.5 below we show how to stay on these constraints. The system is expressed by

$$\mathbf{g} = \begin{bmatrix} y_3 - y_1' \\ y_4 - y_2' \\ -y_1 y_5 - m y_3' \\ -y_2 y_5 - m y_4' - mg \\ m(y_3^2 + y_4^2) - m g y_2 - l^2 y_5 \end{bmatrix}.$$

The partial derivatives are

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}} + c_j \frac{\partial \mathbf{g}}{\partial \mathbf{y}'} = \begin{bmatrix} -c_j & 0 & 1 & 0 & 0 \\ 0 & -c_j & 0 & 1 & 0 \\ -y_5 & 0 & -m c_j & 0 & -y_1 \\ 0 & -y_5 & 0 & -m c_j & -y_2 \\ 0 & -mg & 2 m y_3 & 2 m y_4 & -l^2 \end{bmatrix}.$$

Our main program **DRDDASL3** integrates this system over the interval $0 \le t \le 10$. The subroutine **DDASSF3** illustrates additional features that our software permits. For the values **IRES=1** and **IRES=2,** function and partial derivative information is provided. For **IRES=3** and **IRES=4** we factor and then solve a system of five algebraic equations. In this example the matrix is reduced to upper triangular form using five plane rotations. The rotation data that eliminates the entry at row 3, column 1 also eliminates the entry at row 4, column 2. Thus the first rotation does double duty. The triangularization requires three more plane rotations, applied between the main diagonal and row 5, columns 2, 3, and 4. To avoids divides in the solving step, signaled by **IRES=4**, we reciprocate the diagonals of the upper triangular matrix during the decomposition. It it not required that a solution of the linear system be provided in the routine **DDASS3**. We override the default solver to give an example of this feature applied to this special problem. [The code and output listing are available in the download.]

C.4 Variational Equations with an ODE Test Problem

The test problem E5, see Section C.2 on page 10, involves the vector of parameters $\mathbf{p} = [b_0,\ b_1,\ b_2, b_3, b_4]^T$. In applications where this vector itself must be varied, one often needs to compute the variational equations $\partial \mathbf{y} / \partial p_i,\ i = 0,\ 1,\ 2,\ 3,\ 4$. Total differentiation of the DAE system $\mathbf{g}(t, \mathbf{y}, \mathbf{y}') = \mathbf{0}$ with respect to $\mathbf{p}$ results in the variational DAE system

$$\frac{d\mathbf{g}}{d\mathbf{p}} \equiv \frac{\partial \mathbf{g}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{p}} + \frac{\partial \mathbf{g}}{\partial \mathbf{y}'} \frac{\partial \mathbf{y}'}{\partial \mathbf{p}} + \frac{\partial \mathbf{g}}{\partial \mathbf{p}} = 0.$$

In this ODE example note that

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}} = \begin{bmatrix} -b_1 - b_2 y_3 & 0 & -b_2 y_1 & 0 \\ b_1 & -b_3 y_3 & -b_3 y_2 & 0 \\ b_1 - b_2 y_3 & -b_3 y_3 & -b_2 y_1 - b_3 y_2 & b_4 \\ b_2 y_3 & 0 & b_2 y_1 & -b_4 \end{bmatrix}$$

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}'} = -I_4, \text{ and}$$

$$\frac{\partial \mathbf{g}}{\partial \mathbf{p}} = \begin{bmatrix} 0 & -y_1 & -y_1 y_3 & 0 & 0 \\ 0 & y_1 & 0 & -y_2 y_3 & 0 \\ 0 & y_1 & -y_1 y_3 & -y_2 y_3 & y_4 \\ 0 & 0 & y_1 y_3 & 0 & -y_4 \end{bmatrix}$$

Thus the variational part of the DAE system is

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{p}} - \frac{\partial \mathbf{y}'}{\partial \mathbf{p}} + \frac{\partial \mathbf{g}}{\partial \mathbf{p}} = 0.$$

The initial conditions are

$$\frac{\partial \mathbf{y}}{\partial \mathbf{p}}\rfloor_{t=0} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The twenty-four unknowns for **DDASLX** are the four state equations for $\mathbf{y}$ and $\mathbf{y}'$, followed by the twenty variational equations involving $\partial \mathbf{y} / \partial \mathbf{p}$ and $\partial \mathbf{y}' / \partial \mathbf{p}$. However, this combined system has a simple derivative matrix structure: There are six diagonal blocks each with the same 4×4 matrix, namely $(\partial \mathbf{g} / \partial \mathbf{y}) - c_j I_4$. The right-hand sides are different for each block. This observation obviates the need to deal with a 24×24 matrix. Instead we store and factor the 4×4 matrix. With each request to solve a system, we have reduced the problem to the same 4×4 matrix and six different right-hand sides. Thus the functionality we have provided within **DDASLX** for storing derivatives and solving systems has saved storage and work when

dealing with variational equations. This is implemented within the main program using reverse communication. The main program **DRDDASL4** integrates the system and variational equations over the sequence of intervals $[t_{i-1}, t_i]$, $i = 1, ..., k$. Partial derivative formulas are used **(INFO(5)=1)** in the integration. [The code and output listing are available in the download.]

C.5 Swinging Pendulum Problem and Higher Index Constraints

Here we deal with the drift issue mentioned in Section C.3. To this end we consider the "index 1" and "index 0" problem with the additional constraints, including the constraint of total energy,

$$\mathbf{g} = \begin{bmatrix} y_3 - y_1' \\ y_4 - y_2' \\ -y_1 y_5 - m y_3' \\ -y_2 y_5 - m y_4' - mg \\ m(y_3^2 + y_4^2) - mgy_2 - l^2 y_5 \\ y_1 y_3 + y_2 y_4 \\ y_1^2 + y_2^2 - l^2 \\ (1/2)(y_3^2 + y_4^2) + gy_2 \end{bmatrix} = 0.$$

The first five rows of this $\mathbf{g} = \mathbf{F}$ are provided as part of the DAE. The last three rows or $\mathbf{G} = 0$ are satisfied using the option to project onto constraints. The 8×5 derivative matrix is

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}} + c_j \frac{\partial \mathbf{g}}{\partial \mathbf{y}'} = \begin{bmatrix} -c_j & 0 & 1 & 0 & 0 \\ 0 & -c_j & 0 & 1 & 0 \\ -y_5 & 0 & -mc_j & 0 & -y_1 \\ 0 & -y_5 & 0 & -mc_j & -y_2 \\ 0 & -mg & 2my_3 & 2my_4 & -l^2 \\ y_3 & y_4 & y_1 & y_2 & 0 \\ 2y_1 & 2y_2 & 0 & 0 & 0 \\ 0 & g & y_3 & y_2 & 0 \end{bmatrix}.$$

The main program **DRDDASL5** and subroutines, **DDASSFA** for "index 1" and **DDASSFB** for "index 0", combine to use forward communication to evaluate $\mathbf{g}$ and required partial derivatives. See Section C.8 for the equivalent simulation but with numerical differentiation used in place of analytic partial derivatives.

[The code and output listing are available in the download.]

C.6 A Nonlinear Elliptic PDE Using an Iterative Linear Solver

This example shows how **DDASLX** can be applied to a system of ODEs, but with a large number of unknowns. The linear solve step required by the integrator is implemented with a "GMRES" iterative solver, [7]. The code for the solver is located in the file **drddasl6.f** that evaluates the residual function and the partial derivatives.

This example primarily illustrates setting program options that allow replacement of the default linear solver and eliminating the dense partial derivative matrix storage requirements. The problem, see [8], is a nonlinear, elliptic equation on the unit square,

$$F(u) = \nabla^2 u + \lambda e^u = 0.$$

We use the boundary condition is $u = 0$ on the edge of the square, and $\lambda = 6$ for the parameter. For illustration, we solve for an approximate solution $F(u) = 0$ by "continuation."

That is, we introduce the artificial continuation variable t and integrate the equation $F(u) = \partial u/\partial t$, with the initial values $u = 0$. We continue integrating in $t > 0$ as $\partial u/\partial t \to 0$. When we fix a sufficiently large $t > 0$, $F(u) = 0$, approximately.

Our conversion to a discrete problem uses a nine-point formula. This formula is based on a novel blending of the difference approximations $(1/3)\Delta_h U$ plus the rotated formula $(2/3)\Delta_h^{\times} U$. This development is found on pages 192–194, [9]. The blending parameters are reversed from those found on page 194. This strong second order difference scheme relies on a uniform grid, in both space dimensions. This yields the following system of ODEs, using the well-known Method of Lines:

Define

$$u_{i,j} = u(x_i, y_j), x_i = ih, i = 0, 1, \ldots, n+1;$$

$$y_j = jh,\ j = 0, 1, \ldots, n+1;\ h = 1/(n+1)$$

The ODE system becomes

$$g_{i,j} = -8u_{i,j} + u_{i-1,j-1} + u_{i-1,j} + u_{i-1,j+1} + u_{i,j-1} + u_{i,j+1} + u_{i+1,j-1} + u_{i+1,j} + u_{i+1,j+1} + 3h^2 \lambda e^{u_{i,j}} - 3h^2 \frac{\partial u_{i,j}}{\partial t} = 0.$$

This system has n^2 components, and the initial values are set to $u_{i,j} = 0$ for all i, j. The edge values for $u_{i,j}$ remain fixed at the value zero, while the interior values assume non-zero values as the integration proceeds.

[The code and output listing are available in the download.]

C.7 A Banded Linear System

This example illustrates using **DDASLX** in the case of banded matrices $D = (\partial \mathbf{g}/\partial \mathbf{y}) + c(\partial \mathbf{g}/\partial \mathbf{y}')$.

Define the linear system as $\mathbf{g} = \mathbf{A}\mathbf{y} - \mathbf{y}'$. The $n \times n$ matrix $\mathbf{A}$ is constant, and lower triangular. It is also banded. The respective lower and upper bandwidths

are $ml = 5$ and $mu = 0$. These are defined in the same way as found in Linpack, [3].

The matrix size is $n = 25$ and the initial value is $\mathbf{y}(0) = \mathbf{y}_0 = \mathbf{e}_1$, the first column of the $n \times n$ identity matrix, $\mathbf{I}_n$.

All diagonal values of $\mathbf{A}$ are real and negative, so the analytic solution $\mathbf{y}(t) = e^{\mathbf{A}t}\mathbf{y}_0$ is bounded for $t > 0$. Note that $D = \mathbf{A} - c\mathbf{I}_n$ has the same bandwidth dimensions as $\mathbf{A}$.

This problem is integrated over a sequence of intervals $[t_{i-1}, t_i]$, $i = 1, \ldots, k$, $(k = 6)$. Starting with $t_0 = 0$ and $t_1 = 0.01$, each $t_i = 10t_{i-1}$, $i = 2, \ldots, k$. All intervals are essentially an order of magnitude longer than the previous one. An absolute error tolerance is used.

The main program **DRDDASL7** first integrates this system with forward communication to define the residual function **g** and the banded partial matrix D. The Linpack banded solver that comes with **DDASLX** is used for the linear algebra in this first case.

A second integration uses reverse communication to define everything. This includes evaluating the residual function, the partial matrix, and solving linear systems with this partial matrix.

[The code and output listing are available in the download.]

C.8 Swinging Pendulum Problem and Higher Index Constraints - Numerical Differentiation

This example is identical to Section C.5 except for the use of numerical instead of analytic derivatives. The numerical differentiation routine **DJACG**, Chapter 8.4, is used. This example may serve as a guide for a DAE integration when analytic derivatives of **g**, including constraint equations, must be computed numerically.

[The code and output listing are available in the download if the option for numerical differentiation is included. The documentation for the numerical differentiation can be found at `http://mathalacarte.com/cb/mom.fcg/ya89`.]

D. Functional Description

Subroutine **DDASLX** uses the backward differentiation formulas of orders one through five, as described in [1, pp. 115–129] or [10], to solve a system of the above form for y and y'. Values for y and y' at the initial time must be given as input in arrays **Y(1:NEQ)** and **YPRIME(1:NEQ)**. These values must be consistent. Thus if t, y, y' are the given initial values, they must satisfy $\mathbf{F}(t, y, y') = 0$. However, see **INFO(11)** above. The routine solves the system from **T** to **TOUT**. One can continue the solution to get results at additional values of **TOUT**. This is the *interval* mode of operation. Intermediate results can also be obtained by using the *intermediate-output* capability.

References

1. K. E. Brenan, S. L. Campbell, and L. R. Petzold, **Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations**, Society for Industrial and Applied Mathematics, Philadelphia (1996).
2. "Richard J. Hanson and Fred T. Krogh", **"Solving Constrained Differential-Algebraic Systems Using Projections"**. Technical report, "Visual Numerics" (Nov. 2008). Available from `http://mathalacarte.com/fkrogh/pub/SolvingC.pdf`.
3. J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, **LINPACK Users' Guide**, Society for Industrial and Applied Mathematics, Philadelphia (1979) 320 pages.
4. Gustaf Söderlind, *Digital filters in adaptive time-stepping*, **ACM Trans. on Math. Software 29**, 1 (March 2003) 1–26.
5. C. W. Gear, *Differential–algebraic equation index transformations*, **SIAM J. on Scientific Computing 9**, 1 (Jan. 1988) 39–47.
6. W. H. Enright and J. D. Pryce, *Two FORTRAN packages for assessing initial value methods*, **ACM Trans. on Math. Software 13**, 1 (March 1987) 1–27.
7. G. H. Golub and C. F. van Loan, **Matrix Computations, Third Edition**, Johns Hopkins, Baltimore and London (1996) 549 pages.
8. M. E. Hayder, D. E. Keyes, and P. Mehrotra, **A Comparison of PETSC Libary and HPF Implementations of an Archetypal PDE Computation**. TU Doc. NASA/CR-97-206278, NASA Langley Research Center, Hampton, VA (1997).
9. G. E. Forsythe and W. R. Wasow, **Finite Difference Methods for Partial Differential Equations**, J. Wiley and Sons, New York (1960).
10. Linda R. Petzold, **Description of DASSL: A Differential/Algebraic System Solver**. Technical Report SAND82-8637, Sandia National Laboratories (1982).

E. Error Procedures and Restrictions

The routine **DDASLX** is not "thread safe." Nor is it permitted to use the routine recursively. For example you must not call **DDASLX** in the routine **DDASF**.

See page 7 for error conditions and the flags returned in **IDID**.

All error diagnostics and messages are handled by calls to the library subroutines MESS and DMESS or SMESS of Chapter 19.3. The user can change the printing and/or stopping actions taken by the error message program or change the file to which such messages are sent by calling MESS.

F. Supporting Information

The source language is ANSI Fortran 77. The original code author is L. R. Petzold. Changes include a subroutine name change, using one external subprogram for system information, installation of reverse communication, handling constraints, computing initial derivative values, adding Söderlind's stepsize selection algorithm, and writing this document. These were made by R. J. Hanson during the period 1998–2008, with advice from F. T. Krogh.

Entry	Required Files
DDASLX	amach, daxpy, dcopy, ddas1, ddasco, ddasdb, ddasf, ddasgh, ddasin, ddasj, ddaslv, ddaslx, ddasnm, ddastp, ddaswt, ddot, dgbfa, dgbsl, dgefa, dgesl, dmess, dnrm2, drot, drotg, dscal, dswap, idamax, mess
DDASLS	d1mach, daxpy, dcopy, dscal, dswap, dasum, dgbfa, dgbsl, dgefa, dgesl
SDASLX	amach, dcopy, dnrm2, isamax, mess, saxpy, scopy, sdas1, sdasco, sdasdb, sdasf, sdasgh, sdasin, sdasj, sdaslv, sdaslx, sdasnm, sdastp, sdaswt, sdot, sgbfa, sgbsl, sgefa, sgesl, smess, snrm2, srot, srotg, sscal, sswap
SDASLS	r1mach, saxpy, scopy, sscal, sswap, sasum, sgbfa, sgbsl, sgefa, sgesl

---

DRDDASL1

```
      PROGRAM drddasl1
c>> 2009-10-21 DRDDASL1 Hanson/Krogh Fixed initialization.
c>> 2008-10-26 DRDDASL1 Krogh Moved Format statements up for C conv.
c>> 2008-10-24 DRDDASL1 Krogh Removed in INCLUDE statement & cDEC$...
c>> 2008-09-02 DRDDASL1 Hanson added starting computation of y'
c>> 2008-08-26 DRDDASL1 Hanson added row dimensions to evaluators
c>> 2006-04-10 DRDDASL1 Krogh Removed declaration of unused E.
c>> 2002-05-29 DRDDASL1 Krogh Changes for C conversion problem.
c>> 2001-10-11 DRDDASL1 R. J. Hanson Document Example Code,

c  Solve a C. W. Gear index=2 problem.
c  Reduce it to an index=1, solve it.
c  Reduce it further to index=0, solve it.
c  Compare results, which are equivalent but
c  not exactly equal.

c--D replaces "?": DR?DASL1, ?DASLX, ?DASLS, ?DASSF1, ?DASSF0
      EXTERNAL ddassf0,ddassf1
      INTEGER ndig,ntimes
      DOUBLE PRECISION tol
c++S Default NDIG = 4
c++  Default NDIG = 8
c++ Substitute for NDIG below
      PARAMETER (ndig=8)
      PARAMETER (tol=10.d0**(-ndig))
      INTEGER liw,lrw,maxcon,neq,ldc,ltd
c     Set number of equations:
      PARAMETER (neq=2)
c     Set maximum number of constraints.
      PARAMETER (maxcon=2)
c     Work space sizes:
      PARAMETER (liw=30+neq)
      PARAMETER (lrw=45+(5+2*maxcon+4)*neq+neq**2)
      PARAMETER (ntimes=10)
      PARAMETER (ldc=2*neq,ltd=neq)
```

```
      INTEGER i,info(16),idid,iwork(liw)

      DOUBLE PRECISION t,y(neq),yprime(neq),tout,rtol(neq),atol(neq),
     & rwork(lrw),soln1(ntimes,neq),soln0(ntimes,neq),
     & solp1(ntimes,neq),solp0(ntimes,neq)
      DOUBLE PRECISION c(ldc,ltd),ftol,rnktol

      INTEGER kount0,kount1
      COMMON /counts/ kount0,kount1

   60 FORMAT (6x,'Example Results for a Transformed Index-2 DAE Problem'
     &//10x,'T',11x,'Y1/Y2',9x,'Y1P/Y2P')
   70 FORMAT (6x,'Differences, (Index-1) - (Index-0) Values'//10x,'T',
     &11x,'Y1/Y2',9x,'Y1P/Y2P')
   80 FORMAT (1p,3e15.6/15x,2e15.6/' ')
   90 FORMAT (6x,'Index-1 and Index-0 residual evalutions:',2i5)
  100 FORMAT (6x,'Initial conditions for y,y'' at t=0, index 1 system')
  110 FORMAT (6x,'Initial conditions for y,y'' at t=0, index 0 system')

c     Tolerances:
      DO 10 i=1,neq
        atol(i)=tol
        rtol(i)=tol
   10 CONTINUE
c     Setup options:
      DO 20 i=1,16
        info(i)=0
   20 CONTINUE
c     Use partial derivatives provided in evaluator routines:
      info(5)=2
c     Constrain solution with 1 constraint:
      info(10)=1
c     Compute the initial value of YPRIME(*):
      t=0
      ftol=tol
      rnktol=tol
c     Assign initial y, and guess for y', then get initial y'.
      y(1) = 1.D0
      y(2) = 0.D0
      yprime(1)=0.D0
      yprime(2)=0.D0
      CALL ddasls (ddassf1, neq, t, y, yprime, info, ftol, rnktol, c,
     & ldc, ltd, idid, rwork, lrw, iwork, liw)
      WRITE (*,60)
      WRITE (*,100)
      WRITE (*,80) t,y(1),yprime(1),y(2),yprime(2)

c     Allow up to 5000 steps:
      info(12)=5000

      DO 30 i=1,ntimes
c     Integrate from T=I-1 to TOUT=T+1.  Final TOUT=10.
        t=i-1
        tout=t+1
        CALL ddaslx (ddassf1, neq, t, y, yprime, tout, info, rtol, atol,
     &    idid, rwork, lrw, iwork, liw)
        WRITE (*,80) tout,y(1),yprime(1),y(2),yprime(2)
c     Save solution and derivative for comparison to index 0 values.
        soln1(i,1)=y(1)
```

```
          soln1(i,2)=y(2)
          solp1(i,1)=yprime(1)
          solp1(i,2)=yprime(2)
   30   CONTINUE

c       Start integration again.
        info(1)=0
        DO 40 i=1,neq
          atol(i)=tol
          rtol(i)=tol
   40   CONTINUE
c       Switch from 1 to 2 constraints, and use the index 0 system.
        info(10)=2
        WRITE (*,70)
        t=0.D0
c       Assign initial y, and guess for y', then get initial y'.
        y(1) = 1.D0
        y(2) = 0.D0
        yprime(1)=0.D0
        yprime(2)=0.D0
        CALL ddasls (ddassf0, neq, t, y, yprime, info, ftol, rnktol, c,
     & ldc, ltd, idid, rwork, lrw, iwork, liw)
        WRITE (*,110)
        WRITE (*,80) t,y(1),yprime(1),y(2),yprime(2)

        DO 50 i=1,ntimes
c       Integrate from T=I-1 to TOUT=T+1.  Final TOUT=10.
          t=i-1
          tout=t+1
          CALL ddaslx (ddassf0, neq, t, y, yprime, tout, info, rtol, atol,
     &    idid, rwork, lrw, iwork, liw)
c       Use solution and derivative differences for comparison
c       with index 1 values.
          soln0(i,1)=soln1(i,1)-y(1)
          soln0(i,2)=soln1(i,2)-y(2)
          solp0(i,1)=solp1(i,1)-yprime(1)
          solp0(i,2)=solp1(i,2)-yprime(2)
          WRITE (*,80) tout,soln0(i,1),solp0(i,1),soln0(i,2),solp0(i,2)
   50   CONTINUE
c       Print number of residual evaluations for index 1 and index 0
c       problems.
        WRITE (*,90) kount1,kount0
        END

        SUBROUTINE ddassf1 (t, y, yprime, delta, d1, ldd, cj, ires, rwork,
     & iwork)
c
c       Routine for the Gear index=2 problem.
c       One equation is differentiated to reduce it to index 1,
c       with a constraint on the index 2 equation.
        DOUBLE PRECISION t,y(*),yprime(*),delta(*),d1(ldd,*),cj,rwork(*),
     & eta,one, two, zero
        INTEGER ires,iwork(*),ldd
        INTEGER kount0,kount1
        COMMON /counts/ kount0,kount1
        one=1.d0
        two=2.d0
        zero=0.d0
        eta=10.d0
```

```
c This is the setup call.
      IF (ires.eq.0) THEN
        kount1=0
      END IF

c The system residual value.
      IF (ires.eq.1) THEN
        delta(1)=yprime(1)+eta*t*yprime(2)+(one+eta)*y(2)-sin(t)
c This second equation comes from differentiating the second equation in
c section C.1, and subtracting the result from the first equation.
        delta(2)=y(2)-two*sin(t)
c Count function evaluations.
        kount1=kount1+1
      END IF

c The partial of the iteration matrix with respect to y.  This is an
c index 1 system.  d1 is set to 0 prior to all calls here.  Partials
c are based on equations above.  Note that \partial y'_i / y_i is c_j.
      IF (ires.eq.2) THEN
        d1(1,1)=cj
        d1(1,2)=(one+eta)+cj*eta*t
        d1(2,2)=one
      END IF

c The constraining equation after the corrector has converged.
c Both partials and residuals are required.  This is for the second
c equation in C.1.
      IF (ires.eq.5) THEN
        d1(3,1)=one
        d1(3,2)=eta*t
        delta(3)=y(1)+eta*t*y(2)-cos(t)
      END IF
c The values of IRES=6,7 and 8 occur for the starting procedure
c that solves for y'.  First the partials with respect to  t of what
c is computedb when ires is 1.  Here and below we are computing
c partials of f not of the iteration matrix.
      IF (ires.eq.6) THEN
        delta(1)=eta*yprime(2)-cos(t)
        delta(2)= -two*cos(t)
      END IF
c Compute the partial w.r.t y' of the equations defined when IRES=1
      IF (ires.eq.7) THEN
        d1(1,1)=one
        d1(1,2)=eta*t
      END IF
c Compute the partial w.r.t y of the equations defined when IRES=1
      IF (ires.eq.8) THEN
        d1(1,2)=one+eta
        d1(2,2)=one
      END IF
      END

      SUBROUTINE ddassf0 (t, y, yprime, delta, d0, ldd, cj, ires, rwork,
     & iwork)
c
c     Routine for the Gear index=2 problem.
c     One equation is differentiated twice to reduce it to index 0.
c     This gives constraints on the index 2 and index 1 equations.
      DOUBLE PRECISION t,y(*),yprime(*),delta(*),d0(ldd,*),cj,rwork(*),
```

```
     & eta, one, two, zero
      INTEGER i,ires,iwork(*),j,ldd
      INTEGER kount0,kount1
      COMMON /counts/ kount0,kount1
      one=1.d0
      two=2.d0
      zero=0.d0
      eta=10.d0
c This is the setup call.
      IF (ires.eq.0) THEN
        kount0=0
      END IF

c The system residual value.
      IF (ires.eq.1) THEN
        delta(1)=yprime(1)+eta*t*yprime(2)+(one+eta)*y(2)-sin(t)
        delta(2)=yprime(2)-two*cos(t)
c Count function evaluations.
        kount0=kount0+1
      END IF

c The mixed partial derivative matrix.
c This is an index 0 system.
      IF (ires.eq.2) THEN
        d0(1,1)=cj
        d0(1,2)=(one+eta)+cj*eta*t
        d0(2,2)=cj
      END IF

c The constraining equations after the corrector has converged.
c Both partials and residuals are required.
      IF (ires.eq.5) THEN
        d0(3,1)=one
        d0(3,2)=eta*t
        d0(4,1)=zero
        d0(4,2)=one
        delta(3)=y(1)+eta*t*y(2)-cos(t)
        delta(4)=y(2)-two*sin(t)
      END IF
c The partial w.r.t y' for the starting procedure
c Since this is an index 0 system the cases IRES=6,8 will not occur
      IF (ires.eq.7) THEN
        d0(1,1)=one
        d0(1,2)=eta*t
        d0(2,2)=one
      END IF
      END
```

ODDDASL1

```
      Example Results for a Transformed Index-2 DAE Problem

          T            Y1/Y2          Y1P/Y2P
      Initial conditions for y,y' at t=0, index 1 system
  0.000000E+00   1.000000E+00   0.000000E+00
                 0.000000E+00   2.000000E+00

  1.000000E+00  -1.628912E+01  -2.847694E+01
                 1.682942E+00   1.080605E+00
```

```
2.000000E+00   -3.678804E+01   -2.449373E+00
                1.818595E+00   -8.322937E-01

3.000000E+00   -9.457193E+00    5.643603E+01
                2.822400E-01   -1.979985E+00

4.000000E+00    5.989056E+01    6.818434E+01
               -1.513605E+00   -1.307287E+00

5.000000E+00    9.617609E+01   -8.228807E+00
               -1.917849E+00    5.673243E-01

6.000000E+00    3.449003E+01   -1.093527E+02
               -5.588310E-01    1.920341E+00

7.000000E+00   -9.122422E+01   -1.193430E+02
                1.313973E+00    1.507805E+00

8.000000E+00   -1.584428E+02    2.503480E+00
                1.978716E+00   -2.910000E-01

9.000000E+00   -7.509246E+01    1.553490E+02
                8.242370E-01   -1.822261E+00

1.000000E+01    1.079652E+02    1.792387E+02
               -1.088042E+00   -1.678143E+00

   Differences, (Index-1) - (Index-0) Values

      T            Y1/Y2          Y1P/Y2P
   Initial conditions for y,y' at t=0, index 0 system
0.000000E+00    1.000000E+00    0.000000E+00
                0.000000E+00    2.000000E+00

1.000000E+00   -5.444534E-09    1.148811E-07
                5.286331E-10    5.637443E-09

2.000000E+00    9.030664E-09   -5.975787E-07
               -4.492540E-10    1.429224E-08

3.000000E+00    7.262546E-10   -5.326531E-07
               -8.426299E-11    2.284928E-09

4.000000E+00    3.102966E-08    4.156549E-07
               -9.139132E-10    1.607341E-08

5.000000E+00    2.861000E-09    2.237059E-06
               -1.723777E-11   -3.744174E-08

6.000000E+00   -1.730818E-08   -5.157416E-08
                2.452695E-10   -2.679479E-11

7.000000E+00   -5.656071E-08   -5.471411E-07
                6.758687E-10   -3.088880E-08

8.000000E+00   -1.684640E-08   -3.323030E-06
                2.047635E-10    3.999238E-08
```

```
9.000000E+00   1.188216E-08   1.039281E-06
              -1.710334E-10   -5.074183E-09

1.000000E+01   5.870513E-08   -6.847629E-07
              -7.596022E-10   -2.048200E-08

   Index-1 and Index-0 residual evalutions:  796  672
```