8.4 Computing Numerical Derivatives, Gradients and Jacobians

A. Purpose
This subroutine computes approximate gradient vectors and Jacobian matrices of vector functions \( f(y) \) in several variables. This routine is used to compute approximate derivatives for problems where only function information is available. To approximate a scalar derivative function, consider it an equivalent \( 1 \times 1 \) Jacobian matrix. Either one-sided or central differences can be used.

B. Usage

B.1 Program Prototype, Double Precision

```fortran
INTEGER MODE, M, N, LDFJAC, IOPT(k), LWK, IWK(LWK \geq 21), LIWK

DOUBLE PRECISION Y(\geq N), F(\geq M),
FJAC(LDFJAC,\geq N), YSCALE(1 or \geq N),
FAC(\geq N), WK(LWK \geq 3*M+18)

CALL DJACG(MODE, M, N, Y, F,
FJAC, LDFJAC, YSCALE, FAC,
IOPT, WK, LWK, IWK, LIWK)
```

See the description of IOPT for the \( k \) required.

```fortran
Assign values to M, N, LDFJAC and IOPT().
Set MODE = 0.
```

10 continue

C Approximate FJAC(,) as the \( M \times N \) Jacobian matrix of first partial derivatives of \( f(y) \) with respect to \( y \), evaluated at \( Y() \):

```fortran
CALL DJACG(MODE, M, N, Y, F,
FJAC, LDFJAC, YSCALE, FAC,
IOPT, WK, LWK, IWK, LIWK)
```

if (MODE .gt. 0) then
  Compute WK() as an M-vector of function values \( f(y) \) evaluated at perturbed \( y \) in \( Y() \). go to 10
end if

Here the process is normally completed, if there are no error conditions. Approximations to the partials are in FJAC(,), with related scaling and performance data in FAC() and IWK().

B.2 Argument Definitions

MODE [inout] On initial entry set MODE = 0. The routine DJACG will normally return a number of times with MODE = J, the index of the derivative being computed. A return value of MODE=0 means the approximate derivatives are computed. A return value of MODE = K < 0 means that argument number \( -K \) has an error condition.

For J > 0 the calling code should compute WK() as a function of \( Y() \) and call DJACG again, not altering MODE.

M [in] Number of terms in \( F() \) and number of rows of data in FJAC(,). Must have M .gt. 0.

N [in] Number of terms in \( Y() \) and number of columns of data in FJAC(,). Must have N .gt. 0.

Y(,) [inout] Initially must contain the vector \( y \) at which the partial will be computed. Contains perturbed \( y \) on each return with MODE = J > 0. On final return with MODE = 0, contains the original \( y \). Only component J is perturbed with each return, but more than one evaluation for a given J may occur. After DJACG computes FAC() for the problem, one evaluation per variable is typically needed if central differences are not used..

F(,) [in] On entries with MODE \geq 0, the user has function values in \( F() \), i.e., \( F(I) = f_I(y) \). The user should initialize \( F()=f(y) \) before the routine is entered with MODE = 0.

FJAC(,) [inout] , The numerical derivatives, i.e., \( FJAC(i, j) = \partial f_I / \partial y_j \). For default usage, this is designated as [out] except in columns receiving special treatment, as noted in IOPT().

LDFJAC [in] Declared first dimension of the array FJAC(,). Require LDFJAC \geq M or N=1.

YSCALE(,) [in] An array of length 1 or N. The user can provide representative sizes for \( y \) values in the array YSCALE(,). The user can also use YSCALE(,) to give appropriate directions for the differences. If YSCALE(1) =0 then \( Y() \) itself is used for the scaling.

FAC(,) [inout] An array of length N. The value FAC(J) contains the factor for differencing variable J. If FAC(1)=0, the array FAC() is set to the square root of machine precision on the first call to DJACG. Unless the user wishes to initialize FAC(), this array should not be altered between subsequent calls to DJACG.

The user may provide FAC() values if desired. If the user provides these values, FAC(J) should be set to a value between 0 and 1. The routine DJACG ensures that FACMIN \leq FAC(J) \leq FACMAX. The values for FACMIN and FACMAX are saved in the storage locations WK(3M+1), WK(3M+2) when DJACG exits with MODE .gt. 0.

IOPT(,) [in] An integer array defining the methods used to compute the derivatives. The default is to use one-sided differences to compute derivatives. Entries in this array are interpreted as follows.
0 Use the current settings for all remaining variables.
   The starting setting is to use one-sided differences.

k > 0 Use the current settings for all variables from
the last specified up to and including variable k.

−1 Set to use one-sided differences. (Not needed at
the beginning since this is the default state.)

−2 Set to use central differences.
   This will typically yield more accuracy than the
   one-sided differences, but with the expense of
   an additional function evaluation per variable.
   The increment used for central differencing is
   \[ T = \text{macheps}^{-1/6} \] times the increment used
   in one-sided differencing.

To change this factor for succeeding variables,
assign a new value between calls with MODE =
J in the storage location FAC(J).

The default value \[ T = \text{macheps}^{-1/6} \] is based
on the approximate relation \[ T \cdot \text{FAC}(J) = \text{macheps}^{2/3}. \] This value is near an optimal
choice, under certain conditions on higher
derivatives, provided \[ \text{FAC}(J) = \text{macheps}^{1/2}. \]
Sometimes larger or smaller values of FAC(J)
will give more accuracy than the default.

−3 Set to accumulate the result from whatever type
of differences have been specified above into initial
values provided in FJAC(1:M,J). This must be
followed by a number \( \geq 0 \) as any number \( < 0 \)
will turn this off.

This partial consists of the prior computed values
FJAC(1:M,J) plus additional values that are
computed with divided differences. The value F() should equal the part of the function to be
differenced.

−4 Skip variables. This must be followed by a value
\( \geq 0 \) as any number \( < 0 \) will turn this off.

No function evaluation is made when skipping a
variable. It is assumed that this partial is com-
puted in the user’s program in some way other
than with divided differences.

An an illustration, suppose column 3 is not to
be computed and column 4 is to be accumu-
lated. The rest of the columns are computed
with one-sided differences and no additional options.
These requirements are designated by: IOPT(1:7) =
2, −4, 3, −3, 4, −1, 0.

\textbf{IKW()} [inout] An integer array whose dimension is
at least LIWK. The first 10 positions of IWK() con-
tain diagnostic information, which can normally be
ignored. The remaining locations are used for scratch
storage.

IKW(1) gives the number of times the function, i.e.
f(y), was computed.

IKW(2) gives the number of columns in which three
attempts were made to increase a percentage factor
for differencing (i.e. a component in the FAC() array)
but the computed \( \Delta J \) remained too small relative to
Y(J) or YSCALE(J). In such cases the percentage
factor is set to the square root of the unit roundoff
of the machine.

IKW(3) gives the number of columns in which the
computed \( \Delta J \) was zero to machine precision because
Y(J) or YSCALE(J) was zero. In such cases \( \Delta J \)
is set to the square root of the unit roundoff.

IKW(4) gives the number of Jacobian columns which
had to be recomputed because the largest difference
formed in the column was close to zero relative to
\textit{scale}, where

\[
\text{scale} = \max(|f_i(y)|, |f_i(y + \Delta)|)
\]

and \( i \) denotes the row index of the largest difference
in the column currently being processed. IKW(10)
gives the last column where this occurred.

IKW(5) gives the number of columns whose largest
difference is close to zero relative to \textit{scale} after the
column has been recomputed.

IKW(6) gives the number of times \textit{scale} information
was not available for use in the roundoff and trunca-
tion error tests. this occurs when

\[
\min(|f_i(y)|, |f_i(y + \Delta)|) = 0.
\]

where \( i \) is the index of the largest difference for the
column currently being processed.

IKW(7) gives the number of times the increment for
differencing \( \Delta J \) was computed and had to be in-
creased because \( Y(J) + \Delta J - Y(J) \) was too small
relative to \( Y(J) \) or YSCALE(J).

IKW(8) gives the number of times a component of the
FAC() array was reduced because changes in function
values were large and excess truncation error was sus-
ppected. IKW(9) gives the last column in which this
occurred.

IKW(9) gives the index of the last column where the
corresponding component of the FAC() array had to be
reduced because excessive truncation error was sus-
ppected.
IWK(10) gives the index of the last column where the difference was small and the column had to be recomputed with an adjusted increment, as in IWK(4). The largest derivative in this column may be inaccurate due to excessive roundoff error.

LIWK [in] The required length of the array IWK(), LIWK ≥ 21.

B.3 Modifications for Single Precision

For single precision usage change the DOUBLE PRECISION statements to REAL and change the name DJACG to SJACG. It is recommended that one use the CCISION statements to REAL and change the name of this package for better reliability and accuracy.

C. Example

C.1 A Gradient Computation

The function is \( f(y_1, y_2) = a \exp(by_1) + cy_1y_2^2 \). Its gradient vector, or \( 1 \times 2 \) Jacobian matrix, is

\[
\begin{bmatrix}
\frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial y_2}
\end{bmatrix} = \begin{bmatrix}
ab \exp(by_1) + cy_2^2, 2cy_1y_2
\end{bmatrix}
\]

This formula is used for comparison to the computed results. Values of the parameters and variables in the program are:

\[ a = 2.5, \quad b = 3.4, \quad c = 4.5 \]
\[ y_1 = 2.1, \quad y_2 = 3.2 \]

This driver illustrates how to:

1. Compute approximate derivatives using one-sided divided differences (default)
2. Use one-sided differences on the first component and analytically compute the second component
3. Accumulate a known term of the first component with a differenced term that is not known a priori
4. Use central differences for both partials, getting more accuracy with additional function evaluations

[The code and output listing are shown below.]

D. Functional Description

Let \( y \) denote the vector given initially in \( Y() \). Let \( \epsilon \) denote the machine precision, \( macheps \). This is provided by the functions D1MACH(4) and R1MACH(4) in single precision. (See Chapter 19.1).

Let \( e_j \) denote the \( n \)-vector that is all zeros except for the \( j^{th} \) component which is one.

For each \( i \) and \( j \) compute

\[
\text{FJAC}(i,j) = \frac{f_i(y + h_j e_j) - f_i(y)}{h_j}
\]

The error in this difference approximation to the derivative \( \partial f_i/\partial y_j \) is bounded by the magnitude of

\[ h_j M_2/2 + \delta/h_j \]

where \( M_2 \) denotes the magnitude of \( ||\partial^2 f/\partial^2 y_j|| \) evaluated at some point on the line segment from \( y \) to \( y + h_j e_j \), and \( \delta \) is a bound on the error in computing \( ||f(y)|| \).

A key feature of DJACG is its heuristic for efficiently estimating values of \( \Delta_j = h_j = fac_j \times \text{scale}_j \) to maintain precision. The algorithm for choosing \( fac_j \) is found in Salane, [1]. August, 1986. A user should save values of \( fac_j \) between computations of partials, particularly when values of \( y \) do not change much and several evaluations of the partials are anticipated.

The routine DJACG is re-entrant and thread-safe. This is done by avoiding the use of elementary functions, except SQRT(). All required scalars are saved in the working arrays WK() and IWK() during reverse communication.

An evaluation of the function \( f(y) \) can use DJACG as part of that computation. This feature will be useful when approximating higher order derivatives of a smooth function, or using threaded algorithms to concurrently compute the Jacobian matrix columns. To use the re-entrant or threaded functionality, separate storage copies of the arguments are usually required.

References


E. Error Procedures

Require MODE = 0 or > 0 on any entry to DJACG. A returned value of MODE = 0 is normal, meaning the derivatives are approximated. Values of MODE = K < 0 mean that argument number \(-K\) has an error condition. Additional diagnostic information about numerical cancellation or excessive truncation error is found in IWK(2:10).

F. Supporting Information

The source language is ANSI Fortran 77.

Entry Required Files

DJACG AMACH
SJACG AMACH

Designed and programmed by D. A. Salane, Sandia Labs. (1986). Modified by R. J. Hanson, Rice University, (June, 2002) with advice from F. T. Krogh.
DRDJACG1

program DRDJACG1
   c>> 2006–04–12 DRDJACG1 Hanson — Reduced lengths of djacg work arrays.
   c>> 2003–07–08 DRDJACG1 Hanson — Check for MODE < 0.
   c>> 2003–07–07 DRDJACG1 Krogh — Changed 3 arg max to 2 arg for C conv.
   c>> 2002–06–21 DRDJACG1 R. J. Hanson Example 1 Code, with Download
   c Demo driver for DJACG, using numerical derivatives for a gradient.
   c
   c—— D replaces "?": DR?JACG1, ?JACG
   c
   c The function used for this demo is \( f(y_1, y_2) = a \exp (b y_1) + c y_1^2 \).
   c Its gradient vector is \((df/\partial y_1, df/\partial y_2) = (a b \exp (b y_1) + c y_1, 2 c y_1 + y_2)\).
   c This is used for comparison of the computed results with actual
   c results.
   c
   C This driver shows how to:
   C
   C A. Compute approximate derivatives using one-sided divided
   C differences
   C B. Use one-sided differences on the first component and
   C analytically compute the second component
   C C. Accumulate a known term of the first component with a
   C differenced term that is not known a priori
   C D. Use central differences on both components
   C (No checking for error conditions, i.e. MODE < 0.)
   C
   C Define sizes and parameters.

   INTEGER I, MODE, M, N, LDFJAC, LWK, LIWK
   PARAMETER(M=1, N=2, LDFJAC=M, LWK=3*M+18, LIWK=21)
   INTEGER IOPT(04), IWK(LIWK)
   DOUBLE PRECISION Y(N), F(M), FJAC(LDFJAC,N), XSCALE(N),
   . FAC(N), WK(LWK), ACTUAL(LDFJAC,N), RE(4,2)
   DOUBLE PRECISION A, B, C, F2, U, MAXERR, DIMACH
   CHARACTER*55 WHAT(4)
   CHARACTER*1 uv
   DATA WHAT /
   . 'One sided partials, default settings',
   . 'One sided partial, second partial known and skipped',
   . 'One sided partials, first partial accumulated',
   . 'Central difference partials' /

   C Define data and point of evaluation:
   A=2.5D0
   B=3.4D0
   C=4.5D0

   Y(1)=2.1D0
   Y(2)=3.2D0

   C Machine precision, for measuring errors
   U=DIMACH(4)
   C Set defaults for increments and scaling:
   FAC(1)=0.D0
   XSCALE(1)=0.D0

8.4–4 Computing Numerical Derivatives, Gradients and Jacobians July 11, 2015
**C**  
Compute true values of partials.

\[
\text{ACTUAL}(1,1) = A \times \exp(B \times Y(1)) + C \times Y(2)^2 \]
\[
\text{ACTUAL}(1,2) = 2 \times C \times Y(1) \times Y(2) \]

**C**  
A. No variable gets special treatment

\[
\text{IOPT}(1) = 0 
\]

**C**  
Start each problem with MODE=0. Other starting values are errors. Values < 0 or > N are caught.

\[
\text{MODE} = 0 
\]

10 **CONTINUE**

\[
\text{WK}(1) = A \times \exp(B \times Y(1)) + C \times Y(1) \times Y(2)^2 
\]

**C**  
This sets the function value used in forming one-sided differences.

\[
\text{IF} (\text{MODE} \ . \text{eq} . \ 0) \ \text{THEN} 
\]

\[
F(1) = \text{WK}(1) 
\]

**END IF**

CALL DJACG(MODE,M,N,Y,F,
          . FJAC,LDFJAC,XSCALE,FAC,IOPT,
          . WK,LWK,IKW,LIWK)

\[
\text{IF} (\text{MODE} \ . \text{gt} . \ 0) \ \text{GO TO} \ 10 
\]

**C**  
Check for an error condition.

\[
\text{IF} (\text{MODE} \ . \text{lt} . \ 0) \ \text{THEN} 
\]

\[
\text{PRINT} \ '( ' ' \text{Initial error in argument number } ' ' , I2 ) ' , -\text{MODE} 
\]

**GO TO** 60

**END IF**

**C**  
Check the relative accuracy of one-sided differences.

**C**  
They should be good to about half-precision.

\[
\text{FJAC}(1,1) = (\text{FJAC}(1,1) - \text{ACTUAL}(1,1)) / \text{ACTUAL}(1,1) 
\]
\[
\text{FJAC}(1,2) = (\text{FJAC}(1,2) - \text{ACTUAL}(1,2)) / \text{ACTUAL}(1,2) 
\]
\[
\text{RE}(1,1) = \text{FJAC}(1,1) / \sqrt{U} 
\]
\[
\text{RE}(1,2) = \text{FJAC}(1,2) / \sqrt{U} 
\]

**C**  
B. Skip variable number 2.

\[
\text{IOPT}(1) = 1 
\]
\[
\text{IOPT}(2) = -4 
\]
\[
\text{IOPT}(3) = 2 
\]

\[
\text{MODE} = 0 
\]

20 **CONTINUE**

\[
\text{WK}(1) = A \times \exp(B \times Y(1)) + C \times Y(1) \times Y(2)^2 
\]

\[
\text{IF} (\text{MODE} \ . \text{eq} . \ 0) \ \text{THEN} 
\]

\[
F(1) = \text{WK}(1) 
\]

**END IF**

CALL DJACG(MODE,M,N,Y,F,
          . FJAC,LDFJAC,XSCALE,FAC,IOPT,
          . WK,LWK,IKW,LIWK)

\[
\text{IF} (\text{MODE} \ . \text{gt} . \ 0) \ \text{GO TO} \ 20 
\]

**C**  
Check for an error condition.

\[
\text{IF} (\text{MODE} \ . \text{lt} . \ 0) \ \text{THEN} 
\]

\[
\text{PRINT} \ '( ' ' \text{Initial error in argument number } ' ' , I2 ) ' , -\text{MODE} 
\]

**GO TO** 60

**END IF**

\[
\text{FJAC}(1,1) = (\text{FJAC}(1,1) - \text{ACTUAL}(1,1)) / \text{ACTUAL}(1,1) 
\]
\[
\text{FJAC}(1,2) = (\text{FJAC}(1,2) - \text{ACTUAL}(1,2)) / \text{ACTUAL}(1,2) 
\]
RE(2,1)=FJAC(1,1)/sqrt(U)
RE(2,2)=FJAC(1,2)/sqrt(U)

C  C. Accumulate a part of the first partial.
IOPT(1)=-3
IOPT(2)= 1
C  Shift to using one-sided differences for the
C  rest of the variables.
IOPT(3)=-1
IOPT(4)=0

MODE=0

30  CONTINUE
C  Since part of the partial is known, evaluate what is
C to be differentiated.
IF (MODE .ne. 2) WK(1)=A*EXP(B*Y(1))
IF (MODE .eq. 0) THEN
C  Start with part of the derivative that is known.
F(1)=WK(1)
FJAC(1,1)=C*Y(2)**2
C  This is the function value for the partial wrt y_2.
F2=C*Y(1)*Y(2)**2
END IF
IF (MODE .eq. 2) THEN
C  The function value for the second partial has the part removed
C  that depends on the first variable only.
F(1)=F2
WK(1)=C*Y(1)+Y(2)**2
END IF
CALL DJACG(MODE,M,N,Y,F,
. FJAC,LDFJAC,XSCALE,FAC,IOPT,
. WK,LWK,IWK,LIWK)
IF (MODE .gt. 0) GO TO 30
C  Check for an error condition.
IF (MODE .lt. 0) THEN
  PRINT '( 'Initial error in argument number ',I2)',-MODE
  GO TO 60
END IF

FJAC(1,1)=(FJAC(1,1)-ACTUAL(1,1))/ACTUAL(1,1)
FJAC(1,2)=(FJAC(1,2)-ACTUAL(1,2))/ACTUAL(1,2)
RE(3,1)=FJAC(1,1)/sqrt(U)
RE(3,2)=FJAC(1,2)/sqrt(U)

C  D. Use central differences and get more accuracy.
C  Twice the function evaluations are needed.
IOPT(1)=-2
IOPT(2)= 0
C  Set the increment used at the default value.
C  This value must be assigned when using central differences.
WK(3*M+3)=0.D0

MODE=0

40  CONTINUE
WK(1)=A*EXP(B*Y(1))+C*Y(1)*Y(2)**2
IF (MODE .eq. 0) THEN
    F(1)=WK(1)
END IF
CALL DJACG(MODE,M,N,Y,F, .
    .FIAC,LDJAC,XSCALE,FAC,IOPT, .
    .WK,LIWK,IWK,LIWK)
IF (MODE .gt. 0) GO TO 40

C Check for an error condition.
IF (MODE .lt. 0) THEN
    PRINT ' ( ' ' Initial error in argument number ' ',I2 ) ' ,-MODE
    GO TO 60
END IF

C Check the relative accuracy of central differences.
C They should be good to about two thirds-precision.
FJAC(1,1)=(FJAC(1,1)-ACTUAL(1,1))/ACTUAL(1,1)
FJAC(1,2)=(FJAC(1,2)-ACTUAL(1,2))/ACTUAL(1,2)
F2=(3.D0*U)**(2.0/3.0)
RE(4,1)=FJAC(1,1)/F2
RE(4,2)=FJAC(1,2)/F2

C Output the results and what is expected.
PRINT ' ( ' ' Rel Err of partials , f= a*exp(b*y_1)+c*y_1*(y_2)**2. ' '/ .
    . 'Case df/dy_1 df/dy_2 , u=macheps**.5 , v=(3*macheps)**(2/3) ' ) '
MAXERR=0.D0
UV='u'
DO 50 I=1,4
    MAXERR=max(MAXERR, max(ABS(RE(1,1)), ABS(RE(1,2))))
    IF (I .eq. 4) UV='v'
    PRINT ' (I3 , 2(F7.2 , A1, 2 x ) , A55) ' ,
        I,RE(1,1), uv, RE(1,2), uv, WHAT(1)
50 CONTINUE

C All expected relative errors (in units of truncation error)
C should not exceed 8. If they do there may be an error.
IF (MAXERR .le. 8.D0) THEN
    PRINT ' ( ' ' Numbers above with absolute values .le. 8 are ' ', .
        . 'considered acceptable.' ) '
ELSE
    PRINT ' ( ' ' Numbers above with absolute values .gt. 8 are ' ', .
        . 'considered unacceptable.' ) '
END IF
60 CONTINUE

END

ODDJACG1

Rel Err of partials , f= a*exp(b*y_1)+c*y_1*(y_2)**2.
Case df/dy_1 df/dy_2 , u=macheps**.5 , v=(3*macheps)**(2/3)
  1  3.62u  3.97u One sided partials , default settings
  2  3.62u  0.00u One sided partial , second partial known and skipped
  3  3.60u  0.34u One sided partials , first partial accumulated
  4  4.11v  −0.77v Central difference partials
Numbers above with absolute values .le. 8 are considered acceptable.