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 DSI48S 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

\[ \dot{y}_1 = -(p_1 + p_3) y_1^2 \]  
\[ \dot{y}_2 = p_1 y_1^2 - p_2 y_2 \]

with initial conditions

\[ y_1(0) = 1 \]  
\[ y_2(0) = 0. \]

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,

\[ f_1 = \dot{y}_1 + (p_1 + p_3) y_1^2 \]  
\[ f_2 = \dot{y}_2 - p_1 y_1^2 + p_2 y_2. \]

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

\texttt{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, \( \text{DELTA} \) is a double precision array of length \( NY \) returning the values of the residuals at \( (T, Y, YDOT) \), \( \text{ires} \) is an error return code set to a nonzero value if an error has occurred during residual evaluation (not modified otherwise), \( \text{ichvar} \) is equal to 1 if the \( T, Y, \) or \( YDOT \) have changed values since the last call to \( \text{RES} \), and \( \text{rpar} \) and \( \text{ipar} \) 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:

---

4
SUBROUTINE RES(NY,T,Y,YD,T,DELTA,RES,ICHVAR,RPAR,IPAR)
C
IMPLICIT NONE
INTEGER RES,JRES,IPAR(*)
DOUBLE PRECISION T,Y(NY),YDOT(NY),DELTA(NY),RPAR(100)
C
DELTA(1)=YD(T)+RPAR(IPAR(1))+RPAR(IPAR(3))*Y(1)**2
DELTA(2)=YD(T)-RPAR(IPAR(1)+RPAR(IPAR(2))*Y(1)**2+RPAR(IPAR(2))*Y(2)
C
RETURN
END

where \( Y(1)=y_i, \ YD(T)=y_i, \ i=1,2, \) and \( RPAR(IPAR(J))=\mu_j, \ j=1,2,3. \) 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 \( y \) 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,zzzirm,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 zzzjcnn.
C### zzzme - number of entries in zzzderiv.
C### zzzirn - row numbers for entries in zzzderiv.
C### zzzjcnn - column numbers for entries in zzzderiv.
C###
C### double precision zzzderiv(*)
C### integer zzzme, zzzirn(zzzme), zzzjcnn(zzzme)
C###
C### integer zzziv(*)
C###
C### zzzmodel - unique key for this model.
C###
C### integer zzzmodel
C###
C### Define elementary variables and adjoints
C###
double precision zzzv1, zzzvbar1
C###
double precision zzzv2, zzzvbar2
C###
double precision zzzv3, zzzvbar3
C###
double precision zzzv4, zzzvbar4
C###
double precision zzzv5, zzzvbar5
C###
double precision zzzv6, zzzvbar6
C###
double precision zzzv7, zzzvbar7
C###
double precision zzzv8, zzzvbar8
C###
double precision zzzv9, zzzvbar9
C###
double precision zzzv10, zzzvbar10
C###
double precision zzzv11, zzzvbar11

C###
C### Active variable offsets and loop control variables
C###
integer deltaoft
integer yoft
integer ydotoft
integer zzzn
integer zzzm
integer zzi
integer zziindex

C### Variable offsets for independent, dependent, and
C### non-common block local active variables.
deltaoft=0
yoft=deltaoft+ny
dottoft=yoft+ny

C### Number of independent and dependent variables.
zzzn=ny+ny
zzzs=ny

C### Create unique identifier for this model.
call GETMD.ID(zzzismodel,'rseed')

C### Create SVM for functions/subroutines.
call CREATESV(zzzismodel,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)=dottoft+zzzi
end do

C### Initialize independent variable gradients to Cartesian basis vectors.
call DSETIVCV(1,zzzn,zzziw)

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 DSM2(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,ydof(2)+1)

C###
goto 1111

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
    zzziv(zzzindx)=deltaoft+zzzi
end do
call DPFRVS(1,zzzm,zzziv,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, zzzrn and zzzjn 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, zzzrn, and zzzjn must be dimensioned to at least the maximum number of nonzero entries expected in the Jacobian. Let \( \text{DELT}\equiv f \) and \( Y\cdot\text{D}\equiv y, \dot{y} \). Calling the subroutine above computes the following information:

\[
J(\dot{y}, y; t; p) = \begin{pmatrix}
\frac{\partial f}{\partial \dot{y}} & \frac{\partial f}{\partial y}
\end{pmatrix}
\]

(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:

\[
\alpha \frac{\partial f}{\partial \dot{y}} + \frac{\partial f}{\partial y}
\]

(8)

Consequently, partial derivatives with respect to \( \dot{y} \) and \( y \) must have the same column indices (\( \dot{y} \) is represented as a function of \( y \) and \( t \)). Furthermore, DSL48S requires three additional pieces of information: NJYD\( \text{DT} \), the number of elements in the sparse Jacobian matrix with respect to time derivatives, JYD\( \text{DT} \)(1:NJYD\( \text{DT} \)), the indices in the sparse Jacobian that are partial derivatives with respect to time derivatives, and JD\( \text{TYPE} \)(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:

```c
SUBROUTINE JAC(NY,T,Y,YD\( \text{DT} \),AJ\( \text{AC} \),NJ\( \text{AC} \),I\( \text{ROW} \),JC\( \text{OL} \), 1  IYD\( \text{DT} \),NJYD\( \text{DT} \),ICH\( \text{VAR} \),RP\( \text{AR} \),IP\( \text{AR} \),IRES,J\( \text{DEV} \)"
C
IMPLICIT NONE
INTEGER IP\( \text{AR} \)(*), IRES, JC\( \text{OL} \)(*), I\( \text{ROW} \)(*), IYD\( \text{DT} \)(*), 1  NY, NJ\( \text{AC} \), NJYD\( \text{DT} \),ICH\( \text{VAR} \),J\( \text{DEV} \)(*), D\( \text{OBLE} \) PRECISION AJ\( \text{AC} \)(*), RP\( \text{AR} \)(*), T, Y(*), YD\( \text{DT} \)(*)
C
DOUBLE PRECISION DRA\( \text{F} \)WRK
INTEGER ID\( \text{WRK} 
COMMON /DA\( \text{EPA} \CK/ DDP\( \text{R} \)WK,ID\( \text{WRK} 
DIMENSION DDP\( \text{R} \)(2),ID\( \text{WRK} 
C
INTEGER I
EXTERNAL RESAD
C
\footnote{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.}
```

9
CALL RESAD(NY,T,Y,YDUT,DDPW RK, IRES, ICHVAR, RP AR, IPAR,
1 A JAC, NJAC, IRow, JCOL, IDPW RK)
C......SET JACOBIAN INFORMATION N JYDUT AND JYDUT
NJYDUT=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 JYDUT.
1 JCOL(I)=JCOL(I)-NY
NJYDUT=NJYDUT+1
JYDUT(NJYDUT)=I
JTYPE(I)=1
ELSE
JTYPE(I)=-2
END IF
END DO
C
RETURN
END

where arguments NY, T, Y, YDUT, ICHVAR, RP AR, 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, NJYDUT and JYDUT are described above, and JTYPE 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 IDPW RK 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 NJYDUT, JYDUT, and JTYPE. 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.

\[
\frac{\partial f}{\partial y} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial p_i} = \frac{\partial f}{\partial p_i} \quad i = 1, \ldots, n_p
\]  

where \( n_y \) is the number of states and \( n_p \) is the number of parameters. The partial derivatives \( \frac{\partial y}{\partial p_i} \) and \( \frac{\partial y}{\partial p_i} \) are the sensitivities of interest. DSL48S requires a subroutine to return the values of \( \frac{\partial f}{\partial p_i} \),
where $I = 1, \ldots, n_p$. Like the residual and Jacobian routines, the user provides these partial derivatives through a subroutine passed to DLS45. The interface of this sensitivity equation right-hand-sides subroutine is

\begin{verbatim}
SUBROUTINE SENRHS(NEQ,T,Y,YDOT,DELTA,IVES,RPAR,IPAR)
\end{verbatim}

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 $\frac{\partial f}{\partial p_1}$, entries $DELTA(2*NY+1:3*NY)$ are filled with $\frac{\partial f}{\partial p_2}$, and so on. In other words, after calling RES and SENRHS, $DELTA$ contains

\[
\begin{pmatrix}
\frac{\partial f}{\partial p_1} \\
\vdots \\
\frac{\partial f}{\partial p_{NP}}
\end{pmatrix}
\]

(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

\begin{verbatim}
GENERATE DERIVS
ROOT : res
OUTFILE : resadp
SUFFIX : adp
SPARSESTORAGE : false
INDEPENDENT : rpar(ipar(1:3))
DEPENDENT : delta
END
\end{verbatim}

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

```csharp
C### - DAEPACK Version 1.0 - Copyright (C) MIT
C### - DERIVATIVE COMPUTATION -
C### - GENERATED: Fri May 19 17:59:57 2000
$ zzzlmd,zzzderiv,zzziv
имв independent { rpar(ipar(1:3)) }
имв 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
$\text{zzz index}=\text{zzz index}+1$
$\text{zzz iv}(\text{zzz index})=\text{rparoft}+\text{ipar}(\text{zzzi})$
end do

C### Initialize independent variable gradients to Cartesian basis vectors.
call DSETIVCV(1, zzzn, zzziv)

C### Compute elementary variables
zzzv1=\text{ydot}(1)
zzzv2=\text{rpar}(\text{ipar}(1))
zzzv3=\text{rpar}(\text{ipar}(3))
zzzv4=\text{zzz v}^2+\text{zzz v}^3
zzzv5=y(1)
zzzv6=2
zzzv7=\text{zzz v}^5+\text{zzz v}^6
zzzv8=\text{zzz v}^4+\text{zzz v}^7
zzzv9=\text{zzz v}^1+\text{zzz v}^8

C### Compute elementary partial derivatives
zzzvbar9=1.0d0
zzzvbar8=zzzvbar9
zzzvbar4=zzzvbar8*zzzv7
zzzvbar3=zzzvbar4
zzzvbar2=zzzvbar4

C###
delta(1)=zzzv9

C###
call DSV2(deltaoft+1,1,  
$\text{zzzvbar3, rparoft}+\text{ipar}(3), 1, \text{zzzvbar2, rparoft}+\text{ipar}(1), 1)$

C### Compute elementary variables
zzzv1=\text{ydot}(2)
zzzv2=\text{rpar}(\text{ipar}(1))
zzzv3=y(1)
zzzv4=2
zzzv5=zzz v^3+zzz v^4
zzzv6=zzz v^2+zzz v^5
zzzv7=zzz v^1-zzz v^6
zzzv8=\text{rpar}(\text{ipar}(2))
zzzv9=y(2)
zzzv10=zzz v^8+zzz v^9
zzzv11=zzz v^7+zzz v^10

C### Compute elementary partial derivatives
zzzvbar11=1.0d0
zzzvbar10=zzzvbar11
zzzvbar8=zzzvbar10*zzz v^9
zzzvbar7=zzzvbar11
zzzvbar6=-zzzvbar7  
zzzvbar2=zzzvbar6*zzzv5

C###
  delta(2)=zzzv11
C###
  call DSV2(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###
  zzzind=0
  do zzzi=1,ny
     zzzind=zzzind+1
     zzziv(zzzind)=deltaoft+zzzi
  end do
  call DCPVD(1,zzzm,zzziv,zzzldm,zzzderiv)
C### Delete SVM for functions/subroutines.
  call DELESV(zzzmodel,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 zzziv 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

$$\frac{\partial f}{\partial p} = \left( \begin{array}{ccc} \frac{\partial f}{\partial p_1} & \frac{\partial f}{\partial p_2} & \frac{\partial f}{\partial p_3} \end{array} \right)$$ \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 Dl48s interface requirements:

```
SUBROUTINE SENRHS(NEQ,T,Y,YDPT,DELTA,IRES,RPAR,IPAR)
C
IMPLICIT NONE
INTEGER IRES,IPAR(*),NEQ
DOUBLE PRECISION T,Y(*),YDPT(*),DELTA(*),RPAR(*)
C
DOUBLE PRECISION DDPWRK
```
EXTERNAL RESADP
EXTERNAL NY,ICHVAR

C......NUMBER OF STATES IS EQUAL TO THE TOTAL NUMBER OF EQUATIONS
C PLUS ONE.
  NY=NEQ/4
C
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
T. Maly and L.R. Petzold, "Numerical methods and software for
sensitivity analysis of differential-algebraic equations",

This is a two dimensional ODE with three parameters. The problem has the form:

\[ \begin{align*}
\dot{y}(1) &= -(\alpha_1 + \alpha_3) {y}(1)^2 \\
\dot{y}(2) &= \alpha_1 {y}(1)^2 - \alpha_2 {y}(2) \\
y(1)(0) &= 1 \\
y(2)(0) &= 0
\end{align*} \]

where the \( y \)'s are state variables, \( \dot{y} \)'s are the corresponding time derivatives, and the parameters have the values:

\[ \begin{align*}
\alpha_1 &= 0.9875 \\
\alpha_2 &= 0.2566 \\
\alpha_3 &= 0.3323
\end{align*} \]

The resulting ODE and sensitivity system defines an 8x8 system of equations.

IMPLICIT NONE
DOUBLE PRECISION T, Y(8), YDOT(8), TOUT, RTOL(1),
REAL TOL(1), RWORK(1000), BOUND(4), RPAR(3), TSTOP
INTEGER NEQ, NY, NP, NJAC, IDID, LWF, LW, IW(1000),
1 INFO(26), IRLIST(5), JCLIST(5), JYDOT(5), JDTYPE(5),
2 NJYDOT, IPAR(100), I, IRES, ICHVAR
EXTERNAL RES, JAC, SENSRS

DOUBLE PRECISION DDPWRK(2)
INTEGER IDPWRK(4)
COMMON /DAEPACK/ DDPWRK, IDPWRK

Set dimensions, \( NY = \) number of states, \( NP = \) number of parameters,
\( NEQ = \) number of state and sensitivity equations.
\( NY = 2 \)
\( NP = 3 \)
\( NEQ = NY \times (NP+1) \)

Initial time and maximum initial time step
\( T = 0.00 \)
\( TSTOP = 4.00 \)
\( TOUT = 10.00 \)

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 *** 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......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......time derivatives of states
    YDÔT(1)=-(RPAR(1)+RPAR(3))*Y(1)**2
    YDÔT(2)=RPAR(1)*Y(1)**2

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

C    Set up Jacobian sparsity pattern and related information.
C    CALL RESAD(NY,T,Y,YDÔT,RWÔRK,IRES,IChVÅrå,RPAR,IPAR,
1         RWÔRK(NY+1),NJAC,IRLiST,JCLiST,IDPWRK)

C    ADJUST INDICES OF JCLiST CORRESPONDING TO YDÔT (I.E., MAKE THEM
C    THE SAME AS Y) AND FILL JYDÔT AND JDTYPE. ASSUME ALL DERIVATIVES
C    ARE ANALYTICAL (I.E., NO CONSTANT DERIVATIVE ENTRIES).

C    NJYDÔT=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
            NJYDÔT=NJYDÔT+1
            JYDÔT(NJYDÔT)=I
ELSE
C...........THIS IS A DERIVATIVE W.R.T. A STATE VARIABLE.
       JTYPE(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)',
       '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, YD, TOUT, INFO,
                  RTOL, ATOL, IDID, RWK, LW, IWK,
                  LIW, RIPAR, IPAR, JAC, NJAC, IRLIST, JCLST, JYDOT, NJYDOT, JDTYPE,
                  BOUND)
    WRITE(3,1000)T, (Y(I), I=1,NEQ)
    TOUT=TSTOP
    IF (T.LE.(TSTOP-ATOL(I))) 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, IDWK and DDWK. IDWK is dimensioned at least max(NP,2*NY) and DDWK 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 RIPAR 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 RWK and IWK). This can be readily obtained by calling RESAD, the generated Jacobian code. After the sparsity pattern is returned, it is modified to account for the fact that $y$ 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 $p_2$.

![States Y(1) and Y(2) versus Time](image)

Figure 1: State trajectories for example problem. Solid line is $y_1(t)$ and dashed line is $p_2(t)$.

4 Conclusion

This document describes how to generate the necessary code required by DSL4SS automatically with other DAEPACK components. Additional information about how to use DSL4SS 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

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 $\partial y_2/\partial p_1$.


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 $\partial y_2/\partial p_3$. 