1993

Comparison of IMSL/IDL with the IMSL Math Library and Exponet Graphics

Xingkang Fu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
93-025

COMPARISON OF IMSL/IDL WITH THE IMSL MATH LIBRARY AND EXPONENT GRAPHICS

Xingkang Fu
John R. Rice

CSD-TR-93-025
April 1993

COMPARISON OF IMSL/IDL WITH THE IMSL MATH LIBRARY AND EXPONENT GRAPHICS

Xingkang Fu[1] and John R. Rice
Computer Science Department
Purdue University
West Lafayette, IN 47907
Technical Report CSD-TR-93-025
CAPO Report CER-93-15
April 1993

Abstract

This report makes a comparison of an interaction graphics system (IDL) with a graphics library (Exponent Graphics). The objective is evaluate the ease of use and flexibility of the two approaches for typical "small" applications. Nine applications are used. In addition, the applications are programmed in ordinary Fortran and, for two applications, in ELLPACK. The principal conclusion in IDL is easier to use (has shorter codes) and is more versatile. In only four of the applications could Exponent Graphics produce output comparable to that of IDL with reasonable effort.

[1]Work supported by IMSL research grant.

COMPARISON OF IMSL/IDL WITH THE IMSL MATH LIBRARY AND EXPONENT GRAPHICS

Xingkang Fu and John R. Rice
Department of Computer Science
Purdue University
April 1, 1993

The approach here is to take a number of simple applications and program them both in IMSL/IDL and in Fortran using the IMSL Math Library and Exponent Graphics. One then compares the results and draws conclusions about the computation advantage of the two problem solving approaches. Two of the applications use program written in ELLPACK also.

The applications included in this study are as follows: Page

A simple comparison method is the length of the code used for the application. The following table lists the lines of executable code for the applications. The entry "Fortran+Exponent" means that the IMSL Math Library is used along with the Exponent Graphics. If no entry is given, then the Exponent Graphics could not be used directly to provide the graphical output similar to that of IMSL/IDL. The entry "Fortran" means that IMSL Math Library is used but no graphical output is generated.

Table 1. Comparison of executable statements count for solutions with IMSL/IDL, Math Library plus Exponent Graphics(Fortran+Exponent), Math Library without graphics(Fortran), and the ELLPACK system(ELLPACK)

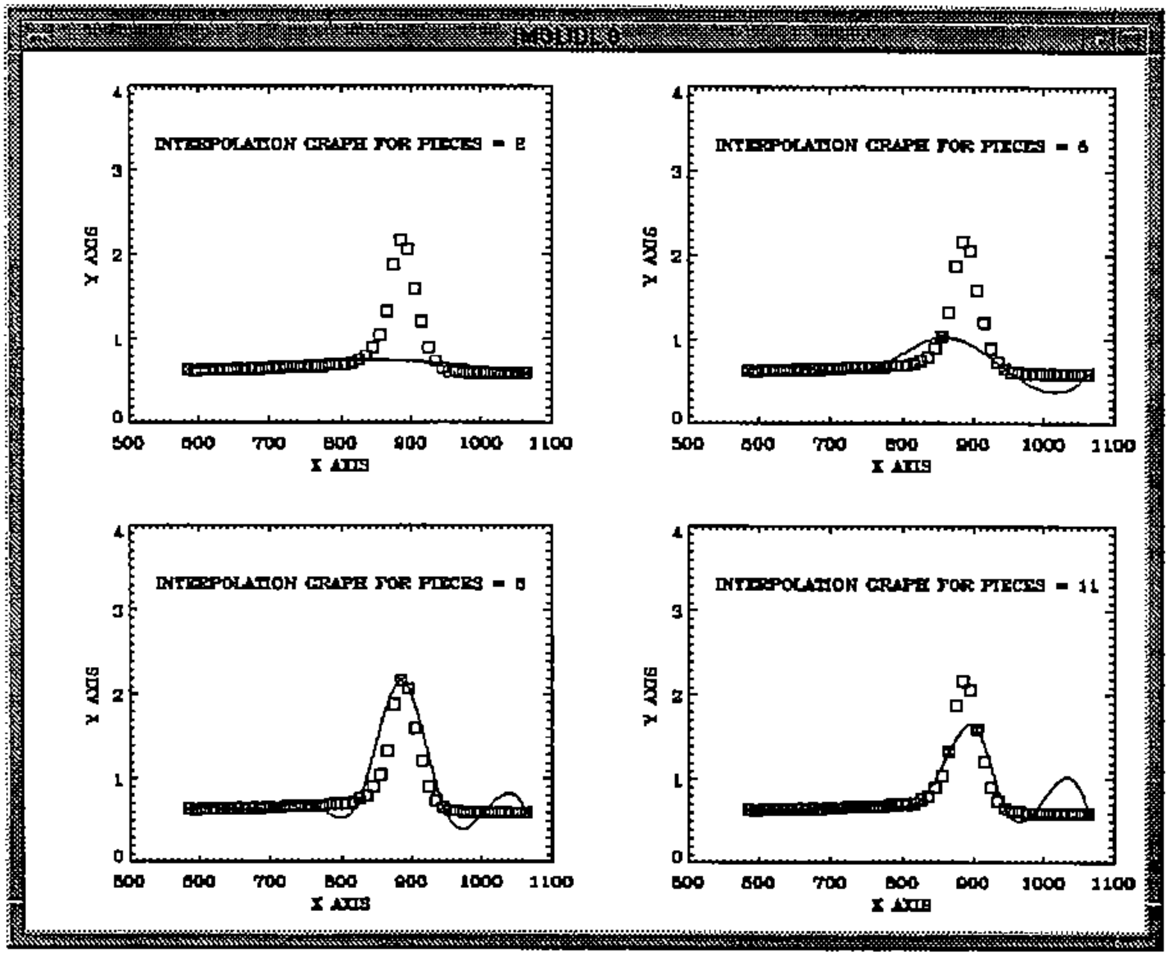
Application	IMSL/IDL	Fortran+Exponent	Fortran	ELLPACK
5.3	35	58	38	xx
5.5	49	92	82	xx
7.2	50	xx	60	xx
7.3	51	xx	75	xx
8.8	18	30	27	xx
9.2	38	xx	58	xx
animation	33	xx	xx	xx
10.1	52	90	79	16
10.2	75	xx	100	54

CUBIC SPLINE INTERPOLATION OF TITANIUM BY CURVES OF 2, 5, 8, 11 PIECES

IMSL/IDL program:

```
;       LIBRARY APPLICATION 5.3
;
;       CUBIC SPLINE INTERPOLATION OF TITANIUM BY CURVES OF
;       2, 5, 8, 11 PIECES.
;
;       THE GIVEN TITANIUM DATA IS WELL KNOWN AS PHYSICAL DATA WHICH IS
;       DIFFICULT TO REPRESENT WELL BY A MATHEMATICAL MODEL. THE INTERPOLATION
;       POINTS ARE MORE OR LESS EQUALLY SPACED BETWEEN 585 AND 1085.
;       XPTS   - INPUT ABSCISSAE
;       TDAT   - INPUT ORDINATES
;       XDATA  - DATA POINTS ABSCISSAE AS BREAKPOINTS
;       FDATA  - THE ORDINATE VALUES AT THE ABOVE ABSCISSAE
;       N - THE NUMBER OF INPUT POINTS ( 1 + NUMBER OF PIECES )
;         = 3,6,9,12

pro libappl53
;
;   INITIALIZATION
;
    xpts = findgen(49)*10 + 585.0
    tdat = fltarr(49)
    openr, lun, '5.3.tdat.dat', /get_lun
    readf, lun, tdat
    free_lun, lun
;
;   MAKE FOUR PLOTS ON THE DISPLAY
    !p.multi = [0,2,2,0,0]
;
    loadct,12
    for n = 3, 12, 3 do begin
        xdata = fltarr(n)
        fdata = fltarr(n)
        xdata(0) = xpts(0)
        fdata(0) = tdat(0)
        xdata(n-1) = xpts(48)
        fdata(n-1) = tdat(48)
        dm = 49/(n-1)
        for i = 1, n-2 do begin
            j = fix(dm*i)
            xdata(i) = xpts(j)
            fdata(i) = tdat(j)
        endfor
;
;   CUBIC SPLINE INTERPOLATION
        pp = csinterp(xdata, fdata)
;
```

```
;   GET VALUE AT 193 POINTS
        ppval = spvalue(findgen(193)*2.5+585.0,pp)
;
        plot, findgen(193)*2.5+585.0, ppval, yrange = [0,4], $
      xtitle = 'X AXIS', ytitle = 'Y AXIS', color = 60
        oplot, xpts, tdat, psym = 6, color = 112
        oplot, xdata, fdata, psym = 7, color = 200
        case n of
             3:     xyouts,540,3.25,'!6INTERPOLATION GRAPH FOR PIECES = 2', $
                    /data
             6:     xyouts,540,3.25,'!6INTERPOLATION GRAPH FOR PIECES = 5', $
                    /data
             9:     xyouts,540,3.25,'!6INTERPOLATION GRAPH FOR PIECES = 8', $
                    /data
             else:  xyouts,540,3.25,'!6INTERPOLATION GRAPH FOR PIECES = 11', $
                    /data
        endcase
    endfor
    end
```

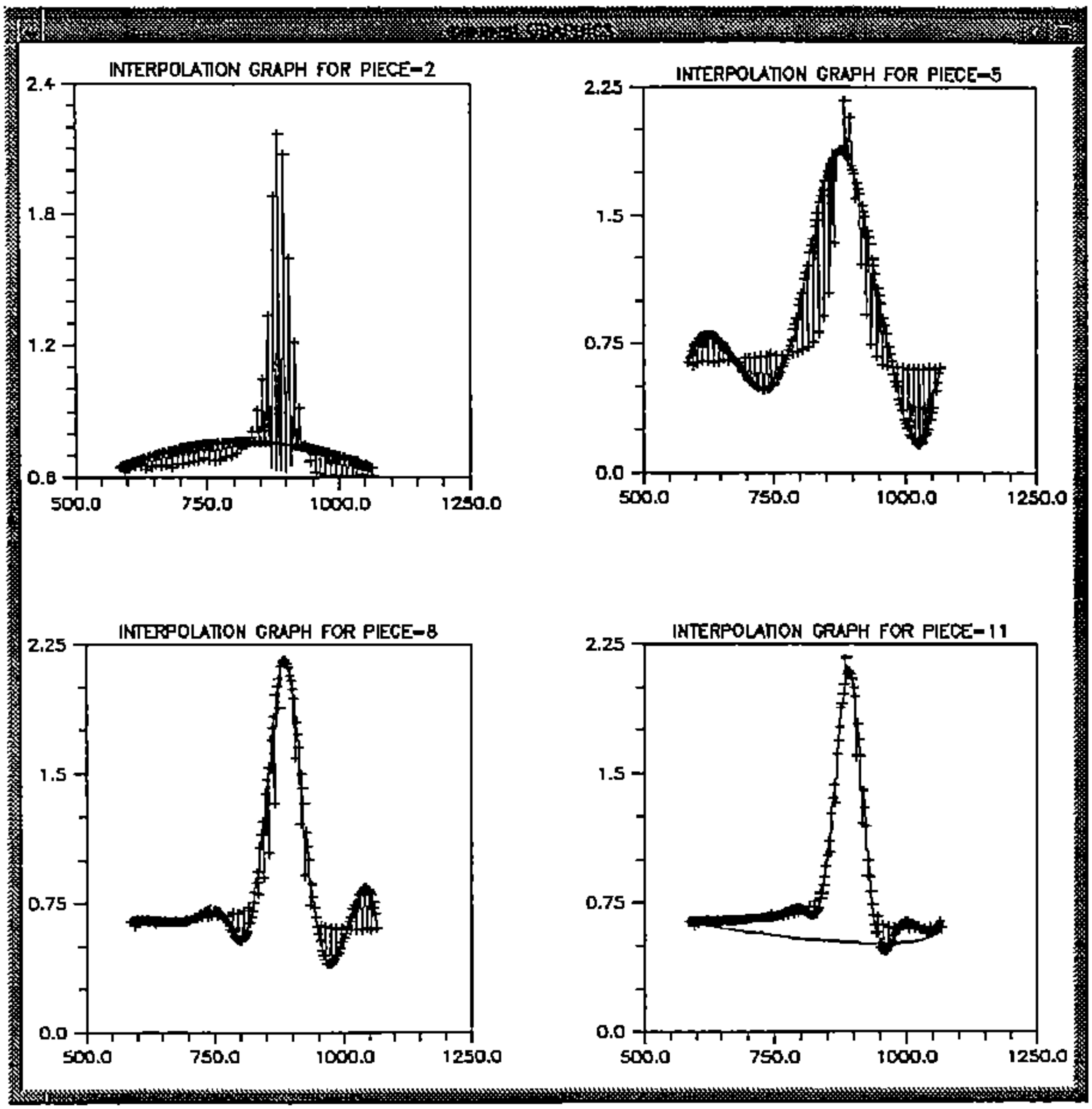


The Fortran program using the IMSL Math Library and Exponent Graphics:

```
      REAL XDATA(12),FDATA(12),BREAK(12),CSCOEF(4,12),ABSC(193)
      REAL ORDI(193,2),RANGE(4),TDAT(49,1),XPTS(49),DM,H
      INTEGER N,VALUE(4),VALIND,IUNIT
      CHARACTER *2 SYMBOL
      EXTERNAL PLOTP,CSVAL,CSINT
C
C                    THE GIVEN TITANIUM DATA.
      DATA TDAT/.644,.622,.638,.649,.652,.639,.646,.657,.652,.655,
     * .644,.663,.663,.668,.676,.676,.686,.679,.678, .683,
     * .694,.699,.710,.730,.763,.812,.907,1.044,1.336,1.881,
     * 2.169,2.075,1.598,1.211,.916,.746,.672,.627,.615,.607,
     * .606,.609,.603,.601,.603,.601,.611,.601,.608/
C
      DATA RANGE / 575.0,1065.0,3.0,-1.0 /
      DATA VALUE/3,6,9,12/
C                             INITIALIZATIONS.
      VALIND = 1
      H = 2.50
      IUNIT = 1
C                         INITIALIZE THE ABSCISSAE.
```

```
      DO 12 I = 1,49
        XPTS(I) = 575.0 + I*10.0
 12   CONTINUE
 15   N = VALUE(VALIND)
C
C                    CHOOSE THE INTERPOLATION POINTS.
      XDATA(1) = XPTS(1)
      FDATA(1) = TDAT(1,1)
      XDATA(N) = XPTS(49)
      FDATA(N) = TDAT(N,1)
      DM = 49.0/FLOAT(N-1)
      DO 25 I = 2,N-1
        J = INT(1 + DM*(I-1))
        XDATA(I) = XPTS(J)
        FDATA(I) = TDAT(J,1)
 25   CONTINUE
C
C  COMPUTE THE CUBIC SPLINE INTERPOLANT WITH THE 'NOT-A-KNOT' CONDITION
C
      CALL CSINT(N,XDATA,FDATA,BREAK,CSCOEF)
C
C              CALCULATE THE INTERPOLATION VALUES AT 193 POINTS.
      RANGE(3) = 0.0
      NINTV = N - 1
      DO 35 J = 1,193
         ABSC(J) =  585.0 + (J-1.0)*H
         ORDI(J,2) = CSVAL(ABSC(J),NINTV,BREAK,CSCOEF)
         ORDI(J,1) = ORDI(J,2)
         IF (MOD(J,4).EQ.1) ORDI(J,1) = TDAT(INT(J/4)+1,1)
         IF (ABS(ORDI(J,2)).GT.RANGE(3)) THEN
            RANGE(3) = ABS(ORDI(J,2))
         ENDIF
 35   CONTINUE
      RANGE(4) = -1.0*RANGE(3)
C
C     USING EXPONENT GRAPHICS TO DISPLAY
C
      GOTO (100, 200, 300, 400) VALIND
 100  CALL SCATR(193,ABSC,ORDI)
      CALL EGSGL('.1 use$', 'scatr2.d1$')
      GOTO 500
 200  CALL EGQSPN(1,2)
      CALL SCATR(193,ABSC,ORDI)
      CALL EGSGL('.2 use$', 'scatr2.d2$')
      GOTO 500
 300  CALL EGQSPN(1,3)
      CALL SCATR(193,ABSC,ORDI)
      CALL EGSGL('.3 use$', 'scatr2.d3$')
      GOTO 500
```

```
400   CALL EGQSPN(1,4)
      CALL SCATR(193,ABSC,ORDI)
      CALL EGSGL('.4 use$', 'scatr2.d4$')
      CALL EFMPLT(1,2,2,IUNIT, ' ')
500   CONTINUE
      VALIND = VALIND + 1
      IF (VALIND.NE.5) GO TO 15
      END
```



From these two examples, one can see that IMSL/IDL is much simpler. It only uses two statements:

```
pp = csinterp(xdata, fdata)
ppval = spvalue(findgen(193)*2.5+585.0,pp)
```

to compute the interpolation and find the values at the 193 points, while the second program uses:

```
      CALL CSINT(N,XDATA,FDATA,BREAK,CSCOEF)
C
C              CALCULATE THE INTERPOLATION VALUES AT 193 POINTS.
      RANGE(3) = 0.0
      NINTV = N - 1
      DO 35 J = 1,193
         ABSC(J) =  585.0 + (J-1.0)*H
         ORDI(J,2) = CSVAL(ABSC(J),NINTV,BREAK,CSCOEF)
         ORDI(J,1) = ORDI(J,2)
         IF (MOD(J,4).EQ.1) ORDI(J,1) = TDAT(INT(J/4)+1,1)
         IF (ABS(ORDI(J,2)).GT.RANGE(3)) THEN
            RANGE(3) = ABS(ORDI(J,2))
         ENDIF
 35   CONTINUE
      RANGE(4) = -1.0*RANGE(3)
```

IMSL/IDL hides the Fortran loop.

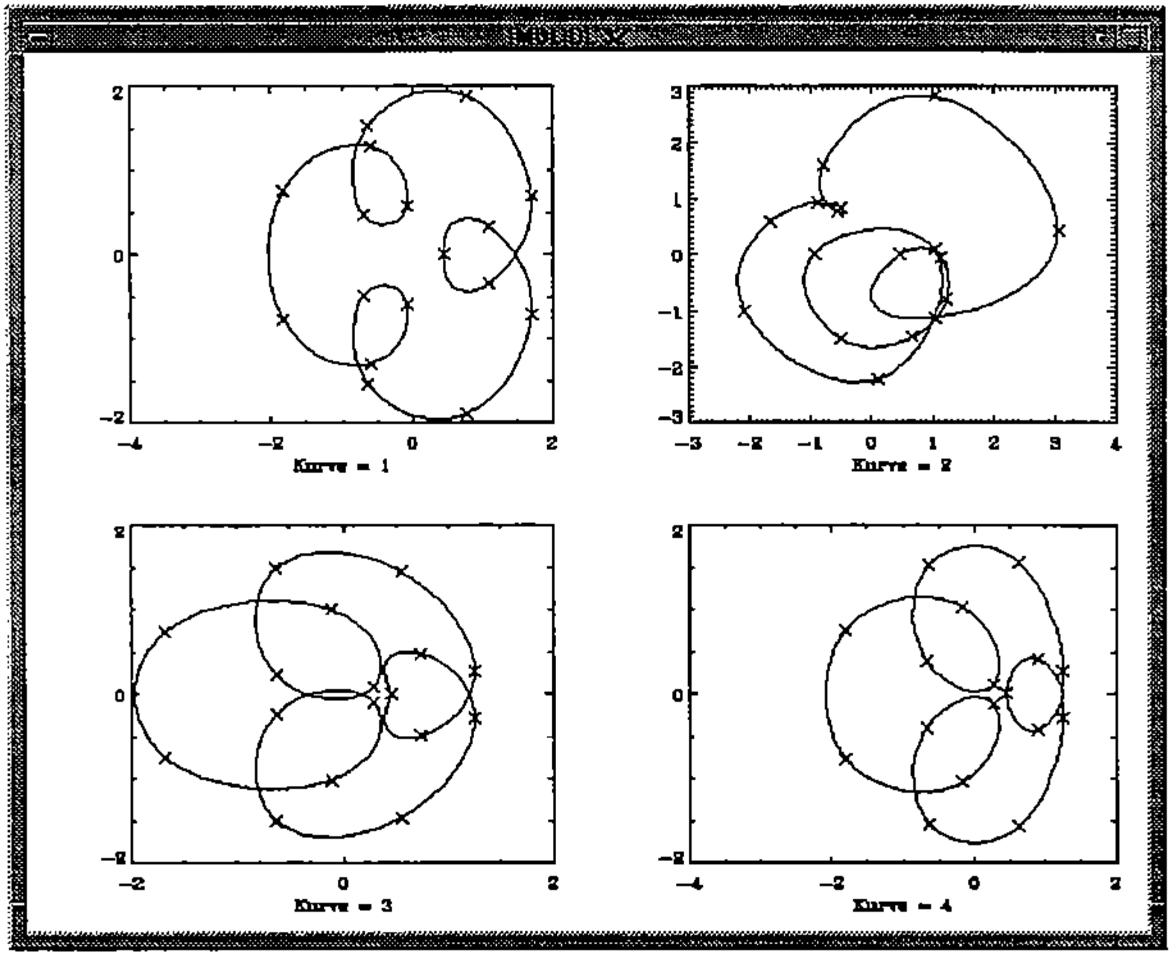
As for the graphic display part, Exponent Graphics provides external control data file so that one can change the graphics without recompiling the program. But the tree structure mechanism seems rather complicated compared with the facility that IMSL/IDL uses. With IMSL/IDL, one can change the colors of the display without modification of the program. One can run the program which displays the graphics, and then run *xloadct* which is a library routine. It lists 16 color tables and one can dynamically change the color of the display by clicking the mouse button on the color table chosen. Certainly this can not change the marks. With IMSL/IDL one can chose the size of the graphics window. A good approach would be to combine the dynamic capability of IMSL/IDL with the flexibility of the Exponent Graphics mechanism.

PRETTY CURVES WITH LOOPS

IMSL/IDL program:

```
;       LIBRARY APPLICATION 5.5
;       PRETTY CURVES WITH LOOPS FOR DATA SETS OF 4NX POINTS BY CHOOSING
;       NX AS 6,12,18,24. CHOOSE EACH SET OF NX POINTS AS EQUALLY DISTRIBUTED
;       ON THE CIRCLE. THE PARAMETRIC REPRESENTATION OF THE CIRCLE IS BEING
;       USED. THE 4 SETS CHOSEN HAVE PHASE DIFFERENCE BETWEEN THEM. THE
;       4NX POINTS ARE INTERPOLATED BY CUBIC SPLINES AND THE RESULTING CURVES
;       ARE PLOTTED.
;
;       KURVE  = SELECTION FOR 1 OF THE 4 CURVES
;       N      = THE NUMBER OF LOOPS IN THE CURVE
;       M      = THE NUMBER OF SQUEEZES IN THE CURVE
;       NX     = THE NUMBER OF INTERPOLATION POINTS USED = 6,12,18,24
;       RT     = 1.25 -- PARAMETER USED TO DEFINE CURVE
;       RB     = .8   --  PARAMETER USED TO DEFINE CURVE
;
;       DT     - ANGLE INCREMENT BEING INTERPOLATED
;       PARAM  - THE VALUES OF THE PARAMATER VARIABLE
;       XP     - THE ABSCISSAE AS FUNCTION OF PARAM
;       YP     - THE ORDINATES AS FUNCTION OF PARAM

    function x, t, kurve
    common params, rt, rb, m, n
        case Kurve of
          1: x = rt*cos(t) - rb*cos((n+1)*t)
          2: x = rt*cos(t) - rb*cos((n+1)*t)*exp(sin(m*t))
          3: x = rt*cos(t)/(1.+sin(m*t)^2) - rb*cos((n+1)*t)
          4: x = rt*cos(t)/(1.+sin(m*t)^4) - rb*cos((n+1)*t)
          else: x = 0.0
        endcase
        return, x
    end

    function y, t, kurve
    common params, rt, rb, m, n
        case kurve of
          1: y = rt*sin(t) - rb*sin((n+1)*t)
          2: y = rt*sin(t) - rb*sin((n+1)*t)*exp(sin(m*t))
          3: y = rt*sin(t)/(1.+sin(m*t)^2) - rb*sin((n+1)*t)
          4: y = rt*sin(t)/(1.+sin(m*t)^4) - rb*sin((n+1)*t)
          else: y = 0.0
        endcase
        return, y
    end

pro 155
;
;   INITIALIZATION
```

```
;
    common params, rt, rb, m, n

    p.multi = [0,2,2,0,0]
    rt = 1.25
    rb = 0.8
    m = 2
    n = 3
    loadct, 13
    for kurve = 1, 4 do begin
        window, /free
        for nx = 6, 24, 6 do begin
         dt = 2.*!pi/(nx-1)
    param = findgen(nx)
;   1
    xp = x(findgen(nx)*dt,kurve)
    yp = y(findgen(nx)*dt,kurve)
;   2
;           xp = fltarr(nx)
;           yp = fltarr(nx)
;           for i=0,nx-2 do begin
;               t = i*dt
;               xp(i) = x(t,kurve)
;               yp(i) = y(t,kurve)
;           endfor
;
    xp(nx-1) = xp(0)
    yp(nx-1) = yp(0)
    px = csinterp(param,xp)
    py = csinterp(param,yp)
    pxval = spvalue(findgen(124)*(nx-1)/123,px)
    pyval = spvalue(findgen(124)*(nx-1)/123,py)
            if (nx .eq. 18) then
               plot, pxval, pyval
         oplot, xp, yp, psym = 7, color = 112
            endif
        endfor
    endfor
end
```



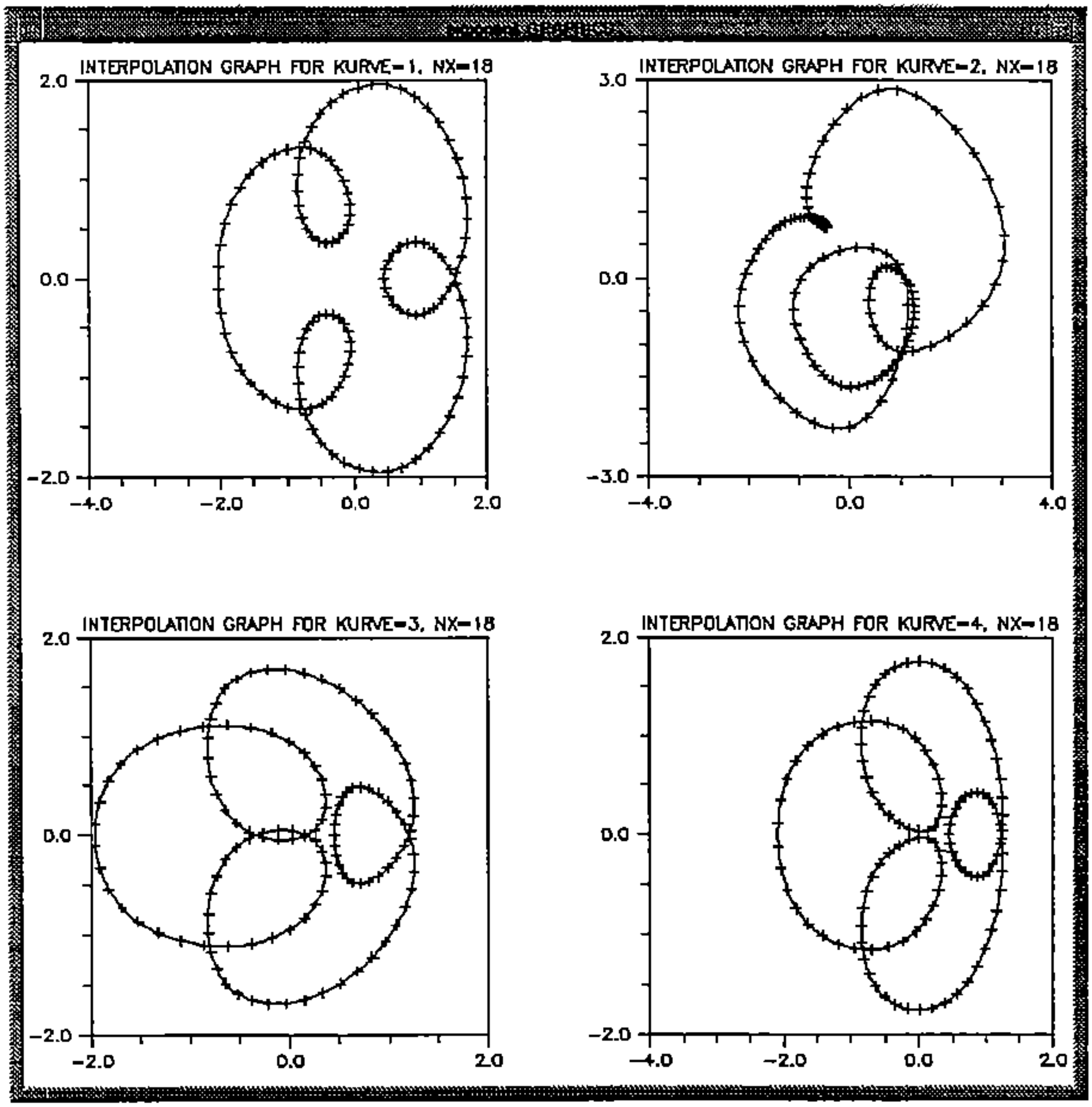
The corresponding Fortran program using the IMSL Math Library and Exponent Graphics is as follows:

```
C                              LIBRARY APPLICATION 5.5
C      PRETTY CURVES WITH LOOPS FOR DATA SETS OF 4NX POINTS BY CHOOSING
C      NX AS 6,12,18,24. CHOOSE EACH SET OF NX POINTS AS EQUALLY DISTRIBUTED
C      ON THE CIRCLE. THE PARAMETRIC REPRESENTATION OF THE CIRCLE IS BEING
C      USED. THE 4 SETS CHOSEN HAVE PHASE DIFFERENCE BETWEEN THEM. THE
C      4NX POINTS ARE INTERPOLATED BY CUBIC SPLINES AND THE RESULTING CURVES
C      ARE PLOTTED.
C
C      KURVE  = SELECTION FOR 1 OF THE 4 CURVES
C      N      = THE NUMBER OF LOOPS IN THE CURVE
C      M      = THE NUMBER OF SQUEEZES IN THE CURVE
C      NX     = THE NUMBER OF INTERPOLATION POINTS USED = 6,12,18,24
C      RT     = 1.25 -- PARAMETER USED TO DEFINE CURVE
C      RB     = .8  --  PARAMETER USED TO DEFINE CURVE
C
C      T      - VALUE OF ANGLE USED
C      DT     - ANGLE INCREMENT BEING INTERPOLATED
C      PARAM  - THE VALUES OF THE PARAMATER VARIABLE
C      XP     - THE ABSCISSAE AS FUNCTION OF PARAM
C      YP     - THE ORDINATES AS FUNCTION OF PARAM
```

```
C       ABSC   - ABSCISSAE USED FOR PLOTTING
C       ORDI   - ORDINATE USED FOR PLOTTING
C       BREAK1 - BREAKPOINTS FOR PIECEWISE CUBIC REPRESENTATION FOR ABSC
C       BREAK2 - BREAKPOINTS FOR PIECEWISE CUBIC REPRESENTATION FOR ORDI
C       CSCOE1 - MATRIX(4 BY NX) OF COEFFICIENTS OF CUBIC PIECES FOR ABSC
C       CSCOE2 - MATRIX(4 BY NX) OF COEFFICIENTS OF CUBIC PIECES FOR ORDI
C       RANGE -  CONSISTS OF ENDPOINTS ON THE X-AXIS & Y-AXIS
C
        REAL XP(24),YP(24),T,PI
        REAL BREAK1(24),BREAK2(24), CSCOE1(4,24),CSCOE2(4,24)
        REAL ORDI(150,1),RANGE(4),PI,ABSC(150),PARAM(24)
        INTEGER N,NX,M
        CHARACTER SYMBOL*1
        EXTERNAL PLOTP,CSVAL,CSPER,CONST
        COMMON/PARAMS/RT,RB,N,M,PI
C                                   INITIALIZATIONS.
        PI = CONST('PI')
        RT = 1.25
        RB = .8
        N  = 3
        M  = 2
C                                   SELECT CURVE
        DO 100 KURVE = 1,4
C                            LOOP OVER NUMBER OF INTERPOLATION POINT
        DO 100 NX = 6,24,6
        DT = 2.*PI/(NX - 1)
C                        CHOOSE THE POINTS AS THE FUNCTION OF A PARAMETER.
        DO 20 I = 1,NX-1
            PARAM(I) = I - 1
            T      = (I-1)*DT
            XP(I) = X(T,KURVE)
            YP(I) = Y(T,KURVE)
   20   CONTINUE
C                         MAKE LAST POINTS EXACTLY EQUAL TO FIRST
        PARAM(NX) = NX-1
        XP(NX)    = XP(1)
        YP(NX)    = YP(1)
C                         COMPUTE THE CUBIC SPLINE INTERPOLANT
C                          FOR X AND Y FUNCTIONS.
        CALL CSPER(NX,PARAM,XP,BREAK1,CSCOE1)
        CALL CSPER(NX,PARAM,YP,BREAK2,CSCOE2)
C                        COMPUTE THE 124 DATA VALUES FOR PLOTTING.
        DVAL = (NX-1)/123.
        DO 30 I = 1,124
           VAL = (I-1)*DVAL
           ABSC(I) = CSVAL(VAL,NX-1,BREAK1,CSCOE1)
           ORDI(I,1) = CSVAL(VAL,NX-1,BREAK2,CSCOE2)
   30   CONTINUE
C                            COMPUTE SIZE OF PLOT
```

```
      RANGE(1) = ABSC(1)
      RANGE(2) = ABSC(1)
      RANGE(3) = ORDI(1,1)
      RANGE(4) = ORDI(1,1)
      DO 40 I = 2,124
          RANGE(1) = MIN(RANGE(1),ABSC(I))
          RANGE(2) = MAX(RANGE(2),ABSC(I))
          RANGE(3) = MIN(RANGE(3),ORDI(I,1))
          RANGE(4) = MAX(RANGE(4),ORDI(I,1))
 40   CONTINUE

C               PLOT THE CURVES WITH MANY MORE POINTS THAN DATA
C                               ONLY FOR NX = 18
      IF (NX .EQ. 18) THEN
         GOTO (50, 60, 70, 80) KURVE
 50      CALL SCATR(124,ABSC,ORDI)
         CALL EGSGL('.1 use$', 'scatr5.d1$')
         GOTO 100
 60      CALL EGQSPN(1,2)
         CALL SCATR(124,ABSC,ORDI)
         CALL EGSGL('.2 use$', 'scatr5.d2$')
         GOTO 100
 70      CALL EGQSPN(1,3)
         CALL SCATR(124,ABSC,ORDI)
         CALL EGSGL('.3 use$', 'scatr5.d3$')
         GOTO 100
 80      CALL EGQSPN(1,4)
         CALL SCATR(124,ABSC,ORDI)
         CALL EGSGL('.4 use$', 'scatr5.d4$')
         CALL EFMPLT(1,2,2,IUNIT, ' ')
      ENDIF
  100 CONTINUE
     END

      REAL FUNCTION X(T,KURVE)
      REAL T,RT,RB
      COMMON / PARAMS / RT,RB,N,M,PI
      X = 0.0
      IF (KURVE.EQ.1)  THEN
                  X = RT*COS(T) - RB*COS((N+1)*T)
        ELSE IF (KURVE.EQ.2)  THEN
                  X = RT*COS(T) - RB*COS((N+1)*T)*EXP(SIN(M*T))
        ELSE IF (KURVE.EQ.3)  THEN
                  X = RT*COS(T)/(1.+SIN(M*T)**2) - RB*COS((N+1)*T)
        ELSE IF (KURVE.EQ.4)  THEN
                  X = RT*COS(T)/(1.+SIN(M*T)**4) - RB*COS((N+1)*T)
      END IF
      RETURN
      END
```

```
REAL FUNCTION Y(T,KURVE)
REAL T,RT,RB
INTEGER N,M,KURVE
COMMON / PARAMS / RT,RB,N,M,PI
Y = 0.0
IF (KURVE.EQ.1)  THEN
            Y = RT*SIN(T) - RB*SIN((N+1)*T)
  ELSE IF (KURVE.EQ.2)  THEN
            Y = RT*SIN(T) - RB*SIN((N+1)*T)*EXP(SIN(M*T))
  ELSE IF (KURVE.EQ.3)  THEN
            Y = RT*SIN(T)/(1.+SIN(M*T)**2) - RB*SIN((N+1)*T)
  ELSE IF (KURVE.EQ.4)  THEN
            Y = RT*SIN(T)/(1.+SIN(M*T)**4) - RB*SIN((N+1)*T)
END IF
RETURN
END
```



The function *findgen* is very useful. It returns an array whose elements contain the values of their subscripts. This function is extensively used especially for a graph display where the abscissae are functions of 0 to n. Without this function, one has to write a loop to get the array. In IMSL/IDL, data type of a variable changes dynamically. This is not good for large applications. But one interesting aspect is that one does not need to rewrite the function x and y no matter whether one uses method 1 or method 2 to obtain xp and yp.

One disadvantage of IMSL/IDL is that one cannot save the compiled module. Each time one exits IMSL/IDL session, the compiled module is lost and the next time when one invokes IMSL/IDL, one has to recompile the module. This is not desirable especially for large programs. Another disadvantage of IMSL/IDL is that the language does not have the facility of passing a function name to a user defined procedure.

COMPARE THREE QUADRATURE METHODS

IMSL/IDL program:

```
;                          LIBRARY APPLICATION 7.2
;     TO COMPARE THE ACCURACY OBTAINED BY USING THREE INTEGRATION
;     METHODS. THE METHODS ARE
;
;     (1) COMPOSITE TRAPEZOIDAL RULE
;     (2) INTEGRATION OF THE HERMITE CUBICS INTERPOLANT
;     (3) INTEGRATION OF THE CUBIC SPLINE INTERPOLANT
;
;     THE FOLLOWING FUNCTIONS ARE USED.
;     (1) EXP(X)
;     (2) X**3 - X**2
;     (3) SIN(2*X)
;     (4) 3./(1. + 50*(X-.2)**4)
;
;     WE DIVIDE THE INTERVAL INTO 20 PARTS. ( K = 20 )
;
;     XDATA  - ABSCISSAE OF THE INTERPOLATION POINTS
;     YDATA  - ORDINATE OF THE INTERPOLATION POINT
;     H      -THE LENGTH OF THE INTERVAL USED FOR COMPOSITE TRAPEZOIDAL RULE
;     K      - NUMBER OF INTERPOLATION POINTS - 1
;     VAL    - USED TO COMPUTE THE COMPOSITE TRAPEZOIDAL RULE

    pro f1
    common x, xdata, ydata, val, k
    ydata = exp(xdata)
    val = total(exp((findgen(k-1)+1)/k))
    print, 'RESULT FOR F(X) = EXP(X)'
    comput
    end

    pro f2
    common x, xdata, ydata, val, k
    ydata = xdata^3-xdata^2
    val = total(((findgen(k-1)+1)/k)^3-((findgen(k-1)+1)/k)^2)
    print, 'RESULT FOR F(X) = X**3 - X**2'
    comput
    end

    pro f3
    common x, xdata, ydata, val, k
    ydata = sin(2*xdata)
    val = total(sin(2*(findgen(k-1)+1)/k))
    print, 'RESULT FOR F(X) = SIN(2*X)'
    comput
    end

    pro f4
```

```
    common x, xdata, ydata, val, k
    ydata = 3.0/(1.0 + 50.0*(xdata-0.2)^4)
    val = total(3.0/(1.0 + 50.0*((findgen(k-1)+1)/k-0.2)^4))
    print, 'RESULT FOR F(X) = 3.0/(1 + 50*(X-.2)**4)'
    comput
    end

    pro comput
    common x, xdata, ydata, val, k
    h = 1.0/k
    val = (val + (ydata(0)+ydata(k))/2.0)*h
    print,val, format = $
              '("COMPOSITE TRAPEZOIDAL RULE                = ",F13.10)'
    pp = csshape(xdata,ydata)
    ppeval = spinteg(0.0,1.0,pp)
    print,ppeval, format = $
              '("INTEGRATION OF HERMITE CUBIC INTERPOLANT = ",F13.10)'
    pp = csinterp(xdata,ydata)
    ppeval = spinteg(0.0,1.0,pp)
    print,ppeval, format = $
              '("INTEGRATION OF CUBIC SPLINE INTERPOLANT  = ",F13.10, /)'
    end

pro 172
    common x, xdata, ydata, val, k
    k = 20
    xdata = findgen(k+1)/k
    xdata = ((1.0+xdata)^2-1.0)/3.0
    f1
    f2
    f3
    f4
end
```

RESULTS

```
--------------------------------------------------------------------------------
RESULT FOR F(X) = EXP(X)
COMPOSITE TRAPEZOIDAL RULE              =  1.7186399698
INTEGRATION OF HERMITE CUBIC INTERPOLANT =  1.7182828188
INTEGRATION OF CUBIC SPLINE INTERPOLANT  =  1.7182817459

RESULT FOR F(X) = X**3 - X**2
COMPOSITE TRAPEZOIDAL RULE              = -0.0831250101
INTEGRATION OF HERMITE CUBIC INTERPOLANT = -0.0833311379
INTEGRATION OF CUBIC SPLINE INTERPOLANT  = -0.0833333284

RESULT FOR F(X) = SIN(2*X)
COMPOSITE TRAPEZOIDAL RULE              =  0.7074832916
INTEGRATION OF HERMITE CUBIC INTERPOLANT =  0.7080752850
INTEGRATION OF CUBIC SPLINE INTERPOLANT  =  0.7080735564
```

```
RESULT FOR F(X) = 3.0/(1 + 50*(X-.2)**4)
COMPOSITE TRAPEZOIDAL RULE                    =  1.8046340942
INTEGRATION OF HERMITE CUBIC INTERPOLANT =  1.8056036234
INTEGRATION OF CUBIC SPLINE INTERPOLANT  =  1.8056387901
```

If IMSL/IDL had the facility of passing function names as a parameter to user defined procedure, one might change the program to be:

```
function f1, xdata
print, 'RESULT FOR F(X) = EXP(X)'
return exp(xdata)
end

function f2, xdata
print, 'RESULT FOR F(X) = X**3 - X**2'
return, xdata^3-xdata^2
end

function f3, xdata
print, 'RESULT FOR F(X) = SIN(2*X)'
return, sin(2*xdata)
end

function f4, xdata
print, 'RESULT FOR F(X) = 3.0/(1 + 50*(X-.2)**4)'
return, 3.0/(1.0 + 50.0*(xdata-0.2)^4)
end

pro comput, f, xdata
ydata = f(xdata)
val = total(f(findgen(k-1)+1)/k))
val = val + ydata(0) + ydata(20)
print,val, format = $
            '("COMPOSITE TRAPEZOIDAL RULE                    = ",F13.10)'
pp = csshape(xdata,ydata)
ppeval = spinteg(0.0,1.0,pp)
print,ppeval, format = $
            '("INTEGRATION OF HERMITE CUBIC INTERPOLANT = ",F13.10)'
pp = csinterp(xdata,ydata)
ppeval = spinteg(0.0,1.0,pp)
print,ppeval, format = $
            '("INTEGRATION OF CUBIC SPLINE INTERPOLANT  = ",F13.10)'
print
end

pro 172
    k = 20
    xdata = findgen(k+1)/k
```

```
    xdata = ((1.0+xdata)^2-1.0)/3.0
    comput, f1, xdata
    comput, f2, xdata
    comput, f3, xdata
    comput, f4, xdata
end
```

Apparently, the second program would be clearer and more succinct.
The corresponding Fortran program using the IMSL Math Library is as follows:

```
C                            LIBRARY APPLICATION 7.2
C     TO COMPARE THE ACCURACY OBTAINED BY USING THREE INTEGRATION
C     METHODS. THE METHODS ARE
C
C     (1) COMPOSITE TRAPEZOIDAL RULE
C     (2) INTEGRATION OF THE HERMITE CUBICS INTERPOLANT
C     (3) INTEGRATION OF THE CUBIC SPLINE INTERPOLANT
C
C     THE FOLLOWING FUNCTIONS ARE USED.
C     (1) EXP(X)
C     (2) X**3 - X**2
C     (3) SIN(2*X)
C     (4) 3./(1. + 50*(X-.2)**4)
C
C     WE DIVIDE THE INTERVAL INTO 20 PARTS. ( K = 20 )
C
C     XDATA  - ABSCISSAE OF THE INTERPOLATION POINTS
C     YDATA  - ORDINATE OF THE INTERPOLATION POINT
C     BREAK  - THE BREAK POINTS OF THE PIECEWISE CUBIC REPRESENTATION
C     CSCOEF - MATRIX OF LOCAL COEFFICIENTS OF THE CUBIC PIECES
C     H      -THE LENGTH OF THE INTERVAL USED FOR COMPOSITE TRAPEZOIDAL RULE
C     K      - NUMBER OF INTERPOLATION POINTS - 1
C     VAL    - USED TO COMPUTE THE COMPOSITE TRAPEZOIDAL RULE
C
      REAL XDATA(21),K,F1,F2,F3,F4,H
      EXTERNAL F1,F2,F3,F4
      COMMON XDATA,K,H
C                                  INITIALIZATIONS
      K = 21.0
      H = 1.0/(K - 1.0)
      DO 10 I = 0,20,1
          XDATA(I+1) = ( (1+I*H)**2 - 1.0 )/3.0
 10   CONTINUE
      WRITE(6,*) ' RESULT FOR F(X) = EXP(X) '
      CALL COMPUT(F1)
      WRITE(6,*)
      WRITE(6,*) ' RESULT FOR F(X) = X**3 - X**2'
      CALL COMPUT(F2)
      WRITE(6,*)
      WRITE(6,*) ' RESULT FOR F(X) = SIN(2*X) '
```

```
      CALL COMPUT(F3)
      WRITE(6,*)
      WRITE(6,*)' RESULT FOR F(X) = 3.0/(1 + 50*(X-.2)**4)'
      CALL COMPUT(F4)
      END

      SUBROUTINE COMPUT(F)
C           COMPUTE THE THREE INTEGRALS FOR THE GIVEN FUNCTION F
      REAL F
      EXTERNAL F
      COMMON XDATA,K,H
      REAL XDATA(21),YDATA(21),BREAK(21),CSCOEF(4,21),VAL
      EXTERNAL CSINT,CSAKM,CSITG
C                          INITIALIZATIONS
      VAL =  0.0
      DO 11 I = 1,19
         VAL = VAL + F(I/20.0)
         YDATA(I+1) = F(XDATA(I+1))
 11   CONTINUE
C              COMPUTE THE COMPOSITE TRAPEZOIDAL RULE
      YDATA(1) = F(0.0)
      YDATA(21) = F(1.0)
      VAL = (VAL + (YDATA(1)+YDATA(21))/2.0)*H
      WRITE(6,91) VAL
  91  FORMAT(' COMPOSITE TRAPEZOIDAL RULE                  = ',F12.10)
C              COMPUTE THE HERMITE CUBIC INTERPOLATION
      CALL CSAKM(21,XDATA,YDATA,BREAK,CSCOEF)
      WRITE(6,92) CSITG(0.0,1.0,20,BREAK,CSCOEF)
  92  FORMAT(' INTEGRATION OF HERMITE CUBIC INTERPOLANT = ',F12.10)
C               COMPUTE THE CUBIC SPLINE INTERPOLATION
      CALL CSINT(21,XDATA,YDATA,BREAK,CSCOEF)
      WRITE(6,93) CSITG(0.0,1.0,20,BREAK,CSCOEF)
  93  FORMAT(' INTEGRATION OF CUBIC SPLINE INTERPOLANT  = ',F12.10)
      END

      REAL FUNCTION F1(X)
      REAL X
      INTRINSIC EXP
      F1 = EXP(X)
      RETURN
      END

      REAL FUNCTION F2(X)
      REAL X
      F2 = X**3 - X**2
      RETURN
      END

      REAL FUNCTION F3(X)
```

```
      REAL X
      INTRINSIC SIN
      F3 = SIN(2*X)
      RETURN
      END

      REAL FUNCTION F4(X)
      REAL X
      F4 = 3.0/(1.0 + 50.0*(X-0.2)**4)
      RETURN
      END
```

RESULTS

```
--------------------------------------------------------------------------------
RESULT FOR F(X) = EXP(X)
COMPOSITE TRAPEZOIDAL RULE                 = 1.7186399698
INTEGRATION OF HERMITE CUBIC INTERPOLANT = 1.7182828188
INTEGRATION OF CUBIC SPLINE INTERPOLANT  = 1.7182819843

RESULT FOR F(X) = X**3 - X**2
COMPOSITE TRAPEZOIDAL RULE                 = -.0831249952
INTEGRATION OF HERMITE CUBIC INTERPOLANT = -.0833311304
INTEGRATION OF CUBIC SPLINE INTERPOLANT  = -.0833333358

RESULT FOR F(X) = SIN(2*X)
COMPOSITE TRAPEZOIDAL RULE                 = 0.7074832916
INTEGRATION OF HERMITE CUBIC INTERPOLANT = 0.7080752850
INTEGRATION OF CUBIC SPLINE INTERPOLANT  = 0.7080735564

RESULT FOR F(X) = 3.0/(1 + 50*(X-.2)**4)
COMPOSITE TRAPEZOIDAL RULE                 = 1.8046340942
INTEGRATION OF HERMITE CUBIC INTERPOLANT = 1.8056036234
INTEGRATION OF CUBIC SPLINE INTERPOLANT  = 1.8056387901
```

EVALUATE THE SENSITIVITY OF INTEGRATION METHODS

IMSL/IDL program:

```
function f1, x
r = random(1,/double)
return, (1.d0 + x)*(.002d0*r(0)+.999d0)
end

function f2, x
r = random(1,/double)
return, (x^4 + x*x + 1)*(.002d0*r(0) + .999d0)
end

function f3, x
r = random(1,/double)
return, sin(x)*(.002d0*r(0)+.999d0)
end

function f4, x
r = random(1,/double)
return, sin(x*20.0d0)*(.002d0*r(0)+.999d0)
end

function f5, x
r = random(1,/double)
return, (1.d0/(1.0d0 + 40.0d0*x*x))*(.002d0*r(0)+.999d0)
end

function f6, x
r = random(1,/double)
if (x > 0) then begin
   return, x*x*(.002d0*r(0)+.999d0)
endif else begin
        return, -x*x*(.002d0*r(0)+.999d0)
endelse
end

pro 173
randomopt, set=10
a = -1.0d0
b = 1.0d0
ans = intfcn('f1', a, b, /double, err_est=errest, err_abs=0.0d0, $
             err_rel=0.01d0, max_sub=1000, rule=2)
print,ans,format = '("RESULT FOR FUNCTION I    =     ",d20.17)'
print,errest,format = '("ABSOLUTE ERROR ESTIMATE =     ",d20.17, /)'
ans = intfcn('f2', a, b, /double, err_est=errest, err_abs=0.0d0, $
      err_rel=0.01d0, max_sub=1000, rule=2)
print,ans,format = '("RESULT FOR FUNCTION II   =     ",d20.17)'
print,errest,format = '("ABSOLUTE ERROR ESTIMATE =     ",d20.17, /)'
ans = intfcn('f3', a, b, /double, err_est=errest, err_abs=0.0d0, $
```

```
        err_rel=0.01d0, max_sub=1000, rule=2)
    print,ans,format = '("RESULT FOR FUNCTION III =      ",d20.17)'
    print,errest,format = '("ABSOLUTE ERROR ESTIMATE =      ",d20.17, /)'
    ans = intfcn('f4', a, b, /double, err_est=errest, err_abs=0.0d0, $
        err_rel=0.01d0, max_sub=1000, rule=2)
    print,ans,format = '("RESULT FOR FUNCTION IV  =      ",d20.17)'
    print,errest,format = '("ABSOLUTE ERROR ESTIMATE =      ",d20.17, /)'
    ans = intfcn('f5', a, b, /double, err_est=errest, err_abs=0.0d0, $
        err_rel=0.01d0, max_sub=1000, rule=2)
    print,ans,format = '("RESULT FOR FUNCTION V   =      ",d20.17)'
    print,errest,format = '("ABSOLUTE ERROR ESTIMATE =      ",d20.17, /)'
    ans = intfcn('f6', a, b, /double, err_est=errest, err_abs=0.0d0, $
        err_rel=0.01d0, max_sub=1000, rule=2)
    print,ans,format = '("RESULT FOR FUNCTION VI  =      ",d20.17)'
    print,errest,format = '("ABSOLUTE ERROR ESTIMATE =      ",d20.17, /)'
end
```

```
                          RESULTS
--------------------------------------------------------------------------
RESULT FOR FUNCTION I   =      2.00006692290557098
ABSOLUTE ERROR ESTIMATE =      0.01146662585000681

RESULT FOR FUNCTION II  =      3.06650023517439996
ABSOLUTE ERROR ESTIMATE =      0.03064606778448305

% INTFCN: Warning: ROUNDOFF_CONTAMINATION.
          Roundoff error has been detected. The requested tolerances,
          "ERR_ABS" = 0.000000e+00 and "ERR_REL" = 1.000000e-02 cannot
          reached.
RESULT FOR FUNCTION III =      0.00000148998466897
ABSOLUTE ERROR ESTIMATE =      0.00081770084898793

% INTFCN: Warning: ROUNDOFF_CONTAMINATION.
          Roundoff error has been detected. The requested tolerances,
          "ERR_ABS" = 0.000000e+00 and "ERR_REL" = 1.000000e-02 cannot
          reached.
RESULT FOR FUNCTION IV  =      0.00002184360167393
ABSOLUTE ERROR ESTIMATE =      0.00464273563325588

RESULT FOR FUNCTION V   =      0.44713921783942639
ABSOLUTE ERROR ESTIMATE =      0.00384964923094636

% INTFCN: Warning: ROUNDOFF_CONTAMINATION.
          Roundoff error has been detected. The requested tolerances,
          "ERR_ABS" = 0.000000e+00 and "ERR_REL" = 1.000000e-02 cannot
          reached.
RESULT FOR FUNCTION VI  =     -0.00000035163600409
ABSOLUTE ERROR ESTIMATE =      0.00072768477839928
```

The corresponding Fortran program using the IMSL Math Library is as follows:

```
C                        LIBRARY APPLICATION 7.3
C       TO EVALUATE THE SENSITIVITY OF INTEGRATION METHODS TO ROUND OFF
C       FOR FOLLOWING FUNCTIONS. THE INTEGRALS ON [-1.0,1.0] FOR THE
C       CHOSEN FUNCTIONS ARE EXACTLY KNOWN. DURING THE EVALUATION OF THE
C       INTEGRALS THEIR VALUES ARE PERTURBED BY MULTIPLYING BY
C       (1.0 + EPS) WHERE EPS IS A RANDOM NUMBER DISTRIBUTED IN [-.001,.001]
C
C       F(X) = 1 + X
C       F(X) = X**4 + X*X + 1
C       F(X) = SIN(X)
C       F(X) = SIN(20*X)
C       F(X) = 1/(1 + 40*X*X)
C       F(X) = X**2*SIGN(X)
C
C       A      - LOWER LIMIT OF INTEGRATION
C       B      - UPPER LIMIT OF INTEGRATION
C       ERRABS - ABSOLUTE ACCURACY DESIRED
C       ERRREL - RELATIVE ACCURACY DESIRED
C       IRULE  - PARAMETER FOR CHOICE OF QUADRATURE
C       RESULT - ESTIMATE OF THE INTEGRAL
C       ERREST - ESTIMATE OF THE ABSOLUTE VALUE OF THE ERROR
C
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        DOUBLE PRECISION F1,F2,F3,F4,F5,F6,RLIST(1000),ELIST(1000),A
        INTEGER IRULE,MAXSUB,NEVAL,NSUBIN,IORD(1000)
        DOUBLE PRECISION B,ERRABS,ERREST,RESULT,ALIST(1000),BLIST(1000)
        EXTERNAL F1,F2,F3,F4,F5,F6,RNSET,DQ2AG
C                          INITIALIZATIONS
       A      = -1.0D0
       B      = 1.0D0
       ERRABS = 0.0D0
       ERRREL = 0.01D0
       IRULE  = 2
       MAXSUB = 1000
C
C                 INITIALIZE SEED VALUE OF RANDOM NUMBER GENERATOR.
       CALL RNSET(10)
C                 COMPUTE THE INTEGRALS FOR VARIOUS FUNCTIONS.
       CALL DQ2AG(F1,A,B,ERRABS,ERRREL,IRULE,RESULT,ERREST,MAXSUB,NEVAL
     * ,NSUBIN,ALIST,BLIST,RLIST,ELIST,IORD)
       PRINT *, 'RESULT FOR FUNCTION II  = ',RESULT
       PRINT *, 'ABSOLUTE ERROR ESTIMATE = ',ERREST

       PRINT *
       CALL DQ2AG(F3,A,B,ERRABS,ERRREL,IRULE,RESULT,ERREST,MAXSUB,NEVAL
     * ,NSUBIN,ALIST,BLIST,RLIST,ELIST,IORD)
       PRINT *, 'RESULT FOR FUNCTION III = ',RESULT
       PRINT *, 'ABSOLUTE ERROR ESTIMATE = ',ERREST
```

```
      PRINT *
      CALL DQ2AG(F4,A,B,ERRABS,ERRREL,IRULE,RESULT,ERREST,MAXSUB,NEVAL
     * ,NSUBIN,ALIST,BLIST,RLIST,ELIST,IORD)
      PRINT *, 'RESULT FOR FUNCTION IV   = ',RESULT
      PRINT *, 'ABSOLUTE ERROR ESTIMATE = ',ERREST

      PRINT *
      CALL DQ2AG(F5,A,B,ERRABS,ERRREL,IRULE,RESULT,ERREST,MAXSUB,NEVAL
     * ,NSUBIN,ALIST,BLIST,RLIST,ELIST,IORD)
      PRINT *, 'RESULT FOR FUNCTION V    = ',RESULT
      PRINT *, 'ABSOLUTE ERROR ESTIMATE = ',ERREST

      PRINT *
      CALL DQ2AG(F6,A,B,ERRABS,ERRREL,IRULE,RESULT,ERREST,MAXSUB,NEVAL
     * ,NSUBIN,ALIST,BLIST,RLIST,ELIST,IORD)
      PRINT *, 'RESULT FOR FUNCTION VI   = ',RESULT
      PRINT *, 'ABSOLUTE ERROR ESTIMATE = ',ERREST

      END
C
C     THE 6 FUNCTIONS WHOSE VALUES ARE PERTURBED BY ROUND OFF ERRORS.
      DOUBLE PRECISION FUNCTION F1(X)
      DOUBLE PRECISION X,DRNUNF
      F1 = (1 + X)*(.002*DRNUNF() + .999)
      RETURN
      END

      DOUBLE PRECISION FUNCTION F2(X)
      DOUBLE PRECISION X,DRNUNF
      EXTERNAL DRNUNF
      F2 = (X**4 + X*X + 1)*(0.999 + .002*DRNUNF())
      RETURN
      END

      DOUBLE PRECISION FUNCTION F3(X)
      DOUBLE PRECISION X,DRNUNF
      EXTERNAL DRNUNF
      INTRINSIC DSIN
      F3 = DSIN(X)*(0.999 + .002*DRNUNF())
      RETURN
      END

      DOUBLE PRECISION FUNCTION F4(X)
      DOUBLE PRECISION X,DRNUNF
      EXTERNAL DRNUNF
      INTRINSIC DSIN
      F4 = DSIN(X*20.0)*(0.999 + .002*DRNUNF())
      RETURN
```

```
      END

      DOUBLE PRECISION FUNCTION F5(X)
      DOUBLE PRECISION X,DRNUNF
      EXTERNAL DRNUNF
      F5 = (1.0/(1. + 40*X*X))*(0.999 + .002*DRNUNF())
      RETURN
      END

      DOUBLE PRECISION FUNCTION F6(X)
      DOUBLE PRECISION X,DRNUNF
      EXTERNAL DRNUNF
      IF (X.GT.0.0) THEN
         F6 = X*X*(0.999 + .002*DRNUNF())
      ELSE
         F6 = -X*X*(0.999 + .002*DRNUNF())
      ENDIF
      RETURN
      END
```

RESULTS

```
--------------------------------------------------------------------------------
RESULT FOR FUNCTION  I  =     2.0000669229672
ABSOLUTE ERROR ESTIMATE =     1.1466626362397D-02

RESULT FOR FUNCTION II  =     3.0665002352608
ABSOLUTE ERROR ESTIMATE =     3.0646068623752D-02


 *** WARNING  ERROR 2 from DQ2AG.  Roundoff error has been detected.  The
 ***          requested tolerances, ERRABS = 0.000000000000000D+00 and ERRREL
 ***          = 1.000000000000000D-02 cannot be reached.
RESULT FOR FUNCTION III =     1.4899847133067D-06
ABSOLUTE ERROR ESTIMATE =     8.1770085773151D-04


 *** WARNING  ERROR 2 from DQ2AG.  Roundoff error has been detected.  The
 ***          requested tolerances, ERRABS = 0.000000000000000D+00 and ERRREL
 ***          = 1.000000000000000D-02 cannot be reached.
RESULT FOR FUNCTION IV  =     2.1843602324972D-05
ABSOLUTE ERROR ESTIMATE =     4.6427357676711D-03

RESULT FOR FUNCTION V   =    0.44713921785273
ABSOLUTE ERROR ESTIMATE =     3.8496493918691D-03


 *** WARNING  ERROR 2 from DQ2AG.  Roundoff error has been detected.  The
 ***          requested tolerances, ERRABS = 0.000000000000000D+00 and ERRREL
 ***          = 1.000000000000000D-02 cannot be reached.
```

```
RESULT FOR FUNCTION VI  =     -3.5163601449313D-07
ABSOLUTE ERROR ESTIMATE =      7.2768478362914D-04
```

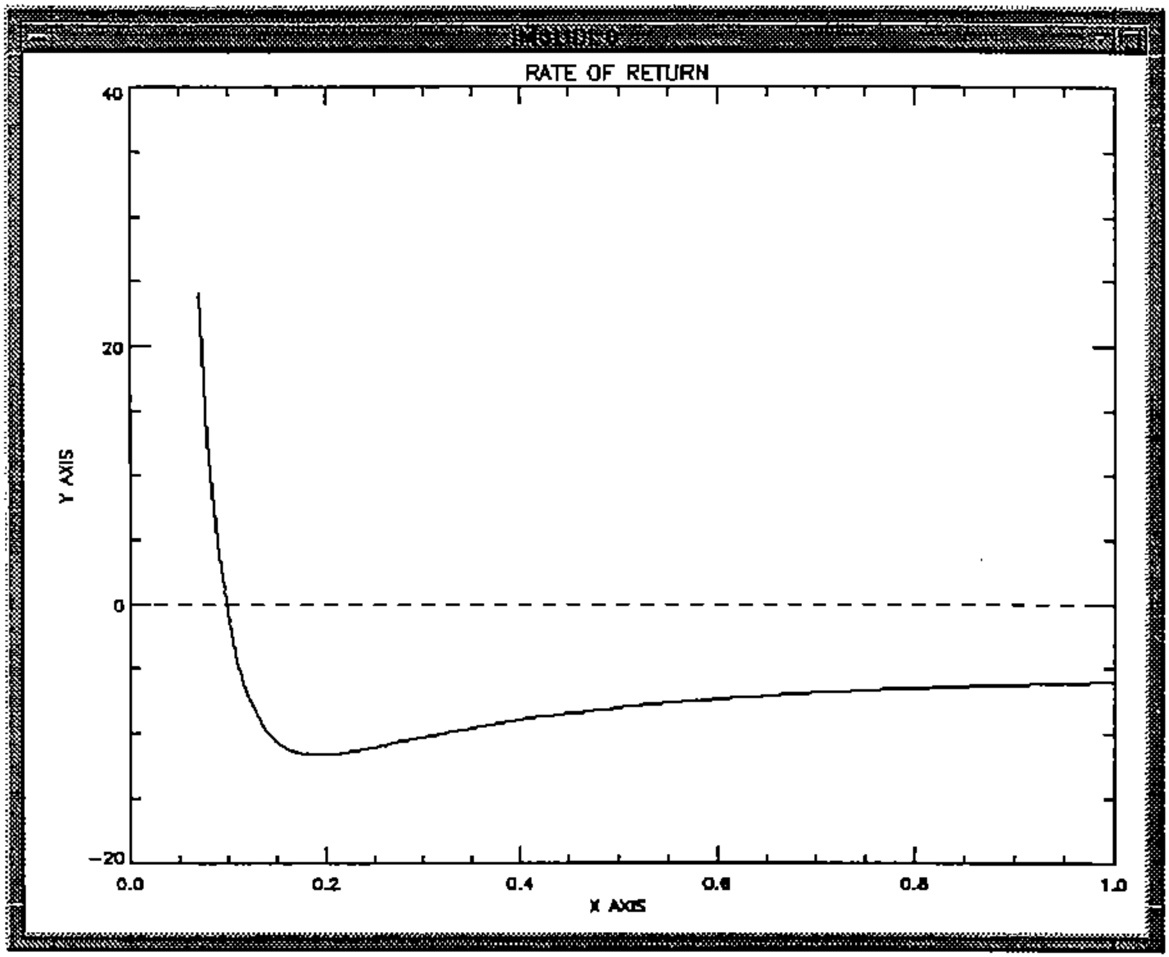
Both programs produce pretty much the same results, but the IMSL/IDL version is shorter than the IMSL Math Library version.

RATE OF RETURN ON AN INVESTMENT IN FORESTRY PRODUCTS

IMSL/IDL program:

```
function f1, x
   y = 1.0 + x
   return, 20.0/(y^15) + 36.0/(y^25) + 40.0/(y^33) + 475.0/(y^40) $
          -1.12*(y^40 - 1.0)/(x*(y^40)) - 6.0/(Y^4) - 3.0/(Y^8) - 4.5
end

function f2, x
   y = 1.0 + x
   return, 20.0*x*y^25 + 36.0*x*y^15 + 40.0*x*y^7 + 475.0*x $
          -1.12*(y^40-1) - 6*x*y^36 - 3*x*y^32 - 4.5*x*y^40
end

pro 188
    absc = findgen(94)/100.0 + 0.07
    ordi = f1(absc)
    plot, absc, ordi, title = "RATE OF RETURN", $
          xtitle = "X AXIS", ytitle = "Y AXIS", back = 255, color = 0
    plots, [0,1], [0,0], linestyle = 2, color = 0
    zero1 = zerofcn("f1", xguess = findgen(1)+0.1)
    print, "THE ZERO OF THE RATIONAL FUNCTION IS  ", zero1
    zero2 = zerofcn("f2", xguess = findgen(1)+0.1)
    print, "THE ZERO OF THE POLYNOMIAL FUNCTION IS", zero2
end
```

```
                         RESULTS
--------------------------------------------------------------------------------
THE ZERO OF THE RATIONAL FUNCTION IS      0.0986472
THE ZERO OF THE POLYNOMIAL FUNCTION IS    0.0986473
```



The corresponding Fortran program using the IMSL Math Library and Exponent Graphics is as follows:

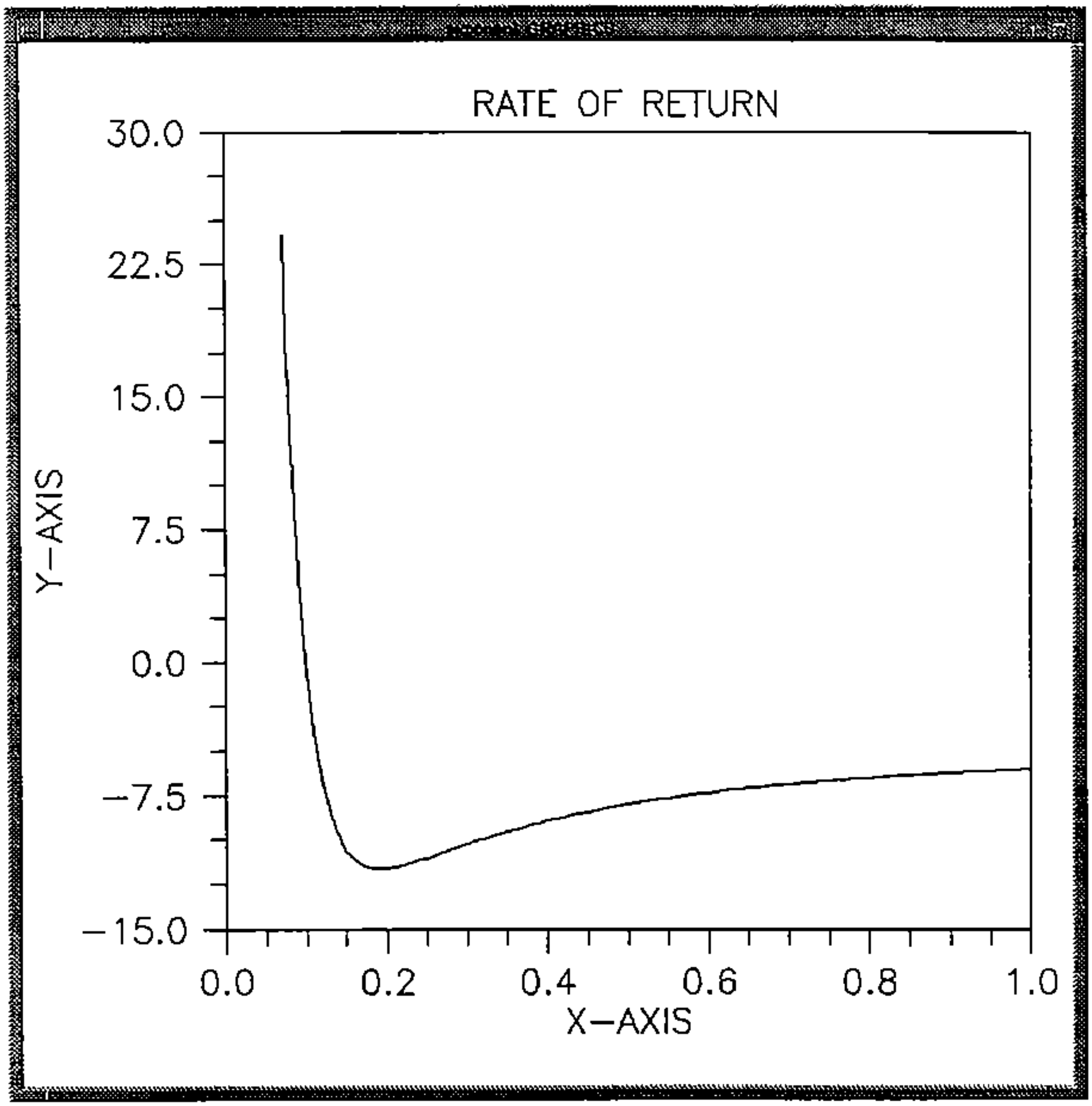
```
C                             LIBRARY APPLICATION 8.8
C      THE GIVEN RATIONAL FUNCTION HAS A POSITIVE ROOT X NEAR ZERO,
C      AFTER PLOTTING THE FUNCTION, WE TRANSFORM IT INTO A
C      POLYNOMIAL AND COMPARE THE NEW ROOT WITH THE PREVIOUS ONE.
C
C      NDATA  - NUMBER OF POINTS USED FOR PLOTTING
C      ABSC   - ABSCISSAE OF POINTS USED FOR PLOTTING
C      ORDI   - ORDINATES OF POINTS USED FOR PLOTTING
C      RANGE  - ENDPOINTS OF THE 2 AXIS
C      XGUESS - THE INITIAL GUESS OF THE ZERO OF THE POLYNOMIA
C      X      - THE ZERO OF THE POLYNOMIAL
C      INFO   - NUMBER OF ITERATIONS USED FOR FINDING THAT ZERO
C
       REAL ABSC(100),ORDI(100,1),RANGE(4),X(1),XGUESS(1),INFO(1)
       EXTERNAL PLOTP,ZREAL,F1,F2
C                                INITIALIZATIONS
       NDATA = 94
       NFUN = 1
       DO 10 I = 1,94
          ABSC(I) = 0.06 + I/100.0
```

```
         ORDI(I,1) = F1(ABSC(I))
 10    CONTINUE
C                            PLOT THE GIVEN FUNCTION.
C                     AFTER SETTING PAGE WIDTH = 76, DEPTH = 45
C      CALL PAGE(-1,76)
C      CALL PAGE(-2,45)
C      CALL PLOTP(NDATA,NFUN,ABSC,ORDI,100,1,RANGE,SYMBOL,'X AXIS',
C    *             'Y AXIS','RATE OF RETURN')
       CALL SCATR(94,ABSC,ORDI)
       CALL EGSGL('.1 use$', 'scatr8.d1$')
       CALL EFMPLT(1,1,1,IUNIT, ' ')
       XGUESS(1) = 0.1
C
C                     CALL THE NONLINEAR EQUATION SOLVER
C                       FOR THE GIVEN RATIONAL FUNCTION.
       CALL ZREAL(F1,1.0E-5,1.0E-5,1.0E-5,1.0E-2,1,100,XGUESS,X,INFO)
       WRITE(6,*)
       WRITE(6,*) 'THE ZERO OF THE RATIONAL FUNCTION IS   ', X(1)
C
C                      CALL THE NONLINEAR EQUATION SOLVER
C                         FOR THE POLYNOMIAL OBTAINED.
C
       CALL ZREAL(F2,1.0E-5,1.0E-5,1.0E-5,1.0E-2,1,100,XGUESS,X,INFO)
       WRITE(6,*) 'THE ZERO OF THE POLYNOMIAL FUNCTION IS ', X(1)
       END

      REAL FUNCTION F1(X)
C                            THE GIVEN RATIONAL FUNCTION.
C                          MODEL OF INVESTMENT RETURN IN FORESTRY
      REAL X,Y
      Y = 1 + X
      F1 = 20.0/(Y**15) + 36.0/(Y**25) + 40.0/(Y**33) + 475.0/(Y**40)
     *     -1.12*(Y**40 - 1)/(X*(Y**40)) - 6/(Y**4) - 3/(Y**8) - 4.5
      RETURN
      END
C
      REAL FUNCTION F2(X)
C                            THE POLYNOMIAL OBTAINED BY TRANSFORMING F1.
      REAL X,Y
      Y = 1 + X
      F2 = 20.0*X*Y**25 + 36.0*X*Y**15 + 40.0*X*Y**7 + 475.0*X
     *     -1.12*(Y**40-1) - 6*X*Y**36 - 3*X*Y**32 - 4.5*X*Y**40
      RETURN
      END
```

RESULTS

```
------------------------------------------------------------------------------
THE ZERO OF THE RATIONAL FUNCTION IS       9.86472E-02
```

```
THE ZERO OF THE POLYNOMIAL FUNCTION IS     9.86473E-02
```



Still the IMSL/IDL program is shorter than the program using the IMSL Math Library and Exponent Graphics.

PRESURE AND VELOCITY DISTRIBUTION FROM DIFFERENCE EQUATIONS

IMSL/IDL program:

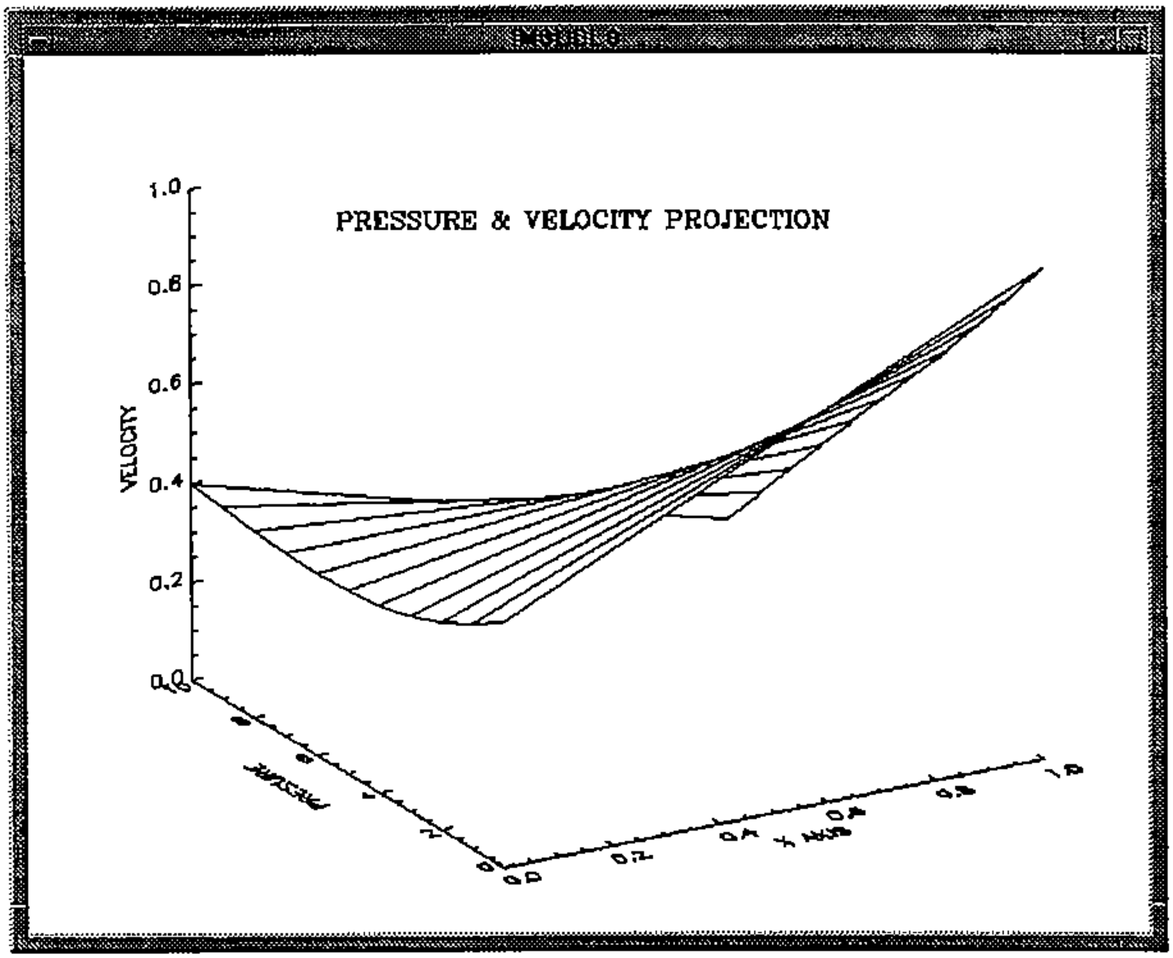
```
;       LIBRARY APPLICATION 9.2
;       CODED BY: XINGKANG FU
;
;       COMPUTE PRESURE AND VELOCITY DISTRIBUTIONS IN A MANIFOLD
;       AS MODELED BY A SYSTEM OF DIFFERENCE EQUATIONS
;
;       PLEFT(J)  = PRIGHT(J-1) - F(J-1/2) U(J-1/2)**2 DELTAX
;       U(J+1/2)  = CON1 (U(J-1/2) - SQRT( CON2 U(J-1/2)**2 + CON3 PLEFT(J) ))
;       PRIGHT(J) = PLEFT(J) + ( U(J-1/2)**2 - U(J+1/2)**2 )/2
;
;       WHERE
;         PLEFT(J), PRIGHT(J)   ARE THE PRESURES IN THE MANIFOLD TO THE
;                   LEFT AND TO THE RIGHT OF PORT J
;         U(J+1/2)        IS THE FLOW VELOCITY BETWEEN PORTS J AND J+1
;         F(J-1/2)        IS THE FLOW FRICTION FACTOR BETWEEN PORTS J-1 AND J
;         DELTAX          IS THE DISTANCE BETWEEN ADJACENT PORTS
;       AND
;         CON1 = 2 / ( 2 + DELTX )
;         CON2 = DELTAX**4 / 4
;         CON3 = DELTAX**2 ( 2 + DELTAX**2 )
;
;         THE FRICTION FACTOR IS TAKEN AS  F(J-1/2) = FO U(J-1/2)**(-1/4)
;         WHERE FO IS A CONSTANT
;
;       FOR INITIAL VELOCITY U(1/2) = 1 AND PRESURE PRIGHT(0) = MO**2/2,
;       THE VELOCITIES U(J+1/2) AND PRESURES P(J) ARE COMPUTED
;
;       IN THE LIMIT OF O DELTAX AND AN INFINITE NUMBER OF PORTS, THE
;       SYSTEM OF DIFFERENCE EQUATIONS BECOMES THE DIFFERENTIAL EQUATION
;       PROBLEM GIVEN BY
;
;          DP/DX = U SQRT( 2P ) - FO U**(7/4),     P(0) = MO**2/2
;          DU/DX = - SQRT( 2P ),                   U(0) = 1
;
;       WHERE
;         P   IS PRESSURE, STORED IN  Y(0)  IN THE PROGRAM
;         U   IS VELOCITY, STORED IN  Y(1)  IN THE PROGRAM
;         X   IS POSITION ALONG THE MANIFOLD
;
;       THE PROGRAM BELOW SOLVES BOTH OF THESE PROBLEMS
;
;
FUNCTION DERIV, T, Y
    YP = Y
    YP(1) = -SQRT(2.0*Y(0))
    YP(0) = -YP(1)*Y(1) - 1.5*ABS(Y(1))^(1.75)
```

```
    RETURN, YP
END
PRO L92
;
;   DECLARATION AND INITIALIZATION
;
    F0     = 1.5
    M0     = 1.0
    U0     = 1.0
    MAXSTP = 10
    PLEFT  = FLTARR( MAXSTP )
    PRIGHT = FLTARR( MAXSTP + 1 )
    U      = FLTARR( MAXSTP + 1 )
    DELTAX = 1./MAXSTP
    DELSQR = DELTAX^2
    CON1   = 2. / ( 2. + DELSQR )
    CON2   = DELSQR^2/4.
    CON3   = DELSQR*( 2. + DELSQR )
    U(0)      = 1.0
    PRIGHT(0) = 0.5
;
;   ASSIGN THE INITIAL VALUE
;
    Y = [M0^2/2.0, U0]
    T = FINDGEN( MAXSTP + 1 ) / MAXSTP
;
;   SOLVE THE ODE. USING DEFAULT VALUES EXCEPT TOLORANCE
;
    Y = ODE(T, Y, 'DERIV', TOL=0.0005, /R_K_V)
;
;   PRINT TITLE
;
    PRINT,'              |     DIFFERENTIAL EQUATION   |' $
         + '     DIFFERENCE   EQUATION     |'
    PRINT,'   J     X    |     P(X)       U(X)    -DU/DX |' $
         + '  PLEFT(J)    U(J)   PRIGHT(J) |'
    PRINT, 0, 0, Y(*,0), SQRT( 2. * Y(0,0)), $
    FORMAT='(I4,F8.3,(" |"),F8.3,F10.3,F10.3,(" |"),28x,("   |"))'
;
;   FIND SOLUTION OF DIFFERENCE EQUATION AND PRINT RESULT
;
    UDIEFE = SQRT( 2. * Y(0,*) )
    FOR ISTEP = 0, MAXSTP-1 DO BEGIN
        PLEFT(ISTEP)   = PRIGHT(ISTEP) - F0*ABS(U(ISTEP))^1.75*DELTAX
        U(ISTEP+1)     = CON1*( U(ISTEP) $
                       - SQRT( CON2 * U( ISTEP )^2 + CON3*PLEFT(ISTEP) ) )
        PRIGHT(ISTEP+1) = PLEFT(ISTEP) + ( U(ISTEP) - U(ISTEP+1) ) $
                                       * ( U(ISTEP) + U(ISTEP+1) ) / 2.
        PRINT, ISTEP+1, (ISTEP+1)*DELTAX, Y(*,ISTEP+1), UDIEFE(ISTEP+1), $
```

```
              PLEFT(ISTEP), U(ISTEP+1), PRIGHT(ISTEP+1), $
              FORMAT = '(I4, F8.3, (" |"), F8.3, F10.3, F10.3,' $
                     + '(" |"), F8.3, F10.3, F10.3, ("   |") )'
    ENDFOR
;
;   LOAD STAND GARMA II COLOR TABLE
;
    LOADCT,5
;
;   PRODUCE A 3D VIEW OF THE ODE. RESULT
;
    !P.MULTI = 0
    SURFACE, Y, BACKGROUND = 200, COLOR = 40, XTITLE = 'X AXIS', $
            YTITLE = 'PRESURE', ZTITLE = 'VELOCITY', CHARSIZE = 2
    XYOUTS, 0.5, 0.8, '!6PRESSURE & VELOCITY PROJECTION', $
            /NORMAL, COLOR = 99, CHARSIZE = 1.5, ALIGN = 0.5
END
```

RESULTS

		DIFFERENTIAL EQUATION			DIFFERENCE EQUATION		
J	X	P(X)	U(X)	-DU/DX	PLEFT(J)	U(J)	PRIGHT(J)
0	0.000	0.500	1.000	1.000			
1	0.100	0.455	0.902	0.954	0.350	0.911	0.435
2	0.200	0.421	0.809	0.918	0.307	0.829	0.379
3	0.300	0.397	0.718	0.891	0.271	0.751	0.333
4	0.400	0.381	0.630	0.873	0.242	0.678	0.294
5	0.500	0.373	0.544	0.863	0.218	0.608	0.263
6	0.600	0.371	0.457	0.861	0.200	0.542	0.238
7	0.700	0.375	0.371	0.866	0.187	0.478	0.219
8	0.800	0.382	0.284	0.874	0.178	0.417	0.205
9	0.900	0.390	0.196	0.884	0.173	0.356	0.197
10	1.000	0.398	0.107	0.892	0.172	0.295	0.192



The corresponding Fortran program using the IMSL Math Library and Exponent Graphics is as follows:

```
C COMPUTE PRESURE AND VELOCITY DISTRIBUTIONS IN A MANIFOLD
C AS MODELED BY A SYSTEM OF DIFFERENCE EQUATIONS
C
C PLEFT(J)  = PRIGHT(J-1) - F(J-1/2) U(J-1/2)**2 DELTAX
C U(J+1/2)  = CON1 (U(J-1/2) - SQRT( CON2 U(J-1/2)**2 + CON3 PLEFT(J) ))
C PRIGHT(J) = PLEFT(J) + ( U(J-1/2)**2 - U(J+1/2)**2 )/2
C
C WHERE
C   PLEFT(J), PRIGHT(J)   ARE THE PRESURES IN THE MANIFOLD TO THE
C                  LEFT AND TO THE RIGHT OF PORT J
C   U(J+1/2)       IS THE FLOW VELOCITY BETWEEN PORTS J AND J+1
C   F(J-1/2)       IS THE FLOW FRICTION FACTOR BETWEEN PORTS J-1 AND J
C   DELTAX         IS THE DISTANCE BETWEEN ADJACENT PORTS
C AND
C   CON1 = 2 / ( 2 + DELTX )
C   CON2 = DELTAX**4 / 4
C   CON3 = DELTAX**2 ( 2 + DELTAX**2 )
C
C   THE FRICTION FACTOR IS TAKEN AS  F(J-1/2) = FO U(J-1/2)**(-1/4)
C   WHERE FO IS A CONSTANT
```

```
C
C FOR INITIAL VELOCITY U(1/2) = 1 AND PRESURE PRIGHT(0) = MO**2/2,
C THE VELOCITIES U(J+1/2) AND PRESURES P(J) ARE COMPUTED
C
C IN THE LIMIT OF 0 DELTAX AND AN INFINITE NUMBER OF PORTS, THE
C SYSTEM OF DIFFERENCE EQUATIONS BECOMES THE DIFFERENTIAL EQUATION
C PROBLEM GIVEN BY
C
C     DP/DX = U SQRT( 2P ) - FO U**(7/4),       P(0) = MO**2/2
C     DU/DX = - SQRT( 2P ),                     U(0) = 1
C
C WHERE
C    P    IS PRESSURE, STORED IN  Y(1)  IN THE PROGRAM
C    U    IS VELOCITY, STORED IN  Y(1)  IN THE PROGRAM
C    X    IS POSITION ALONG THE MANIFOLD
C
C THE PROGRAM BELOW SOLVES BOTH OF THESE PROBLEMS
C
      PARAMETER (MAXPAR = 50, NEQN = 2)
      REAL      MO, PARAM(MAXPAR), Y(NEQN)
      REAL      PRIGHT(50), PLEFT(50), U(50)
      COMMON    FO
      EXTERNAL  DERIV
C                              SET TOLERENCE FOR IVPRK AND CONSTANTS
C                              FOR THE PROBLEM
      DATA TOL / 0.0005/, FO / 1.5/, MO / 1.0/, UO / 1.0/
C
C                              SET OUTPUT UNIT NUMBER
      CALL UMACH( 2, NOUTPT )
C                              SET DEFAULT VALUES OF PARAM (USED BY IVPRK)
      CALL SSET( MAXPAR, 0.0, PARAM, 1 )
C
C                              SET INITIAL CONDITIONS AND CONSTANTS
      X         = 0.0
      Y(1)      = MO**2 / 2.
      Y(2)      = UO
      MAXSTP    = 10
      DELTAX    = 1./MAXSTP
      DELSQR    = DELTAX**2
      CON1      = 2. / ( 2. + DELSQR )
      CON2      = DELSQR**2/4.
      CON3      = DELSQR*( 2. + DELSQR )
      PRIGHT(1) = Y(1)
      U(1)      = Y(2)
      PPLOT(1) = Y(1)
      UPLOT(1) = Y(2)
C
      WRITE( NOUTPT, 1000 ) 0, X, Y, SQRT( 2. * Y(1) )
 1000 FORMAT( 13X,'|',5X,'DIFFERENTIAL EQUATION',3X,'|',
```

```
     A  5X,'DIFFERENCE   EQUATION',5X,'|'/
     B  3X,'J',5X,'X',3X,'|',4X,'P(X)',6X'U(X)',4X,'-DU/DX',
     C  ' | ','PRIGHT(J)',4X,'U(J)',4X,'PLEFT(J) |'/
     D  I4,F8.3,' |',F8.3,2F10.3,' |',31X,'|')
      IDO = 1
      DO 1030 ISTEP = 1, MAXSTP
        XEND   = ISTEP * DELTAX
C                               FIND SOLUTION OF DIFFERENTIAL EQUATIONS
C                               AT NEXT TIME STEP
        CALL IVPRK( IDO, NEQN, DERIV, X, XEND, TOL, PARAM, Y )
C                               FIND SOLUTION OF DIFFERENCE EQUATION AT
C                               NEXT TIME STEP
        PPLOT(ISTEP+1) = Y(1)
        UPLOT(ISTEP+1) = Y(2)
        PLEFT(ISTEP) = PRIGHT(ISTEP) - FO*ABS( U(ISTEP) )**1.75*DELTAX
        U(ISTEP+1)    = CON1*( U(ISTEP)
     A                  - SQRT( CON2 * U(ISTEP)**2 + CON3*PLEFT(ISTEP) ) )
        PRIGHT(ISTEP+1) = PLEFT(ISTEP) + ( U(ISTEP) - U(ISTEP+1) )
     A                                  * ( U(ISTEP) + U(ISTEP+1) ) / 2.
        UDIFFE = SQRT( 2. * Y(1) )
        WRITE( NOUTPT, 1010 ) ISTEP, X, Y, UDIFFE,
     A                          PLEFT(ISTEP), U(ISTEP+1), PRIGHT(ISTEP+1)
 1010   FORMAT(I4,F8.3,' |',F8.3,2F10.3,' |',F8.3,2F10.3,'   |')
        IF( UDIFFE .LT. 0. .OR. U(ISTEP+1) .LT. 0. ) THEN
          WRITE( NOUTPT, 1020 )
 1020     FORMAT(/' *** EXECUTION TERMINATED BECAUSE OF NEGATIVE ',
     A      'VELOCITY')
                                                              GO TO 1040
        END IF
 1030 CONTINUE
C
 1040 CONTINUE
C
      IDO = 3
      CALL IVPRK( IDO, NEQN, DERIV, X, XEND, TOL, PARAM, Y)
      STOP
      END
      SUBROUTINE DERIV( NEQN, X, Y, YPRIME)
C  EVALUATE RIGHT SIDES OF DIFFERENTIAL EQUATIONS
C  Y(1) IS PRESSURE  P,  YPRIME(1) IS DP/DX
C  Y(2) IS VELOCITY  U,  YPRIME(2) IS DU/DX
      REAL Y(NEQN), YPRIME(NEQN)
      COMMON    FO
C
      YPRIME(2) = - SQRT( 2. * Y(1) )
      YPRIME(1) = - YPRIME(2) * Y(2) - FO * ABS( Y(2) ) ** 1.75

      RETURN
      END
```

RESULTS

J	X	P(X)	U(X)	-DU/DX	PRIGHT(J)	U(J)	PLEFT(J)
		DIFFERENTIAL EQUATION			DIFFERENCE EQUATION		
0	0.000	0.500	1.000	1.000			
1	0.100	0.455	0.902	0.954	0.350	0.911	0.435
2	0.200	0.421	0.809	0.918	0.307	0.829	0.379
3	0.300	0.397	0.718	0.891	0.271	0.751	0.333
4	0.400	0.381	0.630	0.873	0.242	0.678	0.294
5	0.500	0.373	0.544	0.863	0.218	0.608	0.263
6	0.600	0.371	0.457	0.861	0.200	0.542	0.238
7	0.700	0.375	0.371	0.866	0.187	0.478	0.219
8	0.800	0.382	0.284	0.874	0.178	0.417	0.205
9	0.900	0.390	0.196	0.884	0.173	0.356	0.197
10	1.000	0.398	0.107	0.892	0.172	0.295	0.192

IMSL/IDL provides the *surface* command to view the result in 3 dimensions which is very convenient. Although Exponent Graphics has the surface facility(FNP3D and EF3PLT), it is not as convenient as that of IMSL/IDL, and sometimes it is not even feasible to use the surface facility of Exponent Graphics. For example, the above IMSL/IDL program uses one statement:

```
SURFACE, Y, BACKGROUND = 200, COLOR = 40, XTITLE = 'X AXIS', $
        YTITLE = 'PRESURE', ZTITLE = 'VELOCITY', CHARSIZE = 2
```

to show the 3D picture. But it seems to me that the surface facility(FNP3D) of Exponent Graphics is not suitable in the above Fortran program. The syntax of FNP3D is

```
FNP3D(FCN, ISHADE, NX, NY, AX, BX, AY, BY).
```

Where AX, BX, AY, BY are left edge, right edge, bottom edge and top edge, respectively, of the domain and FCN(X,Y) is a function to be plotted. NX and NY are number of points in the X and Y direction at which the function is to be evaluated. In the above example, we already have the values of Y and Z and there is no simple function to describe the relations of the pressure and velocity, thus it is not possible to use FNP3D in this example. My observation is that the 3D facilities of IMSL/IDL are more powerful and easier to use than those of Exponent Graphics. If we have the function FCN(X,Y), we just evaluate FCN and use the *surface* command of IMSL/IDL. But having the data, it is not always easy to construct a corresponding FCN.

This program runs correctly if it does not follow the runing of IMSL/IDL program 155. If 155 runs first and then one tries to run this program, IMSL/IDL gives bunch of error messages. This occurs because in 155, Y is defined as a function and in 192 Y is defined as an array. In other words, within one IMSL/IDL session, the name used remains effective and one has to use different names for different objects even though they are in different programs, or one has to exit the IMSL/IDL session and enter another IMSL/IDL session. This is not desirable. If IMSL/IDL can provide a function which eliminates the effect of names defined in previous programs, one does not need to exit the IMSL/IDL session before running another program, one may simply invoke this function and remove the effect of program ran before.

The following program reveals a bug in IMSL/IDL.

```
function der, t, y
   return, [-2.0*t*y(0)*y(1), -1.0/(t*y(0)*exp(2*t))]
end
```

```
pro 197
    y = [0.1353352832, 1.0]
    t = findgen(4)+1.0
    y = ode(t, y, 'der', tol=0.0005, hinit=0.01, /r_k_v)
    print, y
end
```

The current version of IMSL/IDL can not solve problems which are not autonomous, i.e. the right-hand side does depend explicitly on t(time). This bug has been reported to IMSL and the bug is to be corrected for the Î2 beta version 2.0.0 of IMSL/IDL. One shortcoming of IMSL/IDL is that the documents do not provide a good variety of examples. For instance, no example for the function ODE ever uses the parameter t. If the documents includes more variety of examples, this problem might have been discovered before its release and certainly more variety of examples will help the users.

ANIMATION EXAMPLE

The animation facility provided by the IMSL/IDL allows one to examine the data and results of computation visually and dynamically. The facility to read in picture data and reproduce it as an animation is very efficient. The following program uses the animation facility.

```
FUNCTION L92DERIV, T, Y
    YP = Y
    YP(1) = -SQRT(2.0*Y(0))
    YP(0) = -YP(1)*Y(1) - 1.5*ABS(Y(1))^(1.75)
    RETURN, YP
END

PRO CR92ANI
;
;   DECLARATION AND INITIALIZATION
;
    MAXSTP = 10
    T = FINDGEN( MAXSTP + 1 ) / MAXSTP
    ANIMATE = BYTARR(320,256,10)
;
;   LOAD STAND GARMA II COLOR TABLE
;
    LOADCT,5
;
    FOR I = 1, MAXSTP DO BEGIN
;
;       ASSIGN THE INITIAL VALUE
;
        U0 = 1.0*I
        M0 = 1.0*I
        Y = [M0^2/2.0, U0]
;
;       SOLVE THE ODE. USING DEFAULT VALUES EXCEPT TOLORANCE
;
        Y = ODE(T, Y, 'L92DERIV', TOL=0.0005, /R_K_V)
;
;       PRODUCE A 3D VIEW OF THE ODE. RESULT
;
    WINDOW, /FREE, XSIZE = 320, YSIZE = 256
        SURFACE, Y, BACKGROUND = 200, COLOR = 40, XTITLE = 'X AXIS', $
                 YTITLE = 'PRESURE', ZTITLE = 'VELOCITY', CHARSIZE = 1.5
        XYOUTS, 0.6, 0.9, '!6PRESSURE & VELOCITY PROJECTION', $
                /NORMAL, COLOR = 99, ALIGN = 0.5
        ANIMATE(*,*,I-1) = TVRD()
    ENDFOR
    OPENW, LUN, 'ANI92.DAT', /GET_LUN
    WRITEU, LUN, ANIMATE(*,*,*)
    FREE_LUN, LUN
END
```

This program produces 10 pictures and saves them into a file.

```
PRO ANI92
DISPLAY = BYTARR(320,256,10)
OPENR, LUN, 'ANI92.DAT', /GET_LUN
READU, LUN, DISPLAY
FREE_LUN, LUN
WINDOW, XSIZE = 320, YSIZE = 256, /FREE
MOVIE, DISPLAY, ORDER = 0
END
```

And this program produces the animation using the data created by the previous program.

SOLVE AN ELLIPTIC PROBLEM USING ORDINARY FINITE DIFFERENCES

IMSL/IDL program:

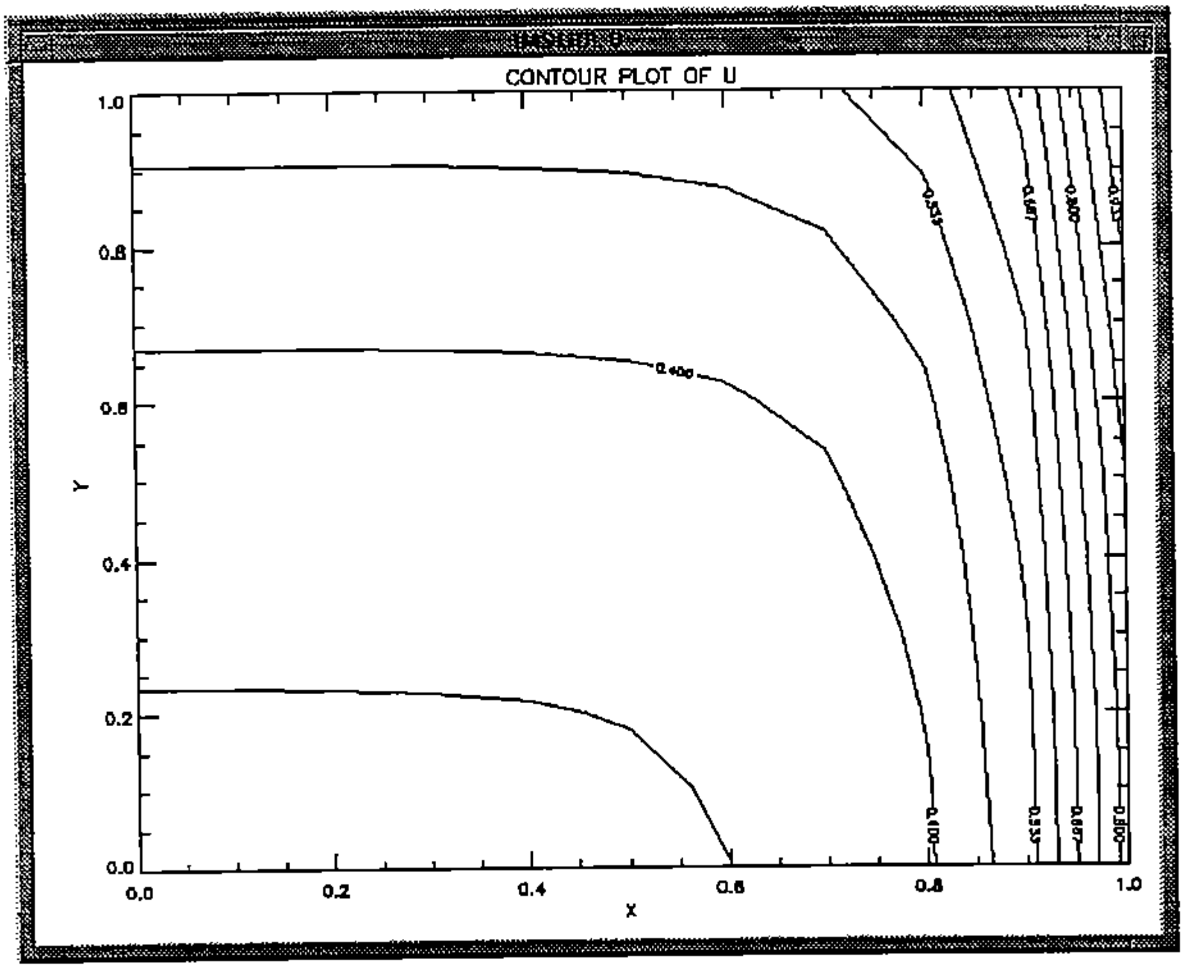
```
function bound, idx, ngrid, len
    if (idx le ngrid) then return, 1
    if (idx ge (len-ngrid)) then return, 1
    if ((idx mod ngrid) eq 0) then return, 1
    if ((idx mod ngrid) eq 1) then return, 1  else return, 0
end

function pdeval, x, y, ngrid
    pt = (y - 1.0)/(ngrid - 1.0)
    return, -49.5*cosh(pt)/cosh(1.0)
end

function bval, x, y, ngrid
    xpt = (x - 1.0)/(ngrid - 1.0)
    ypt = (y - 1.0)/(ngrid - 1.0)
    return, 0.5*(cosh(10.0*xpt)/cosh(10.0) + cosh(ypt)/cosh(1.0))
end

pro 1101
    ngrid = 11
    len = ngrid*ngrid
    a = fltarr(len,len)
    b = fltarr(len)
    uout = fltarr(len)
    u = fltarr(ngrid,ngrid)

    space = 1.0/(ngrid - 1)
    star = 1.0/(space*space)
    space2 = -100.0 - star*4.0
    for y = 1, ngrid do begin
       for x = 1, ngrid do begin
          idx = ngrid*(y-1) + x
          if (bound(idx,ngrid,len)) then begin
             a(idx-1,idx-1) = 1.0
             b(idx-1) = bval(x,y,ngrid)
          endif else begin
             if (idx+ngrid-1 lt len) then a(idx-1, idx+ngrid-1) = star
             a(idx-1, idx) = star
             a(idx-1, idx-1) = space2
             if (idx-2 ge 0) then a(idx-1, idx-2) = star
             if (idx-ngrid-1 ge 0) then a(idx-1, idx-ngrid-1) = star
             b(idx-1) = pdeval(x,y,ngrid)
          endelse
       endfor
    endfor
    uout = lusol(b,a)
```

```
    print,'VALUES ON THE GRID'
    for i = ngrid-1, 0, -1 do begin
        u(*,i) = uout(i*ngrid:(i+1)*ngrid-1)
        print, float(i)/10, uout(i*ngrid:(i+1)*ngrid-1), $
               format = '(/,f3.1,("I"),12(f6.3))'
        print, '   I'
    endfor
    print,'   I----------------------------------------', $
          + '------------------------'
    print,findgen(11)/10.0, format = '(3x,("I"),12(f6.1))'
    contour,u,findgen(11)/10.0, findgen(11)/10.0, nlevels = 11, back=255, $
            color = 0, /follow, xtitle = 'X', ytitle = 'Y', $
            title = 'CONTOUR PLOT OF U'
end
```

RESULTS

```
--------------------------------------------------------------------------------
VALUES ON THE GRID

1.0I 0.500 0.500 0.500 0.500 0.501 0.503 0.509 0.525 0.568 0.684 1.000
   I

0.9I 0.464 0.464 0.465 0.465 0.466 0.468 0.474 0.491 0.536 0.653 0.964
   I

0.8I 0.433 0.433 0.434 0.434 0.435 0.437 0.444 0.461 0.506 0.624 0.933
   I

0.7I 0.407 0.407 0.407 0.407 0.408 0.411 0.417 0.434 0.479 0.598 0.907
   I

0.6I 0.384 0.384 0.384 0.385 0.386 0.388 0.395 0.412 0.457 0.575 0.884
   I

0.5I 0.365 0.365 0.366 0.366 0.367 0.369 0.376 0.393 0.438 0.556 0.865
   I

0.4I 0.350 0.350 0.351 0.351 0.352 0.354 0.361 0.378 0.423 0.541 0.850
   I

0.3I 0.339 0.339 0.339 0.339 0.340 0.343 0.349 0.366 0.411 0.530 0.839
   I

0.2I 0.331 0.331 0.331 0.331 0.332 0.334 0.341 0.358 0.403 0.521 0.831
   I

0.1I 0.326 0.326 0.326 0.326 0.327 0.329 0.336 0.352 0.397 0.515 0.826
   I
```

```
0.0I 0.324 0.324 0.324 0.324 0.325 0.327 0.333 0.349 0.392 0.508 0.824
   I
   I-------------------------------------------------------------------
   I   0.0   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9   1.0
```



IMSL/IDL dose not have a band storage format, so it needs a large array to store the coefficient matrix, although the array has a lot of zero entries. This is not desirable especially if the coefficient matrix is very large. The IMSL/IDL's library routines are simple but do not have as much variety as the IMSL Math Library. IMSL/IDL has various array operations which make our output easier than the corresponding Fortran program. In order to print and plot the U values, we need to change the one dimensional array *uout* to a two dimensional array *u*. Instead of using a loop, we can simply specify the column index of *u* and the subrange of *uout*.

```
        u(*,i) = uout(i*ngrid:(i+1)*ngrid-1)
        print, float(i)/10, uout(i*ngrid:(i+1)*ngrid-1), $
               format = '(/,f3.1,("I"),12(f6.3))'
```

The corresponding ELLPACK program is as follows:

```
equation.       uxx + uyy - 100.0*u = -49.5/cosh(1.0)*cosh(y)
boundary.       u = true(0.0,y) on x = 0
```

```
               u = true(1.0,y) on x = 1
               u = true(x,0.0) on y = 0
               u = true(x,1.0) on y = 1
grid.          11 x points $ 11 y points
dis.           5 point star
sol.           band ge
out.           table(u) $ plot(u)

subprograms.
               function true(x,y)
               real x, y
               true = 0.5*(cosh(10.0*x)/cosh(10.0) + cosh(y)/cosh(1.0))
               return
               end
end.
```

RESULTS

```
--------------------------------------------------------------------------------
 Start Interactive ELLPACK session
1
 -----------------------
  discretization module
 -----------------------



    5 - p o i n t   s t a r

    domain                           rectangle
    discretization                     uniform
    number of equations                     81
    max no. of unknowns per eq.              5
    matrix is                        symmetric

    execution successful




 -----------------
  solution module
 -----------------

   b a n d   g e

   number of equations       81
   lower bandwidth            9
   upper bandwidth            9
   required workspace      2349
```

```
   execution successful
1
 ----------------
  ellpack output
 ----------------



          +++++++++++++++++++++++++++++++++++++++++++++
          +                                           +
          +    table of u      on  11 x  11  grid     +
          +                                           +
          +++++++++++++++++++++++++++++++++++++++++++++


     x-abscissae are
     ---------------
     0.000000E+00    1.000000E-01    2.000000E-01    3.000000E-01
     4.000000E-01    5.000000E-01    6.000000E-01    7.000000E-01
     8.000000E-01    9.000000E-01    1.000000E+00

        y = 1.000000E+00
        ----------------
     5.000454E-01    5.000700E-01    5.001708E-01    5.004570E-01
     5.012398E-01    5.033692E-01    5.091579E-01    5.248935E-01
     5.676677E-01    6.839397E-01    1.000000E+00

        y = 9.000000E-01
        ----------------
     4.644043E-01    4.644398E-01    4.645576E-01    4.648822E-01
     4.657558E-01    4.680960E-01    4.743540E-01    4.910576E-01
     5.355291E-01    6.534743E-01    9.643589E-01

        y = 8.000000E-01
        ----------------
     4.334106E-01    4.334520E-01    4.335796E-01    4.339254E-01
     4.348457E-01    4.372859E-01    4.437430E-01    4.607957E-01
     5.057303E-01    6.238284E-01    9.333652E-01

        y = 7.000000E-01
        ----------------
     4.067542E-01    4.067987E-01    4.069315E-01    4.072878E-01
     4.082300E-01    4.107134E-01    4.172474E-01    4.344157E-01
     4.794667E-01    5.975407E-01    9.067088E-01

        y = 6.000000E-01
        ----------------
```

3.841683E-01	3.842142E-01	3.843493E-01	3.847104E-01
3.856619E-01	3.881619E-01	3.947231E-01	4.119269E-01
4.570049E-01	5.750576E-01	8.841228E-01	

y = 5.000000E-01

3.654268E-01	3.654731E-01	3.656089E-01	3.659712E-01
3.669251E-01	3.694295E-01	3.759971E-01	3.932086E-01
4.382916E-01	5.563379E-01	8.653814E-01	

y = 4.000000E-01

3.503422E-01	3.503879E-01	3.505230E-01	3.508841E-01
3.518355E-01	3.543354E-01	3.608966E-01	3.781003E-01
4.231784E-01	5.412312E-01	8.502967E-01	

y = 3.000000E-01

3.387634E-01	3.388076E-01	3.389402E-01	3.392966E-01
3.402388E-01	3.427221E-01	3.492560E-01	3.664243E-01
4.114753E-01	5.295494E-01	8.387181E-01	

y = 2.000000E-01

3.305747E-01	3.306157E-01	3.307431E-01	3.310888E-01
3.320092E-01	3.344494E-01	3.409063E-01	3.579590E-01
4.028937E-01	5.209920E-01	8.305293E-01	

y = 1.000000E-01

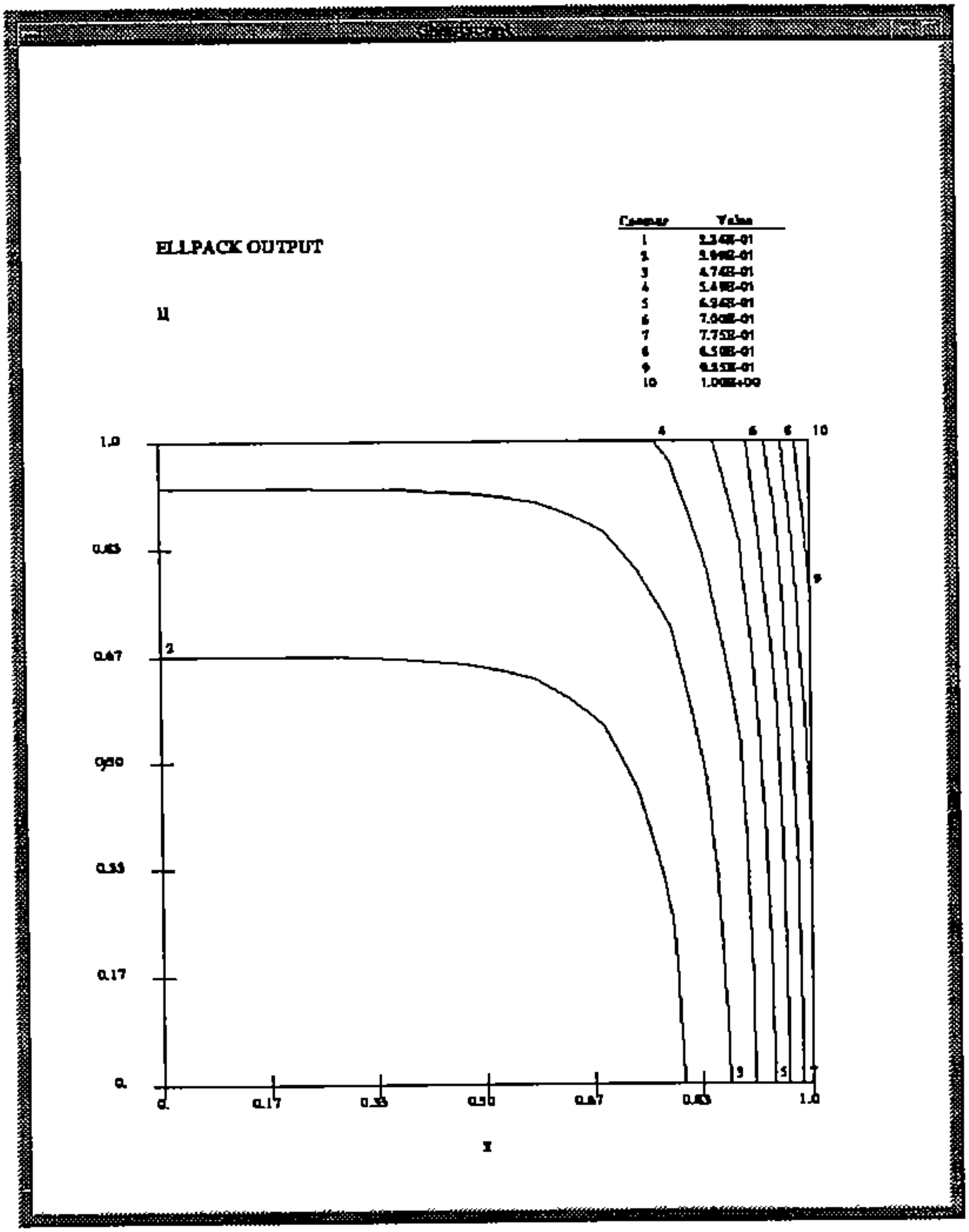
3.256940E-01	3.257292E-01	3.258467E-01	3.261714E-01
3.270450E-01	3.293851E-01	3.356430E-01	3.523466E-01
3.968182E-01	5.147637E-01	8.256486E-01	

y = 0.000000E+00

3.240725E-01	3.240972E-01	3.241979E-01	3.244842E-01
3.252669E-01	3.273962E-01	3.331850E-01	3.489207E-01
3.916948E-01	5.079668E-01	8.240271E-01	

----
ellpack output
----

contour plot of u
grid 20 by 20
execution successful



ELLPACK is designed to solve partial different equations, it is very convenient to specify the equation, the boundary conditions, discretization methods, solvers, and the output. It has a varity of discretization methods and solvers. No wonder the ELLPACK program is much shorter than both of the IMSL/IDL program and the Fortran program using the IMSL Math Library and Exponent Graphics.

The corresponding Fortran program using the IMSL Math Library and Exponent Graphics is as follows:

```
C                       LIBRARY APPLICATION 10.1
C  APPLY A 5-POINT STAR FINITE DIFFERENCE METHOD TO SOLVE AN ELLIPTIC PROBLEM
C  IN THE UNIT SQUARE 0<= X,Y <=1 WITH GRID SPACING OF 1/10.  SOLVE THE SYSTEM
C  WITH BAND MATRIX FACTORING USING LFTRB AND BAND MATRIX SOLVING USING LFTIRB
C  WHICH USES ITERATIVE REFINEMENT.
```

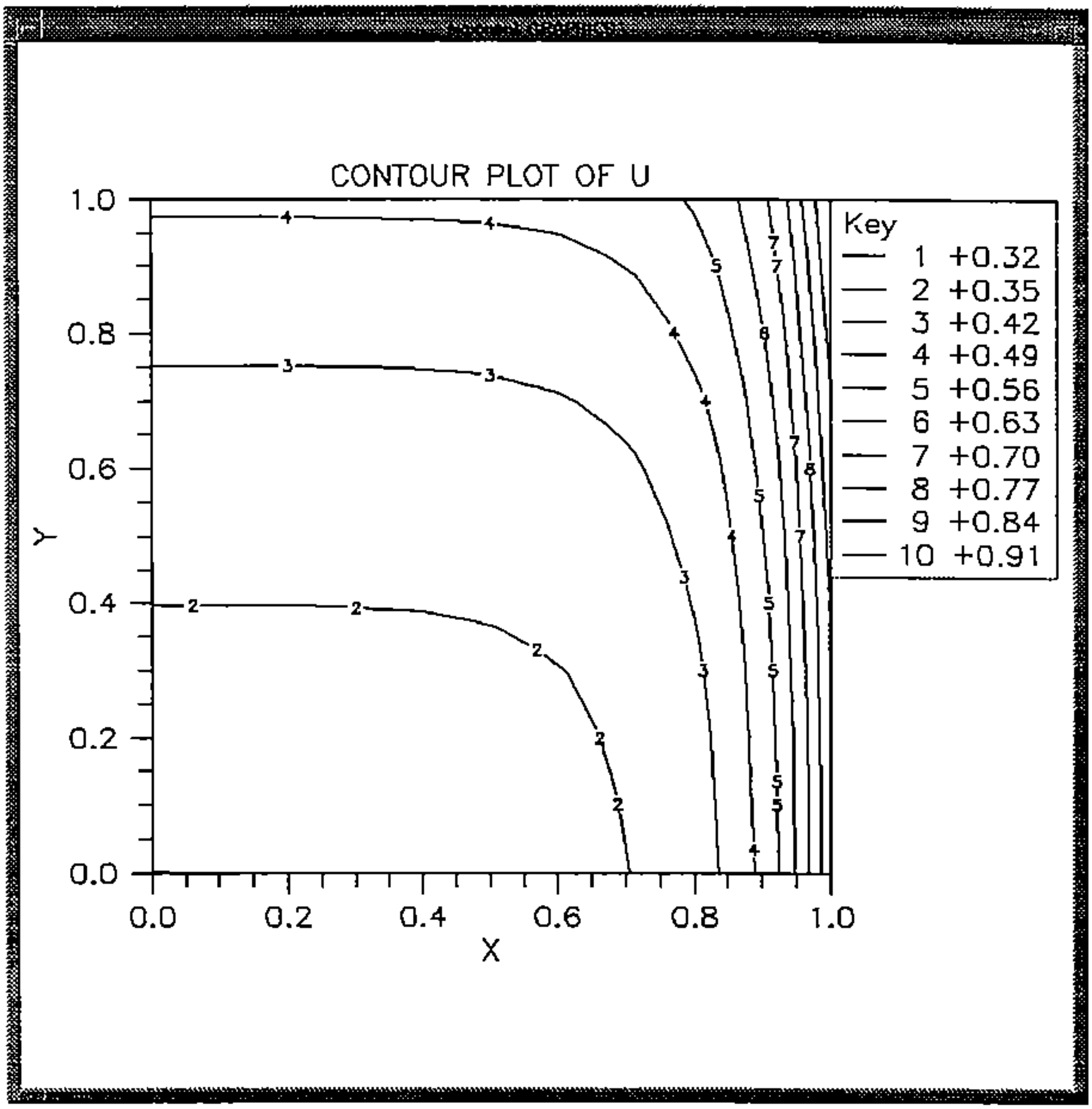
```
C
C  INPUT:
C    BVAL    - FUNCTION TO CALCULATE VALUES FOR BOUNDARY CONDITIONS ON THE GRID
C    PDEVAL  - FUNCTION TO CALCULATE B-VECTOR(RIGHT SIDE OF PDE) IN AU=B
C    BOUND   - FUNCTION TO CHECK IF AN (X,Y) POINT IS ON THE GRID BOUNDARY
C    A       - BAND MATRIX OF 5-POINT STAR COEFFICIENTS
C    B       - VECTOR OF KNOWN VALUES IN AX=B
C    STAR    - COEFFICIENT ON FINGERS OF THE 5-POINT STAR
C    SPACE2  - COEFFICIENT OF THE MIDDLE OF THE 5-POINT STAR
C    SPACE   - SPACING BETWEEN GRID POINTS
C    NROWS   - COUNT OF DIAGONAL AN UPPER AN LOWER CODIAGONALS
C    NGRID   - NUMBER OF GRID POINTS ON X AND Y AXIS
C    IDX     - INDEX OF EACH GRID POINT. (0,0) IS 1, (0,1) IS 2, ETC.
C    X       - X AXIS VALUE
C    Y       - Y AXIS VALUE
C    FAC     - WORK SPACE ARRAY FOR BAND MATRIX SOLVERS
C
C  OUTPUT:
C    UOUT    - VECTOR OF VALUES ON THE GRID.  UOUT IS U IN AU=B
C    RES     - RESIDUAL VECTOR AT IMPROVED SOLUTION FROM ITERATIVE REFINEMENT
C
C-----------------------------------------------------------------------------
      PARAMETER (NGRID=11, LEN=NGRID*NGRID, LDFAC=3*NGRID+1)
C
      REAL A(-NGRID:NGRID, 1:LEN), B(LEN), STAR, SPACE2, FAC(LDFAC,LEN)
      REAL IPVT(LEN), UOUT(LEN), RES(LEN), SPACE
      INTEGER X, Y, IDX, NROWS
      LOGICAL BOUND
      EXTERNAL LFTRB, LFIRB
C                         INITIALIZE COMPUTATION
      SPACE = 1.0/(NGRID - 1)
      STAR = 1.0/(SPACE*SPACE)
      SPACE2 = -100.0 - STAR*4.0
C                         LOOP OVER GRID POINTS
C                    PUT VALUES IN MATRIX A, VECTOR B
      NROWS = 2*NGRID + 1
      DO 10 Y = 1, NGRID
        DO 10 X = 1, NGRID
          IDX = NGRID*(Y-1) + X
          IF (BOUND(IDX,NGRID,LEN)) THEN
              A(0,IDX) = 1.0
              B(IDX) = BVAL(X,Y,NGRID)
            ELSE
              A(-NGRID, IDX+NGRID) = STAR
              A(-1, IDX+1) = STAR
              A(0, IDX) = SPACE2
              A(1, IDX-1) = STAR
              A(NGRID, IDX-NGRID) = STAR
              B(IDX) = PDEVAL(X,Y,NGRID)
```

```
            ENDIF
10      CONTINUE
        CALL LFTRB(LEN, A, NROWS, NGRID, NGRID, FAC, LDFAC, IPVT)
        CALL LFIRB(LEN, A, NROWS, NGRID, NGRID, FAC, LDFAC,
     &                   IPVT, B, 1, UOUT, RES)
C                        PRINT VALUES OF SOLUTION U
        CALL TABLE(LEN, NGRID, SPACE, UOUT)
        STOP
        END
C
        SUBROUTINE TABLE(LEN, NGRID, H, UOUT)
        parameter(ncv = 10, g = 11)
        INTEGER I, J, LEN, NGRID
        REAL UOUT(LEN), H
        real u(g,g)
        real grid(g)
        real cval(ncv)
C                          GET OUTPUT DEVICE NUMBER
        CALL UMACH(2,NOUT)
        WRITE(NOUT, '(''VALUES ON THE GRID'')')
        J = LEN - NGRID + 1
        AXIS = H*NGRID - H
        DO 30 N=1, NGRID
          grid(n) = (n-1)*0.1
          do 31 i = j,j+ngrid-1
             u(i-j+1,ngrid+1-n) = uout(i)
31        continue
          I = J
          WRITE(NOUT,'(/,F3.1,''I'',12(F6.3))')
     *             AXIS,(UOUT(I),I=J,J+NGRID-1)
          AXIS = AXIS - H
          J = J - NGRID
          WRITE (NOUT,'(3X,''I'')')
30      CONTINUE
        WRITE(NOUT,'(3X,''I---------------------------------------'',
     *             ''--------------------------'')')
        AXIS = 0.0
        WRITE(NOUT,'(3X,''I'',12(F6.1))')
     &     (AXIS+(I-1)*H,I=1,NGRID)
        call grctr(ngrid,ngrid,grid,grid,u,ngrid,5,ncv,cval)
        call egsgl('.1 use$', 'grctr1.d1$')
        call efsplt(0,' ')
        RETURN
        END
C
        LOGICAL FUNCTION BOUND(IDX,NGRID,LEN)
C                          CHECK (X,Y) ON BOUNDARY
        INTEGER IDX, NGRID, LEN
        IF (IDX .LE. NGRID) THEN
```

```
          BOUND = .TRUE.
        ELSE IF( IDX .GE. LEN-NGRID    ) THEN
          BOUND = .TRUE.
        ELSE IF( MOD(IDX,NGRID) .EQ. 0 ) THEN
          BOUND = .TRUE.
        ELSE IF( MOD(IDX,NGRID) .EQ. 1 ) THEN
          BOUND = .TRUE.
        ELSE
          BOUND = .FALSE.
      ENDIF
      RETURN
      END
C
      REAL FUNCTION PDEVAL(X,Y,NGRID)
C                                  RIGHT SIDE OF PDE
      INTEGER X,Y,NGRID
      REAL PT
      PT = (Y - 1.0)/(NGRID - 1.0)
      PDEVAL = -49.5*COSH(PT)/COSH(1.0)
      RETURN
      END
C
      REAL FUNCTION BVAL(X,Y,NGRID)
C                                   BOUNDARY VALUES
      INTEGER X,Y,NGRID
      REAL XPT, YPT
      XPT = (X - 1.0)/(NGRID - 1.0)
      YPT = (Y - 1.0)/(NGRID - 1.0)
      BVAL = 0.5*(COSH(10.0*XPT)/COSH(10.0) + COSH(YPT)/COSH(1.0))
      RETURN
      END
```

```
                              RESULTS
--------------------------------------------------------------------------------
VALUES ON THE GRID

1.0I 0.500 0.500 0.500 0.500 0.501 0.503 0.509 0.525 0.568 0.684 1.000
   I

0.9I 0.464 0.464 0.465 0.465 0.466 0.468 0.474 0.491 0.536 0.653 0.964
   I

0.8I 0.433 0.433 0.434 0.434 0.435 0.437 0.444 0.461 0.506 0.624 0.933
   I

0.7I 0.407 0.407 0.407 0.407 0.408 0.411 0.417 0.434 0.479 0.598 0.907
   I

0.6I 0.384 0.384 0.384 0.385 0.386 0.388 0.395 0.412 0.457 0.575 0.884
```

0.5	0.365	0.365	0.366	0.366	0.367	0.369	0.376	0.393	0.438	0.556	0.865
0.4	0.350	0.350	0.351	0.351	0.352	0.354	0.361	0.378	0.423	0.541	0.850
0.3	0.339	0.339	0.339	0.339	0.340	0.343	0.349	0.366	0.411	0.530	0.839
0.2	0.331	0.331	0.331	0.331	0.332	0.334	0.341	0.358	0.403	0.521	0.831
0.1	0.326	0.326	0.326	0.326	0.327	0.329	0.336	0.352	0.397	0.515	0.826
0.0	0.324	0.324	0.324	0.324	0.325	0.327	0.333	0.349	0.392	0.508	0.824
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0



Although the Exponent Graphics provides the *contour* facility, it is not as easy to use as that of IMSL/IDL. The IMSL/IDL uses one statement to contour plot the *u* values,

```
contour,u,findgen(11)/10.0, findgen(11)/10.0, nlevels = 11, back=255, $
        color = 0, /follow, xtitle = 'X', ytitle = 'Y', $
        title = 'CONTOUR PLOT OF U'
```

while the Exponent Graphics uses three call statements,

```
call grctr(ngrid,ngrid,grid,grid,u,ngrid,5,ncv,cval)
call egsgl('.1 use$', 'grctr1.d1$')
call efsplt(0,' ')
```

Beside these three call statements, one has to set up an external control file, several arrays and variables which makes the program error prone and look awkward.

SOLVE A PARABOLIC PROBLEM

IMSL/IDL program:

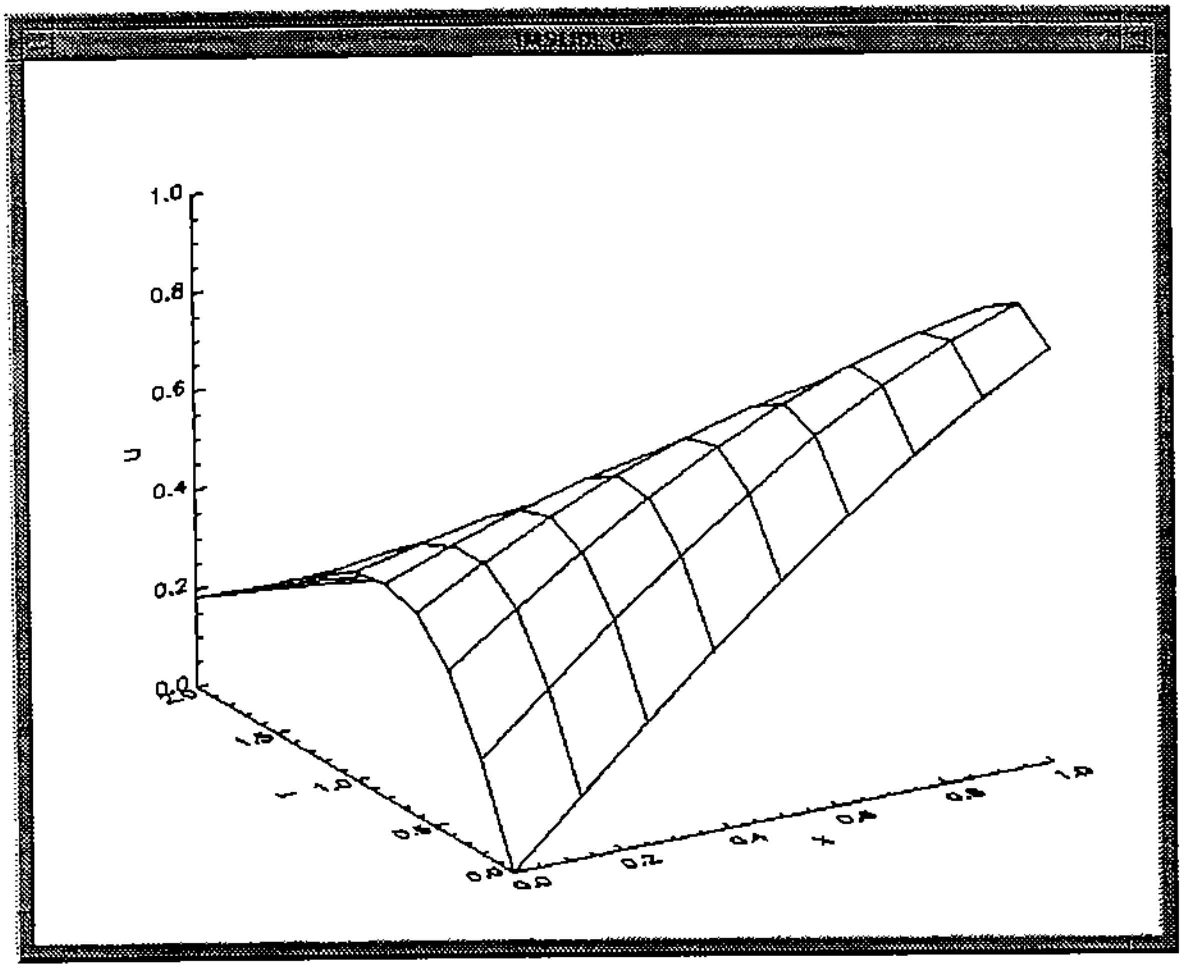
```
pro tableb,u,igrid,kgrid,iprn,kprn,dspace,dtime,range
    subx = indgen((igrid-1)/iprn+1)*iprn
    suby = indgen((kgrid-1)/kprn+1)*kprn
    emax = 0.0
    x = range(0) + findgen(igrid)*dspace
    for k = kgrid,1, -kprn do begin
        t = range(2) + dtime*(k-1.0)
        print, t, u(subx,k-1), format =' (F4.2,("I"),12(X,F6.3))'
        for i = 1, igrid do begin
            y = abs(u(i-1,k-1)-sin(t+x(i-1))/(1.0+t*t))
            if (emax lt y) then emax = y
        endfor
    endfor
    print,'     I---------------------------------------', $
          + '--------------------------'
    print, x(subx), format = '(5X,11(X,F6.3))'
    print, emax, format = '((" MAX ERROR ="),(F15.8))'
    y = u(subx,*)
    z = y(*,suby)
;   Contour, z, findgen(9)/8.0,findgen(11)/5.0, back = 255, color = 0
    surface, z, findgen(9)/8.0,findgen(11)/5.0, back = 255, color = 0, $
           xtitle = 'X', ytitle = 'T', ztitle = 'U', charsize = 2.0
end

function bval, i, k, igrid, kgrid, range
   x = range(0) + (i-1.0)*(range(1) - range(0))/(igrid-1.0)
   t = range(2) + (k-1.0)*(range(3) - range(2))/(kgrid-1.0)
   return, (sin(x+t))/(1.0 + t*t)
end

pro setb, u, igrid, kgrid, range
    u(*,0) = bval(findgen(igrid)+1,1,igrid,kgrid,range)
    u(0,*) = bval(1,findgen(kgrid)+1,igrid,kgrid,range)
    u(igrid-1,*) = bval(igrid,findgen(kgrid)+1,igrid,kgrid,range)
end

pro crank, left, center, right, k, dspace, dtime, range
    t = range(2) + (k-1)*dtime
    const1 = dtime/(4.0*dspace)
    const2 = dtime*t/(dspace*dspace*(1.0+t*t))
    left = -const2 + const1
    right = -const2 - const1
    center = 2.0*const2
end

pro l102
    igrid = 17
```

```
    kgrid = 41
    a = fltarr(igrid, igrid)
    b = fltarr(igrid)
    u = fltarr(igrid, kgrid)
    uout = fltarr(igrid)
    range = [0.0, 1.0, 0.0, 2.0]
    iprn = (igrid - 1)/8
    kprn = (kgrid - 1)/10
    dspace = (range(1)-range(0))/(igrid-1.0)
    dtime = (range(3)-range(2))/(kgrid-1.0)
    setb, u, igrid, kgrid, range
    for k = 2, kgrid do begin
        for i = 0, igrid-1 do begin
            if (i eq 0) then begin
               a(0,0) = 1.0
               a(0,1) = 0.0
               b(0) = u(0,k-1)
            endif else if (i eq igrid-1) then begin
               a(i, i-1) = 0.0
               a(i,i) = 1.0
               b(i) = u(i,k-1)
            endif else begin
               crank, left, center, right, k, dspace, dtime, range
               a(i,i+1) = right
               a(i,i) = 1.0 + center
               a(i,i-1) = left
               crank, left, center, right, k-1, dspace, dtime, range
               b(i) = u(i,k-2)-left*u(i-1,k-2)- $
                        center*u(i,k-2)-right*u(i+1,k-2)
            endelse
        endfor
        uout = lusol(b,a)
;   CALL TSTEP(U,IGRID,KGRID,K,UOUT,IGRID) in the Fortran program
        u(*,k-1) = uout(*)
    endfor
    tableb,u,igrid,kgrid,iprn,kprn,dspace,dtime,range
end
```

RESULTS

2.00I	0.182	0.170	0.156	0.139	0.120	0.099	0.076	0.053	0.028
1.80I	0.230	0.221	0.209	0.194	0.176	0.155	0.132	0.106	0.079
1.60I	0.281	0.278	0.270	0.258	0.243	0.223	0.200	0.174	0.145
1.40I	0.333	0.338	0.337	0.331	0.320	0.304	0.283	0.257	0.228
1.20I	0.382	0.398	0.407	0.410	0.406	0.397	0.381	0.359	0.331
1.00I	0.421	0.451	0.475	0.491	0.499	0.499	0.492	0.477	0.455
0.80I	0.437	0.487	0.529	0.563	0.588	0.603	0.610	0.606	0.594
0.60I	0.415	0.488	0.552	0.609	0.655	0.692	0.718	0.732	0.735
0.40I	0.336	0.432	0.522	0.603	0.675	0.737	0.787	0.825	0.850

```
0.20I  0.191  0.307  0.418  0.523  0.619  0.706  0.782  0.846  0.896
0.00I  0.000  0.125  0.247  0.366  0.479  0.585  0.682  0.768  0.841
    I-----------------------------------------------------------------
       0.000  0.125  0.250  0.375  0.500  0.625  0.750  0.875  1.000
 MAX ERROR =     0.00023976
```



In this IMSL/IDL program, in order to produce a surface plot of the u value, we have to extract some entries from a big array to form a new small array, the facility of indexing an array using another array relieves much of the work.

```
subx = indgen((igrid-1)/iprn+1)*iprn
suby = indgen((kgrid-1)/kprn+1)*kprn
        .
        .
        .
    print, t, u(subx,k-1), format =' (F4.2,("I"),12(X,F6.3))'
        .
        .
        .
print, x(subx), format = '(5X,11(X,F6.3))'
print, emax, format = '((" MAX ERROR ="),(F15.8))'
```

```
    y = u(subx,*)
    z = y(*,suby)
    surface, z, findgen(9)/8.0,findgen(11)/5.0, back = 255, color = 0, $
             xtitle = 'X', ytitle = 'T', ztitle = 'U', charsize = 2.0
```

Instead of using a loop to extract the entries, one can use two index arrays *subx, suby* as index of the big array(u) and form a new small array(z). With the array operations and dynamic type of IMSL/IDL, one can use a single statement instead of Fortran loops. For example, the *TSTEP* subroutine in the Fortran program can be replaced by a single IMSL/IDL statement.

```
        u(*,k-1) = uout(*)
```

Although ELLPACK is not designed to solve time dependent problems, its flexibility of allowing Fortran statements in the ELLPACK program makes the ELLPACK very powerful. Besides the time dependent problems, it also can solve nonlinear problems, system of elliptic problems, etc. Below is the ELLPACK program to solve the parabolic problem.

```
options.         iilevl = 0

declarations.
                 real table(11,9), emax
global.
                 common /gcommon/ t, deltat, nstep

equition.        u - deltat/2.0*ux-deltat*t/(1+t*t)*uxx = pders(x,y)

boundary.        u = sin(t)/(1+t*t) on x = 0
                 u = sin(1+t)/(1+t*t) on x = 1
                 uy = 0.0 on y = 0
                 uy = 0.0 on y = 1

grid.            17 x points $ 3 y points

fortran.
                 emax = 0.0
                 tstart = 0.0
                 tstop  = 2.0
                 deltat = (tstop - tstart)/40
                 dspace = 1.0/16
                 dspace2 = 2*dspace
                 nsteps = int((tstop-tstart)/deltat+0.5)
                 do 10 nstep = 0, nsteps
                    t = tstart + deltat*nstep
dis.                5 point star
sol.                band ge

fortran.
c
c                find max error and set up output table
c
                    do 40 i = 1, 17
```

```
                emax = max(emax,abs(u((i-1)*dspace,1)-
     a                 sin(t+(i-1)*dspace)/(1.+t*t)))
 40           continue
              if (mod(nstep,4) .eq. 0) then
                do 20 i = 1, 9
                  table(nstep/4+1,i) = u(dspace2*(i-1),1)
 20             continue
              endif
c
 10         continue
c
c           print the results
c
            do 30 j = 11, 1, -1
            write (6,'(x,f4.2,''|'',12(x,f6.3))')
     a           float(j-1)/5,(table(j,k),k=1,9)
 30         continue
            write (6,'(5x,''|'',66(1h-))')
            write (6,'(6x,11(x,f6.3))') (dspace2*(i-1.0),i=1,9)
            write (6,88) emax
 88         format('  max error =',f15.8)

subprograms.
            function pders(x,y)
            real x, y
            common /gcommon/ t, deltat, nstep
            t = t - deltat
            if (nstep .eq. 0) then
              pders = sin(x+t)/(1+t*t) + deltat/2.0*
     a                (cos(x+t)/(1+t*t) +
     b                 2.0*t/(1+t*t)*(-sin(x+t)/(1+t*t)))
c
c    cos(x+t)/(1+t*t) is u0x, -sin(x+t)/(1+t*t) is u0xx
c
            else
              pders = u(x,y) + deltat/2.0*(ux(x,y)
     a                +2.0*t/(1+t*t)*uxx(x,y))
            endif
            t = t + deltat
            return
            end
end.
```

RESULTS

2.00	0.182	0.170	0.156	0.139	0.120	0.099	0.076	0.053	0.028
1.80	0.230	0.221	0.209	0.194	0.176	0.155	0.132	0.106	0.079
1.60	0.281	0.278	0.270	0.258	0.243	0.223	0.200	0.174	0.145
1.40	0.333	0.338	0.337	0.331	0.320	0.304	0.283	0.257	0.228

```
1.20|  0.382  0.398  0.407  0.410  0.406  0.397  0.381  0.359  0.331
1.00|  0.421  0.451  0.475  0.491  0.499  0.499  0.492  0.477  0.455
0.80|  0.437  0.487  0.529  0.563  0.588  0.603  0.610  0.606  0.594
0.60|  0.415  0.488  0.552  0.609  0.655  0.692  0.718  0.732  0.735
0.40|  0.336  0.432  0.522  0.603  0.675  0.737  0.787  0.825  0.850
0.20|  0.191  0.307  0.418  0.523  0.619  0.706  0.782  0.846  0.896
0.00|  0.000  0.125  0.247  0.366  0.479  0.585  0.682  0.767  0.841
    |-----------------------------------------------------------------
       0.000  0.125  0.250  0.375  0.500  0.625  0.750  0.875  1.000
 max error =     0.00030294
```

The corresponding Fortran program using the IMSL Math Library and Exponent Graphics is as follows:

```
C                          LIBRARY APPLICATION 10.2
C  APPLY THE CRANK-NICHOLSON DISCRETIZATION TO SOLVE A PARABOLIC PROBLEM
C  WITH ABOUT THREE DIGITS ACCURACY.
C
C  PRINCIPAL VARIABLES:
C    A         - BAND MATRIX OF EXPLICIT DESCRETIZATION COEFFICIENTS
C    B         - VECTOR OF KNOWN  BOUNDARY VALUES IN AU=B
C    FAC       - WORK SPACE ARRAY FOR BAND MATRIX SOLVERS
C    IPVT,RES- WORK SPACE ARRAYS FOR BAND MATRIX SOLVERS
C    UOUT      - U VALUES FOUND AT ONE TIME STEP BY LINEAR EQUATION SOLVER
C    CENTER    - COEFFICIENT IN CENTER OF FINITE DIFFERENCE FORMULA
C    LEFT      - COEFFICIENT TO THE LEFT OF CENTER
C    RIGHT     - COEFFICIENT TO THE RIGHT OF CENTER
C    RANGE     - RANGE OF TIME AND SPACE VALUES TO COVER
C    DSPACE    - SPACING BETWEEN GRID POINTS OF SPACE OR X
C    DTIME     - SPACING BETWEEN GRID POINTS OF TIME OR T
C    IGRID     - NUMBER OF SPACE GRID POINTS ON X AXIS
C    I         - INDEX FROM 1 TO IGRID
C    X         - SPACE AXIS VALUE, IT IS HORIZONTAL
C    KGRID     - NUMBER OF TIME GRID POINTS ON T AXIS
C    K         - INDEX FROM 1 TO KGRID
C    T         - TIME AXIS VALUE, IT IS VERTICAL
C    IPRN      - INTERVALS IN SPACE INDEX TO TABLE U
C    KPRN      - INTERVALS IN TIME INDEX TO TABLE U
C
C  SUBPROGRAMS
C    BVAL      - FUNCTION COMPUTES VALUES FOR BOUNDARY CONDITIONS ON THE GRID
C    CRANK     - SUBROUTINE COMPUTES DIFFERENCE COEFFS FOR CRANK-NICOLSON
C    UFUNC     - FUNCTION TO CALCULATE B-VECTOR IN AU=B
C    TABLEB    - SUBROUTINE TO TABLE SOLUTION OF PARABOLIC PROBLEM
C    SETB      - SETS BOUNDARY & INITIAL VALUES IN U ARRAY
C
C  OUTPUT:
C    U    - ARRAY OF VALUES ON THE GRID.
C
C-----------------------------------------------------------------------------
```

```
      PARAMETER (IGRID=17,KGRID=41)
      REAL A(3,IGRID), B(IGRID), FAC(4,IGRID), RANGE(4)
      REAL IPVT(IGRID), UOUT(IGRID), RES(IGRID), DSPACE, DTIME
      REAL LEFT, CENTER, RIGHT, U(IGRID,KGRID)
      INTEGER I,K, IPRN, KPRN
      EXTERNAL LFTRB, LFIRB, UMACH
C                                        SET PROBLEM DOMAIN RANGES
      DATA RANGE / 0.0, 1.0, 0.0, 2.0 /
C                          SET UP OUTPUT PRINTING AND GRID STEPS
      IPRN = (IGRID - 1)/8
      KPRN = (KGRID - 1)/10
      DSPACE = (RANGE(2) - RANGE(1))/(IGRID - 1)
      DTIME  = (RANGE(4) - RANGE(3))/(KGRID - 1)
C             PUT BOUNDARY AND INITIAL CONDITIONS INTO U ARRAY
      CALL SETB(U, IGRID, KGRID, RANGE)
C                          CALCULATE THE COEFFICIENT MATRIX AND B VECTOR
      DO 10 K = 2, KGRID
        DO 5 I = 1, IGRID
C           IF POINT ON SPACE BOUNDARY,MAKE SIMPLE ASSIGNMENT FOR U
C                 SPECIAL A(1,1) AND A(3,NGRID) VALUES DOE TO
C                 IMSL BAND MATRIX SSTORAGE FORMAT
            IF ( I .EQ. 1.)    THEN
               A(1, 1)    = 0.0
               A(2, I)    = 1.0
               A(1, 2)    = 0.0
               B(I)       = U(I,K)
            ELSE IF( I .EQ. IGRID ) THEN
               A(3, I-1) = 0.0
               A(2, I)    = 1.0
               A(3, I)    = 0.0
               B(I)       = U(I,K)
            ELSE
C                    GET CRANK-NICOLSON SPACE DISCRETIZATION AT TIME K
               CALL CRANK(LEFT,CENTER,RIGHT,K,DSPACE,DTIME,RANGE)
               A(1,I+1)   = RIGHT
               A(2,I)     = 1. + CENTER
               A(3,I-1)   = LEFT
C                    GET CRANK-NICOLSON SPACE DISCRETIZATION AT TIME K-1
               CALL CRANK(LEFT,CENTER,RIGHT,K-1,DSPACE,DTIME,RANGE)
               B(I)      = U(I,K-1)
     &               - LEFT*U(I-1,K-1)-CENTER*U(I,K-1)-RIGHT*U(I+1,K-1)
            ENDIF
  5     CONTINUE
C                                 FACTOR THE COEFFICIENT MATRIX
        CALL LFTRB(IGRID, A, 3, 1, 1, FAC, 4, IPVT)
C                SOLVE THE SYSTEM FOR UOUT( = U ON NEXT TIME LINE)
        CALL LFIRB(IGRID, A, 3, 1, 1, FAC, 4, IPVT, B, 1, UOUT, RES)
C                                 MAKE ANOTHER TIME STEP
        CALL TSTEP(U,IGRID,KGRID,K,UOUT,IGRID)
```

```
   10 CONTINUE
C                                 TABLE THE U VALUES
      CALL TABLEB(U,IGRID,KGRID,IPRN,KPRN,DSPACE,DTIME,RANGE)
      STOP
      END
C
      SUBROUTINE CRANK(LEFT,CENTER,RIGHT,K,DSPACE,DTIME,RANGE)
      REAL LEFT, CENTER, RIGHT, DSPACE,DTIME,RANGE(4),T
      REAL CONST1, CONST2
      INTEGER K
      T = RANGE(3) + (K-1)*DTIME
C                      COMPUTE COEFFICIENTS OF DISCRETIZATION
      CONST1 = DTIME/(4.*DSPACE)
      CONST2 = DTIME*T/(DSPACE*DSPACE*(1.0 + T*T))
      LEFT   = -CONST2 + CONST1
      RIGHT  = -CONST2 - CONST1
      CENTER =  2.*CONST2
      RETURN
      END
C
      SUBROUTINE TSTEP(U,IGRID,KGRID,K,UOUT)
      REAL U(IGRID,KGRID),UOUT(IGRID)
      INTEGER K
C                PLACE UOUT VALUES IN U ARRAY
      DO 10 I=2,IGRID-1
         U(I,K) = UOUT(I)
   10 CONTINUE
      RETURN
      END
C
      SUBROUTINE SETB(U,IGRID,KGRID,RANGE)
      REAL U(IGRID,KGRID), RANGE(4)
      INTEGER IGRID, KGRID
C                     SET VALUES FROM INITIAL CONDITION
      DO 5 I=1,IGRID
         U(I,1) = BVAL(I,1,IGRID,KGRID,RANGE)
5     CONTINUE
C                     SET VALUES FROM BOUNDARY CONDITIONS AT X = 0, 1
      DO 10 K=1,KGRID
         U(1,K) = BVAL(1,K,IGRID,KGRID,RANGE)
         U(IGRID,K) = BVAL(IGRID,K,IGRID,KGRID,RANGE)
10    CONTINUE
      RETURN
      END
C
      SUBROUTINE TABLEB(U,IGRID,KGRID,IPRN,KPRN,DSPACE,DTIME,RANGE)
      REAL U(IGRID,KGRID),RANGE(4),T,DTIME,DTIME
      INTEGER IGRID,KGRID,IPRN,KPRN
C                           GET OUTPUT UNIT NUMBER
```

```
      CALL UMACH(2,NOUT)
      EMAX = 0.
      DO 20 K=KGRID,1, -KPRN
        T = RANGE(3) + DTIME*(K-1.0)
        WRITE(NOUT,'(X,F4.2,''I'',12(X,F6.3))')
     &        T,(U(I,K),I=1,IGRID,IPRN)
        DO 10 I =1, IGRID
          X = RANGE(1)+(I-1)*DSPACE
          EMAX = MAX(EMAX,ABS(U(I,K)-SIN(T+X)/(1.+T*T)))
   10   CONTINUE
   20 CONTINUE
      WRITE(NOUT,99)
   99 FORMAT(5X,'I',66(1H-))
      WRITE (NOUT,'(6X,11(X,F6.3))')
     &        (RANGE(1)+DSPACE*(I-1.0),I=1,IGRID,IPRN)
      WRITE(NOUT,88) EMAX
   88 FORMAT('  MAX ERROR =',F15.8)
      RETURN
      END
C
      REAL FUNCTION BVAL(I,K,IGRID,KGRID,RANGE)
C                                BOUNDARY VALUE = EXACT VALUE
      INTEGER I,K,IGRID,KGRID
      REAL X, T, RANGE(4)
      X = RANGE(1) + (I-1.0)*(RANGE(2) - RANGE(1))/(IGRID-1.0)
      T = RANGE(3) + (K-1.0)*(RANGE(4) - RANGE(3))/(KGRID-1.0)
      BVAL = (SIN(X+T))/(1.0 + T*T)
      RETURN
      END
```

RESULTS

```
-------------------------------------------------------------------------------
 2.00I  0.182  0.170  0.156  0.139  0.120  0.099  0.076  0.053  0.028
 1.80I  0.230  0.221  0.209  0.194  0.176  0.155  0.132  0.106  0.079
 1.60I  0.281  0.278  0.270  0.258  0.243  0.223  0.200  0.174  0.145
 1.40I  0.333  0.338  0.337  0.331  0.320  0.304  0.283  0.257  0.228
 1.20I  0.382  0.398  0.407  0.410  0.406  0.397  0.381  0.359  0.331
 1.00I  0.421  0.451  0.475  0.491  0.499  0.499  0.492  0.477  0.455
 0.80I  0.437  0.487  0.529  0.563  0.588  0.603  0.610  0.606  0.594
 0.60I  0.415  0.488  0.552  0.609  0.655  0.692  0.718  0.732  0.735
 0.40I  0.336  0.432  0.522  0.603  0.675  0.737  0.787  0.825  0.850
 0.20I  0.191  0.307  0.418  0.523  0.619  0.706  0.782  0.846  0.896
 0.00I  0.000  0.125  0.247  0.366  0.479  0.585  0.682  0.768  0.841
     I------------------------------------------------------------------
        0.000  0.125  0.250  0.375  0.500  0.625  0.750  0.875  1.000
  MAX ERROR =      0.00023970
```

Again, for the same reason as in application 9.2(PRESURE AND VELOCITY DISTRIBUTION FROM DIFFERENCE EQUATIONS), the *surface* facility of the Exponent Graphics is not readily applicable in this program.