10/7-2-90 75 (1)

ornl

ORNL/M-1121

OAK RIDGE NATIONAL LABORATORY

MARTIN MARIETTA

Automated Sensitivity Analysis with the Gradient Enhanced Software System (GRESS)

J. E. Horwedel

DO NOT MICROFILM COVER

OPERATED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831; prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161.
NTIS price codes—Printed Copy: A04 Microfiche A01

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/M--1121

DE90 011291

Engineering Physics and Mathematics Division

AUTOMATED SENSITIVITY ANALYSIS WITH THE GRADIENT ENHANCED SOFTWARE SYSTEM (GRESS)

J. E. Horwedel*

DATE PUBLISHED — May 1990

Contract Program: DOE Project No. 3410-6926

*Computing and Telecommunications Division

NOTICE: This document contains information of a preliminary nature. It is subject to revision or correction and therefore does not represent a final report.

Prepared by the
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831
operated by
MARTIN MARIETTA ENERGY SYSTEMS, INC.
for the
U.S. DEPARTMENT OF ENERGY
under contract DE–AC05–84OR21400

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am grateful to Brian Worley for his continued support of the development activities related to GRESS and ADGEN. I am also pleased to acknowledge the efforts by R. E. Maerker in specifying the requirements for several new utility routines that improve the sensitivity report generated by an ADGEN application. Finally, I appreciate the careful typing of this report by Wanda Merriweather.

ABSTRACT

GRESS automates the implementation of the well-known adjoint method for performing a comprehensive sensitivity analysis of existing FORTRAN 77 computer models. The GRESS ADjoint matrix GENerator (ADGEN) option is used to calculate first derivatives and sensitivities of selected model results with respect to all input data. This report provides a small sample problem used to exercise most of the major program options for a GRESS/ADGEN application. A description of the input and output from each processing step as well as the method for controlling the application is presented. The report is designed to aid users of the GRESS ADGEN option.

1. INTRODUCTION

ADGEN was developed as a GRESS option that provides the capability of automated implementation of the adjoint sensitivity methods into existing FORTRAN 77 models.[1-3] The GRESS Version 0.0 User's Manual describes how to use GRESS for both the CHAIN and ADGEN applications. However, recent improvements in the GRESS ADGEN option render some of the information included in the GRESS Version 0.0 User's Manual obsolete.[4-6] Specifically the processing steps in performing an ADGEN application are different. This report is supplemental to the documentation of earlier versions of GRESS.

The flow chart shown in Fig. 1 illustrates the processing steps for a GRESS application. A FORTRAN 77 program is input to the GRESS precompiler (EXAP). EXAP creates a new FORTRAN program that when compiled and linked with the GRESS run-time library is capable of calculating derivatives for each real equation along with the normally calculated result. For an ADGEN application, these derivatives are written to a computer disk in a structure that can easily be solved by back substitution. Utility programs are then used to solve the matrix for user selected results of interest. A report of sensitivities and derivatives for selected results with respect to all input data is generated. The actual sensitivity reported is normalized and expressed as as percentage. Input data is identified either automatically as any data that is entered via a FORTRAN read statement or manually by the user through the insertion of subroutine calls to the GRESS run-time library.

The major improvement in GRESS Version 1.0 was the implementation of Forward and Back Reduction algorithms as presented in Refs. 4 and 5. The implementation of these algorithms significantly reduces the amount of data storage required to perform an adjoint application with GRESS.

Appendixes A and B contain GRESS utility routine descriptions and commands to the precompiler. These appendixes supersede sections 4.10 and 4.11 in the GRESS Version 0.0 User's Manual.

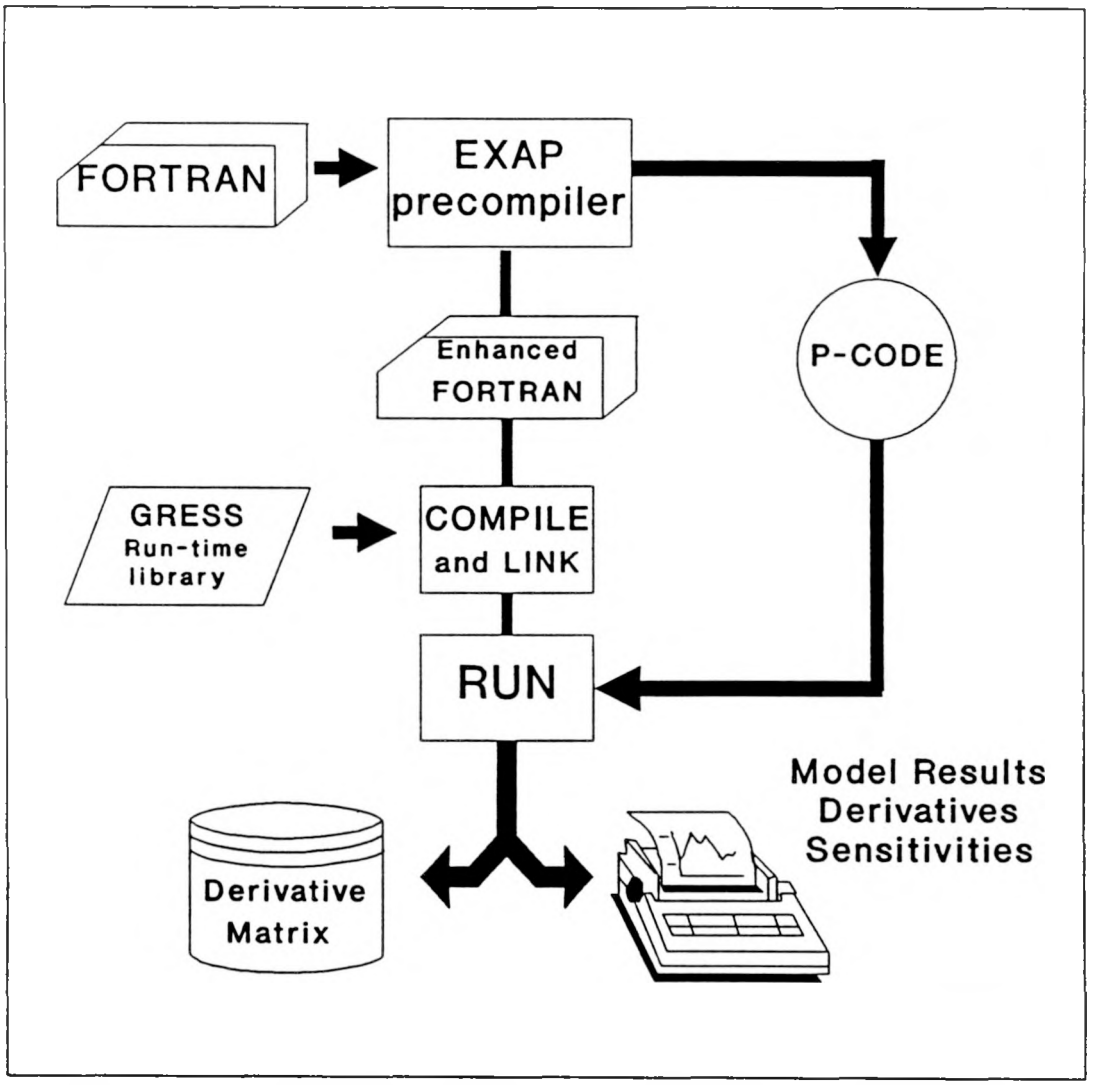

Fig. 1. Processing steps for a GRESS application.

2. APPLICATION INFORMATION

Provided in this section is information on how to do a GRESS ADGEN application. A simple FORTRAN program is presented as an example. The processing steps, data sets created, output information, and utility programs necessary to perform a complete application are shown. There are four steps in performing an ADGEN application: 1) precompile with EXAP; 2) compile and link with GRESS run-time library; 3) execute the enhanced code; and 4) solve the matrix.

2.1. SAMPLE PROBLEM TEST.FOR

Shown in Fig. 2 is a simple FORTRAN program to be used to demonstrate the GRESS ADGEN option.

```
C       GRESS/ADGEN test program
        READ(5,1000)B,C,D
        X = B + D
        Y = D**2 + B**2
        R = 7.0*X + D**2
        S = Y**2
1000    FORMAT(3E15.5)
        END
```

Fig. 2. A simple FORTRAN program, TEST.FOR.

Note specifically that FORTRAN variables B, C, and D are input via a READ statement which means they will automatically be treated as independent parameters. FORTRAN variables R and S will be chosen as results of interest. That is, the first derivatives and sensitivities of variables R and S with respect to B, C, and D will be calculated and reported. The modifications to prepare the code for an ADGEN application are shown in Fig. 3. Subroutine calls are included to identify the purpose of the run (SETRXX), to select results of interest (POTRXX), and to "clear" the matrix buffers (CLEARXX). A description of how to use these routines and others is included in Appendix A. Also the first line in Fig. 3., *COMMENTS ON, is a command to the precompiler. Commands to the precompiler are described in Appendix B. The *COMMENTS ON command will cause any comments beginning with a 'C' in column one to be included in the enhanced code. The default is to not include comments in the enhanced code.

```
*COMMENTS ON
                CALL SETRXX('ADJOINT')
C               GRESS/ADGEN test program
                READ(5,1000)B,C,D
                X = B + D
                Y = D**2 + B**2
                R = 7.0*X + D**2
                CALL POTRXX(R,' R ')
                S = Y**2
                CALL POTRXX(S,' S ')
                CALL CLEARXX
1000            FORMAT(3E15.5)
                END
```

Fig. 3. Program TEST.FOR prepared for an ADGEN application.

2.2. PRECOMPILATION

Once the code is prepared for precompilation, make the appropriate logical unit assignments and execute EXAP. The default logical units are described in Table 1.

Table 1. Default logical units used by EXAP

Data Set	Logical Unit	Description
Enhanced code	7	EXAP enhanced code
Input code	50	Program to be enhanced
P-code	80	Binary pseudo-machine code

The following commands will make logical unit assignments and execute EXAP to enhance TEST.FOR.

```
$ASSIGN/USER_MODE   TEST.FOR FOR050
$ASSIGN/USER_MODE   EH.FOR FOR007
$ASSIGN/USER_MODE   PC80.DAT FOR080
$RUN EXAP
```

The two data sets created during precompilation are the enhanced source program and the P-code. The P-code is an unformatted data set that becomes input data to the enhanced code. The P-code contains information that GRESS uses to solve each equation. A more complete description of the P-code and the enhancement process is included in Reference 2. A listing of the enhanced version of TEST.FOR (i.e., EH.FOR) is included as Appendix C.

2.3. COMPILING AND LINKING THE ENHANCED CODE

The FORTRAN 77 compiler and link editor used with the code prior to enhancement are also used to compile and link the enhanced code. The object module for the enhanced code must be linked with the GRESS run-time library. This example assumes that the GRESS run-time library (EXLIB.OLB) is in the same directory as the enhanced code and that the enhanced code is named EH.FOR.

```
$FOR EH
$LINK EH,EXLIB/LIB
```

2.4. EXECUTING THE ENHANCED CODE

Table 2. shows the logical units used during execution of the enhanced code.

Table 2. Logical units required by the enhanced code.

Data Set	Logical Unit	Description
Parameter Dictionary	42	selected parameters
Response Dictionary	43	selected results
Adjoint matrix[a]		
DERIV.DAT	46	derivative values
COLUMN.DAT	47	column numbers
NPAIRS.DAT	48	non-zero derivative count

[a]The adjoint matrix is represented by three unformatted data sets containing the derivative values, column numbers, and number of non-zero derivatives for each row in the matrix.

For this example three numbers (2.0, 3.0, 5.0) were entered into a file named USER.DAT with a 3E15.5 format. To execute the enhanced version of TEST.FOR (i.e., EH) and create the adjoint matrix enter the following commands.

```
$ASSIGN/USER_MODE   PC80.DAT FOR080
$ASSIGN/USER_MODE   USER.DAT FOR005
$RUN EH
```

2.5. SOLVING THE ADJOINT MATRIX

The two utility programs used to solve the adjoint matrix are BREDUCE and BSOLVE. BREDUCE implements the Back Reduction algorithm discussed in Refs. 4 and 5 to create a "reduced" form of the adjoint matrix. The BREDUCE step actually only needs to be executed when working with large models; however, it is included here for purposes of demonstration. One could proceed directly to the BSOLVE step. After executing the BREDUCE step there will be two copies of the adjoint matrix data sets. Only the latest version needs to be kept. To execute BREDUCE and then solve the matrix enter the following.

```
$RUN BREDUCE
$RUN BSOLVE
```

The BSOLVE program will request a value for the smallest sensitivity to report (i.e., CUTOFF). In practice sensitivities that are less than 0.01 are of little interest. Entering a value of 0.0 for CUTOFF will result in all sensitivities being reported. The output from the BSOLVE program is a report of derivatives and sensitivities written to logical unit 45. The sensitivity report for TEST.FOR is shown in Fig. 4.

Row number for response = 6 Number of parameters = 3

RESPONSE 1 R = 7.400000D+01

ROW	NAME	DERIVATIVE	SENSITIVITY
1	B	7.00000E+00	1.89189E-01
2	C	0.00000E+00	0.00000E+00
3	D	1.70000E+01	1.14865E+00

Row number for response = 7 Number of parameters = 3

RESPONSE 2 S = 8.410000D+02

ROW	NAME	DERIVATIVE	SENSITIVITY
1	B	2.32000E+02	5.51724E-01
2	C	0.00000E+00	0.00000E+00
3	D	5.80000E+02	3.44828E+00

Fig. 4. Derivative and sensitivity report generated by BSOLVE.

3. CONCLUSIONS AND RECOMMENDATIONS

The sample problem presented in this report is meant both as a supplement to previous documentation and as an aid to GRESS users. For a more complete discussion of the processing steps and data sets created Refs. 1 - 4 are recommended.

REFERENCES

1. Oblow, E. M. "GRESS, Gradient-Enhanced Software System," ORNL/TM-9658, Oak Ridge National Laboratory, Oak Ridge, TN (1985).
2. Horwedel, J. E., B. A. Worley, Oblow, E. M., Pin, F. G., and Wright, R. Q., "GRESS Version 0.0 User's Manual," ORNL/TM-10835, Oak Ridge National Laboratory (1988).
3. Worley, B. A., Pin, F. G., Horwedel, J. E., and Oblow, E. M., "ADGEN-ADjoint GENerator For Computer Models," ORNL-11037, Oak Ridge National Laboratory (1989).
4. Horwedel, J. E., "Matrix Reduction Algorithms for GRESS and ADGEN," ORNL/TM-11261, Oak Ridge National Laboratory (1989).
5. Horwedel, J. E., Raridon, R. J., and Wright, R. Q., "Sensitivity Analysis of AIRDOS-EPA Using ADGEN with Matrix Reduction Algorithms," ORNL/TM-11373, Oak Ridge National Laboratory (1989).
6. Horwedel, J. E., Wright, R. Q., Maerker, R. E., "Sensitivity Analysis of EQ3," ORNL/TM-11407, Oak Ridge National Laboratory (1990).

APPENDIX A

GENERAL UTILITY ROUTINES USED IN GRESS

Utility routines are used to control the application of the enhanced code. The following pages provide a description of each utility function. The format is one utility function per page. Each page includes at least one example on how to use the routine.

UTILITY ROUTINE

Name: AUTOXX (LUN, NUMP)

Function: To set the maximum number of parameters to be declared.

Application: CHAIN

Arguments:
1) LUN = -1 is required
2) NUMP - maximum number of parameters to be declared

Argument Types: Integer*4

How to use it: CALL AUTOXX must be made after CALL SETRXX but before calling any other library routines. A call to AUTOXX is required if derivatives are to be calculated by forward propagation using the chain rule.

Note: The present version does not support positive values for argument LUN.

Example:

1) Specifying a maximum of five parameters in an enhanced code

```
      .
      .
      .
DATA X /4.0/
CALL INITXX(D00001)
CALL SETRXX('CHAIN')
CALL AUTOXX(-1,5)
      .
      .
      .
```

2) Specifying a maximum of twenty parameters in an unenhanced code

```
      .
      .
      .
CALL SETRXX('CHAIN')
XPZ = 5.500
      .
      .
      .
CALL AUTOXX(-1,20)
      .
      .
```

UTILITY ROUTINE

Name: BSOLXX (LUN1,LUN2,CUTOFF)

Function: To solve the adjoint matrix in virtual memory. This is a test routine and should only be used for small models.

Application: Adjoint

Arguments:
1) LUN1 - read CUTOFF from lun1. A zero value means use argument three as the CUTOFF value.
2) LUN2 - write sensitivity report to lun2.
3) CUTOFF - magnitude of the smallest sensitivity to report. A value of zero will result in all sensitivities being reported.

Argument Type:
1) INTEGER*4
2) INTEGER*4
3) REAL*4

How to use it: A call to BSOLXX may be used in place of a call to CLEARXX at the end of program execution. A senstivity report will be written to LUN2.

Examples:

1) To report sensitivities that are greater than 1.0E-4 to logical unit 95

```
      .
      .
CALL BSOLXX(0,95,1.0E-4)
STOP
END
```

UTILITY ROUTINE

Name: CLEARXX
CLEADXX

Function: To clear the forward matrix buffers.

Application: Adjoint

Arguments: NONE

How to use it: Generally, CALL CLEARXX should be inserted in the main program immediately before the STOP statement. CLEARXX must be the last executed call before ending the run. If the program exits at some point other than in the main program, it will be necessary to insert CALL CLEARXX at that point. Use CLEADXX if responses are double precision and you are using G_float option on the FORTRAN compiler.

Example:

1) Normal exit from a main program with a STOP statement

```
PROGRAM MAIN
  .
  .
CALL CLEARXX
STOP
END
```

2) Possible exit from subroutine and no STOP statement

```
PROGRAM MAIN
  .
  .
  .
CALL THIS
CALL CLEARXX
END
SUBROUTINE THIS
  .
  .
  .
IF(UNHAPPY) CALL CLEARXX
IF(UNHAPPY) STOP
RETURN
END
```

UTILITY ROUTINE

Name: DECLARXX (CHAR)

Function: To turn the declaration of parameters on or off.

Application: CHAIN

Arguments: 'ON' or 'OFF'

Argument Type: CHARACTER

How to use it: CALL DECLARXX is used to limit the number of parameters added to the parameter dictionary. The default is 'ON', meaning that any parameters read in will automatically be added to the paramameter dictionary. To limit the number of parameters, use DECLARXX to turn off the automatic declaration at the start of the program. To have specific parameters added to the parameter dictionary, use DECLARXX to turn on the declaration of parameters, either immediately before the read statement in the unenhanced model or immediately before calling parameter declaration routines in the enhanced model.

Example:

1) Using DECLARXX to cause only C to be declared a parameter.

```
      .
      .
      .
   DATA X /4.0/
   CALL INITXX(D00001)
   CALL SETRXX('ADJOINT')
   CALL DECLARXX('OFF')
      .
      .
      .
   READ (100,1000)A,B
   CALL INNRXX(A,'  A  ')
   CALL INNRXX(B,'  B  ')
   CALL DECLARXX('ON')
   READ(100,1001)C
   CALL INNRXX(C,'  C  ')
   CALL DECLARXX('OFF')
      .
      .
      .
```

UTILITY ROUTINE

Name: DEFIXX (VAR)

Function: To declare a parameter for a GRESS run. No gradients are computed until at least one parameter is defined. Each call adds a new parameter. There is an upper limit of 200 parameters. Each additional parameter declaration increases the amount of memory and execution time required by the enhanced code.

Application: CHAIN

Arguments: 1) VAR - program variable to be declared a parameter

Argument Type: REAL*4 or Real*8

How to use it: Insert CALL DEFIXX after the variable has been initialized or defined. Subroutines SETRXX and AUTOXX must be called prior to CALL DEFIXX.

EXAMPLE:

1) Declare Y to be a parameter for a GRESS run.

```
      .
      .
READ(LUN,100) Y
CALL DEFIXX(Y)
      .
      .
      .
X = 2.0 * Y + Z
      .
      .
      .
```

2) Declare D(5) to be a parameter.

```
      .
      .
      .
D(J) = B(J)**2
IF(J.EQ.5) CALL DEFIXX(D(J))
      .
      .
      .
```

UTILITY ROUTINE

Name: DEFAXX (ARRAY, NELEM, NTYPE)

Function: To declare elements in an array as parameters for a GRESS run. No gradients are computed until at least one parameter is defined. Each call adds a new array to the list of parameters. Each element in the array counts as one parameter. There is an upper limit of 200 parameters. Each additional parameter declaration increases the amount of memory and execution time required by the enhanced code.

Application: CHAIN

Arguments:

1) ARRAY - array to be declared a parameter
2) NELEM - number of elements in array to be declared
3) NTYPE - argument type

Argument Type:

1) REAL*4 or REAL*8
2) INTEGER*4
3) INTEGER*4

How to use it: Insert CALL DEFAXX after the array has been initialized or defined. Subroutines SETRXX and AUTOXX must be called prior to CALL DEFAXX. For multi-dimensional arrays, the number of elements specified must take into consideration how FORTRAN treats dimensioned variables. DEFAXX will define the array elements as parameters, sequentially, in order of their location in memory.

Example:

1) Declare the elements in array Y as parameters for a GRESS run.

```
      .
      .
      .
  DIMENSION Y(10),X(10)
      .
      .
  READ(LUN,100) Y
  NUMP=10
  NTYPE=1
  CALL DEFAXX(Y(1),NUMP, NTYPE)
      .
      .
      .
  X(I) = 2.0 * Y(I) + Z
      .
      .
```

UTILITY ROUTINE

Name: DLIBXX (VARIABLE, NAME)

Function: To add a double precision variable to the parameter dictionary.

Arguments:
1) VARIABLE to be included
2) name or description of VARIABLE

Argument type:
1) REAL*8
2) CHARACTER*N (N < 12)

How to use: Insert CALL DLIBXX after the point in the code where a value is assigned to VARIABLE.

EXAMPLE:

1) To declare X to be a parameter in an adjoint application.

```
REAL*8 X
   .
   .
   .
X = 2.0D+01 * Y + Z
CALL DLIBXX(X,' X ')
   .
   .
   .
```

UTILITY ROUTINE

Name: DLINXX (VARIABLE, NAME, COUNTER)

Function: To add a double precision variable to the parameter dictionary with a user defined counter.

Arguments:
1) VARIABLE to be included
2) name or description of VARIABLE
3) user defined counter

Argument type:
1) REAL*8
2) CHARACTER*N (N < 12)
3) INTEGER*4

How to use: Insert CALL DLINXX after the point in the code where a value is assigned to VARIABLE.

EXAMPLE:

1) To declare X to be a parameter in an adjoint application with the integer ICOUNT as a user defined counter to help identify the result in the sensitivity report.

```
REAL*8 X
  .
  .
  .
ICOUNT=ICOUNT+1
X = 2.0D+01 * Y + Z
CALL DLINXX(X,' X ',ICOUNT)
  .
  .
  .
```

UTILITY ROUTINE

Name: DRAYXX (ARRAY, N1, N2, N3, N4, NDIMS, NAME)

Function: To add the elements in a double precision array to the parameter dictionary.

Arguments:
1) ARRAY with up to four dimensions
2-5) N1-N4 dimensions 1 to 4 of ARRAY
6) NDIMS - actual number of dimensions
7) name or description of ARRAY

Argument type:
1) REAL*8
2-5) INTEGER*4
6) INTEGER*4
7) CHARACTER*N (N < 12)

How to use: Insert CALL DRAYXX after the point in the code where values have been assigned to ALL elements to be included in the parameter dictionary. However, the call must be made prior to any element of ARRAY being used to assign a value to any other variable in the FORTRAN program. The values for N2-N4 must be set to at least 1, even if that dimension does not exist. For example, if an array has only two dimensions, arguments N3 and N4 must be assigned the value of 1.

EXAMPLE:

1) To declare the elements in array X to be a parameters in an adjoint application.

```
      REAL*8 X(10,5)
        .
        .
        .
       DO 20 I=1,10
         DO 10 J=1,5
           X(I,J) = 2.0D+01 * Y(I,J) + Z
10       CONTINUE
20     CONTINUE
       CALL DRAYXX( X, 10, 5, 1, 1, 2, 'X')
        .
        .
        .
```

UTILITY ROUTINE

Name: FILEXX (LUN)

Function: To alter the logical unit number for all printed output generated by run-time library routines. The default logical unit number for printed output from the utility routines is 6.

Applicaton: CHAIN

Arguments: LUN - Logical unit number for printed output

Argument Type: INTEGER*4

How to use it: If the user chooses to have all or part of the calculated gradients from a GRESS application written to a file other than unit 6, simply call FILEXX with an integer argument specifying the desired unit number. The assignment stays active until the end of the run, or until FILEXX is called again.

Examples:

1) To print all gradient results in a GRESS application to logical unit 90.

```
      .
      .
      .
LUN=90
CALL FILEXX(LUN)
      .
      .
      .
```

2) To temporarily change the logical unit during a GRESS run for selected results

```
      .
      .
      .
LUN=70
CALL FILEXX(LUN)
CALL BSOLXX(A)
LUN=6
CALL FILEXX(LUN)
      .
      .
      .
```

UTILITY ROUTINE

Name: GETNXX (X, N, Z)

Function: To retrieve an individual derivative

Application: CHAIN

Arguments:
1) X - any program variable
2) N - parameter number
3) Z - storage location

Argument Type:
1) REAL*4 or REAL*8
2) INTEGER*4
3) REAL*4

How to use it: GETNXX returns the derivative of X with respect to the N^{th} declared parameter. As mentioned earlier, parameters have an "ordinal" number corresponding to the sequence in which they are declared. It is necessary to provide a REAL*4 variable as the third argument for storing the retrieved derivative.

Example:

1) Retrieve the derivative of A with respect to the first, second, and fourth declared parameters. Store the derivatives in the first three locations in array ZZ.

```
      .
      .
      .
  CALL GETNXX(A,1,ZZ(1))
  CALL GETNXX(A,2,ZZ(2))
  CALL GETNXX(A,4,ZZ(3))
      .
      .
      .
```

UTILITY ROUTINE

Name: GETGXX (X, Z)

Function: To retrieve an individual derivative using symbol name

Application: CHAIN

Arguments:
1) X - any program variable
2) Z - an array

Argument Type:
1) REAL*4 or REAL*8
2) REAL*4

How to use it: GETGXX returns the derivatives of X with respect to the N declared parameters. Z must be dimensioned by at least N to hold the derivative of X with respect to each declared parameter.

Example:

1) Retrieve the derivative of A with respect to all declared parameters. Store the derivatives in array B.

```
      .
      .
  DIMENSION B(8)
  CALL AUTOXX(-1,8)
      .
      .
      .
  CALL GETGXX(A,B)
      .
      .
      .
```

UTILITY ROUTINE

Name: NINDXX (N)

Function: To return the current number of declared parameters.

Application: CHAIN

Arguments: N - number of declared parameters

Argument Type: Integer

How to use it: At any point during program execution, the number of parameters presently declared is returned as an integer argument.

Examples:

1) To check the number of declared parameters in a GRESS run.

```
      .
      .
      .
CALL NINDXX (N)
IF(N.GT.3) THEN
      .
      .
      .
```

UTILITY ROUTINE

Name: POTRXX (X, 'CHAR')
POTDXX (X, 'CHAR')

Function: To declare a variable as a potential response of interest during an adjoint application run. A list of potential responses will be printed at the end of the run with row number and name.

Application: Adjoint

Arguments:
1) X: program variable to be declared
2) name or description of variable

Argument Type:
1) REAL*4 or REAL*8
2) CHARACTER*n (n < 24)

How to use it: Insert CALL POTRXX immediately following the line defining the variable. CLEARXX must be called at the end of the run. Use POTDXX if X is REAL*8. POTDXX is used exactly the same way as POTRXX.

Examples:

1) Declare X to be a potential response of interest.

```
      .
      .
      .
X = 2.0 * Y + Z
CALL POTRXX(X,'  X   ')
      .
      .
      .
```

2) Declare D(5) to be a response of interest.

```
      .
      .
      .
D(J) = B(J)**2
IF(J.EQ.5) CALL POTRXX(D(J),'  D of 5 ')
      .
      .
      .
```

UTILITY ROUTINE

Name: PRNTSS (X)

Function: To print the sensitivities of a dependent variable with respect to the declared parameters when using the *SENSON command.

Application: CHAIN

Arguments: Any program variable

Argument Type: REAL*4 or REAL*8

How to use it: At any point during program execution, the gradient of a dependent variable may be printed by a call to PRNTSS.

Examples:

1) To print gradients at two places in a GRESS run.

```
      .
      .
READ(LUN,100) Y
CALL DEFIXX(Y)
      .
      .
      .
X = 2.0 * Y + Z
CALL PRNTSS(X)
      .
      .
      .
 Z = X*B
 CALL PRNTSS(Z)
      .
      .
      .
```

UTILITY ROUTINE

Name: PRNTXX (X)

Function: To print the gradients and sensitivities of a dependent variable with respect to the declared parameters.

Application: CHAIN

Arguments: Any program variable

Argument Type: REAL*4 or REAL*8

How to use it: At any point during program execution, the gradient of a dependent variable may be printed by a call to PRNTXX.

Examples:

1) To print gradients at two places in a GRESS run.

```
      .
      .
READ(LUN,100) Y
CALL DEFIXX(Y)
      .
      .
      .
X = 2.0 * Y + Z
CALL PRNTXX(X)
      .
      .
      .
Z = X*B
CALL PRNTXX(Z)
      .
      .
      .
```

UTILITY ROUTINE

Name: REDUXX

Function: To create a reduced form of the adjoint matrix to be solved later by the BSOLVE program. This is a test routine that implements the Back Reduction algorithm directly in the enhanced code.

Application: Adjoint

Arguments: NONE

How to use it: A call to REDUXX may be used in place of a call to CLEARXX at the end of program execution.

Examples:

1) To create a reduce form of the adjoint matrix

```
      .
      .
CALL REDUXX
STOP
END
```

UTILITY ROUTINE

Name: RLIBXX (VARIABLE, NAME)

Function: To add a single precision variable to the parameter dictionary.

Arguments:
1) VARIABLE to be included
2) name or description of VARIABLE

Argument type:
1) REAL*4
2) CHARACTER*N (N < 12)

How to use: Insert CALL RLIBXX after the point in the code where a value is assigned to VARIABLE.

EXAMPLE:

1) To declare X to be a parameter in an adjoint application.

```
REAL*4 X
   .
   .
   .
X = 2.0D+01 * Y + Z
CALL RLIBXX(X,' X ')
   .
   .
   .
```

UTILITY ROUTINE

Name: RLINXX (VARIABLE, NAME, COUNTER)

Function: To add a single precision variable to the parameter dictionary with a user defined counter.

Arguments:
1) VARIABLE to be included
2) name or description of VARIABLE
3) user defined counter

Argument type:
1) REAL*4
2) CHARACTER*N (N < 12)
3) INTEGER*4

How to use: Insert CALL RLINXX after the point in the code where a value is assigned to VARIABLE.

EXAMPLE:

1) To declare X to be a parameter in an adjoint application with the integer ICOUNT as a user defined counter to help identify the result in the sensitivitiy report.

```
REAL X
  .
  .
  .
ICOUNT=ICOUNT+1
X = 2.0D+01 * Y + Z
CALL RLINXX(X,' X ',ICOUNT)
  .
  .
  .
```

UTILITY ROUTINE

Name: RRAYXX (ARRAY, N1, N2, N3, N4, NDIMS, NAME)

Function: To add the elements in a single precision array to the parameter dictionary.

Arguments:
1) ARRAY with up to four dimensions
2-5) N1-N4 dimensions 1 to 4 of ARRAY
6) NDIMS - actual number of dimensions
7) name or description of ARRAY

Argument type:
1) REAL*4
2-6) INTEGER*4
7) CHARACTER*N (N < 12)

How to use: Insert CALL RRAYXX after the point in the code where values have been assigned to ALL elements to be included in the parameter dictionary. However, the call must be made prior to any element of ARRAY being used to assign a value to any other variable in the FORTRAN program. The values for N2-N4 must be set to at least 1, even if that dimension does not exist. For example, if an array has only two dimensions, arguments N3 and N4 must be assigned the value of 1.

EXAMPLE:

1) To declare the elements in array X to be a parameters in an adjoint application.

```
      REAL*4 X(10,5,5)
          .
          .
          .
        DO 20 I=1,10
          DO 15 J=1,5
            DO 10 K=1,5
              X(I,J,K) = 2.0D+01 * Y(I,J,K) + Z
10          CONTINUE
15        CONTINUE
20      CONTINUE
        CALL RRAYXX( X, 10, 5, 5, 1, 3, 'X')
          .
          .
          .
```

UTILITY ROUTINE

Name: SETRXX (CHAR)

Function: To specify the purpose for the run (i.e., chain rule propagation of derivatives or adjoint matrix generation).

Application: CHAIN or Adjoint

Arguments: 'CHAIN' or 'ADJOINT'

Argument Type: CHARACTER*5 or CHARACTER*7

How to use it: CALL SETRXX must be the first executed line in the unenhanced code, or the line immediately following CALL INITXX in the enhanced code. If CALL SETRXX is NOT made, no derivatives will be calculated.

Examples:

1) Enhanced code ready for chain rule propagation of derivatives.

```
      .
      .
      .
    DATA X /4.0/
    CALL INITXX(D00001)
    CALL SETRXX('CHAIN')
      .
      .
      .
```

2) Unenhanced code preparing for adjoint matrix generation.

```
      .
      .
      .
    DATA X /4.0/
    CALL SETRXX('ADJOINT')
      .
      .
      .
```

APPENDIX B

DETAILED DESCRIPTION OF PRECOMPILER COMMANDS

Commands to the precompiler are used to control the creation of the enhanced code. The following pages provide a description of each command. The format is one command per page. Each page includes at least one example on how to use the routine. All commands to the precompiler must begin with the asterisk (i.e., *) in column 1. An asterisk was chosen because it is a legal comment to FORTRAN 77 compilers and does not impede syntax checking using the FORTRAN compiler.

PRECOMPILATION COMMAND

Name: *COMMENTS ON/OFF

Purpose: To cause EXAP to pass comments from code being translated to the enhanced code.

Notes: Only comments indicated with a lowercase or uppercase C in column one are passed. Blank lines and comments indicated by an asterisk in column one are not passed. Use COMMENTS OFF if you want to selectively pass comments. The default is comments off.

Examples:

```
*COMMENTS ON
      DIMENSION X(100)
      COMMON/ ALPHA/ Y,Z
          .
          .
          .
```

PRECOMPILATION COMMAND

Name: *ECHO ON/OFF

Purpose: To cause EXAP to echo the line of code being translated as a comment in the enhanced code. This can be very helpful in debugging.

How to use it: ECHO can be used to echo the entire code as comments in the enhanced code, or to selectively echo part of the enhanced code. The default is ECHO OFF.

Example:

1) To echo one line in the enhanced code as a comment.

```
      DIMENSION X(100)
      COMMON/ ALPHA/ Y,Z
          .
          .
          .
*ECHO ON
     X(I)=Z*Y + 5.0
*ECHO OFF
```

PRECOMPILATION COMMAND

Name: *EXAP ON/OFF

Purpose: To selectively specify lines or sections of code to be enhanced, or not enhanced. This is a very dangerous option and should only be used by knowledgeable users.

How to use it: Can be used anywhere within a complete program module or to turn off the enhancement of entire subprograms. Enhancement cannot be turned off within a subroutine and then turned back on within another subroutine. The enhancement must be turned on to process the declaration section of a subprogram if it is turned on anywhere within that subprogram.

Examples:

1) To prevent one line from being enhanced

```
      DIMENSION X(100)
      COMMON/ ALPHA/ Y,Z
         .
         .
         .
*EXAP OFF
      X(I)=Z*Y + 5.0
*EXAP ON
```

PRECOMPILATION COMMAND

Name: *SENSON

Purpose: To cause GRESS to propagate sensitivities rather than derivatives.

Notes: This is a risky option.[a] It is available for testing only. It is available with both ADGEN and CHAIN. As an equation is solved the sensitivity of the result is calculated with respect to the terms on the right-hand-side of the equation. The sensitivities are propagated similar to the way derivatives are propagated. If you are using the CHAIN option, then the PRNTSS utility should be used to print sensitivities as opposed to the PRNTXX utility. It is recommended that you only use this option if you are using G_float and the exponents for your derivatives are exceeding the allowable range for F_float representation (i.e., approximately E-38 to E+38).

How to use it: Should be at the top of the main program beginning in column one.

Example:

```
*SENSON
      DIMENSION X(100)
      COMMON/ ALPHA/ Y,Z
            .
            .
            .
```

[a]The *SENSON option has undergone limited testing. In situations where the result from and equation is 0.0, the sensitivity for that equation is set to 0.0. However, if the derivative is <u>not</u> zero, it is possible for the effect of a parameter dependency to be lost. Under those circumstances the *SENSON option <u>may</u> give an incorrect value for a reported sensitivity.

PRECOMPILATION COMMAND

Name: *WSPSIZE

Purpose: To override the default work space size.

How to use it: Must begin in column one.

Example: To set the work space at 2 million words:

```
*WSPSIZE 2 000 000
   DIMENSION X(100)
   COMMON/ ALPHA/ Y,Z
        .
        .
        .
```

Comment: Both CHAIN and ADGEN options use a common block area that can vary in size dependent on the application. The default work space size is usually set at 8 million 4 byte words; however, the actual size is installation dependent. This command is useful if for any reason you desire a larger or smaller work space.

APPENDIX C

LISTING OF THE ENHANCED VERSION OF TEST.FOR.

```
      COMMON XD0001
C     GRESS/ADGEN test program
      DOUBLE PRECISION D00001(5)
      INTEGER I00001(5)
      REAL R00001(5)
      CALL INITXX(XD0001)
      CALL SETRXX('ADJOINT')
      READ(5,*)B,C,D
      CALL INSRXX(B,'B')
      CALL INSRXX(C,'C')
      CALL INSRXX(D,'D')
      CALL INTPXX(1,X,B,D,4H$)
      CALL INTPXX(7,Y,D,2,B,2,4H$)
      CALL INTPXX(19,R,7.0,X,D,2,4H$)
      CALL POTRXX(R,' R ')
      CALL INTPXX(31,S,Y,2,4H$)
      CALL POTRXX(S,' S ')
      CALL CLEARXX
      END
      SUBROUTINE INITXX(D00001)
      IMPLICIT INTEGER (A-Z)
      PARAMETER (WSP_SIZE = 8000000)
      COMMON/ZZZZZZ/LIMIT(WSP_SIZE+2)
      COMMON /ZZZZZZ01/ CODE(36)
C
      CALL PREPXX(D00001,WSP_SIZE,36,80)
      RETURN
      END
```

INTERNAL DISTRIBUTION

1. B. R. Appleton
2. M. B. Emmett
3–10. J. E. Horwedel
11. R. E. Maerker
12. F. C. Maienschein
13. R. W. Roussin
14. B. A. Worley
15. R. Q. Wright
16. EPMD Reports Office
17–18. Laboratory Records Department
19. Laboratory Records, ORNL–RC
20. Document Reference Section
21. Central Research Library
22. ORNL Patent Section

EXTERNAL DISTRIBUTION

23. J. J. Dorning, Department of Nuclear Engineering and Engineering Physics, Thornton Hall University of Virginia, Charlottesville, Virginia 22901
24. R. M. Haralick, Department of Electrical Engineering, University of Washington, Seattle, Washington 98195
25. James E. Leiss, 13013 Chestnut Oak Drive, Gaithersburg, Maryland 20878
26. Neville Moray, Department of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urban, Illinois 61801
27. Mary F. Wheeler, Mathematics Department, University of Houston, 4800 Calhoun, Houston, Texas 77204-3476
28–37. Office of Scientific and Technical Information, P. O. Box 62, Oak Ridge, Tennessee 37831
38. Office of the Assistant Manager for Energy Research and Development, Department of Energy, Oak Ridge Operations, P. O. Box 2001, Oak Ridge, Tennessee 37831

DO NOT MICROFILM THIS PAGE