DAEPACK

DSL48S Supplementary Manual

Version 1.0

Using DAEPACK Generated Code with DSL48S

John E. Tolsma

November 25, 2000

Contents

1 Overview **4**

2 Generating Necessary Code **4**

3 Simulation Executive **15**

4 Conclusion **20**

This document describes how to use the DAEPACK code generation components to generate automatically the additional code required when using DSL48S for numerical integration and parametric sensitivity analysis. A small example problem is provided to illustrate the entire process.

1 Overview

DAEPACK numeric component DSL48S is a large-scale differential-algebraic equation (DAE) integrator based on the popular code DASSL. DSL48S contains a number of extensions including exploitation of sparsity using the Harwell library MA48 routines and an efficient staggered-corrector sensitivity algorithm. A detailed description of DSL48S is beyond the scope of this paper, and more information can be found in [2, 3]. This document supplements the DSL48S manual [2] by describing how the additional information required by DSL48S (e.g., sparse Jacobian matrix, analytical partial derivatives with respect to model parameters, and sparsity information) can be generated automatically with other DAEPACK components.

A small example problem, adapted from [4], is used to illustrate the entire process, starting from a Fortran code representing the model residuals and ending with state and sensitivity trajectories obtained with DSL48S. The model equations are

$$\dot{y}_1 = -(p_1 + p_3)y_1^2 \tag{1}$$
$$\dot{y}_2 = p_1 y_1^2 - p_2 y_2 \tag{2}$$

with initial conditions

$$y_1(0) = 1 \tag{3}$$
$$y_2(0) = 0. \tag{4}$$

Although this problem is very small and the additional information can be readily generated by hand, it serves as an illustrative example and the accompanying code can be used as a template for larger, more complicated examples.

2 Generating Necessary Code

Obviously, a necessary piece of information required when numerically integrating a system of equations is some representation of the system of equations of interest. We assume that these equations are available in the form of a Fortran subroutine returning the residuals of the model equations, i.e., $f(\dot{y}, y, t; p) = 0$. In this example,

$$f_1 = \dot{y}_1 + (p_1 + p_3)y_1^2 \tag{5}$$
$$f_2 = \dot{y}_2 - p_1 y_1^2 + p_2 y_2. \tag{6}$$

DSL48S allows this residual routine to be passed as an argument to the integrator. The residual subroutine must have the following interface:

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

where `NY` is the number of states, `T` is the independent variable, `Y` and `YDOT` are double precision arrays of length `NY` holding the current values of the states and time derivatives, respectively, `DELTA` is a double precision array of length `NY` returning the values of the residuals at (`T`, `Y`, `YDOT`), `IRES` is an error return code set to a nonzero value if an error has occurred during residual evaluation (not modified otherwise), `ICHVAR` is equal to 1 if the `T`, `Y`, or `YDOT` have changed values since the last call to `RES`, and `RPAR` and `IPAR` are double precision and integer arrays, respectively, containing any additional user-supplied information (in this example, the values of the parameters). The entire residual subroutine for this example is shown below:

```
      SUBROUTINE RES(NY,T,Y,YDOT,DELTA,IRES,ICHVAR,RPAR,IPAR)
C
      IMPLICIT NONE
      INTEGER IRES,IPAR(*),NY,ICHVAR
      DOUBLE PRECISION T,Y(NY),YDOT(NY),DELTA(NY),RPAR(100)
C
      DELTA(1)=YDOT(1)+(RPAR(IPAR(1))+RPAR(IPAR(3)))*Y(1)**2
      DELTA(2)=YDOT(2)-RPAR(IPAR(1))*Y(1)**2+RPAR(IPAR(2))*Y(2)
C
      RETURN
      END
```

where `Y(I)`$=y_i$, `YDOT(I)`$=\dot{y}_i$, $i = 1, 2$, and `RPAR(IPAR(J))`$=p_j$, $j = 1, 2, 3$. For many numerical integrators this information is sufficient for performing the calculation (partial derivatives can be approximated by finite differences and dense linear algebra performed). However, DSL48S requires additional information in order to perform the calculation more efficiently. In particular, DSL48S requires the sparse Jacobian of the model residuals (partial derivatives of f with respect to y and $\dot{y}$ returned in *sparse triplet* (*coordinate*) form[1]), the sparsity pattern, and partial derivatives of f with respect to the parameters.[1] Fortunately, other components in the DAEPACK library can be used to generate this additional information automatically with little user intervention. It is assumed that the reader is familiar with the use of the the DAEPACK code generation components (see the manual for details [5]).

First, the sparse Jacobian of the model equations with respect to the states and time derivatives is generated. The specification file for generating this code with the DAEPACK derivative code generator is shown below:

```
GENERATE DERIVS
   ROOT          : res
   OUTFILE       : resad
   SUFFIX        : ad
   SPARSESTORAGE : true
   INDEPENDENT   : y,ydot
   DEPENDENT     : delta
END
```

(This is required when using the code generation components with the command-line version of DAEPACK. Alternatively, these same options can be specified through the GUI available on the Windows 9x and Windows NT platforms.) The resulting code is shown below:

```
C###  - DAEPACK Version 1.0 - Copyright (C) MIT
C###  - DERIVATIVE COMPUTATION -
C###  - GENERATED: Fri May 19 17:59:57 2000
      subroutine resad(ny,t,y,ydot,delta,ires,ichvar,rpar,ipar,
     $ zzzderiv,zzzne,zzzirn,zzzjcn,zzziw)
!!independent { y, ydot }
!!dependent   { delta }
```

[1] Alternatively, DSL48S can be provided with a dense sparsity pattern and specified to compute all partial derivatives numerically, however, as shown in this paper, this is unnecessary.

```
      implicit none
      integer ichvar
      integer ny
      double precision t
      integer ires
      double precision delta(ny)
      integer ipar(*)
      double precision y(ny)
      double precision ydot(ny)
      double precision rpar(100)
C###  Additional arguments for partial derivative computation
C###     zzzderiv - partial derivative array stored in sparse matrix
C###                format, pattern contained in zzzirn and zzzjcn.
C###     zzzne    - number of entries in zzzderiv.
C###     zzzirn   - row numbers for entries in zzzderiv.
C###     zzzjcn   - column numbers for entries in zzzderiv.
      double precision zzzderiv(*)
      integer zzzne,zzzirn(zzzne),zzzjcn(zzzne)
C###     zzziw    - integer workspace array.
      integer zzziw(*)
C###    zzzimodel - unique key for this model.
      integer zzzimodel
C###  Define elementary variables and adjoints
      double precision zzzv1,zzzvbar1
      double precision zzzv2,zzzvbar2
      double precision zzzv3,zzzvbar3
      double precision zzzv4,zzzvbar4
      double precision zzzv5,zzzvbar5
      double precision zzzv6,zzzvbar6
      double precision zzzv7,zzzvbar7
      double precision zzzv8,zzzvbar8
      double precision zzzv9,zzzvbar9
      double precision zzzv10,zzzvbar10
      double precision zzzv11,zzzvbar11
C###
C###  Active variable offsets and loop control variables
      integer deltaoft
      integer yoft
      integer ydotoft
      integer zzzn
      integer zzzm
      integer zzzi
      integer zzzindx
C###  Variable offsets for independent, dependent, and
C###  non-common block local active variables.
      deltaoft=0
```

```
      yoft=deltaoft+ny
      ydotoft=yoft+ny
C###  Number of independent and dependent variables.
      zzzn=ny+ny
      zzzm=ny
C###  Create unique identifier for this model.
      call GETMDLID(zzzimodel,'resad')
C###  Create SVM for functions/subroutines.
      call CREATESV(zzzimodel,1,zzzn)
C###  Construct mapping between independent variable offsets and indices
      zzzindx=0
      do zzzi=1,ny
         zzzindx=zzzindx+1
         zzziw(zzzindx)=yoft+zzzi
      end do
      do zzzi=1,ny
         zzzindx=zzzindx+1
         zzziw(zzzindx)=ydotoft+zzzi
      end do
C###  Initialize independent variable gradients to Cartesian basis vectors.
      call DSETIVCV(1,zzzn,zzziw)
C###
C###  Compute elementary variables
      zzzv1=ydot(1)
      zzzv2=rpar(ipar(1))
      zzzv3=rpar(ipar(3))
      zzzv4=zzzv2+zzzv3
      zzzv5=y(1)
      zzzv6=2
      zzzv7=zzzv5**zzzv6
      zzzv8=zzzv4*zzzv7
      zzzv9=zzzv1+zzzv8
C###  Compute elementary partial derivatives
      zzzvbar9=1.0d0
      zzzvbar8=zzzvbar9
      zzzvbar7=zzzvbar8*zzzv4
      zzzvbar5=zzzvbar7*zzzv6*zzzv5**(zzzv6-1.0d0)
      zzzvbar1=zzzvbar9
C###
      delta(1)=zzzv9
C###
      call DSVM2(deltaoft+1,1,
     $           zzzvbar5,yoft+1,1,
     $           zzzvbar1,ydotoft+1,1)
C###
C###  Compute elementary variables
```

```
      zzzv1=ydot(2)
      zzzv2=rpar(ipar(1))
      zzzv3=y(1)
      zzzv4=2
      zzzv5=zzzv3**zzzv4
      zzzv6=zzzv2*zzzv5
      zzzv7=zzzv1-zzzv6
      zzzv8=rpar(ipar(2))
      zzzv9=y(2)
      zzzv10=zzzv8*zzzv9
      zzzv11=zzzv7+zzzv10
C###  Compute elementary partial derivatives
      zzzvbar11=1.0d0
      zzzvbar10=zzzvbar11
      zzzvbar9=zzzvbar10*zzzv8
      zzzvbar7=zzzvbar11
      zzzvbar6=-zzzvbar7
      zzzvbar5=zzzvbar6*zzzv2
      zzzvbar3=zzzvbar5*zzzv4*zzzv3**(zzzv4-1.0d0)
      zzzvbar1=zzzvbar7
C###
      delta(2)=zzzv11
C###
      call DSVM3(deltaoft+2,1,
     $           zzzvbar9,yoft+2,1,
     $           zzzvbar3,yoft+1,1,
     $           zzzvbar1,ydotoft+2,1)
C###
      goto 11111
C###  Construct derivative matrix and sparsity pattern before returning.
11111 continue
C###  Construct mapping between dependent variable offsets and indices
      zzzindx=0
      do zzzi=1,ny
         zzzindx=zzzindx+1
         zzziw(zzzindx)=deltaoft+zzzi
      end do
      call DCPRVS(1,zzzm,zzziw,zzzderiv,zzzne,zzzirn,zzzjcn)
C###  Delete SVM for functions/subroutines.
      call DELETESV(zzzimodel,1)
C###  Done
      return
      end
```

The original interface of the residual routine is augmented with five additional arguments: `zzzderiv` is a double precision array holding the nonzero entries of the Jacobian matrix[2], `zzzne` is an integer returning the number of nonzero entries, `zzzirn` and `zzzjcn` are integer arrays holding the row and column indices, respectively, of the sparse Jacobian matrix, and `zzziw` is an integer array with dimension equal to the number of rows or columns in the resulting matrix, whichever is larger (in this example, `zzziw` must have four entries). Arrays `zzzderiv`, `zzzirn`, and `zzzjcn` must be dimensioned to at least the maximum number of nonzero entries expected in the Jacobian. Let `DELTA` $\equiv f$ and `Y`,`YDOT` $\equiv y, \dot{y}$. Calling the subroutine above computes the following information:

$$J(\dot{y}, y, t; p) = \left(\begin{array}{c|c} \frac{\partial f}{\partial y} & \frac{\partial f}{\partial \dot{y}} \end{array} \right). \tag{7}$$

The order of the dependent and independent variables in the resulting Jacobian matrix is given by their order in the specification file. Notice that the model residuals are also computed during the Jacobian evaluation. This is done with very little additional cost due to the exploitation of common subexpressions between the model equations and their derivatives.

DSL48S uses the Jacobian routine to compute the following iteration matrix:

$$\alpha \frac{\partial f}{\partial \dot{y}} + \frac{\partial f}{\partial y} \tag{8}$$

Consequently, partial derivatives with respect to $\dot{y}$ and y must have the same column indices ($\dot{y}_i$ is represented as a function of y_i and t). Furthermore, DSL48S requires three additional pieces of information: `NJYDOT`, the number of elements in the sparse Jacobian matrix with respect to time derivatives, `JYDOT(1:NJYDOT)`, the indices in the sparse Jacobian that are partial derivatives with respect to time derivatives, and `JDTYPE(1:zzzne)`, indicating how the partial derivatives are computed (see DSL48S documentation [2]). In addition, DSL48S requires a specific interface for the Jacobian subroutine. (Like the residual routine, the Jacobian routine is also passed to DSL48S through the argument list.) All of this information can be obtained by placing the following wrapper around the generated code:

```
      SUBROUTINE JAC(NY,T,Y,YDOT,AJAC,NJAC,IROW,JCOL,
     1               JYDOT,NJYDOT,ICHVAR,RPAR,IPAR,IRES,JDTYPE)
C
      IMPLICIT NONE
      INTEGER IPAR(*), IRES, JCOL(*), IROW(*), JYDOT(*),
     1        NY, NJAC, NJYDOT,ICHVAR,JDTYPE(*)
      DOUBLE PRECISION AJAC(*), RPAR(*), T, Y(*), YDOT(*)
C
      DOUBLE PRECISION DDPWRK
      INTEGER IDPWRK
      COMMON /DAEPACK/ DDPWRK,IDPWRK
      DIMENSION DDPWRK(2),IDPWRK(4)
C
      INTEGER I
      EXTERNAL RESAD
C
```

[2]The entries in this array are the nonzero entries based on structural information, i.e., some of the elements of `zzzderiv` may be zero, however, they are not identically zero because the variable doesn't appear in the equation.

```
      CALL RESAD(NY,T,Y,YDOT,DDPWRK,IRES,ICHVAR,RPAR,IPAR,
     1   AJAC,NJAC,IROW,JCOL,IDPWRK)
C.....SET JACOBIAN INFORMATION NJYDOT AND JYDOT
      NJYDOT=0
      DO I=1,NJAC
         IF(JCOL(I).GT.NY) THEN
C........THIS ENTRY IS A PARTIAL DERIVATIVE W.R.T. A TIME DERIVATIVE
C        ADJUST COLUMN INDEX AND ADD POSITION TO JYDOT.
            JCOL(I)=JCOL(I)-NY
            NJYDOT=NJYDOT+1
            JYDOT(NJYDOT)=I
            JDTYPE(I)=-2
         ELSE
            JDTYPE(I)=1
         END IF
      END DO
C
      RETURN
      END
```

where arguments `NY`, `T`, `Y`, `YDOT`, `ICHVAR`, `RPAR`, `IPAR`, and `IRES` are the same as those described for the residual routine, `AJAC` returns the nonzero entries of the Jacobian matrix, `NJAC` is the number of nonzero entries in the Jacobian, `IROW` and `JCOL` are the row and column indices, respectively, of the Jacobian matrix, `NJYDOT` and `JYDOT` are described above, and `JDTYPE` an integer array containing the type of derivative (see [2]). The DAEPACK common block contains a double precision array `DDPWRK` (used to hold the residual values that are computed during the Jacobian evaluation) and an integer array `IDPWRK` for the workspace required by the DAEPACK generated code.

Note that this wrapper is generic and can be used for any problem (where the generated code has the same interface as shown in this code and the DAEPACK common block workspaces are dimensioned to the necessary size).

The purpose of this wrapper is to provide DSL48S with the required Jacobian subroutine interface, to adjust the column indices of the partial derivatives with respect to time derivatives, and to determine `NJYDOT`, `JYDOT`, and `JDTYPE`. Since this code determines the sparsity pattern as it evaluates the Jacobian, it is not necessary to generate additional code specifically for this purpose (although, DAEPACK provides a component that can be used to generate code that simply returns the sparsity pattern of a given model). This is the only additional code that must be generated when performing numerical integration.

When performing a parametric sensitivity analysis, DSL48S also requires partial derivatives of the model equations with respect to the parameters, used to compute the sensitivity equation right-hand-sides. The sensitivity equations are a system of $n_p n_y$ linear differential equations shown below.

$$\frac{\partial f}{\partial \dot{y}}\frac{\partial \dot{y}}{\partial p_i} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial p_i} = -\frac{\partial f}{\partial p_i} \qquad i = 1, \ldots, n_p \tag{9}$$

where n_y is the number of states and n_p is the number of parameters. The partial derivatives $\partial \dot{y}/\partial p_i$ and $\partial y/\partial p_i$ are the sensitivities of interest. DSL48S requires a subroutine to return the values of $\partial f/\partial p_i$,

$i = 1, \ldots, n_p$. Like the residual and Jacobian routines, the user provides these partial derivatives through a subroutine passed to DSL48S. The interface of this sensitivity equation right-hand-sides subroutine is

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

where `NEQ` is the dimension of the original model and sensitivity equations (`NEQ=NP*(NY+1)` where `NP` is the number of parameters and `NY` is the number of states) and the other arguments are the same as those described for the residual and Jacobian routines. In this subroutine, entries `NY+1` through `NP*(NY+1)` of `DELTA` are filled with the partial derivatives of the model equations with respect to each of the `NP` parameters. Specifically, entries `DELTA(NY+1:2*NY)` are filled with $\partial f/\partial p_1$, entries `DELTA(2*NY+1:3*NY)` are filled with $\partial f/\partial p_2$, and so on. In other words, after calling `RES` and `SENRHS`, `DELTA` contains

$$\begin{pmatrix} f \\ \frac{\partial f}{\partial p_1} \\ \vdots \\ \frac{\partial f}{\partial p_{n_p}} \end{pmatrix}. \tag{10}$$

Again, DAEPACK can be used to generate automatically this partial derivative information. In this example, the three parameters are `RPAR(IPAR(1))`$=p_1$, `RPAR(IPAR(2))`$=p_2$, and `RPAR(IPAR(3))`$=p_3$. Given the original residual subroutine and the specification file

```
GENERATE DERIVS
   ROOT          : res
   OUTFILE       : resadp
   SUFFIX        : adp
   SPARSESTORAGE : false
   INDEPENDENT   : rpar(ipar(1:3))
   DEPENDENT     : delta
END
```

the following code is generated with the DAEPACK derivative code generator:

```
C###  - DAEPACK Version 1.0 - Copyright (C) MIT
C###  - DERIVATIVE COMPUTATION -
C###  - GENERATED: Fri May 19 17:59:57 2000
      subroutine resadp(ny,t,y,ydot,delta,ires,ichvar,rpar,ipar,
     $ zzzldm,zzzderiv,zzziw)
!!independent { rpar(ipar(1:3)) }
!!dependent   { delta }
      implicit none
      integer ichvar
      integer ny
      double precision t
      integer ires
      double precision delta(ny)
      integer ipar(*)
      double precision y(ny)
```

```
      double precision ydot(ny)
      double precision rpar(100)
C###  Additional arguments for partial derivative computation
C###     zzzldm   - leading dimension of zzzderiv (i.e., 2-dimensional array
C###                zzzderiv is declared as zzzderiv(zzzldm,n)).
C###     zzzderiv - partial derivative matrix.
      integer zzzldm
      double precision zzzderiv(zzzldm,*)
C###     zzziw    - integer workspace array.
      integer zzziw(*)
C###    zzzimodel - unique key for this model.
      integer zzzimodel
C###  Define elementary variables and adjoints
      double precision zzzv1,zzzvbar1
      double precision zzzv2,zzzvbar2
      double precision zzzv3,zzzvbar3
      double precision zzzv4,zzzvbar4
      double precision zzzv5,zzzvbar5
      double precision zzzv6,zzzvbar6
      double precision zzzv7,zzzvbar7
      double precision zzzv8,zzzvbar8
      double precision zzzv9,zzzvbar9
      double precision zzzv10,zzzvbar10
      double precision zzzv11,zzzvbar11
C###
C###  Active variable offsets and loop control variables
      integer deltaoft
      integer rparoft
      integer zzzn
      integer zzzm
      integer zzzi
      integer zzzindx
C###  Variable offsets for independent, dependent, and
C###  non-common block local active variables.
      deltaoft=0
      rparoft=deltaoft+ny
C###  Number of independent and dependent variables.
      zzzn=3
      zzzm=ny
C###  Create unique identifier for this model.
      call GETMDLID(zzzimodel,'resadp')
C###  Create SVM for functions/subroutines.
      call CREATESV(zzzimodel,1,zzzn)
C###  Construct mapping between independent variable offsets and indices
      zzzindx=0
      do zzzi=1,3
```

```
         zzzindx=zzzindx+1
         zzziw(zzzindx)=rparoft+ipar(zzzi)
      end do
C###  Initialize independent variable gradients to Cartesian basis vectors.
      call DSETIVCV(1,zzzn,zzziw)
C###
C###  Compute elementary variables
      zzzv1=ydot(1)
      zzzv2=rpar(ipar(1))
      zzzv3=rpar(ipar(3))
      zzzv4=zzzv2+zzzv3
      zzzv5=y(1)
      zzzv6=2
      zzzv7=zzzv5**zzzv6
      zzzv8=zzzv4*zzzv7
      zzzv9=zzzv1+zzzv8
C###  Compute elementary partial derivatives
      zzzvbar9=1.0d0
      zzzvbar8=zzzvbar9
      zzzvbar4=zzzvbar8*zzzv7
      zzzvbar3=zzzvbar4
      zzzvbar2=zzzvbar4
C###
      delta(1)=zzzv9
C###
      call DSVM2(deltaoft+1,1,
     $           zzzvbar3,rparoft+ipar(3),1,
     $           zzzvbar2,rparoft+ipar(1),1)
C###
C###  Compute elementary variables
      zzzv1=ydot(2)
      zzzv2=rpar(ipar(1))
      zzzv3=y(1)
      zzzv4=2
      zzzv5=zzzv3**zzzv4
      zzzv6=zzzv2*zzzv5
      zzzv7=zzzv1-zzzv6
      zzzv8=rpar(ipar(2))
      zzzv9=y(2)
      zzzv10=zzzv8*zzzv9
      zzzv11=zzzv7+zzzv10
C###  Compute elementary partial derivatives
      zzzvbar11=1.0d0
      zzzvbar10=zzzvbar11
      zzzvbar8=zzzvbar10*zzzv9
      zzzvbar7=zzzvbar11
```

```
      zzzvbar6=-zzzvbar7
      zzzvbar2=zzzvbar6*zzzv5
C###
      delta(2)=zzzv11
C###
      call DSVM2(deltaoft+2,1,
     $           zzzvbar8,rparoft+ipar(2),1,
     $           zzzvbar2,rparoft+ipar(1),1)
C###
      goto 11111
C###  Construct derivative matrix before returning.
11111 continue
C###  Construct mapping between dependent variable offsets and indices
      zzzindx=0
      do zzzi=1,ny
         zzzindx=zzzindx+1
         zzziw(zzzindx)=deltaoft+zzzi
      end do
      call DCPRVD(1,zzzm,zzziw,zzzldm,zzzderiv)
C###  Delete SVM for functions/subroutines.
      call DELETESV(zzzimodel,1)
C###  Done
      return
      end
```

In this case, the partial derivatives are returned as a dense matrix rather than in sparse triplet form. Three additional arguments have been appended to the end of the original argument list of `RES`: `zzzldm` the leading dimension of two dimensional array `zzzderiv` holding the computed values of the partial derivatives of the model with respect to the parameters and `zzziw` an integer array of dimension equal to the number of rows or columns in the resulting matrix, whichever is larger. The reason a dense derivative matrix is generated will become apparent below. The generated code above returns

$$\frac{\partial f}{\partial p} = \begin{pmatrix} \frac{\partial f}{\partial p_1} & \frac{\partial f}{\partial p_2} & \frac{\partial f}{\partial p_3} \end{pmatrix} \tag{11}$$

as a dense matrix in *column-major order*. Thus, the first column (and contiguous entries) of `zzzderiv` contains partial derivatives with respect to p_1, the second column contains partial derivatives with respect to p_2, and the third column contains partial derivatives with respect to p_3. Again, a wrapper is required around the generated code in order to satisfy the DSL48S interface requirements:

```
      SUBROUTINE SENRHS(NEQ,T,Y,YDOT,DELTA,IRES,RPAR,IPAR)
C
      IMPLICIT NONE
      INTEGER IRES,IPAR(*),NEQ
      DOUBLE PRECISION T,Y(*),YDOT(*),DELTA(*),RPAR(*)
C
      DOUBLE PRECISION DDPWRK
```

```
      INTEGER IDPWRK
      COMMON /DAEPACK/ DDPWRK,IDPWRK
      DIMENSION DDPWRK(2),IDPWRK(4)
C
      EXTERNAL RESADP
      INTEGER NY,ICHVAR
C
C.....NUMBER OF STATES IS EQUAL TO THE TOTAL NUMBER OF EQUATIONS
C     (STATES AND SENSITIVITIES) DIVIDED BY THE NUMBER OF PARAMETERS
C     PLUS ONE.
      NY=NEQ/4
C
      CALL RESADP(NY,T,Y,YDOT,DELTA,IRES,ICHVAR,RPAR,IPAR,
     1   NY,DELTA(NY+1),IDPWRK)
C
      RETURN
      END
```

Since the matrix returned by `RESADP` is column-major ordered, by specifying the leading dimension of the derivative matrix as `NY` and passing `DELTA(NY+1)` as the derivative matrix argument, the desired partial derivatives are computed and placed exactly where they are supposed to be without any additional memory or reordering.

Again, this is a generic wrapper that can be used for other problems provided the generated code has the same interface, the number of states is computed properly (i.e., number of equations divided by the actual number of parameters plus one, and the DAEPACK workspace dimensions are dimensioned to the necessary sizes.

This is all of the information needed to be generated so that DSL48S may be used. The following section describes a simulation executive used to setup the problem and call DSL48S to perform the numerical integration.

3 Simulation Executive

The section above describes how to generate automatically the code required by DSL48S for performing a parametric sensitivity calculation. This section describes how to put this all together to actually perform the calculation. The simulation executive code is shown below. This code is kept as generic as possible to that it may be used as a starting point for other problems.

```
      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     YDOT(1)=-(P1+P3)Y(1)^2
C     YDOT(2)=P1*Y(1)^2 - P2*Y(2)
C     Y(1)(0)=1
C     Y(2)(0)=0
C
C     where the Y's are state variables, YDOT's are the corresponding
C     time derivatives, 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.
      IMPLICIT NONE
      DOUBLE PRECISION T, Y(8), YDOT(8), TOUT, RTOL(1),
     1   ATOL(1), RWORK(1000), BOUND(4),RPAR(3),TSTOP
      INTEGER NEQ, NY, NP, NJAC, IDID, LRW, LIW, IWORK(1000),
     1        INFO(25),IRLIST(5), JCLIST(5), JYDOT(5), JDTYPE(5),
     2        NJYDOT, IPAR(100), I, IRES, ICHVAR
      EXTERNAL RES,JAC,SENRHS
C
C
      DOUBLE PRECISION DDPWRK(2)
      INTEGER IDPWRK(4)
      COMMON /DAEPACK/ DDPWRK, IDPWRK
C
C     Set dimensions, NY = number of states, NP = number of parameters,
C     NEQ = number of state and sensitivity equations.
      NY=2
      NP=3
      NEQ=NY*(NP+1)

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
      INFO(24)=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     Initial Conditions
C
C     *** It is extremely important to provide consistent initial
C     *** conditions for the states and sensitivities here.  The
C     *** consistent values must be supplied by the user, or computed
C     *** using other DAEPACK components
C
```

```
C.....states
      Y(1)=1.0D0
      Y(2)=0.0D0

C.....sensitivities (can be obtained by partial differentiation of
C     the initial conditions for the states alone).
      Y(3)=0.0D0
      Y(4)=0.0D0
      Y(5)=0.0D0
      Y(6)=0.0D0
      Y(7)=0.0D0
      Y(8)=0.0D0

C     Consistent values for the time derivatives can be obtained
C     by a function evaluation of the residual evaluator for the
C     states and sensitivities.
C
C.....time derivatives of states
      YDOT(1)=-(RPAR(1)+RPAR(3))*Y(1)**2
      YDOT(2)=RPAR(1)*Y(1)**2

C.....time derivatives of sensitivities
      YDOT(3)=-Y(1)**2
      YDOT(4)=Y(1)**2
      YDOT(5)=0.0D0
      YDOT(6)=0.0D0
      YDOT(7)=-Y(1)**2
      YDOT(8)=0.0D0

C     Set up Jacobian sparsity pattern and related information.
      CALL RESAD(NY,T,Y,YDOT,RWORK,IRES,ICHVAR,RPAR,IPAR,
     1   RWORK(NY+1),NJAC,IRLIST,JCLIST,IDPWRK)
C
C     ADJUST INDICES OF JCLIST CORRESPONDING TO YDOT (I.E., MAKE THEM
C     THE SAME AS Y) AND FILL JYDOT AND JDTYPE. ASSUME ALL DERIVATIVES
C     ARE ANALYTICAL (I.E., NO CONSTANT DERIVATIVE ENTRIES).
C
      NJYDOT=0
      DO I=1,NJAC
         IF(JCLIST(I).GT.NY) THEN
C..........THIS IS A DERIVATIVE W.R.T. A TIME DERIVATIVE
            JDTYPE(I)=-2
            JCLIST(I)=JCLIST(I)-NY
            NJYDOT=NJYDOT+1
            JYDOT(NJYDOT)=I
```

```
         ELSE
C...........THIS IS A DERIVATIVE W.R.T. A STATE VARIABLE.
            JDTYPE(I)=1
         END IF
       END DO

       OPEN(UNIT=3,FILE='test.out')
       WRITE(3,1001)'TIME','Y(1)','Y(2)','S(1,1)','S(2,1)','S(1,2)',
      1             'S(2,2)','S(1,3)','S(2,3)'
       WRITE(3,1000)T,(Y(I),I=1,NEQ)
10     CONTINUE
       CALL DSL48S(RES, SENRHS, NEQ, T, Y, YDOT, TOUT, INFO,
      1   RTOL, ATOL, IDID, RWORK, LRW, IWORK,
      2   LIW,RPAR,IPAR,JAC,NJAC,IRLIST,JCLIST,JYDOT,NJYDOT,JDTYPE,
      3   BOUND)
       WRITE(3,1000)T,(Y(I),I=1,NEQ)
       TOUT=TSTOP
       IF (T.LE.(TSTOP-ATOL(1))) GOTO 10
       CLOSE(3)

       WRITE(6,*)'DSL48S integration complete'
1000   FORMAT(9E16.8)
1001   FORMAT(9A15)
       END
```

Aside from declaring all of the necessary variables, the first step performed is the declaration of a common block holding the generated code workspace. Two arrays are dimensioned, `IDWRK` and `DDWRK`. `IDWRK` is dimensioned to at least max(`NP`,2*`NY`) and `DDWRK` is dimensioned to at least `NY`. (See description of generated code and wrappers for more details concerning these arrays). A common block is used to avoid allocating arrays within the `JAC` and `SENRHS` subroutines. Next, the problem dimensions (number of states and parameters), the initial and final integration time, the DSL48S `INFO` array (see [2] for more details), the relative and absolute error tolerances, and workspace dimensions are specified. Next, the parameter values and indices are specified in `RPAR` and `IPAR`. This is followed by a specification of the initial conditions for the states and sensitivities and their time derivatives. DSL48S must also be passed the sparsity pattern of the Jacobian matrix (so that the entries can be validated and the necessary workspace partitioned from `RWORK` and `IWORK`). This can be readily obtained by calling `RESAD`, the generated Jacobian code.[3] After the sparsity pattern is returned, it is modified to account for the fact that $\dot{y}$ and y have the same indices and `NJYDOT`, `JYDOT`, and `JDTYPE` are determined (See the DSL48S manual [2] for a description of these arguments). Finally, DSL48S is called repeatedly until the integration is completed. Figures 1 through 4 contain the state and sensitivity trajectories for this problem. Figure 1 contains the state trajectories for a portion of the total integration time. The remaining figures contain the sensitivity trajectories with respect to each of the three parameters. As shown in these figures, the states are initially quite sensitive to the values of the parameters but as the states approach steady-state,

[3] As described earlier, additional code can be generated that computes only the sparsity pattern. However, since `RESAD` has already been generated and returns the desired information, there is no reason to generate additional code. A single Jacobian evaluation can be saved by using the code that only computes the sparsity pattern.

the sensitivity diminishes. Figure 3 shows that the value of y_1 is completely insensitive to the value of p_2. This is expected since the initial conditions are independent of the parameters and the first equation fully determines $y_1(t)$ and this equation is independent of p_2.

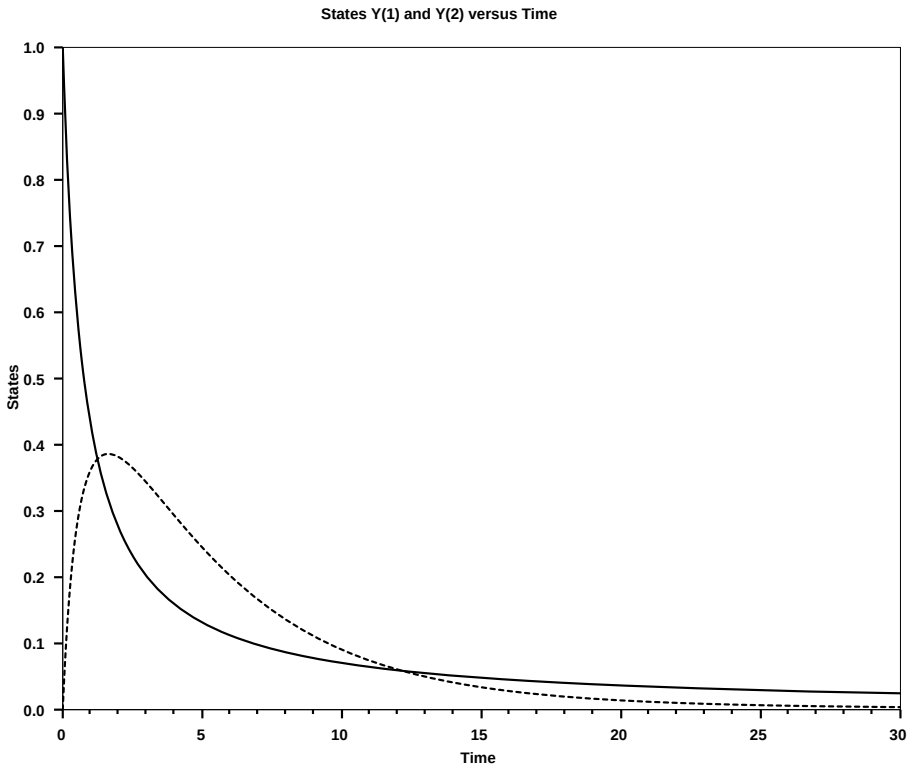

Figure 1: State trajectories for example problem. Solid line is $y_1(t)$ and dashed line is $y_2(t)$.

4 Conclusion

This document describes how to generate the necessary code required by DSL48S automatically with other DAEPACK components. Additional information about how to use DSL48S can be found in [2] and a description of how to use the DAEPACK code generation components can be found in [5]. The code (simulation executive and wrappers) accompanying this paper can be used as a starting point for solving other, more complex, problems.

References

[1] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University

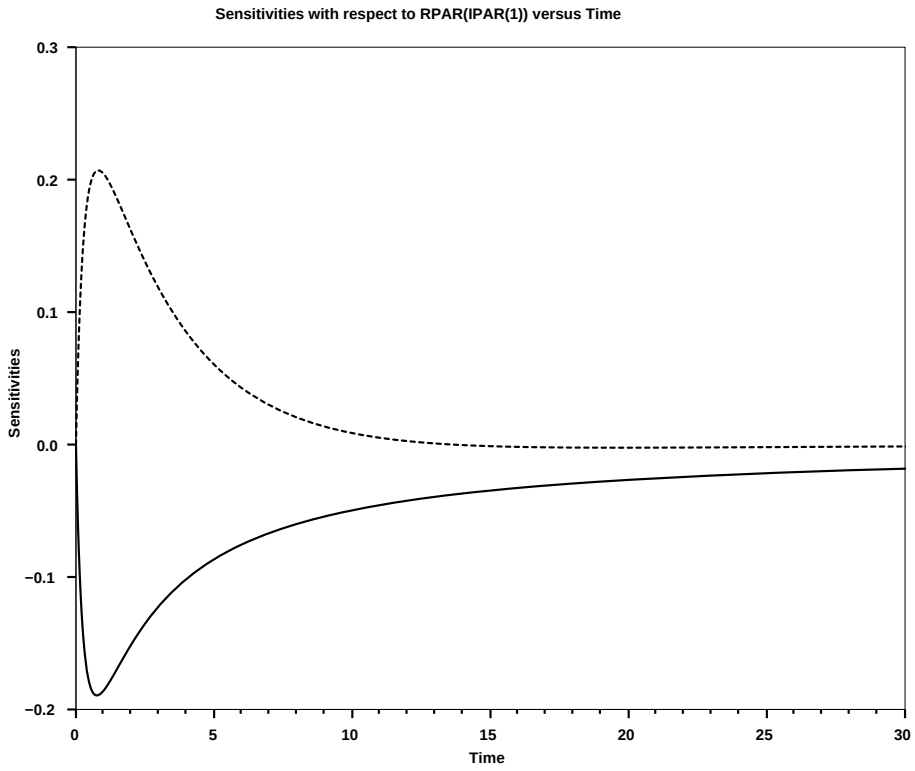

Figure 2: Sensitivities with respect to p_1 versus time for example problem. Solid line is $\partial y_1/\partial p_1$ and dashed line is $\partial y_2/\partial p_1$.

Press, Oxford, UK, 1986.

[2] W. F. Feehery and J. E. Tolsma, *DSL48S user's manual*, Technical Report DSL48S, Massachusetts Institute of Technology, Cambridge, MA, 2000.

[3] W. F. Feehery, J. E. Tolsma, and P. I. Barton, *Efficient sensitivity analysis of large–scale differential–algebraic systems*, Applied Numerical Mathematics, 25 (1997), pp. 41–54.

[4] T. Maly and L. R. Petzold, *Numerical methods and software for sensitivity analysis of differential–algebraic systems*, Applied Numerical Mathematics, 20 (1996), pp. 57–79.

[5] J. E. Tolsma, *DAEPACK code generation user's manual*, Technical Report DAEPACK Code Generation, Massachusetts Institute of Technology, Cambridge, MA, 2000.

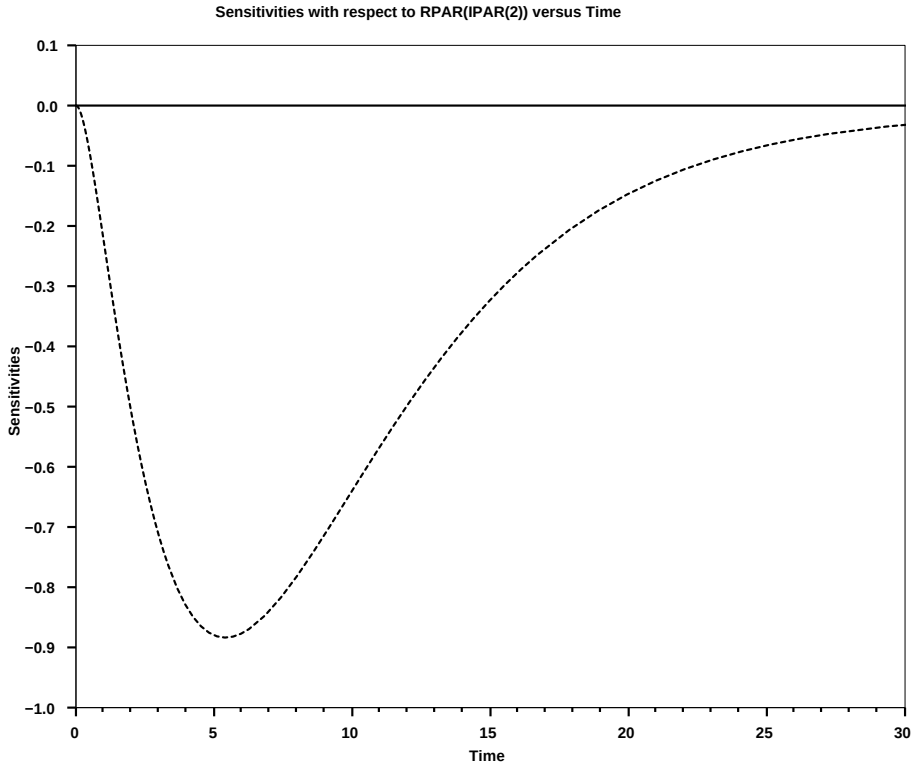

Figure 3: Sensitivities with respect to p_2 versus time for example problem. Solid line is $\partial y_1/\partial p_2$ and dashed line is $\partial y_2/\partial p_2$.

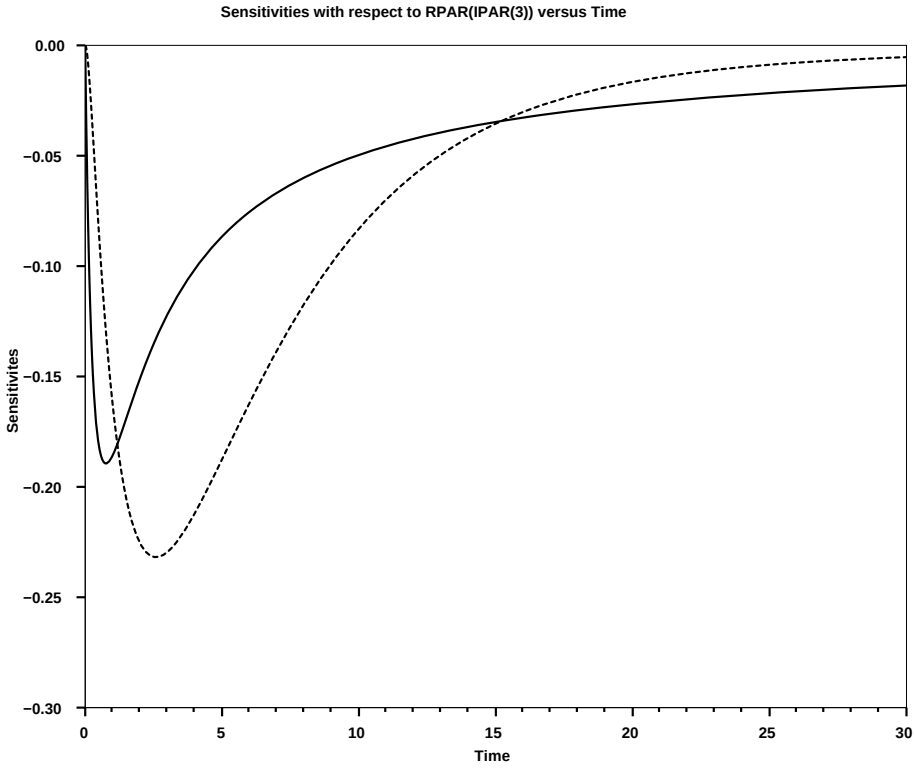

Figure 4: Sensitivities with respect to p_3 versus time for example problem. Solid line is $\partial y_1/\partial p_3$ and dashed line is $\partial y_2/\partial p_3$.