HONORS THESIS

AUTHOR: ROBERT LOUCKS, JR.

FACULTY ADVISOR: DR. EARL MCKINNEY

TOPIC: A brief guide to using common IMSL Math and Statistical subroutines and functions of the IMSL PROTRAN language.
The International Math and Statistical Library (IMSL) is a library of mathematical and statistical FORTRAN subroutines made to interface with a user written FORTRAN main program.

The library comes in two forms:

1) single precision
2) double precision

The single precision and double precision differ in that real parameters passed to the IMSL subroutines are either in single or double precision. To run a FORTRAN program that has calls to IMSL subroutines in single precision form, do the following:

$ FOR programname
$ LINK programname,IMSLS/LIB
$ RUN programname

To run a FORTRAN program that has calls to IMSLS subroutines in double precision form, do the following:

$ FOR programname
$ LINK programname,IMSLD/LIB
$ RUN programname

The following are examples of how to use some of the more basic IMSL subroutines. The subroutines that are explained are:

1) DGEAR
2) DCADRE
3) DVERK
4) ICSCCU
5) LEQIF
6) LINVIF
7) VMULFF
8) ZANLYT
9) ZRPOLY

If anything is not clear or not presented, check the IMSL manuals in RB134.
NAME : DCADRE

PURPOSE : Performs integration over an interval using adaptive Romberg extrapolation. The function must be a single variable function.

CALLING STATEMENT : DCADRE (F,A,B,AERR,RERR,ERROR,IER)

PARAMETERS : F - Single argument real or double precision function subroutine. The function must be declared as external.

A - Beginning real or double precision endpoint of the interval to be integrated.

B - Ending real or double precision endpoint of the interval to be integrated.

AERR - Desired real or double precision absolute error at a single point being integrated.

RERR - Desired real or double precision relative error at a single point being integrated.

ERROR - Estimated bound on real or double precision estimated bound on absolute error over the entire interval being integrated.

IER - An integer error flag. Possible values of interest are:

0 - No error has occurred.

65 - One or more singularities handled successfully. It is a fixed warning, which means that the trouble was handled successfully.

66 - Some small acceptable estimated errors occurred along some subintervals. It is a fixed warning.

131 - Not enough working storage to process function. It is a terminal error.

132 - Error requirements can not be fulfilled or ill behaved integrand. It is a terminal error.

133 - RERR > 0.1 or RERR < 0.0, or too small precision in
* calculations for the machine
to handle. It is a terminal
terminal error.
*
* NOTES : DCADRE is a really a function and should be
used as one. It has no parameter to return the
result.
*
********************************************************************

PROGRAM SHOWDCADRE
EXTERNAL F
REAL F,A,B,AERR,RERR,ERROR,ANSWER,DCADRE
INTEGER IER

* Fill error parameters *
A = 0.0
B = 1.0
RERR = 0.0
AERR = 0.00001

* Call DCADRE function *
ANSWER = DCADRE (F,A,B,AERR,RERR,ERROR,IER)

* Output ANSWER, ERROR, and error flag IER *
WRITE (5,11) ANSWER
11 FORMAT (1X,'ANSWER = ',F12.7)
WRITE (5,22) ERROR
22 FORMAT (1X,'ERROR = ',F12.7)
WRITE (5,33) IER
33 FORMAT (1X,'IER = ',I3)
END

* Function to be integrated *

REAL FUNCTION F (X)
REAL X

F = EXP(X) + SIN (X)
END

ANSWER = 2.1779797
ERROR = 0.0000029
IER = 0
This is an example of the use of the double precision DCADRE function. Notice the only change is that all real parameters going to DCADRE are changed to double precision.

```
PROGRAM SHOWDOUBLEDCADRE
IMPLICIT NONE
EXTERNAL F
DOUBLE PRECISION F,A,B,AERR,RERR,ERROR,ANSWER
DOUBLE PRECISION DCADRE
INTEGER IER

* Fill error parameters *
A = 0.0
B = 1.0
RERR = 0.0
AERR = 0.00000001

* Call DCADRE function *
ANSWER = DCADRE (F,A,B,AERR,RERR,ERROR,IER)

* Output ANSWER, ERROR, and error flag IER *
WRITE (5,11) ANSWER
11 FORMAT (1X,'ANSWER = ',F23.16)
WRITE (5,22) ERROR
22 FORMAT (1X,'ERROR = ',F23.16)
WRITE (5,33) IER
33 FORMAT (1X,'IER = ',I3)

* Function to be integrated *
DOUBLE PRECISION FUNCTION F (X)
IMPLICIT NONE
DOUBLE PRECISION X

* Make sure to use the double precision functions *
F = DEXP(X) + DSIN (X)
END
```

ANSWER = 2.1779795225926145
ERROR = 0.0000000001564953
IER = 0
NAME : DGEAR

PURPOSE : Differential equation solver using variable order Adams Predictor Corrector method or Gears method.

CALLING STATEMENT : DGEAR (N,FCN,FCNJ,X,H,Y,XEND,TOL,
METH,MITER,INDEX,IWK,WK,IER)

PARAMETERS : N - Integer number of first order differential equations to be used.

FCN - Name of the subroutine for evaluating the functions.

FCNJ - Name of the subroutine for computing the Jacobian matrix of partial derivatives if the implementation of DGEAR calls for it. This documentation ignores the possibility as it complicates the use of DGEAR a great deal. Thus FCNJ will be a dummy subroutine consisting of a return statement.

X - The initial real or double precision value of the independent variable. After the return, X is the value at which integration has been completed.

H - The initial real or double precision step size. On return, H will equal the last successful step size. Keep H at or smaller than .001. Larger values will sometimes give access violations for some unknown reason.

Y - Real or double precision single dimension array of the dependent variables. Originally set to the initial values, on return it will equal the computed values at X = XEND.

XEND - Real or double precision endpoint at which X = XEND and a solution is desired. It must be greater than the initial value of X.

TOL - Real or double precision relative error bound. Usually less than or equal to .001.

METH - Integer that indicates which method is to be used. It is referenced only
on the first call unless INDEX = -1.
Possible values and meanings are:

1 - Adams method.
2 - Stiff methods of Gear or backward
differential formulae.

MITER - Integer that indicates the iteration
method. The following values are
the ones that allow the simplest
method of implementing DGEAR.

0 - Functional iteration is used.
No partial derivatives are
needed.
2 - Chord method is used. The
Jacobian is calculated
internally by finite
differences.
3 - Chord method is used. The
Jacobian is replaced by a
diagonal approximation based on
a directional derivative.

INDEX - Integer that indicates type of call
to DGEAR. On return, it is reset
to zero if integration is successful.
Possible values and meanings are:

-1 - Not first call for this problem
and TOL has been reset.
0 - Not first call for this problem.
1 - First call for this problem.
2 - Not first call for this problem,
continue integration, and XEND
must be hit exactly (no
interpolation is to be done).
3 - Not first call, continue
integration executing only one
step and return, thus ignoring
XEND.

IWK - Integer single dimension work array of
length N. It is used only if
MITER = 2.

WK - Real or double precision single dimension
work array of length:

\[(4 \times N) + \text{NMETH} + \text{NMITER}\]

The value of \text{NMETH} depends on the value
of \text{METH} as follows:

If \text{METH} = 1 then \text{NMETH} = N \times 13
If \( \text{METH} = 2 \) then \( \text{NMETH} = N \times 6 \)

The value of \( \text{NMITER} \) depends on the value of \( \text{MITER} \) as follows:

- If \( \text{MITER} = 0 \) then \( \text{NMITER} = 1 \)
- If \( \text{MITER} = 2 \) then \( \text{NMITER} = N \times (N + 1) \)
- If \( \text{MITER} = 3 \) then \( \text{NMITER} = N \)

\( \text{IER} \) - Integer error flag. Possible values and meanings are as follows:

- 0 - Everything is fine.
- 33 - It is a warning that means one of two things:
  1) \( \text{TOL} \) is too small so try increasing it.
  2) The differential equation being solved is stiff so try a \( \text{MITER} \) of 2 or 3.
- 66 - Error test failed. \( \text{H} \) was divided by 10 one or more times and the step was finally completed successfully. This is a fixed warning meaning a problem came up that was handled successfully.
- 67 - A fixed warning that means corrector convergence was not achieved. \( \text{H} \) was divided by 10, one or more times, and the step was completed successfully.
- 132 - A terminal error indicating integration was halted after failure to pass error test even after dividing \( \text{H} \) by a factor of \( 1.0 \times 10^10 \) from its initial value.
- 133 - A terminal error indicating integration was halted after failure to achieve corrector convergence even after dividing \( \text{H} \) by a factor of \( 1.0 \times 10^10 \) from its initial value.
- 134 - A terminal error indicating after some initial success, integration was halted by repeated error test failures or by test on \( \text{TOL} \).
- 135 - A terminal error indicating one of the input parameters \( \text{N} \), \( \text{X} \), \( \text{H} \), \( \text{XEND} \), \( \text{TOL} \), \( \text{METH} \), \( \text{MITER} \), or \( \text{INDEX} \) was specified incorrectly.
- 136 - A terminal error indicating that \( \text{INDEX} = -1 \) and \( \text{XEND} < \text{X} \).

\* NOTES: \( \text{DGEAR} \) can also be used to solve a single equation
by simply setting N to 1, declaring real array Y
as a vector of length 1, and set YPRIME(1) to the
single equation.

i.e.

REAL V(1)
N = 1
YPRIME(1) = SIN(Y(1))

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

PROGRAM SHOWDGEAR
INTEGER W,METH,MITER,INDEX,IWK(2),IER,K
REAL Y(2),WK(22),X,TOL,XEND,H
EXTERNAL FCN,FCNJ

* Initialize parameters *

N = 2
X = 0.0
Y(1) = 0.0
Y(2) = 1.0
TOL = .00001
H = .001
METH = 2
MITER = 3
INDEX = 1

* Write out table header *

WRITE (5,11)
11 FORMAT (2X,'IER',8X,'X',12X,'Y(1)',10X,'Y(2)')

* Main loop that traverses interval and gives answers *
* along the way. *

DO K = 1,10
   XEND = FLOAT(K)

* Call DGEAR *

   CALL DGEAR (N,FCN,FCNJ,X,H,XEND,TOL,METH,+
               MITER,INDEX,IWK,WK,IER)

* If fatal error then stop *

   IF (IER.GT.128) THEN
      WRITE (5,22) IER
      22 FORMAT (1X,'FATAL ERROR. IER = ',I3)
      STOP
   END IF

* Write out answers *
WRITE (5,33) IER,X,Y(1),Y(2)

FORMAT (1X,I3,2X,F12.6,2X,F12.6,2X,F12.6)

END DO
END

* Subroutine FCN to give equations for primes *

SUBROUTINE FCN(N,X,Y,YPRIME)
INTEGER N
REAL Y(N),YPRIME(N),X

YPRIME(1) = 2 * X
YPRIME(2) = Y(2)
RETURN
END

* Dummy subroutine FCNJ *

SUBROUTINE FCNJ(N,X,Y,PD)
INTEGER N
REAL Y(N),PD(N,N),X
RETURN
END

<table>
<thead>
<tr>
<th>IER</th>
<th>X</th>
<th>Y(1)</th>
<th>Y(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000000</td>
<td>1.000003</td>
<td>2.718469</td>
</tr>
<tr>
<td>0</td>
<td>2.000000</td>
<td>4.000002</td>
<td>7.389771</td>
</tr>
<tr>
<td>0</td>
<td>3.000000</td>
<td>9.000004</td>
<td>20.087769</td>
</tr>
<tr>
<td>0</td>
<td>4.000000</td>
<td>16.000002</td>
<td>54.605194</td>
</tr>
<tr>
<td>0</td>
<td>5.000000</td>
<td>25.000004</td>
<td>148.435364</td>
</tr>
<tr>
<td>0</td>
<td>6.000000</td>
<td>36.000004</td>
<td>403.497711</td>
</tr>
<tr>
<td>0</td>
<td>7.000000</td>
<td>48.999996</td>
<td>1096.842773</td>
</tr>
<tr>
<td>0</td>
<td>8.000000</td>
<td>63.999992</td>
<td>2981.587891</td>
</tr>
<tr>
<td>0</td>
<td>9.000000</td>
<td>80.999992</td>
<td>8104.956543</td>
</tr>
<tr>
<td>0</td>
<td>10.000000</td>
<td>99.999992</td>
<td>22031.992188</td>
</tr>
</tbody>
</table>
NAME : DVERK

PURPOSE : Differential equation solver using the Runge-Kutta-Verner fifth and sixth order method.

CALLING STATEMENT : CALL DVERK (N,FCN,X,Y,XEND,TOL,IND,C,NW,W,IER)

PARAMETERS : N - number of equations

FCN - Name of the subroutine for evaluating functions. The user will provide the subroutine and it must be of the following form:

SUBROUTINE FCN(N,X,Y,YPRIME)
REAL Y(N),YPRIME(N)
...

FCN must be declared as external and it cannot alter N,X,Y(1),...,Y(N).

X - The independent variable.
On calling, X should be set to the initial value.
On return, it should equal XEND unless an error occurs.

Y - The dependent variable in the form of a real or double precision vector of length N. On calling, Y(1),...,Y(N) should be the initial values. On return, Y(1),...,Y(N) is replaced with approximate solutions at XEND unless error conditions arise.

XEND - Real or double precision value of X at which a solution is desired.

TOL - Real or double precision error tolerance.

IND - An integer indicator and error flag.
On calling, it should be set to 0 to have options for array C set to default.
On normal return, it should equal 3.
On abnormal return, it should equal one of the following possible values:

-1 - won't occur when using the default options.
-2 - Problem formulating step sizes, TOL is probably too small.
-3 - Error requirements can not be
fulfilled, TOL is probably too small.

C - A real or double precision communications array of length 24. Its values can be set to default if IND = 1 as suggested above.

NW - An integer that should equal the row dimension of array W. NW should be >= N.

W - A real or double precision array of dimension 2. The first dimension should equal NW. The second dimension should be >= 9.

IER - An integer error flag. Possible values and meanings are:

129 - NW < N or TOL <= 0.
130 - Incorrect value for IND.
131 - XEND not changed since previous call or X not set to previous XEND value.

These are all terminal errors!!!!!!

NOTES : DVERK can also be used to solve a single equation. To do this, N must be set to 1, and vectors Y and YPRIME must be of length one.

i.e.

REAL Y(1)
REAL YPRIME(N)
N = 1
YPRIME(1) = COS(Y(1)) - Y(1)

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

PROGRAM SHOWDVERK
INTEGER N,IND,NW,IER,K
REAL Y(2),C(24),W(2,9),X,TOL,XEND
EXTERNAL FCN1

* Set parameters to initial values. *

NW = 2
N = 2
X = 0.0
Y(1) = 0.0
Y(2) = 1.0
TOL = .0001
IND = 1
* Write out header for graph. *

```
WRITE (5,5)
5    FORMAT (9X,'X',11X, 'Y(1) ',10X, 'Y(2) ')
```

* Loop that carries on the evaluation over the interval. *

```
DO K = 1,10
   XEND = FLOAT(K)
```

* Call DVERK subroutine *

```
CALL DVERK(N,FCN1,X,Y,XEND,TOL,IND,C,NW,W,IER)
```

* If error occurs, halt and output error indicators. *

```
IF ((IND.LT.0).OR.(IER.GT.0)) THEN
   WRITE (5,1) IND,IER
   1    FORMAT(1X, 'ERROR HAS OCCURRED. ', 'IND = ', I1, 'IER = ', I3)
   STOP
```

* Else, write out answers for Y(1) and Y(2). *

```
ELSE
   WRITE (5,2) X,Y(1),Y(2)
   2    FORMAT (1X, 3(F12.6, 2X))
END IF
```

* Subroutine that has the YPRIME equations used to find values for Y(1) and Y(2). *

```
SUBROUTINE FCN1 (N,X,Y, YPRIME)
INTEGER N
REAL Y(N), YPRIME(N), X

YPRIME(1) = 2 * X
YPRIME(2) = Y(2)

RETURN
END
```
<table>
<thead>
<tr>
<th>X</th>
<th>Y(1)</th>
<th>Y(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000000</td>
<td>1.000000</td>
<td>2.718281</td>
</tr>
<tr>
<td>2.000000</td>
<td>4.000000</td>
<td>7.389052</td>
</tr>
<tr>
<td>3.000000</td>
<td>9.000001</td>
<td>20.085522</td>
</tr>
<tr>
<td>4.000000</td>
<td>16.000002</td>
<td>54.598095</td>
</tr>
<tr>
<td>5.000000</td>
<td>25.000002</td>
<td>148.412979</td>
</tr>
<tr>
<td>6.000000</td>
<td>36.000004</td>
<td>403.428223</td>
</tr>
<tr>
<td>7.000000</td>
<td>49.000004</td>
<td>1096.631348</td>
</tr>
<tr>
<td>8.000000</td>
<td>64.000008</td>
<td>2980.952393</td>
</tr>
<tr>
<td>9.000000</td>
<td>81.000008</td>
<td>8103.067383</td>
</tr>
<tr>
<td>10.000000</td>
<td>100.000008</td>
<td>22026.416016</td>
</tr>
</tbody>
</table>
* This is an example of using DVERK with only a
* single variable and a single equation.

PROGRAM SHOWDVERK
INTEGER N,IND,NW,IER,K
REAL Y(1),C(24),W(1,9),X,TOL,XEND
EXTERNAL FCN1

* Set parameters to initial values. *

NW = 1
N = 1
X = 0.0
Y(1) = 1.0
TOL = .0001
IND = 1

* Write out header for graph. *

WRITE (5,5)
5 FORMAT (9X,'X',11X,'Y(1)')

* Loop that carries on the evaluation over the interval. *

DO K = 1,10
    XEND = FLOAT(K)
* Call DVERK subroutine *
    CALL DVERK(N,FCN1,X,Y,XEND,TOL,IND,C,NW,W,IER)
* If error occurs, halt and output error indicators. *
    IF ((IND.LT.0).OR.(IER.GT.0)) THEN
        WRITE (5,1) IND,IER
        1 FORMAT (1X,'ERROR HAS OCCURRED. ',IND = ',',I1,' IER = ',',I3)
        STOP
* Else, write out answers for Y(1) and Y(2). *
    ELSE
        WRITE (5,2) X,Y(1)
        2 FORMAT (1X,2(F12.6,2X))
    END IF
END DO
END

* Subroutine that has the YPRIME equation used to *
* find values for Y(1).
SUBROUTINE FCN1 (N,X,Y,YPRIME)
INTEGER N
REAL Y(N),YPRIME(N),X

YPRIME(1) = Y(1)

RETURN
END

<table>
<thead>
<tr>
<th>X</th>
<th>Y(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000000</td>
<td>2.718281</td>
</tr>
<tr>
<td>2.000000</td>
<td>7.389052</td>
</tr>
<tr>
<td>3.000000</td>
<td>20.085522</td>
</tr>
<tr>
<td>4.000000</td>
<td>54.598095</td>
</tr>
<tr>
<td>5.000000</td>
<td>148.412979</td>
</tr>
<tr>
<td>6.000000</td>
<td>403.428223</td>
</tr>
<tr>
<td>7.000000</td>
<td>1096.631348</td>
</tr>
<tr>
<td>8.000000</td>
<td>2980.952393</td>
</tr>
<tr>
<td>9.000000</td>
<td>8103.067383</td>
</tr>
<tr>
<td>10.000000</td>
<td>22026.416016</td>
</tr>
</tbody>
</table>
NAME : ICSCCU

PURPOSE : Used for cubic spline interpolation.

CALLING STATEMENT : CALL ICSCCU(X,Y,NX,C.IC,IER)

PARAMETERS : X - Real or double precision vector of
length NX containing the abscissae of
the NX data points:
(X(I),Y(I)),I = 1,...NX
On calling, vector X must be sorted
such that X(I) < X(I+1) for all I.

Y - Real or double precision vector of
length NX containing the ordinates
or function values of the NX data
points.

NX - The integer number of elements in X
and Y. NX must be >= 2.

C - The real or double precision spline
coefficients. The values of the spline
approximation at T is
S(T) = C(I,3)*(D**3)+C(I,2)*(D**2)+C(I,1)*D+Y(1)
where X(I) < T < X(I+1) and
D = T-X(I).

IC - The integer row dimension of matrix C
as it is specified in the declaration
of array C in the calling program.

IER - The integer error indicator.
Possible values and meanings are:

129 - IC < NX-1
130 - NX < 2
131 - Array X not sorted in ascending
order.

NOTES : N/A

PROGRAM SHOWICSCCU
IMPLICIT NONE
INTEGER NX,IC,I,IER
REAL X(9),Y(9),C(8,3),S,T,YT

* Set values of the X array *

X(1) = 0.0
X(2) = 0.1
X(3) = .23
X(4) = .34
X(5) = .47
X(6) = .59
X(7) = .73
X(8) = .92
X(9) = 1.0

* Set other parameters *
NX = 9
IC = 8

* Find the appropriate array \( Y \) values *
DO I = 1,NX
  \( Y(I) = X(I) / \exp((-4.0 * X(I) + 1.0)) \)
END DO

* Call ICSCCU subroutine *
CALL ICSCCU(X,Y,NX,C,IC,IER)

* Write out error parameters and answer values *
WRITE (5,1) IER
FORMAT (1X,'IER = ',I3)
WRITE (5,2)
FORMAT ('0','6X','X(I)','10X','Y(I)','10X','C(I,1)',
    8X,'C(I,2)',8X,'C(I,3)')
DO I = 1,9
  WRITE (5,3) X(I),Y(I),C(I,1),C(I,2),C(I,3)
END DO

* As an example of evaluating the cubic spline, *
* consider the following: *

* Write out the graph header *
WRITE (5,4)
FORMAT ('0','7X','T','12X','S(T)','10X','Y(T)')

* Find \( S \) AND \( Y(T) \) values at various \( T \)’s and *
* output \( T \), \( S \) AND \( Y(T) \) *
DO I = 1,8
  T = X(I) + .05
  S = C(I,3) * ((T - X(I)) ** 3) +
      C(I,2) * ((T - X(I)) ** 2) +
      C(I,1) * (T - X(I)) + Y(I)
  YT = T / \exp((-4.0 * T + 1.0))
  WRITE (5,5) T,S,YT
IER = 0

<table>
<thead>
<tr>
<th>X(I)</th>
<th>Y(I)</th>
<th>C(I,1)</th>
<th>C(I,2)</th>
<th>C(I,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000000</td>
<td>0.000000</td>
<td>0.407558</td>
<td>0.774832</td>
<td>6.377077</td>
</tr>
<tr>
<td>0.100000</td>
<td>0.054881</td>
<td>0.753836</td>
<td>2.687954</td>
<td>6.377089</td>
</tr>
<tr>
<td>0.230000</td>
<td>0.212317</td>
<td>1.776023</td>
<td>5.175012</td>
<td>12.798703</td>
</tr>
<tr>
<td>0.340000</td>
<td>0.487332</td>
<td>3.379118</td>
<td>9.398583</td>
<td>21.697502</td>
</tr>
<tr>
<td>0.470000</td>
<td>1.133123</td>
<td>6.922813</td>
<td>17.860615</td>
<td>44.965630</td>
</tr>
<tr>
<td>0.590000</td>
<td>2.298753</td>
<td>13.151874</td>
<td>34.048157</td>
<td>62.660057</td>
</tr>
<tr>
<td>0.730000</td>
<td>4.979300</td>
<td>26.369774</td>
<td>60.365295</td>
<td>182.175781</td>
</tr>
<tr>
<td>0.920000</td>
<td>13.418287</td>
<td>69.038223</td>
<td>164.205582</td>
<td>182.182861</td>
</tr>
<tr>
<td>1.000000</td>
<td>20.085537</td>
<td>0.774832</td>
<td>6.377077</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>S(T)</th>
<th>Y(T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.050000</td>
<td>0.023112</td>
<td>0.022466</td>
</tr>
<tr>
<td>0.150000</td>
<td>0.100090</td>
<td>0.100548</td>
</tr>
<tr>
<td>0.280000</td>
<td>0.315655</td>
<td>0.315699</td>
</tr>
<tr>
<td>0.390000</td>
<td>0.682497</td>
<td>0.682762</td>
</tr>
<tr>
<td>0.520000</td>
<td>1.529536</td>
<td>1.531233</td>
</tr>
<tr>
<td>0.640000</td>
<td>3.049300</td>
<td>3.045645</td>
</tr>
<tr>
<td>0.780000</td>
<td>6.471474</td>
<td>6.498288</td>
</tr>
<tr>
<td>0.970000</td>
<td>17.303486</td>
<td>17.279848</td>
</tr>
</tbody>
</table>
NAME: LEQIF

PURPOSE: Solves a set of linear equations in matrix form (AX = B).

CALLING STATEMENT: CALL LEQIF (A, IA, N, MA, B, IB, M, IJOB,
WK, IER)

PARAMETERS: A - On calling, A is the real or double precision N by N matrix containing the coefficient matrix of the equation AX = B. On return, A is replaced by the LU decomposition of a rowwise permutation of A.

IA - The integer row dimension of A.

N - The integer order of matrix A.

MA - The integer number of columns per block. This number must be less than or equal to N. The bigger the number the faster it runs and the more main memory it ties up at once.

B - On calling, B is the real or double precision N by M matrix of the equation AX = B. On return, B will contain the solution solution matrix X of the equation AX = B.

IB - The integer row dimension of B.

M - Integer number of columns of matrix B.

IJOB - An integer option parameter that has the following values and the corresponding meanings:

0 - Factor the matrix and solve the equation AX = B.
1 - Solve the equation AX = B. This implies that LEQIF has already been called with IJOB equal to 0 and now the user wants a solution for a new matrix B. Matrix A has already been factored, thus it is more efficient to not try to factor matrix A again.

WK - Real or double precision work area of
\* length 3 * N.  
\*  
\* IER - Integer error indicator. Possible  
\* values and meanings are as follows:  
\*  
\* 0 - Everything is OK.  
\* 129 - Indicates that matrix A is  
\* singular.  
\*  
\* NOTES : N/A  
\*  
\* *************************************************************************  
  
PROGRAM SHOWLEQIF  
  
IMPLICIT NONE  
INTEGER N,MA,IA,IB,M,IJOB,IER,I,J  
REAL A(5,5),B(5,2),WK(15)  
  
\* Fill matrix A *  
A(1,1) = 0.0  
A(2,1) = 2.0  
A(3,1) = 1.0  
A(4,1) = 1.0  
A(5,1) = 1.0  
A(1,2) = 1.0  
A(2,2) = 0.5  
A(3,2) = 0.5  
A(4,2) = 0.5  
A(5,2) = 0.5  
A(1,3) = 0.0  
A(2,3) = 0.0  
A(3,3) = 3.0  
A(4,3) = 0.0  
A(5,3) = 0.0  
A(1,4) = 0.0  
A(2,4) = 0.0  
A(3,4) = 0.0  
A(4,4) = 0.0  
A(5,4) = 5.0  
A(1,5) = 0.0  
A(2,5) = 0.0  
A(3,5) = 0.0  
A(4,5) = 4.0  
A(5,5) = 0.0  
  
\* Fill matrix B *  
B(1,1) = 2.0  
B(2,1) = 2.0  
B(3,1) = 2.0  
B(4,1) = 2.0  
B(5,1) = 2.0  
B(1,2) = 3.0
B(2,2) = 3.0
B(3,2) = 3.0
B(4,2) = 3.0
B(5,2) = 3.0

* Fill other parameters *

N = 5
MA = 2
IB = 5
IA = 5
IJOB = 0
M = 2

* Write out initial matrix A and B *

WRITE (5,1) (((A(I,J), J = 1,5), I = 1,5))
1 FORMAT (5X,5(F6.2,2X),/ ,5X,5(F6.2,2X),/ ,1X,'A = ',
+ 5(F6.2,2X),/ ,5X,5(F6.2,2X),/ ,5X,5(F6.2,2X))

WRITE (5,2) (((B(I,J), J = 1,2), I = 1,5))
2 FORMAT ('0',4X,2(F6.2,2X),/ ,5X,2(F6.2,2X),/ ,1X,'B = ',
+ 2(F6.2,2X),/ ,5X,2(F6.2,2X),/ ,5X,2(F6.2,2X))

* Call LEQIF to solve AX = B *

CALL LEQIF (A,IA,N,MA,B,IB,M,IJOB,WK,IER)

* Write out error indicator *

WRITE (5,3) IER
3 FORMAT ('0','IER = ',13)

* Write out ending LU decomposition of a rowwise *
* permutation of A *

WRITE (5,4) (((A(I,J), J = 1,5), I = 1,5))
4 FORMAT ('0',4X,5(F6.2,2X),/ ,1X,'NEW ',5(F6.2,2X),/ ,1X,'A = ',
+ 1X,5(F6.2,2X),/ ,5X,5(F6.2,2X),/ ,5X,
+ 5(F6.2,2X))

* Write out solution matrix X that is stored in B *

WRITE (5,5) (((B(I,J), J = 1,2), I = 1,5))
5 FORMAT ('0',4X,2(F6.2,2X),/ ,5X,2(F6.2,2X),/ ,1X,'X = ',
+ 1X,2(F6.2,2X),/ ,5X,2(F6.2,2X),/ ,5X,
+ 2(F6.2,2X))

END
\begin{align*}
\mathbf{A} &= \begin{bmatrix}
0.00 & 1.00 & 0.00 & 0.00 & 0.00 \\
2.00 & 0.50 & 0.00 & 0.00 & 0.00 \\
1.00 & 0.50 & 3.00 & 0.00 & 0.00 \\
1.00 & 0.50 & 0.00 & 0.00 & 4.00 \\
1.00 & 0.50 & 0.00 & 5.00 & 0.00 \\
\end{bmatrix} \\
\mathbf{B} &= \begin{bmatrix}
2.00 & 3.00 \\
2.00 & 3.00 \\
2.00 & 3.00 \\
2.00 & 3.00 \\
\end{bmatrix} \\
\text{IER} &= 0 \\
\text{NEW} \mathbf{A} &= \begin{bmatrix}
0.00 & 1.00 & 0.00 & 0.00 & 0.00 \\
2.00 & 0.50 & 0.00 & 0.00 & 0.00 \\
1.00 & 0.25 & 3.00 & 0.00 & 0.00 \\
1.00 & 0.25 & 0.00 & 0.00 & 4.00 \\
1.00 & 0.25 & 0.00 & 5.00 & 0.00 \\
\end{bmatrix} \\
\mathbf{X} &= \begin{bmatrix}
0.50 & 0.75 \\
2.00 & 3.00 \\
0.17 & 0.25 \\
0.10 & 0.15 \\
0.13 & 0.19 \\
\end{bmatrix}
\end{align*}
NAME: LINV1F

PURPOSE: Performs the inversion of a matrix.

CALLING STATEMENT: CALL LINV1F (A,N,IA,AINV,IDGT,
WKAREA,IER)

PARAMETERS: A - The real or double precision N by N
matrix to be inverted.
On return, it will be replaced by a
LU decomposition of a row permutation
of A.

N - The integer order of matrix A.

IA - The row dimensional matrix A. It
should be equal to N.

AINV - The real or double precision N by N
matrix which will be the inverse
of A.

IDGT - The integer accuracy test indicator.
If IDGT > 0, elements of A will be
assumed to have accuracy to IDGT
decimal digits and an accuracy test
is performed.
If IDGT = 0, no accuracy test is
performed.

WKAREA - The real or double precision
vector of length N that is used
as the work area of ICSCCU.

IER - The integer error indicator.
Possible value and meanings are as
follows:

0 - Everything is OK.
34 - A warning error that indicates
the accuracy test has failed.
129 - A terminal error that indicates
that A is singular, and thus
has no inverse.

NOTES: N/A

PROGRAM SHOWLINV1F
IMPLICIT NONE
INTEGER N,IA,IDGT,IER,I,J
REAL A(3,3),AINV(3,3),WKAREA(3)
* Set values of matrix to be inverted *

\[
\begin{align*}
A(1,1) &= 33.0 \\
A(2,1) &= -24.0 \\
A(3,1) &= -8.0 \\
A(1,2) &= 16.0 \\
A(2,2) &= -10.0 \\
A(3,2) &= -4.0 \\
A(1,3) &= 72.0 \\
A(2,3) &= -57.0 \\
A(3,3) &= -17.0
\end{align*}
\]

* Write out original array *

\[
\text{WRITE (5,4) ((A(I,J),J=1,3),I=1,3))}
\]

\[
\text{FORMAT ('0',10X,3(F12.6,2X),'/',1X, 'A = ',7X,3(F12.6,2X),'/',11X,3(F12.6,2X))}
\]

* Set other LINV1F parameters *

\[
\begin{align*}
N &= 3 \\
IA &= 3 \\
IDGT &= 0
\end{align*}
\]

* Call the LINV1F subroutine *

\[
\text{CALL LINV1F (A,N,IA,AINV,IDGT,WKAREA,IER)}
\]

* Write out error indicator *

\[
\text{WRITE (5,1) IER}
\]

\[
\text{FORMAT ('0',',IER = ',I3)}
\]

* Write out original array after LINV1F has messed it up *

\[
\text{WRITE (5,2) ((A(I,J),J=1,3),I=1,3))}
\]

\[
\text{FORMAT ('0', 'CHANGED', 3X,3(F12.6,2X),'/',1X, 'A = ',7X,3(F12.6,2X),'/',11X,3(F12.6,2X))}
\]

* Write out the inverse of A *

\[
\text{WRITE (5,3) ((AINV(I,J),J=1,3),I=1,3))}
\]

\[
\text{FORMAT ('0',10X,3(F12.6,2X),'/',1X, 'AINV = ',4X,3(F12.6,2X),'/',11X,3(F12.6,2X))}
\]

END
\[
A = \begin{bmatrix}
33.000000 & 16.000000 & 72.000000 \\
-24.000000 & -10.000000 & -57.000000 \\
-8.000000 & -4.000000 & -17.000000
\end{bmatrix}
\]

\[\text{IER} = 0\]

\[
\text{CHANGED} \\
A = \begin{bmatrix}
-8.000000 & -4.000000 & -17.000000 \\
3.000000 & 2.000000 & -6.000000 \\
-4.125000 & -0.250000 & 0.375000
\end{bmatrix}
\]

\[
\text{AINV} = \begin{bmatrix}
-9.666667 & -2.666667 & -32.000000 \\
8.000000 & 2.500000 & 25.500000 \\
2.666667 & 0.666667 & 9.000000
\end{bmatrix}
\]
**NAME:** VMULFF

**PURPOSE:** Used to multiply a L by M matrix and a M by N matrix.

**CALLING STATEMENT:** CALL VMULFF (A,B,L,M,N,IA,IB,C,IC,IER)

**PARAMETERS:**
- **A** - L by M real or double precision matrix.
- **B** - M by N real or double precision matrix.
- **L** - Integer number of rows in matrix A.
- **M** - Integer number of columns in matrix A. It should be the same as the number of rows in matrix B.
- **N** - Integer number of columns in matrix B.
- **IA** - Integer number of rows in matrix A. It should equal L.
- **IB** - Integer number of rows in matrix B. It should equal M.
- **C** - The L by N real or double precision matrix that is the product of A * B. It is where the answer is placed.
- **IC** - Integer row dimension of matrix C. It should equal L.
- **IER** - The integer error indicator. Possible values and meanings are as follows:
  - 0 - Everything is OK.
  - 129 - A, B, or C was dimensioned incorrectly.

**NOTES:** N/A

---

```fortran
PROGRAM SHOWVMULFF
IMPLICIT NONE
INTEGER L,M,N,IA,IB,IC,IER,I,J
REAL A(4,3),B(3,4),C(4,4)

* Initialize matrix A *

A(1,1) = 1.0
A(2,1) = 4.0
A(3,1) = 7.0
```
A(4,1) = -1.0  
A(1,2) = 2.0  
A(2,2) = 5.0  
A(3,2) = 8.0  
A(4,2) = -2.0  
A(1,3) = 3.0  
A(2,3) = 6.0  
A(3,3) = 0.0  
A(4,3) = -3.0

* Initialize Matrix B *

B(1,1) = 4.0  
B(2,1) = 8.0  
B(3,1) = -3.0  
B(1,2) = 3.0  
B(2,2) = 7.0  
B(3,2) = -2.0  
B(1,3) = 2.0  
B(2,3) = 6.0  
B(3,3) = -1.0  
B(1,4) = 1.0  
B(2,4) = 5.0  
B(3,4) = 0.0

* Initialize other parameters *

L = 4  
M = 3  
N = 4  
IA = 4  
IB = 3  
IC = 4

* Call VMULFF subroutine *

CALL VMULFF (A,B,L,M,N,IA,IB,C,IC,IER)

WRITE (5,1) IER  
1 FORMAT (1X,'IER = ',I13)

* Write out error indicator and matrices A, B, and C *

WRITE (5,2) ( (A(I,J),J=1,3),I=1,4))  
2 FORMAT ('0',10X,3(F12.6,2X),/,'A =',7X,  
+ 3(F12.6,2X),/,'1X,10X,3(F12.6,2X),/,'1X,10X,  
+ 3(F12.6,2X))

WRITE (5,3) ( (B(I,J),J=1,4),I=1,3))  
3 FORMAT ('0',10X,4(F12.6,2X),/,'1X,'B =',7X,  
+ 4(F12.6,2X),/,'1X,10X,4(F12.6,2X))

WRITE (5,4) ( (C(I,J),J=1,4),I=1,4))
IER = 0

A =
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 0.000000
-1.000000 -2.000000 -3.000000

B =
4.000000 3.000000 2.000000 1.000000
8.000000 7.000000 6.000000 5.000000
-3.000000 -2.000000 -1.000000 0.000000

A x B =
11.000000 11.000000 11.000000 11.000000
38.000000 35.000000 32.000000 29.000000
92.000000 77.000000 62.000000 47.000000
NAME : ZANLYT

PURPOSE : Finds the zeros of an analytic complex function using the Muller method with deflation.

CALLING STATEMENT : CALL ZANLYT (F,EPS,NSIG,KN,NGUESS, N,X,ITMAX,INFER,IER)

PARAMETERS :  

F - A complex function subroutine, F(Z), which specifies the equation for which roots are to be found. Be sure to declare F as external.

EPS - The first stopping criteria that is given by FP(Z) = F(Z)/P where P = (Z-Z(1))*(Z-Z(2))*...(Z-Z(K-1)) where Z(I),I = 1,K-1 are previously found roots and the absolute complex value of F(Z) and FP(Z) are less than or equal to EPS. EPS can be real or double precision.

NSIG - The second stopping criteria that if two successive approximations to a root agree to the first NSIG digits the root is accepted. NSIG should be an integer.

KN - The integer number of known roots, which are placed in array X. If none are known, set KN to zero.

NGUESS - The integer number of any guesses at possible roots. If no guesses are made, set NGUESS to zero.

N - The integer number of new roots to be found.

X - A complex vector of length KN + N. On calling, X(1) ... X(KN) hold the known roots and X(KN+1) ... X(KN+NGUESS) hold any guesses. On return, X(KN+1) ... X(KN+N) hold all the approximate roots found.

ITMAX - The maximum integer number of iterations per root.
INFER - An integer vector of length KN + N.
INFER(J), on return, will contain the
number of iterations used to find the
Jth root. If convergence was not
obtained, INFER(J) will be greater
than ITMAX.

IER - The integer error indicator.
Possible values and meanings are:

0 - everything is OK.
33 - failure to converge within
     ITMAX iterations for at least
     one of the N new roots.

NOTES : N/A

* *
*....................................................................................
*
* PROGRAM SHOWZANLYT
* IMPLICIT NONE
* EXTERNAL F
* INTEGER INFER(3),I,EPS,NSIG,KN,NGUESS,N,ITMAX,IER
* COMPLEX F,X(3)
* *
* Fill parameters with the proper values *
* EPS = .0001
* NSIG = 4
* KN = 0
* NGUESS = 0
* N = 3
* ITMAX = 100
* *
* Call subroutine ZANLYT *
* CALL ZANLYT (F,EPS,NSIG,KN,NGUESS,N,X,ITMAX,INFER,IER)
* *
* Write out the error parameter *
* WRITE (5,1) IER
1  FORMAT (1X,'IER = ',I3)
* Write out the graph header *
* WRITE (5,2)
2  FORMAT ('0',4X,'REAL Z(I)',5X,'IMAG Z(I)',4X,'INFER(I)')
* Write out graph of real and imaginary part of complex roots and *
* number of iterations ZANLYT took to find them.
* DO I = 1,3
*  WRITE (5,3) X(I),INFER(I)
3  FORMAT (1X,F12.6,2X,F12.6,6X,I4)
* END DO
* Complex function F that contains the equation for * 
* which roots are to be found.                      *

COMPLEX FUNCTION F(Z)                              
IMPLICIT NONE                                        
COMPLEX Z                                             

F = (Z**3) - 1                                       

RETURN                                               
END

IER = 0

<table>
<thead>
<tr>
<th>REAL Z(I)</th>
<th>IMAG Z(I)</th>
<th>INFER(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000000</td>
<td>0.000000</td>
<td>13</td>
</tr>
<tr>
<td>-0.500000</td>
<td>0.866025</td>
<td>5</td>
</tr>
<tr>
<td>-0.500000</td>
<td>-0.866025</td>
<td>5</td>
</tr>
</tbody>
</table>
NAME: ZRPOLY

PURPOSE: Find the zeros of a polynomial with real coefficients.

CALLING STATEMENT: CALL ZRPOLY (A,NDEG,Z,IER)

PARAMETERS: A - Real or double precision vector of length NDEG + 1 containing the coefficients in order of decreasing powers of variables.

NDEG - An integer which is the degree of the polynomial for which the zeros are being found. It must be greater than zero and less than 101.

Z - The complex vector of length NDEG containing the computed roots of the polynomial.

IER - The error indicator that has the following possible values and meanings.

129 - The degree of the polynomial is greater than 100 or less than 1.
130 - The leading coefficient is zero.
131 - Less than NDEG roots have been found. If only K roots have been found, Z(J), J = M+1, ..., NDEG are set to positive machine infinity.

NOTES: N/A

PROGRAM SHOWZRPOLY
IMPLICIT NONE
INTEGER NDEG,IER,I
REAL A(4)
COMPLEX Z(3)

Degree of polynomial for which roots are to be found *
NDEG = 3

Polynomial for which roots are to be found *

A(1) = 1.0
A(2) = 0.0
A(3) = 0.0
A(4) = -1.0

Call ZRPOLY to find roots *
CALL ZRPOLY (A,NDEG,Z,IER)

* Write out error indicator, graph header, and roots *

WRITE (5,1) IER
1 FORMAT (1X,'IER = ',I3)

WRITE (5,2)
2 FORMAT ('0','THE REAL AND IMAGINARY PART ',
+ 'OF EACH ROOT ARE: ')

WRITE (5,3)
3 FORMAT ('0',6X,'REAL',10X,'IMAG')

WRITE (5,4) Z
4 FORMAT (3(IX,FI2.6,2X,FI2.6,/))

END

IER = 0

THE REAL AND IMAGINARY PART OF EACH ROOT ARE:

<table>
<thead>
<tr>
<th>REAL</th>
<th>IMAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>-0.500000</td>
<td>0.866025</td>
</tr>
<tr>
<td>-0.500000</td>
<td>-0.866025</td>
</tr>
</tbody>
</table>
The IMSL MATH/PROTRAN language is a language specially designed for mathematical applications. It is easy to understand and use.

To run a MATH/PROTRAN program one does the following:

$ PROTRAN programname.PRO

This will convert your program to an equivalent FORTRAN program, link it, and run it. It will also create several files in your area. These are used for data files, subroutines, and output. The only file needed is the one called programname.PRE. This file will contain a copy of your original program, any warnings and/or error messages caused by the compilation and running of your program, and any output produced.

The following are examples of some of the MATH/PROTRAN functions. The functions that are explained are:

1) DERIV
2) DIFEQU
3) INTEGRAL
4) INTERPOLATE
5) LINSYS
6) NONLIN
7) POLYNOMIAL

Some of the spaces between fields in the output of the examples have been removed to allow it be easily readable.

If anything is not clear or not presented, check the IMSL MATH/PROTRAN manual in RB134.
**NAME:** DERIV  

**PURPOSE:** Calculates the derivative of a function. Two formats are available. In the first, the function is given, while in the second it is a function defined by a set of data points.

**FIRST FORM -**

**CALLING STATEMENT:**  

$ \text{DERIVATIVE F} \left( D(X) \right) $  

$ [\text{ORDER} = N] $  

$ [\text{NOUTPUT} = \text{NO}] $  

$ [\text{ERRTARGET} = \text{ERR}] $  

$ [\text{STEPSIZE} = H] $  

$ \text{AT U IS V} $  

**PARAMETERS:**  

$ F $ - The real or double precision expression or scalar evaluated in a DEFINE segment to be differentiated.  

$ X $ - The real or double precision scalar for the expression to be differentiated with respect to.  

$ U $ - The real or double precision vector or expression giving the abscissa(e) at which the derivative is to be evaluated.  

$ V $ - The real or double precision vector or scalar which will contain the value(s) of the derivative at abscissa(e) specified at in U.  

$ \text{NO} $ - The integer number of output points. The default is the number of elements in U if U is a vector, or 1 if U is a scalar. In almost all cases, it is sufficient to let it default.  

$ N $ - The integer order of the derivative to be calculated. It can be either 1 or 2. The default is 1.  

$ H $ - The real or double precision initial stepsize for the finite difference approximation. It will be decreased until the derivative estimate stabilizes. The default is 1.0. It is best to let this default and let the computer do the work.  

$ \text{ERR} $ - The real or double precision error
toleration. The min (relative error, absolute error) will be less than ERR. The default is 0.01.

SECOND FORM -

CALLING STATEMENT: $ DERIVATIVE F

ORDER = N]
[NOOUTPUT = NO]
[NOUTPUTS = NP]
AT U
IS V

PARAMETERS: F - The real or double precision vector of ordinates of the data on the curve to be differentiated.

X - The real or double precision vector of abscissae of the data on the curve to be differentiated.

U - The same as in format 1.

V - The same as in format 1.

N - The same as in format 1.

NO - The same as in format 1.

NP - The integer number of data points. By default it is the number of elements in X.

NOTES: To get double precision just change all real variables to double precision.

* This is an example of format 1. *
$ DECLARATIONS
REAL VECTOR U(4), V(4)
REAL SCALAR T

* Set U = (.5, 1.0, 1.5, 2.0) *
$ ASSIGN U(I) = .5 * I

$ DERIVATIVE (T)**3
D(T)
ERRTARGET = 0.0001
AT U
IS V

$ PRINT U, V
$ END
* This is an example of format 2. *

$ DECLARATION$
REAL VECTOR X(10), Y(10), U(3), V(3)
$ ASSIGN U = (.3, .6, .9)

* Set X = (.1, .2, .3, .4, .5, .6, .7, .8, .9, 1) *

$ ASSIGN X(I) = 0.1 * I$
$ ASSIGN Y(I) = X(I) ** 3$

$ DERIVATIVE Y$
VS X
ORDER 2
AT U
IS V
$ PRINT V$
$ END$

V

1.80000 3.60001 5.39999
* This is an example of format 1 using a DEFINE for the function. *

$$\begin{align*}
\text{DECLARATIONS} \\
\text{REAL SCALAR T, FF, V} \\
\text{DERIVATIVE FF} \\
\text{D(T)} \\
\text{DEFINE} \\
\text{====} \\
\text{FF = 1.0} \\
\text{IF (T.NE.0.0) FF = (EXP(T) - 1.0)/T} \\
\text{====} \\
\text{AT 0.0} \\
\text{IS V} \\
\text{PRINT V} \\
\text{END} \\
\end{align*}$$

V

0.50000
NAME : DIFEQU

PURPOSE : Solves a system of ordinary differential equations with initial values given.

CALLING STATEMENT : $ DIFEQU Y' = F(X,Y) \\
ON (XA,XB) \\
INITIAL = YA \\
(EQUATIONS = N] \\
(ERRTARGET = ERR) \\
(Absolute] \\
(NOUTPUT = NOJ \\
DEFINE \\
==== \\
(F DEFINED IN FORTRAN) \\
==== \\
SOLUTION = Z \\

PARAMETERS : X - The real or double precision independent scalar as used in the DEFINE segment.

Y - The real or double precision dependent scalar or vector as used in the define segment.

F - The real or double precision function scalar or vector as used in the define segment.

XA - The real or double precision initial value of the independent scalar.

XB - The real or double precision final value of the independent scalar.

YA - The real or double precision initial expression or vector.

N - The integer number of differential equations. The default is the range of YA, if YA is a vector, and 1 otherwise.

ERR - The real or double precision error expression. The default is .001.

ABSOLUTE - This is really a keyword rather than a parameter. If it is specified, the program attempts to keep the absolute error in solution component in proportion to ERR. The default is the min(relative error, absolute error) in each solution component is kept in proportion to ERR.
NO - The integer number of values of the
independent variable between XA and
XB at which the solution is to be
calculated and either printed, stored,
or saved for interpolation. The
default is 100.

Z - May be:

1) If F, Y, and YA are vectors then
   Z is a matrix of dimension NO+1
   by N. Z(i,j) will then hold the
   Jth solution at the point
   XA+(I-1)/NO*(XB-XA).

2) If F and Y are scalars and YA an
   expression, Z is a vector of length
   NO+1. Z(I) will then hold the
   value of the solution at the point
   XA+(I-1)/NO*(XB-XA).

3) The name of two argument FORTRAN
   subroutine that PROTRAN generates to
   evaluate the solution. Z(X,J) will
   return the value of the Jth solution
   at X, interpolated between points
   where the solution was stored, if
   necessary. Z should be declared as
   a real or double precision scalar,
   matching the type of X.

4) If N=1, it can be the name of a
   PROTRAN generated one argument
   FORTRAN subroutine to evaluate the
   solution. Z(X) will return the value
   of the solution at X. Z should be
   declared as a real or double
   precision scalar, matching the type
   of X.

* NOTES : N/A
*

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

$ DECLARATIONS
REAL SCALAR XX
REAL VECTOR YY(2),FF(2),YAA(2)
REAL MATRIX ZSOL(11,2)
$ ASSIGN YAA = (0.0,1.0)
$ DIFEQU YY' = FF(XX,YY)
DEFINE
====
   FF(1) = 2.0 * XX
   FF(2) = YY(2)
ON (0.0,1.0)
INITIAL = YAA
NOUTPUT = 10
SOLUTION = ZSOL

$ PRINT ZSOL
$ END

<table>
<thead>
<tr>
<th>ZSOL</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00000</td>
<td>1.00000</td>
</tr>
<tr>
<td>2</td>
<td>0.01000</td>
<td>1.10517</td>
</tr>
<tr>
<td>3</td>
<td>0.04000</td>
<td>1.22140</td>
</tr>
<tr>
<td>4</td>
<td>0.09000</td>
<td>1.34986</td>
</tr>
<tr>
<td>5</td>
<td>0.16000</td>
<td>1.49182</td>
</tr>
<tr>
<td>6</td>
<td>0.25000</td>
<td>1.64872</td>
</tr>
<tr>
<td>7</td>
<td>0.36000</td>
<td>1.82212</td>
</tr>
<tr>
<td>8</td>
<td>0.49000</td>
<td>2.01375</td>
</tr>
<tr>
<td>9</td>
<td>0.64000</td>
<td>2.22554</td>
</tr>
<tr>
<td>10</td>
<td>0.81000</td>
<td>2.45960</td>
</tr>
<tr>
<td>11</td>
<td>1.00000</td>
<td>2.71828</td>
</tr>
</tbody>
</table>
This is an example of the solution of a single equation output in the form of a FORTRAN function subroutine, which is plotted by P.

```
DECLARATIONS
REAL SCALAR XX, FF, YY, S, ZSOL

DIFEQU YY' = FF(XX, YY)

DEFINE
  FF = (EXP(XX) + YY)/2.0

ON (0.0, 2.0)
INITIAL = 1.0
NOUTPUT = 10
SOLUTION = ZSOL

PLOT ZSOL(S)
FOR (S=0, 2)
END
```
NAME: INTEGRAL

PURPOSE: It calculates the integral of a function. Two formats are available. In the first, the integrand is an expression. In the second, the integrand is a function defined by a set of data points.

FIRST FORM -

CALLING STATEMENT: $ INTEGRAL F FOR (X=A,B) [IS Q] [ERRTARGET = ERR] [ABSOLUTE] [DEFINE] ====[Scalar F defined in FORTRAN ===] PARAMETERS: F - The real or double precision expression or scalar to be integrated.

X - The real or double precision scalar that F is being integrated with respect to.

A - The real or double precision expression that gives the beginning point that at X=A, integration of F is to begin.

B - The real or double precision expression that at X=B, integration is to stop.

Q - The real or double precision scalar that will hold the result of the integration of F. The default is that the result is printed.

ERR - The error control parameter. The default is .001. See ABSOLUTE for further explanation.

ABSOLUTE - This is a keyword that, if given, signifies that the absolute error in the integral must be less than ERR. The default is that the relative error is less than ERR.

DEFINE - In between the two sets of ====[, F can be defined in FORTRAN statements. This is useful if the definition of F takes more than one line.
SECOND FORM -

CALLING STATEMENT - $ INTEGRAL F
   VS X
   [ON (A,B)]
   [NPOINTS = NP]
   [IS Q]

PARAMETERS - F - The real or double precision vector
   That contains the ordinates of the data
   on the curve to be integrated.

X - The real or double precision vector of
   abscissae of the data on the curve to
   be integrated.

A - The same as in format 1.

B - The same as in format 1.

NP - The integer expression that gives the
   number data points. The default is the
   range of X.

Q - The same as in format 1.

NOTES : Format 2 uses a cubic spline form of integration.

This is an example of format 1. *

DECLARATIONS
REAL SCALAR ZZ,ANSWER
INTEGRAL ZZ ** 2.0
FOR (ZZ = 0.0,2.0)
   IS ANSWER
PRINT ANSWER
END

ANSWER

2.66667
* This is an example of format 2. *

\$ DECLARATIONS
REAL VECTOR X(5), Y(5)
REAL SCALAR ANS

\$ ASSIGN X = (0.0, 1.0, 2.0, 3.0, 4.0)
\$ ASSIGN Y = (0.0, 1.0, 4.0, 9.0, 16.0)

\$ INTEGRAL Y
VS X
ON (0.0, 2.0)
IS ANS
\$ PRINT ANS
\$ END

ANS

2.66667

* This is an example of format 1 using a DEFINE for the function. *

\$ DECLARATIONS
DOUBLE PRECISION SCALAR ZZ, FF
\$ INTEGRAL FF
FOR (ZZ = 0.0, 2.0)
DEFINE
=====
    IF (ZZ.LE.1.0) FF = 1.5 * ZZ
    IF (ZZ.GT.1.0) FF = ZZ + 0.75
=====
ERRTARGET = .000000001
ABSOLUTE
\$ END

INTEGRAL =

0.299999999534339D+01
**NAME**: INTERPOLATE

**PURPOSE**: Interpolates to a set of data points using polynomials, piecewise polynomials, or other user specified functions to fit the data. It has two forms.

**FIRST FORM** -

**CALLING STATEMENT**: INTERPOLATE Y VS X BY G

[OPTIONAL]

[NPOINTS = NP]

[NOUTPUT = NO]

[USING INTERPOLANT]

[COEFFICIENTS = C]

**PARAMETERS**: Y - The real or double precision vector of ordinates of the data points.

X - The real or double precision vector of the abscissae of the data points.

G - The name to be given the resulting single argument FORTRAN function subroutine that PROTRAN produces. G should be declared as a single or double precision scalar. The argument of G should be declared as the same type.

NP - An integer expression that is the number of data points. The default is the range of X.

NO - An integer expression that is the number of output points. The default is the range of U.

INTERPOLANT - This is a keyword that signals the type of functions to interpolate the data. It can have the following four possible values (SPLINES is the default):

1) SPLINES - Cubic splines are to be used. Gives high accuracy for smooth data.

2) HERMITES - Quasi-Hermite piecewise polynomials are to be used. Have lower accuracy but may provide better curve fits, especially
for data that is not smooth.

3) \( F(K,X), K, X \) - A linear combination of basis functions of the form \( F(K,X), K=1, \ldots, NP \) is to be used. \( F \) is a real or double precision expression. \( X \) should be a scalar of the same type as \( F \). \( K \) should be an integer scalar. FORTRAN statements embedded in a DEFINE can also be used to define \( F \).

4) POLYNOMIALS - This has the same effect as specifying \( F(K,X), = X^{(K-1)}, K=1, \ldots, NP \), except numerical instability is diminished by using orthogonal polynomials internally.

\[ C \] - A real or double precision vector that on return will contain the coefficients of the interpolatory function. \( C(K) \) is the coefficient of \( F(K,X) \), or of \( X^{(K-1)} \) if the USING POLYNOMIALS is specified. It cannot be given if USING SPLINES (the default) or USING HERMITES is given.

SECOND FORM -

CALLING STATEMENT:
\[
\text{INTERPOLATE Y VS X AT U IS V [INPOINTS = NP] [NOUTPUT = NO] [USING INTERPOLANT] [COEFFICIENTS = C]}
\]

PARAMETERS:
\( Y \) - The same as in the first form.
\( X \) - The same as in the first form.
\( U \) - The real or double precision vector containing a set of abscissae at which the values of interpolatory function are to be evaluated.
V - The real or double precision vector that, on return, will contain the values of the interpolatory function at the abscissae specified in U.

NP - The same as in the first form.

NO - The same as in the first form.

INTERPOLANT - The same as in the first form.

C - The same as in the first form.

NOTES: N/A

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

* This is an example of format 1. *

$ DECLARATIONS
REAL VECTOR C(5),X(5),Y(5)
REAL SCALAR GG,X5
$ ASSIGN X(I) = I
$ ASSIGN Y(I) = X(I) ** 2
$ INTERPOLATE Y VS X
USING POLYNOMIALS
BY GG
COEFFICIENTS = C
$ ASSIGN X5 = GG(1.5)
$ PRINT C,X5
$ END

C
0.00000 0.00000 1.00000 0.00000 0.00000

X5
2.25000
* This is an example of format 2. *

```plaintext
DECLARATIONS
REAL VECTOR X(5), Y(5), U(4), V(4)
ASSIGN X = (1.0, 2.0, 3.0, 4.0, 5.0)
ASSIGN Y(I) = X(I) ** 2
ASSIGN U = (1.5, 2.5, 3.5, 4.5)
INTERPOLATE Y
VS X
AT U
IS V
PRINT U, V
END

U
1.50000 2.50000 3.50000 4.50000
V
2.25000 6.25000 12.25000 20.25000
```

* This is an example of format 1. *

```plaintext
DECLARATIONS
REAL VECTOR C(5), X(5), Y(5)
REAL SCALAR GG, X5
ASSIGN X(I) = I
ASSIGN Y(I) = EXP(X(I))
INTERPOLATE Y
VS X
USING POLYNOMIALS
BY GG
COEFFICIENTS = C
ASSIGN X5 = GG(1.0)
PRINT C, X5
END

C
15.97864 -31.45170 24.77888
-7.57487 0.98733

X5
2.71828
```
* This is an example of format 2. *

\$ DECLARATIONS
REAL VECTOR X(5), Y(5), U(4), V(4)
\$ ASSIGN X = (1.0, 2.0, 3.0, 4.0, 5.0)
\$ ASSIGN Y(I) = EXP(X(I))
\$ ASSIGN U = (1.5, 2.5, 3.5, 4.5)
\$ INTERPOLATE Y
VS X
AT U
IS V
\$ PRINT U,V
\$ END

<table>
<thead>
<tr>
<th>U</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.50000</td>
<td>2.50000</td>
<td>3.50000</td>
<td>4.50000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.54211</td>
<td>12.24243</td>
<td>32.64219</td>
<td>91.37971</td>
<td></td>
</tr>
</tbody>
</table>
**NAME : LINSYS**

**PURPOSE :** Used to solve a system of linear equations.

**CALLING STATEMENT :**

```fortran
LINSYS A \* X = B
[HIGHACCURACY]
[NOSAVE]
[EQUATIONS = N]
[RHS = NRHS]
[POSDEF]
```

**PARAMETERS :**

- **A** - The coefficient matrix of the linear system. It can be real or double precision. It can also be complex. Sample declarations are as follows:

```fortran
REAL MATRIX X
DOUBLE PRECISION MATRIX X
COMPLEX X
DOUBLE COMPLEX MATRIX X
```

- **X** - On return, X will contain the solution of the linear system. It can either be a vector or a matrix, real or double precision. It can also be complex.

- **B** - On return, B will contain the right hand side of the linear system.

- **HIGHACCURACY** - Indicates a high accuracy solution is needed. It will use more CPU time and memory than without it being specified.

- **NOSAVE** - Indicates that matrix A should be used as workspace to save on memory usage. By default, matrix A is left unchanged.

- **N** - The integer expression of the number of equations in the linear system. The default is the row range of B.

- **NRHS** - The integer expression equal to the number of columns of B. The default is the column range of B.

- **POSDEF** - Indicates that the coefficient matrix A is positive definite. The default is that A is not positive definite.
* NOTES : A, X, and B must be of the same type.

DECLARATIONS
REAL MATRIX A(3,3)
REAL VECTOR X(3), B(3)
ASSIGN A = (1.0, 0.0, 0.0)  
       + (0.0, -2.0, 0.0)  
       + (0.0, -2.0, 1.0)  
    B = (1.0, 1.0, 1.0)  
LINSYS A * X = B
HIGHACCURACY
PRINT X
END

X

1.00000  -0.50000  0.00000
NAME: NONLIN

PURPOSE: Solves a system of nonlinear equations. Two forms are available.

FIRST FORM -

CALLING STATEMENT: $ NONLIN F(X) = 0
DEFINE
    ===
      (F defined in FORTRAN)
    ===
      GUESS = GX
      [SOLUTION = XSOL]
      [ERRTARGET = ERR]
      [EQUATIONS = N]

PARAMETERS:
  F - The real or double precision vector as used in the define statement.
  X - The real or double precision independent vector.
  GX - The real or double precision input vector containing an initial guess of the root.
  XSOL - The real or double precision vector that, on return, will hold the solution. The default is that it is printed.
  ERR - The real or double precision expression such that the min(relative error, absolute error) in each parameter will be less than ERR. The default is .001.
  N - The integer expression that is the number of nonlinear equations and the number of unknowns. The default is the range of GX.

SECOND FORM - Used only when a single nonlinear equation is solved.

CALLING STATEMENT: $ NONLIN F(X) = 0
DEFINE
    ===
      (F defined in FORTRAN)
    ===
      IN (XA,XB)
      [SOLUTION = XSOL]
      [ERRTARGET = ERR]
      [EQUATIONS = N]
PARAMETERS:

F - The real or double precision scalar as used in the define statement.

X - The real or double precision independent scalar.

XA and XB - The real or double precision expressions such that somewhere between XA and XB, F(X) changes sign. Thus, there is a real zero for F(X) in that interval.

XSOL - The real or double precision scalar that, on return, will hold the solution. The default is that it is printed.

ERR - The real or double precision expression such that the min(relative error, absolute error) in each parameter will be less than ERR. The default is .001.

N - The integer expression that is the number of nonlinear equations and the number of unknowns. The default is 1.

NOTES:

N/A

--------------------------------------------------------------

This is an example of format 1.

$ DECLARATIONS
REAL VECTOR F(3),X(3),GX(3),XSOL(3)
$ ASSIGN GX = (-1.0,1.0,2.0)
$ NONLIN F(X) = 0
GUESS = GX
SOLUTION = XSOL
DEFINE
===
F(1) = TAN(X(1)) + X(2)**2 + X(3)**3
F(2) = COS(X(1)) - (2 * X(2))
F(3) = (X(1) / X(2)) - SIN(X(3)) + 2.5
===
$ PRINT XSOL
$ END

XSOL

-0.67710  0.38970  0.86715
$\text{DECLARATIONS}$
$\text{REAL SCALAR F,X}$
$\text{NONLIN F(X) = 0}$
$\text{DEFINE}$
$\text{====}$
$\quad F = \cos(X) + (X/3.0) - .7$
$\text{====}$
$\text{IN (0.0,3.14)}$
$\text{ERRTARGET = .000001}$
$\text{END}$

NONLIN SOLUTION =

\[0.1301320E+01\]
NAME: POLYNOMIAL

PURPOSE: Calculates the roots of a polynomial.

CALLING STATEMENT:

POLYNOMIAL A

[ROOTS = X]

[DEGREE = N]

PARAMETERS:

A - The real or double precision vector of coefficients of the polynomial, beginning with the coefficients of the lowest order. The vector can also be complex or double complex.

X - The complex or double complex vector to hold the roots of the polynomial. The default is that the roots are printed.

N - The integer expression that is the degree of the polynomial. The default is the range of A minus 1.

NOTES: N/A

DEKLARATIONS

REAL VECTOR A(4)

ASSIGN A = (1.0, 0.0, 0.0, 1.0)

POLYNOMIAL A

END

ROOTS =

( 0.5000000E+00, 0.8660254E+00)

( 0.5000000E+00, -0.8660254E+00)

(-0.1000000E+01, 0.0000000E+00)