DAEPACK

DSL48S Manual

Version 1.0

Numerical Integration and Parametric Sensitivity Analysis of Large-Scale Differential/Algebraic Systems

William F. Feehery and John E. Tolsma

December 2, 2000

Contents

1 Calling DSL48S **4**

2 Using DSL48S **6**

2.1 What to do on the first call to DSL48S 6

2.2 Output after any return by DSL48S 16

2.3 What to do to continue the integration 17

2.4 Error messages 19

3 The `test.f` program **21**

This document describes how to obtain and use DSL48S, a code for numerical solution and sensitivity analysis of large-scale sparse Differential Algebraic Equations (DAEs). The code was written by Russell Allgor and William Feehery and is derived from the well-known DASSL code that was created by Linda Petzold.

DSL48S provides the following unique features:

- Use of the sparse unstructured linear algebra solver MA48.
- Option for efficient calculation of sensitivities of large-scale systems.
- Dynamic bounds checking algorithm.
- Modifications to allow the use of the dummy derivative method for high-index DAEs.

1 Calling DSL48S

The FORTRAN call to DSL48S is:

```
      CALL DSL48S (RES, SENRHS, NEQ, T, Y, YPRIME, TOUT, INFO,
     +   RTOL, ATOL, IDID, RWORK, LRW, IWORK, LIW, RPAR, IPAR,
     +   JAC, NEJAC, JROW, JCOL, JYDOT, NJYDOT, JDTYPE, BOUND)
```

In the following description of the DSL48S parameters, `EXT` denotes an external function, `IN` denotes a variable that must be set on the first call, `OUT` denotes a variable that DSL48S returns, and `WORK` denotes a workspace.

`RES:EXT` The name of an external user-provided subroutine for calculating the residuals of the DAE. (`USER-DEFINED FUNCTION`)

`SENRHS:EXT` The name of an external user-provided subroutine for calculating the right hand sides of the sensitivity equation. If sensitivities are not solved, this subroutine will not be called, but a dummy subroutine must be provided. (`USER-DEFINED FUNCTION`)

`NEQ:IN` giving the number of equations in the combined DAE and sensitivity system. If there are three equations in the DAE, and two sensitivity parameters, `NEQ` $= 3(2+1) = 9$. (`INTEGER`)

`T:INOUT` Contains the current value of the independent variable. (`DOUBLE PRECISION`)

`Y(*):INOUT` Array of length at least `NEQ` which contains the values of the state and sensitivity variables at the current value of `T`. On the first call, should be set to the initial condition for the DAE and sensitivities. (`ARRAY OF DOUBLE PRECISION`)

`YPRIME(*):INOUT` Array of length at least `NEQ` which contains the values of the time derivatives of the state and sensitivity variables at the current value of `T`. On the first call, should be set to the initial condition for the DAE and sensitivities. (`ARRAY OF DOUBLE PRECISION`)

`TOUT:IN` The value of the independent variable at which the integration should stop (`DOUBLE PRECISION`)

`INFO(25):IN` The basic task of the code is to solve the system from `T` to `TOUT` and return an answer at `TOUT`. `INFO` is an integer array which is used to communicate exactly how you want this task to be carried out. (See below for details.) (`ARRAY OF INTEGER`)

`RTOL,ATOL:INOUT` These quantities represent relative and absolute error tolerances which you provide to indicate how accurately you wish the solution to be computed. You may choose them to be both scalars or else both vectors. Caution: In FORTRAN 77, a scalar is not the same as an array of length 1. Some compilers may object to using scalars for `RTOL`, `ATOL`. (`ARRAY OF DOUBLE PRECISION`)

`IDID:OUT` This scalar quantity is an indicator reporting what the code did. You must monitor this integer variable to decide what action to take next. (`INTEGER`)

`RWORK:WORK` A real work array of length `LRW` which provides the code with needed storage space. (`ARRAY OF DOUBLE PRECISION`)

`LRW:IN` The length of `RWORK`. (See below for required length.) (`INTEGER`)

IWORK:WORK An integer work array of length LIW which provides the code with needed storage space. (ARRAY OF INTEGER)

LIW:IN The length of IWORK. (See below for required length.) (INTEGER)

RPAR(*),IPAR(*):IN These are real and integer parameter arrays which you can use for communication between your calling program and the RES subroutine (and the JAC subroutine), and which may contain information about the parameters. (ARRAY OF DOUBLE PRECISION) (ARRAY OF INTEGER)

JAC:EXT This is the name of a subroutine which you may choose to provide for defining a matrix of partial derivatives described below. (USER-DEFINED FUNCTION)

NEJAC:IN The number of nonzeroes in the sparse jacobian. (INTEGER)

JROW(*):IN An integer array containing the row numbers of the elements in the sparse Jacobian. (ARRAY OF INTEGER)

JCOL(*):IN An integer array containing the column numbers of the elements in the sparse Jacobian. (ARRAY OF INTEGER)

JYDOT(*):IN An integer array containing the Jacobian elements that are with respect to time derivative variables. (ARRAY OF INTEGER)

NJYDOT:IN The number of elements in the time derivative Jacobian. (INTEGER)

JDTYPE:IN Vector specifying the method to be used to calculate the corresponding element of the Jacobian matrix. (ARRAY OF INTEGER)

BOUND(*):IN A real array containing the bounds on the state variables (not the sensitivity variables). (ARRAY OF INTEGER)

2 Using DSL48S

DSL48S solves a DAE of the form $g(t, y, y') = 0$ and performs a sensitivity analysis upon the problem parameters if requested.

Subroutine DSL48S uses the backward differentiation formulae (BDF) of orders one through five to solve a system of the above form for $y(t)$ and $y'(t)$. Values for y and y' at the initial time must be given as input. These values must be consistent, (that is, if t_o, y_o, and y'_o are the given initial values, they must satisfy $g(t_o, y_o, y'_o) = 0$ and all of its first and higher order time derivatives). DSL48S solves the system from `T` to `TOUT`. It is easy to continue the solution to get results at additional `TOUT`. This is the interval mode of operation. Intermediate results can also be obtained easily by using the intermediate-output capability.

DSL48S includes an option to perform a sensitivity analysis of the problem to be solved. Given a DAE depending on a vector of parameters p, $g(t, y, y', p) = 0$, where y, y' and g are vectors of length N_y and p is a vector of length N_p, DSL48S will compute $\frac{\partial y(t)}{\partial p_i}$ for $i = 1 \ldots N_p$. The sensitivities satisfy the equations:

$$\frac{\partial g}{\partial y}\frac{\partial y}{\partial p_i} + \frac{\partial g}{\partial y'}\frac{\partial y'}{\partial p_i} + \frac{\partial g}{\partial p_i} = 0 \quad i = 1 \ldots N_p$$

or a finite difference approximation to this equation. The sensitivity values are stored in the `Y` vector following the solution of the DAE, so that:

$$\texttt{Y} = \begin{bmatrix} y \\ \frac{\partial y}{\partial p_1} \\ \cdot \\ \cdot \\ \frac{\partial y}{\partial p_{N_p}} \end{bmatrix} \qquad \texttt{YPRIME} = \begin{bmatrix} y' \\ \frac{\partial y'}{\partial p_1} \\ \cdot \\ \cdot \\ \frac{\partial y'}{\partial p_{N_p}} \end{bmatrix}$$

Initial conditions must also be specified for the sensitivity variables, and they must be chosen so that they are consistent with the sensitivity equations (and their first and higher order time derivatives) at the initial values.

The following detailed description is divided into subsections:

- Section 2.1: Input required for the first call to DSL48S.
- Section 2.2 : Output after any return from DSL48S.
- Section 2.3 : What to do to continue the integration.
- Section 2.4 : Error messages.

2.1 What to do on the first call to DSL48S

The first call of the code is defined to be the start of each new problem. Read through the descriptions of all the following items, provide sufficient storage space for designated arrays, set appropriate variables for the initialization of the problem, and give information about how you want the problem to be solved.

`RES` Provide a subroutine of the form:

```
SUBROUTINE RES(NY,T,Y,YPRIME,DELTA,IRES,ICHVAR,RPAR,IPAR)
```

where NY is the number of equations EXCLUDING sensitivities to define the system of differential/algebraic equations which is to be solved. For the given values of T, Y and YPRIME, the subroutine should return the residual of the DAE, DELTA $= g$(T, Y, YPRIME). (DELTA is a vector of length NEQ which is output for RES.)

Subroutine RES must not alter T, Y or YPRIME. You must declare the name RES in an external statement in your program that calls DSL48S. You must dimension Y, YPRIME and DELTA in RES.

IRES is an integer flag which is always greater than zero on input. Subroutine RES should alter IRES only if it encounters an illegal value of Y or a stop condition. Set IRES $= -1$ if an input value is illegal, and DSL48S will try to solve the problem without getting IRES $= -1$. If IRES $= -2$, DSL48S will return control to the calling program with IDID $= -11$.

ICHVAR is an integer flag that is set before the call to RES. It equals 1 if the values in Y and YPRIME have changed since the last call to RES. If ICHVAR equals 0, the values have not changed, and it it may be possible to save some work in the RES subroutine.

RPAR and IPAR are real and integer parameter arrays which you can use for communication between your calling program and subroutine RES. They are not altered by DSL48S. If you do not need RPAR or IPAR, ignore these parameters by treating them as dummy arguments. If you are using the sensitivity analysis option (INFO(17) $\neq 0$) and the sensitivity equations are not defined analytically, the parameters of your problem MUST be stored in RPAR as described below and dimensioned in your calling program and in RES as arrays of appropriate length.

SENRHS Provide a subroutine of the form:
SUBROUTINE SENRHS(NEQ,T,Y,YPRIME,DELTA,IRES,RPAR,IPAR)
which is similar to RES, but defines the right hand side vector of the analytic sensitivity residuals. The sensitivity equations have the form $(\partial g/\partial y)s_i + (\partial g/\partial y')s_i' =$ DELTA, $s_i = \frac{\partial y}{\partial p_i}$ is the vector of sensitivity variables for parameter i, and DELTA is the vector that must be defined by the subroutine. SENRHS must not alter the first NY positions of DELTA. DSL48S sets DELTA to a vector of zeroes before the call to SENRHS.

If analytic sensitivity equations are not being used, SENRHS is not called, but a dummy external procedure must be provided to compile the code.

NEQ The value of this constant should be set to the number of differential equations, including the sensitivity equations. Note that in the case of sensitivity analysis, NEQ $= N_y(N_p + 1)$ where N_y is the number of state and algebraic variables, and N_p is the number of sensitivity parameters. (NEQ ≥ 1)

T Set to the initial value of the independent variable. T must be defined as a variable.

Y Set this vector to the initial values of the state and sensitivity variables at the initial point. You must dimension Y of length at least N_y in your calling program for ODE/DAE solutions and of length at least $N_y(N_p + 1)$ for sensitivity analysis solutions.

YPRIME Set this vector to the initial values of the first derivatives of the state and sensitivity variables at the initial point. You must dimension YPRIME at least N_y for ODE/DAE solutions and of length at least $N_y(N_p + 1)$ for sensitivity analysis solutions. DSL48S requires the initial T, Y, and YPRIME to be consistent.

`TOUT` Set it to the first point at which a solution is desired. You must not set `TOUT` = `T`. Integration either forward in `T` (`TOUT` $\geq$ `T`) or backward in `T` (`TOUT` $\leq$ `T`) is permitted.

The code advances the solution from `T` to `TOUT` using step sizes which are automatically selected so as to achieve the desired accuracy. If you wish, the code will return with the solution and its derivative at intermediate steps (intermediate-output mode) so that you can monitor them, but you still must provide `TOUT` in accord with the basic aim of the code.

The first step taken by the code is a critical one because it must reflect how fast the solution changes near the initial point. The code automatically selects an initial step size which is practically always suitable for the problem. By using the fact that the code will not step past `TOUT` in the first step, you could, if necessary, restrict the length of the initial step size.

For some problems it may not be permissible to integrate past a point `TSTOP` because a discontinuity occurs there or the solution or its derivative is not defined beyond `TSTOP`. When you have declared a `TSTOP` point (see `INFO(4)` and `RWORK(1)`), you have told the code not to integrate past `TSTOP`. In this case any `TOUT` beyond `TSTOP` is invalid input.

`INFO` Use the `INFO` array to give the code details about how you want your problem solved. This array should be dimensioned of length 25, though DSL48S uses only the first 23 entries. You must respond to all of the following items, which are arranged as questions. The simplest use of the code corresponds to answering all questions as yes, i.e. setting all entries of `INFO` to 0.

`INFO(1)` This parameter enables the code to initialize itself. You must set it to indicate the start of every new problem.

Is this the first call for this problem?

Yes:

- Set `INFO(1)` = 0 if you want to start the integrator.
- Set `INFO(1)` = -1 if you want to start the integrator without restarting the integration statistics (number of steps, etc.).

No: Not applicable here. See below for continuation calls.

`INFO(2)` How accurate you want your solution to be is specified by the error tolerances `RTOL` and `ATOL`. The simplest use is to take them both scalars. To obtain more flexibility, they can both be vectors. DSL48S must be told your choice.

Are `RTOL` *and* `ATOL` *scalars:*

Yes: Set `INFO(2)`=0.

No: Set `INFO(2)`=1 (both are vectors).

`INFO(3)` The code integrates from `T` in the direction of `TOUT` by 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 specific `TOUT` points, this code will compute them efficiently.

Do you want the solution only at `TOUT` *(and not at the next intermediate step)?*

Yes: Set `INFO(3)` = 0.

No: Set `INFO(3)` = 1.

INFO(4) To handle solutions at a great many specific values TOUT efficiently, 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 TSTOP. Then you must tell the code not to go past TSTOP.

Can the integration be carried out without any restrictions on the independent variable T?

Yes: Set INFO(4)=0.

No: Set INFO(4)=1 and define the stopping point TSTOP by setting RWORK(1) = TSTOP.

INFO(5) To solve differential/algebraic problems it is necessary to use a matrix of partial derivatives of the system of differential equations. If you do not provide a subroutine to evaluate it analytically (see description of the item JAC in the call list), it will be approximated by numerical differencing in this code. Although it is less trouble for you to have the code compute partial derivatives by numerical differencing, the solution will be more reliable if you provide the derivatives via JAC. Sometimes numerical differencing is cheaper than evaluating derivatives in JAC and sometimes it is not - this depends on your problem and the technique you use to evaluate analytic derivatives.

Do you want the code to evaluate the partial derivatives automatically by numerical differences?

Yes: Set INFO(5)=0.

No: Set INFO(5)=1 and provide subroutine JAC for evaluating the matrix of partial derivatives.

INFO(6) Not used.

INFO(7) You can specify a maximum (absolute value of the) stepsize h_{max}, so that the code will avoid passing over very large regions.

Do you want the code to decide on its own maximum stepsize?

Yes: Set INFO(7)=0.

No: Set INFO(7)=1 and define h_{max} by setting RWORK(2) = h_{max}.

INFO(8) Differential/algebraic problems may occasionally suffer from severe scaling difficulties on the first step. If you know a great deal about the scaling of your problem, you can help to alleviate this problem by specifying an initial stepsize h_o.

Do you want the code to define its own initial stepsize?

Yes: Set INFO(8)=0.

No: Set INFO(8)=1 and define h_o by setting RWORK(3) = h_o.

INFO(9) If storage is a severe problem, you can save some locations by restricting the maximum order MAXORD. The default value is 5. for each order decrease below 5, DSL48S requires NEQ fewer locations, however, it is likely to be slower. This option may also be useful if you know that higher order integration methods will experience difficulties when applied to your problem. In any case, you must have $1 \leq$ MAXORD ≤ 5.

Do you want the maximum order to default to 5?

Yes: Set INFO(9)=0.

No: Set INFO(9)=1 and define MAXORD by setting IWORK(3) = MAXORD.

`INFO(10)` If you know that the solutions to your equations will always be nonnegative, it may help to set this parameter. However, it is probably best to try the code without using this option first, and only to use this option if that doesn't work very well.

Do you want DSL48S to solve the problem without invoking any special nonnegativity constraints?

Yes: Set `INFO(10)`=0.

No: Set `INFO(10)`=1.

`INFO(11)` Not currently used.

`INFO(12)` Not currently used.

`INFO(13)` Determines whether DSSL48 should detect when high-index pivoting is necessary. Do not use this option unless your calling code is capable of handling the high-index pivoting-DSL48S does not do it.

Should DSL48S detect the need for possible high-index pivoting?

Yes: Set `INFO(13)` = 1.

No: Set `INFO(13)`=0.

`INFO(14)` Determines whether DSSL48 has detected that high index pivoting may be necessary. This parameter is an output parameter only, unlike the rest of `INFO`. `INFO(14)` is not used if `INFO(13)`=0.

DSL48S has detected that high index pivoting may be necessary:

Yes: `INFO(14)` is set to 1 on return.

No: `INFO(14)` is set to 0 on return.

`INFO(15)` Determines whether DSSL48 will do bounds checking If Yes, then `BOUND` must be an array of length 2`NY`, where the first `NY` positions contains the lower bounds for the unknowns, and the second `NY` positions contains the upper bounds. Unless it is really necessary, this option should probably not be used.

Should DSL48S perform dynamic bounds checking?

Yes: `INFO(15)` = 1.

No: `INFO(15)` = 0.

`INFO(16)` Automatic scaling of the corrector iteration can be performed when using the sparse unstructured matrix option. This requires a small amount of extra work, but improves accuracy guarantees. This option should probably be used in almost all cases.

Should DSL48S perform automatic scaling?

Yes: Set `INFO(16)` = 0.

No: `INFO(16)` = 1.

`INFO(17)` Sensitivity toggle. Computational parameters for sensitivity analysis on DSL48S are accessed through the `INFO` array values $17 \rightarrow 23$.

Do you wish to have DSL48S perform a sensitivity analysis on the given ODE/DAE?

Yes: Set `INFO(17)` = N_p, where N_p equals the number of parameters involved in the system to be solved.

No: Set `INFO(17)` = 0.

`INFO(18)` Finite differencing options toggle for obtaining the sensitivity equations. Note that if `INFO(17)` = 0, then the value of this element is ignored.

Do you wish DSL48S to use a finite difference method (FDM) for the approximation of the sensitivity analysis residuals?

No:

- `INFO(18)` = 0: In this case the external user-provided `SENRHS` routine should compute the $N_y \cdot N_p$ sensitivity right hand sides. It is not necessary to give DSL48S the values of the parameters.
- `INFO(18)` = 1: A first order FDM method is used to calculate the $N_y \cdot N_p$ sensitivity right hand sides.
- `INFO(18)` = 2: A centered second order FDM method is used to calculate the $N_y \cdot N_p$ sensitivity right hand sides.

Yes:

- `INFO(18)` = 3: A first order forward FDM is used.
- `INFO(18)` = 4: A centered second order FDM is used. The sensitivity parameters are stored in `RPAR` by the index values placed in the first N_p locations of `IPAR`. For example, if `IPAR(3)`=4, then parameter(3) is stored in `RPAR(4)`. See the definitions of `IPAR` and `RPAR` below for a more detailed description. Note that `LRW` must now be extended to `LRW` = `LRW` + $4N_y$ and `NEQ` must be extended to `NEQ` = $N_y(N_p + 1)$ so that `Y` and `YPRIME` have dimension $N_y(N_p + 1)$.

`INFO(19)` Sensitivity error control option.

Do you wish DSL48S to include all variables (state and sensitivity) in the error control test?

Yes: Set `INFO(19)` = 0.

No: Set `INFO(19)` = 1 to include ONLY the state variables in the error test for the given problem (i.e. excluding the sensitivity variables). Note: All variables will be included in the convergence test. Although this option will usually make the code execute faster, it is not recommended because the resulting sensitivity answer may be incorrect.

`INFO(20)` Zero sensitivity option.

Do you know that some sensitivity variables are identically zero during the current time step?

Yes: Set `INFO(20)`=1 The sensitivities with respect to parameter k are assumed to be identically zero if `IPAR`$(N_p + k) = 0$. Otherwise `IPAR`$(N_p + k)$ should be set to 1. This option is useful in applications like control parameterization where a large number of sensitivity varibles can be zero for much of the integration.

No: Set `INFO(20)`=0.

`INFO(21)` Perturbation factor option. The selection of the perturbation used in the finite difference approximation to the sensitivity equations is determined by the expression

$$\delta_i = C \max(|\texttt{RPAR}(\texttt{IPAR}(i)|, ||v||)$$

where $||v||$ is the 2-norm of the vector V ($V(j) = WT(N_y + j)/WT(j)$ for $j = 1 \ldots N_y$, WT is a vector of weights determined by `RTOL`, `ATOL` and `Y`) and C is the pertubation factor. The default value for C is $\sqrt{\epsilon}$, where ϵ is the machine roundoff. This option allows the user to change this value.

Do you wish to alter the value of C?

No: Set INFO(21) = 0.

Yes: Set INFO(21) = 1 and set RWORK(10) equal to the desired value.

INFO(22) IPAR marker. Used to start the sensitivity analysis at element INFO(22) of the IPAR vector. For example, setting INFO(22) = 3 causes DSL48S to calculate the sensitivities corresponding to IPAR(3 + i), for i = 1 to INFO(17).

Do you wish DSL48S to use an offset within IPAR*?*

No: Set INFO(22) = 0.

Yes: Set INFO(22) = to the desired offset used in IPAR. Note that DSL48S is incapable of performing an error check (other than ≤ 0) upon this INFO element. Hence it is a potential source of performance errors.

INFO(23) Sensitivity method option. Used to set the method to be used for solving sensitivity equations.

NOTE: THE STAGGERED CORRECTOR METHOD IS RECOMMENDED. The simultaneous corrector method is included only for comparison.

Do you want to use the staggered corrector method?

Yes: Set INFO(16)=0.

No: Set INFO(16)=1 to use the simultaneous corrector method.

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 RTOL and ATOL are scalars. (INFO(2)=0)
- Both RTOL and ATOL are vectors of length 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 ABS(LOCAL ERROR) $<$ RTOL $*$ ABS(Y) + 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. Practically all present day codes, including this one, control the local error at each step and do not even attempt to control the global error directly.

Usually, but not always, 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 a fairly reliable 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 accuracy unreasonable for the machine being used. It will advise you if you ask for too much accuracy and inform you as to the maximum accuracy it believes possible.

RWORK(*) – Dimension this real work array of length LRW in your calling program.

LRW Set it to the declared length of the RWORK array.

Use the following if-then guide to determine the minumum value for LRW:

```
IF (INFO(17).NE.0) THEN
  IF (INFO(18).LE.2) THEN
    NDASSL = (MAXORD+3) * NEQ + NEQ
  ELSE IF (INFO(18).EQ.3) THEN
    NDASSL=(MAXORD+3) * NEQ + 4*NEQ/(INFO(17)+1) + 1
  ELSE IF (INFO(18).EQ.4) THEN
    NDASSL=(MAXORD+3) * NEQ + 4*NEQ/(INFO(17)+1) + 1
  ENDIF
  IF (NFD.GT.0) THEN
    NSENS=NEJAC+5*NEQ/(INFO(17)+1)+NFD
  ELSE
    NSENS=NEJAC
  ENDIF
  NLUDWK = 5 + 5*NEQ/(INFO(17)+1)
ELSE
  NDASSL = (MAXORD+3) * NEQ
  NSENS=0
  NLUDWK = 5 + 5*NEQ
ENDIF

NLUD   = 4*NEJAC
```

NFD is the number of elements of the Jacobian that must be calculated using finite differences.

Note that NLUD gives the amount of space that is estimated to factor the Jacobian. In some cases, NLUD must be up to 10*NEJAC, depending on the form of the Jacobian of the problem you are trying to solve.

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

LIW Set it to the declared length of the IWORK array. Use the following if-then guide to determine the mimimum value of LIW.

```
IF INFO(17).EQ.0) THEN
  NY=NEQ
ELSE
  NY=NEQ/(INFO(17)+1)
ENDIF

NPTRS = 20 + 30
NKEEP = 6*NY + 4*NY/1 + 7
```

```
NIIK48 = 9 + 12 + NKEEP
NIWK48 = 6 * NY
NINDX = 2*NLUD
IF (INFO(16) .NE. 0) THEN
   NSCALE = 0
ELSE
   NSCALE = 3*NY + NEJAC
ENDIF

IF (NFD .GT. 0) THEN
   NFDIFF = 2*NFD + 3*NY + 1
ELSE
   NFDIFF = 0
ENDIF

LENIW = NPTRS + NINDX + NIIK48 + NIWK48 +
        NSCALE + NFDIFF
```

RPAR, IPAR These are parameter arrays, of real and integer type, respectively. You can use them for communication between your program that calls DSL48S and the RES subroutine (and the SENRHS and JAC subroutines). They are not altered by DSL48S. If you do not need RPAR or IPAR, ignore these parameters by treating them as dummy arguments. If you do choose to use them, dimension them in your calling program and in RES (and in the SENRHS and JAC subroutines) as arrays of appropriate length. Depending on the value of INFO(17), RPAR and IPAR may also be used to communicate the parameters of the given problem.

If INFO(17) $\neq 0$, i.e., a sensitivity analysis of the problem has been requested, RPAR is used to hold parameter values and IPAR is used to hold the location of each parameter value within RPAR. For example, suppose that a problem contains three parameters, it is possible to store them in RPAR in the following manner:

$$\begin{aligned} \texttt{RPAR}(2) &= p_1 \\ \texttt{RPAR}(7) &= p_2 \\ \texttt{RPAR}(2) &= p_3 \end{aligned}$$

The first $N_p = 3$ elements of IPAR are then used to locate these parameter values within RPAR:

$$\begin{aligned} \texttt{IPAR}(1) = 2 &\rightarrow p_1 \text{ has been stored in RPAR}(2) \\ \texttt{IPAR}(2) = 7 &\rightarrow p_2 \text{ has been stored in RPAR}(7) \\ \texttt{IPAR}(3) = 5 &\rightarrow p_3 \text{ has been stored in RPAR}(5) \end{aligned}$$

JAC If you have set INFO(5)=0, you can ignore this parameter by treating it as a dummy argument. Otherwise, you must provide a subroutine to return information to compute the iteration matrix of partial derivatives of the form:

$$P = \frac{\partial f}{\partial y} + C_j \frac{\partial f}{\partial y'}$$

where C_j is a scalar. Note that if you are solving for sensitivities, P is the partial derivative matrix of the state variables only.

The subroutine should have the following form:

```
      SUBROUTINE JAC(NY, T, Y, YPRIME, A, NJAC, JROW,
     +          JCOL, JYDOT, NJYDOT, ICHVAR, RPAR, IPAR,
     +          IRTN, JDTYPE)
```

where:

`NY:IN` The number of equations in the DAE (excluding the sensitivity system.

`ICHVAR:IN` Equals 1 if the values in `Y` and `YPRIME` have changed since the last call to JAC.

`A:OUT` Array of length `NEJAC` used to define the matrix of partial derivatives as a sequence of triples.

For example:

$$\texttt{A(K)} = \frac{\partial f_i}{\partial y_j} \quad i = \texttt{JROW(K)} \quad j = \texttt{JCOL(K)}$$

if the index `K` is does not appear in any of the elements of `JYDOT` from 1 to `NYJDOT`. If `K` does appear in the array of `YJDOT` then `A(K)` is defined as follows:

$$\texttt{A(K)} = \frac{\partial f_i}{\partial y'_j} \quad i = \texttt{JROW(K)} \quad j = \texttt{JCOL(K)}$$

Note that `JROW`, `JCOL`, and `JYDOT` must be set before any calls to DSSL48 and that the subroutine `JAC` must not alter `T`, `Y(*)`, `YPRIME(*)`, `JROW(*)`, `JCOL(*)`, `NEQ`, `NJAC`, `JYDOT(*)` or `NJYDOT`.

`NEJAC:IN` The Jacobian is specified in terms of four arrays: two integer arrays indicating the row and column positions in the iteration matrix of the values contained in the third array, and a fourth array that gives the offsets of the derivatives with respect to y'. `NEJAC` is the length of each of the first three vectors. Note that the derivatives of the residuals with respect to `Y` and `YDOT` should be specified as separate triples.

`JROW:IN` An integer array indicating the row index of the nonzero elements of the Jacobian. The specification of `JROW` and `JCOL` defines the sparsity pattern of the merged Jacobian matrix.

`JCOL:IN` An integer array indicating the column index of the nonzero elements of the merged Jacobian. The specification of `JROW` and `JCOL` defines the sparsity pattern of the merged Jacobian matrix.

For example, if `JROW`(1)=1 and `JCOL`(1)=2, then the first equation has a nonzero derivative with respect to the second variable or the time derivative of the second variable. Furthermore, if this derivative is to be calculated analytically, then it will appear as the first element in the array of values specified for the triples.

`JYDOT:IN` Gives the position of derivatives with respect to YDOT in the array of triples. Thus, if `JYDOT`(1)=4, then the fourth value in the Jacobian array of triples corresponds to the value of the derivative with respect to y'_n, where $n =$ `JCOL`(4). Note that the values of `JYDOT` must form an increasing sequence, i.e., `JYDOT`(1) $\leq$ `JYDOT`(2) $\ldots \leq$ `JYDOT`(`NJYDOT`).

NJYDOT:IN The length of the vector JYDOT. This should be equal to the number of nonzero derivatives of the with respect to y'. NJYDOT $= \sum_{j=1}^{\mathtt{NY}} \sum_{i=1}^{\mathtt{NY}} \frac{\partial g_i}{\partial y'_j}$.

JDTYPE:IN Vector of length NEJAC specifying the method to be used to calculate the corresponding element of the Jacobian matrix. The value of JDTYPE(I) specifies the method used to calculate the derivative of the residual equation JROW(I) with respect to the variable Y(JCOL(I)) or YPRIME(JCOL(I)) in the case that an element of JYDOT = I. The values of JDTYPE have the following meaning. Analytic and constant derivatives are calculated using the user-supplied subroutine JAC. Numerical derivatives are calculated by by DSL48S using finite differences.

- -3 : constant derivative with respect to YPRIME
- -2 : analytic derivative with respect to YPRIME
- -1 : numerical derivative with respect to YPRIME
- 0 : numerical derivative with respect to Y
- 1 : analytic derivative with respect to Y
- 2 : constant derivative with respect to Y

2.2 Output after any return by DSL48S

The principal aim of DSL48S is to return a computed solution at TOUT, although it is also possible to obtain intermediate results along the way. To find out whether the code achieved its goal or if the integration process was interrupted before the task was completed, you must check the IDID parameter.

T The solution was successfully advanced to the output value of T.

Y(*) Contains the computed solution approximation at T.

YPRIME Contains the computed derivative approximation at T.

IDID Reports what the code did. Positive values of IDID indicate that the task was completed, while negative values indicate that the task was interrupted.

- IDID = 1 – A step was successfully taken in the intermediate-output mode. The code has not yet reached TOUT.
- IDID = 2 – The integration to TSTOP was successfully completed (T=TSTOP) by stepping exactly to TSTOP.
- IDID = 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 = -1 – A large amount of work has been expended. (About 500 steps)
- IDID = -2 – The error tolerances are too stringent.
- IDID = -3 – The local error test cannot be satisfied because you specified a zero component in ATOL and the corresponding computed solution component is zero. Thus, a pure relative error test is impossible for this component.
- IDID = -6 – DSL48S had repeated error test failures on the last attempted step.
- IDID = -7 – The corrector could not converge.

- IDID = -8 – The matrix of partial derivatives is singular.
- IDID = -9 – The corrector could not converge. There were repeated error test failures in this step.
- IDID =-10 – The corrector could not converge because IRES was equal to minus one.
- IDID =-11 – IRES equal to -2 was encountered and control is being returned to the calling program.
- IDID =-12 – DSL48S failed to compute the initial YPRIME.
- IDID = -13,. . . ,-32 – Not applicable for this code
- IDID = -33 – The code has encountered trouble from which it cannot recover. A message is printed explaining the trouble and control is returned to the calling program. For example, this occurs when invalid input is detected.

RTOL, ATOL These quantities remain unchanged except when IDID = -2. In this case, the error tolerances have been increased by the code to values which are estimated to be appropriate for continuing the integration. However, the reported solution at T was obtained using the input values of RTOL and ATOL.

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

RWORK(3)– Contains the step size H to be attempted on the next step.

RWORK(4)– Contains 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)– Contains the stepsize used on the last successful step.

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

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

IWORK(11)–Contains the number of steps taken so far.

IWORK(12)–Contains the number of calls to RES so far.

IWORK(13)–Contains the number of evaluations of the matrix of partial derivatives needed so far.

IWORK(14)–Contains the total number of error test failures so far.

IWORK(15)–Contains the total number of convergence test failures so far (includes singular iteration matrix failures).

IWORK(18)–Position in RWORK of the start of the Nordsieck array.

IWORK(19)–Contains the number of calls to the SENRHS routine so far.

IWORK(20)–Contains the number of times that the sensitivity equation residuals were evaluated.

2.3 What to do to continue the integration

This code is organized so that subsequent calls to continue the integration involve little (if any) additional effort on your part. You must monitor the IDID parameter in order to determine what to do next.

Recalling that the principal task of the code is to integrate from T to TOUT (the interval mode), usually all you will need to do is specify a new TOUT upon reaching the current TOUT.

Do not alter any quantity not specifically permitted below, in particular do not alter NEQ, T, Y(*), YPRIME(*), RWORK(*), IWORK(*) or the differential equation in subroutine RES. Any such alteration constitutes a new problem and must be treated as such, i.e., you must start afresh.

You cannot change from vector to scalar error control or vice versa (INFO(2)), but you can change the size of the entries of RTOL, ATOL. Increasing a tolerance makes the equation easier to integrate. Decreasing a tolerance will make the equation harder to integrate and should generally be avoided.

You can switch from the intermediate-output mode to the interval mode (INFO(3)) or vice versa at any time.

If it has been necessary to prevent the integration from going past a point TSTOP (INFO(4), RWORK(1)), keep in mind that the code will not integrate to any TOUT beyond the currently specified TSTOP. Once TSTOP has been reached you must change the value of TSTOP or set INFO(4)=0. You may change INFO(4) or TSTOP at any time but you must supply the value of TSTOP in RWORK(1) whenever you set INFO(4)=1.

Do not change INFO(5), IWORK(1), or IWORK(2) unless you are going to restart the code.

*W*hat to do following a completed task:

- If IDID = 1, call the code again to continue the integration another step in the direction of TOUT.
- If IDID = 2 or 3, define a new TOUT and call the code again. TOUT must be different from T. You cannot change the direction of integration without restarting.

*W*hat to do following an interrupted task:

To show the code that you realize the task was interrupted and that you want to continue, you must take appropriate action and set INFO(1) = 1

- If IDID = -1, The code has taken about 500 steps. If you want to continue, set INFO(1) = 1 and call the code again. An additional 500 steps will be allowed.
- If IDID = -2, The error tolerances RTOL, ATOL have been increased to values the code estimates appropriate for continuing. You may want to change them yourself. If you are sure you want to continue with relaxed error tolerances, set INFO(1)=1 and call the code again.
- If IDID = -3, A solution component is zero and you set the corresponding component of ATOL to zero. If you are sure you want to continue, you must first alter the error criterion to use positive values for those components of ATOL corresponding to zero solution components, then set INFO(1)=1 and call the code again.
- IDID = -4,-5 — Cannot occur with DSL48S.
- If IDID = -6, Repeated error test failures occurred on the last attempted step in DSL48S. A singularity in the solution may be present. If you are absolutely certain you want to continue, you should restart the integration. (Provide initial values of Y and YPRIME which are consistent)
- If IDID = -7, Repeated convergence test failures occurred on the last attempted step in DSL48S. An inaccurate or ill-conditioned Jacobian may be the problem. If you are absolutely certain you want to continue, you should restart the integration.
- If IDID = -8, The matrix of partial derivatives is singular. Some of your equations may be redundant. DSL48S cannot solve the problem as stated. It is possible that the redundant equations could be removed, and then DSL48S could solve the problem. It is also possible that a solution to your problem either does not exist or is not unique.

- If `IDID` = -9, DSL48S had multiple convergence test failures, preceeded by multiple error test failures, on the last attempted step. It is possible that your problem is ill-posed, and cannot be solved using this code. Or, there may be a discontinuity or a singularity in the solution. If you are absolutely certain you want to continue, you should restart the integration.
- If `IDID` =-10, DSL48S had multiple convergence test failures because `IRES` was equal to minus one. If you are absolutely certain you want to continue, you should restart the integration.
- If `IDID` =-11, `IRES`=-2 was encountered, and control is being returned to the calling program.
- If `IDID` =-12, DSL48S failed to compute the initial `YPRIME`. This could happen because the initial approximation to `YPRIME` was not very good, or if a `YPRIME` consistent with the initial Y does not exist. The problem could also be caused by an inaccurate or singular iteration matrix.
- `IDID` = -13,..,-32 — Cannot occur with this code.
- If `IDID`= -33, you cannot continue the solution of this problem. An attempt to do so will result in your run being terminated.

2.4 Error messages

The SLATEC error print routine XERMSG is called in the event of unsuccessful completion of a task. This routine is included uncompiled in the DSL48S distribution so that it may be replaced, if necessary for proper integration with your program. If you want to replace XERMSG with your own error routine, see the file `error.f` for the proper form of the error handling subroutines.

Most of these errors are treated as "recoverable errors", which means that (unless the user has directed otherwise) control will be returned to the calling program for possible action after the message has been printed.

In the event of a negative value of `IDID` other than -33, an appropriate message is printed and the "error number" printed by XERMSG is the value of `IDID`. There are quite a number of illegal input errors that can lead to a returned value `IDID`=-33. The conditions and their printed "error numbers" are as follows:

Error number	Condition
1	Some element of `INFO` vector is not zero or one.
2	NEQ .le. 0
3	MAXORD not in range.
4	LRW is less than the required length for `RWORK`.
5	LIW is less than the required length for `IWORK`.
6	Some element of `RTOL` is .lt. 0
7	Some element of `ATOL` is .lt. 0
8	All elements of `RTOL` and `ATOL` are zero.
9	`INFO`(4)=1 and `TSTOP` is behind `TOUT`.
10	HMAX .lt. 0.0
11	`TOUT` is behind `T`.
12	`INFO`(8)=1 and H0=0.0
13	Some element of WT is .le. 0.0
14	`TOUT` is too close to T to start integration.
15	`INFO`(4)=1 and `TSTOP` is behind T.
16	–(Not used in this version)–
17	ML illegal. Either < 0 or >NEQ
18	MU illegal. Either <0 or >NEQ
19	`TOUT` = T.
20	Some element of the DSL48S part of `INFO`[16→20] has not been specified correctly.
21	Sensitivity analysis option has been enabled but `NEQ` does not equal: # of state var.*(# of sensitivity var. + 1) as required.
22	Error test restriction is being applied to illegal values of the Y vector.
23	IPAR contains negative values.
24	Workspace arrays too small for current problem

If DSL48S is called again without any action taken to remove the cause of an unsuccessful return, XERMSG will be called with a fatal error flag, which will cause unconditional termination of the program. There are two such fatal errors:

Error number -998: The last step was terminated with a negative value of `IDID` other than -33, and no appropriate action was taken.

Error number -999: The previous call was terminated because of illegal input (`IDID`=-33) and there is illegal input in the present call, as well. (Suspect infinite loop.)

3 The `test.f` program

```
      PROGRAM TEST
C     This program is a test of the DSL48S code.  The problem
C     is from:
C
C     T. Maly and L.R. Petzold, "Numerical methods and software for
C     sensitivity analysis of differential-algebraic equations",
C     Applied Numerical Mathematics 20:57-79 (1996)
C
C     This is a two dimensional ODE with three parameters.  The problem
C     has the form:
C
C     $X1=-(P1+P3)X1^2
C     $X2=P1*X1^2 - P2*X2
C     X1(0)=1
C     X2(0)=0
C
C     where the X's are state variables, '$' denotes time derivatives,
C     and the parameters have the values:
C
C     P1=0.9875
C     P2=0.2566
C     P3=0.3323
C
C     The resulting ODE and sensitivity system defines an 8x8 system of
C     equations.

      DOUBLE PRECISION T, Y(8), YPRIME(8), TOUT, RTOL(1),
     *   ATOL(1), RWORK(1000), BOUND(4),RPAR(3),TSTOP
      INTEGER NEQ, IDID, LRW, LIW, IWORK(1000), INFO(25),
     +        IRLIST(5),JCLIST(5),IDLIST(2),IDFLAG(5),
     +        NZ,IDNZ, IPAR(3),I
      EXTERNAL RES,JAC,SENRHS

C     Number of equations- ODE and sensitivities
      NEQ=8

C     Initial time and maximum initial time step
      T=0.0D0
      TSTOP=4.0D2
      TOUT=10.0D0

C     Info array parameters- see README for meanings
      INFO(1)=0
      INFO(2)=0
      INFO(3)=1
      INFO(4)=0
      INFO(5)=1
      INFO(6)=1
```

```
      INFO(7)=0
      INFO(8)=0
      INFO(9)=0
      INFO(10)=0
      INFO(11)=0
      INFO(12)=0
      INFO(13)=0
      INFO(14)=0
      INFO(15)=0
      INFO(16)=0
      INFO(17)=3
      INFO(18)=0
      INFO(19)=0
      INFO(20)=0
      INFO(21)=0
      INFO(22)=0
      INFO(23)=0

C     Relative and absolute integration tolerances
      RTOL(1)=1.0D0/1.0D5
      ATOL(1)=RTOL(1)

C     RWORK and IWORK Lengths
      LRW=1000
      LIW=1000

C     Set up RPAR and IPAR with the parameters
      IPAR(1)=1
      RPAR(1)=0.9875D0
      IPAR(2)=2
      RPAR(2)=0.2566D0
      IPAR(3)=3
      RPAR(3)=0.3323D0

C     Set up jacobian
      NZ=5
      IRLIST(1)=1
      IRLIST(2)=1
      IRLIST(3)=2
      IRLIST(4)=2
      IRLIST(5)=2
      JCLIST(1)=1
      JCLIST(2)=1
      JCLIST(3)=2
      JCLIST(4)=1
      JCLIST(5)=2

      IDNZ=2
      IDLIST(1)=1
      IDLIST(2)=3
```

```
      IDFLAG(1)= -3
      IDFLAG(2)= 1
      IDFLAG(3)= -3
      IDFLAG(4)= 1
      IDFLAG(5)= 1

C     Initial Conditions
      Y(1)=1.0D0
      Y(2)=0.0D0
      Y(3)=0.0D0
      Y(4)=0.0D0
      Y(5)=0.0D0
      Y(6)=0.0D0
      Y(7)=0.0D0
      Y(8)=0.0D0

      YPRIME(1)=-(RPAR(1)+RPAR(3))*Y(1)**2
      YPRIME(2)=RPAR(1)*Y(1)**2
      YPRIME(3)=-Y(1)**2
      YPRIME(4)=Y(1)**2
      YPRIME(5)=0.0D0
      YPRIME(6)=0.0D0
      YPRIME(7)=-Y(1)**2
      YPRIME(8)=0.0D0

      OPEN(UNIT=3,FILE='test.out')
      WRITE(3,1001)'TIME','Y(1)','Y(2)','S(1,1)','S(2,1)','S(1,2)',
     +             'S(2,2)','S(1,3)','S(2,3)'
      WRITE(3,1000)T,(Y(I),I=1,8)
10    CONTINUE
      CALL DSL48S(RES, SENRHS, NEQ, T, Y, YPRIME, TOUT, INFO,
     *   RTOL, ATOL, IDID, RWORK, LRW, IWORK,
     *   LIW,RPAR,IPAR,JAC,NZ,IRLIST,JCLIST,IDLIST,IDNZ,IDFLAG,BOUND)
      WRITE(3,1000)T,(Y(I),I=1,8)
      TOUT=TSTOP
      IF (T.LE.(TSTOP-ATOL(1))) GOTO 10
      CLOSE(3)

      WRITE(6,*)'DSL48S integration complete'
1000  FORMAT(9E15.8)
1001  FORMAT(9A15)
      END

*************************************************************

      SUBROUTINE RES(NY,T,Y,YPRIME,DELTA,IRES,ICHVAR,RPAR,IPAR)

      INTEGER IRES,IPAR(*),NY,ICHVAR
      DOUBLE PRECISION T,Y(NY),YPRIME(NY),DELTA(NY),RPAR(*)
```

```
C
      DELTA(1)=YPRIME(1)+(RPAR(IPAR(1))+RPAR(IPAR(3)))*Y(1)**2
      DELTA(2)=YPRIME(2)-RPAR(IPAR(1))*Y(1)**2+RPAR(IPAR(2))*Y(2)

      RETURN
      END

*************************************************************

      SUBROUTINE SENRHS(NEQ,T,Y,YPRIME,DELTA,IRES,RPAR,IPAR)

      INTEGER IRES,IPAR(*),NEQ
      DOUBLE PRECISION T,Y(*),YPRIME(*),DELTA(*),RPAR(*)
C
      DELTA(3)=Y(1)**2
      DELTA(4)=-Y(1)**2
      DELTA(5)=0.0D0
      DELTA(6)=Y(2)
      DELTA(7)=Y(1)**2
      DELTA(8)=0.0D0

      RETURN
      END

*************************************************************

      SUBROUTINE JAC(NEQ,T,Y,YPRIME,AJAC,NJAC,JROW,JCOL,
     &               JYDOT,NJYDOT,ICHVAR,RPAR,IPAR,IRTN)
C
C     Argument List Declarations
      INTEGER IPAR(*), IRTN, JCOL(*), JROW(*), JYDOT(*),
     &        NEQ, NJAC, NJYDOT,ICHVAR
      DOUBLE PRECISION AJAC(*), RPAR(*), T, Y(*), YPRIME(*)
C
C     Local Variables
C
C     First Excutable statement
      AJAC(1)=1.0D0
      AJAC(2)=2*(RPAR(IPAR(1))+RPAR(IPAR(3)))*Y(1)
      AJAC(3)=1.0D0
      AJAC(4)=-2*RPAR(IPAR(1))*Y(1)
      AJAC(5)=RPAR(IPAR(2))

      RETURN
      END
```

References

[1] R. Allgor, *Modeling and Computational Issues in the Development of Batch Processes*, PhD thesis, Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA, 1997.

[2] K. Brenan, S. Campbell, and L. Petzold, *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, SIAM, Philadelphia, PA, 1996.

[3] W. Feehery, *Dynamic Optimization with State Variable Path Constraints*, PhD thesis, Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA, 1998.

[4] W. Feehery, J. Tolsma, and P. Barton, *Efficient sensitivity analysis of large-scale differential-algebraic equations*, Applied Numerical Mathematics, 25 (1997), pp. 41–54.