1993

Comparison of IMSL/IDL with the IMSL Math Library and Exponent 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\textsuperscript{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.

\textsuperscript{1}Work supported by IMSL research grant.
COMPARISON OF IMSL/IDL WITH THE IMSL MATH LIBRARY AND EXONENT 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:

1. Library Application 5.3(Spline interpolation of titanium),
2. Library Application 5.5(Pretty curves with loops),
3. Library Application 7.2(Compare three quadrature methods),
4. Library Application 7.3(Evaluate the sensitivity of integration methods),
5. Library Application 8.8(Rate of return on an investment in forestry products),
6. Library Application 9.2(Presure and velocity distribution from difference equations),
7. Animation of an ODE solution,
8. Library Application 10.1(Solve an elliptic problem using ordinary finite differences),
9. Library Application 10.2(Solve a parabolic problem).

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)

<table>
<thead>
<tr>
<th>Application</th>
<th>IMSL/IDL</th>
<th>Fortran+Exponent</th>
<th>Fortran</th>
<th>ELLPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>35</td>
<td>58</td>
<td>38</td>
<td>xx</td>
</tr>
<tr>
<td>5.5</td>
<td>49</td>
<td>92</td>
<td>82</td>
<td>xx</td>
</tr>
<tr>
<td>7.2</td>
<td>50</td>
<td>xx</td>
<td>60</td>
<td>xx</td>
</tr>
<tr>
<td>7.3</td>
<td>51</td>
<td>xx</td>
<td>75</td>
<td>xx</td>
</tr>
<tr>
<td>8.8</td>
<td>18</td>
<td>30</td>
<td>27</td>
<td>xx</td>
</tr>
<tr>
<td>9.2</td>
<td>38</td>
<td>xx</td>
<td>58</td>
<td>xx</td>
</tr>
<tr>
<td>animation</td>
<td>33</td>
<td>xx</td>
<td>xx</td>
<td>xx</td>
</tr>
<tr>
<td>10.1</td>
<td>52</td>
<td>90</td>
<td>79</td>
<td>16</td>
</tr>
<tr>
<td>10.2</td>
<td>75</td>
<td>xx</td>
<td>100</td>
<td>54</td>
</tr>
</tbody>
</table>
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 = spvalsp(findgen(193)*2.5*585.0, pp)

plot, findgen(193)*2.5*585.0, ppval, yrange = [0,4], $
xtile = '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:

```fortran
REAL XDATA(12),FDATA(12),BREAK(12),CSC0EF(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

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, 149
   XPTS(I) = 575.0 + I*10.0
12 CONTINUE

N = VALUE(VALIND)

C

C CHOOSE THE INTERPOLATION POINTS.

C DATA(1) = XPTS(1)
FDATA(1) = TDAT(1,1)
C DATA(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))
   XD(I) = XPTS(J)
   FD(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 GRAPHS 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 EQQSPN(1,2)
   CALL SCATR(193,ABSC,ORDI)
   CALL EGSGL('.2 use$', 'scatr2.d2$')
   GOTO 500
300 CALL EQQSPN(1,3)
   CALL SCATR(193,ABSC,ORDI)
   CALL EGSGL('.3 use$', 'scatr2.d3$')
   GOTO 500
CALL EQQSPN(1,4)
CALL SCATR(193,ABSC,ORDI)
CALL EGSGL('.4 use$','scatr2.d4$')
CALL EMPLT(1,2,2,IUNIT, ' ')

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:

\[ pp = \text{csinterp}(xdata, fdata) \]
\[ ppval = \text{spvalue}(\text{findgen}(193)*2.5+585.0, pp) \]

to compute the interpolation and find the values at the 193 points, while the second program uses:
CALL CSINT(N,XDATA,FDATA,BREAK,CSCOF)

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,CSCOF)
    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 choose 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 PARAMETER 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)

    xp = x(findgen(nx)*dt,kurve)
    yp = y(findgen(nx)*dt,kurve)

    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 = spval(findgen(124)*(nx-1)/123,px)
    pyval = spval(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:

```
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
T   = VALUE OF ANGLE USED
DT  = ANGLE INCREMENT BEING INTERPOLATED
PARAM = THE VALUES OF THE PARAMETER VARIABLE
XP  = THE ABSCISSAE AS FUNCTION OF PARAM
YP  = THE ORDINATES AS FUNCTION OF PARAM
```
ABSC - ABSCISSAE USED FOR PLOTTING
ORDI - ORDINATE USED FOR PLOTTING
BREAK1 - BREAKPOINTS FOR PIECEWISE CUBIC REPRESENTATION FOR ABSC
BREAK2 - BREAKPOINTS FOR PIECEWISE CUBIC REPRESENTATION FOR ORDI
CSCOE1 - MATRIX(4 BY NX) OF COEFFICIENTS OF CUBIC PIECES FOR ABSC
CSCOE2 - MATRIX(4 BY NX) OF COEFFICIENTS OF CUBIC PIECES FOR ORDI
RANGE - CONSISTS OF ENDPOINTS ON THE X-AXIS & Y-AXIS

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, M
CHARACTER SYMBOL*1
EXTERNAL PLOTP, CSVAL, CSPER, CONST
COMMON/PARMS/RT, RB, N, M, PI

INITIALIZATIONS.
PI = CONST('PI')
RT = 1.25
RB = 0.8
N = 3
M = 2

SELECT CURVE
DO 100 KURVE = 1, 4

LOOP OVER NUMBER OF INTERPOLATION POINT
DO 100 NX = 6, 24, 6
DT = 2.*PI/(NX - 1)

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

MAKE LAST POINTS EXACTLY EQUAL TO FIRST
PARAM(NX) = NX-1
XP(NX) = XP(1)
YP(NX) = YP(1)

COMPUTE THE CUBIC SPLINE INTERPOLANT
FOR X AND Y FUNCTIONS.
CALL CSPER(NX, PARAM, XP, BREAK1, CSCOE1)
CALL CSPER(NX, PARAM, YP, BREAK2, CSCOE2)

COMPUTE THE 124 DATA VALUES FOR PLOTTING.
DVAL = (NX-1)/123.
DO 30 I = 1, 124
VAL = (I-1)*DVAL
ABSC(I) = CSVAL(VALL, NX-1, BREAK1, CSCOE1)
ORDI(I, 1) = CSVAL(VALL, NX-1, BREAK2, CSCOE2)
30 CONTINUE

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 EGSLG(1. use$, 'scatr5.d1$')
   GOTO 100
60 CALL EGQSPN(1,2)
   CALL SCATR(124,ABSC,ORDI)
   CALL EGSLG(1.2 use$, 'scatr5.d2$')
   GOTO 100
70 CALL EGQSPN(1,3)
   CALL SCATR(124,ABSC,ORDI)
   CALL EGSLG(1.3 use$, 'scatr5.d3$')
   GOTO 100
80 CALL EGQSPN(1,4)
   CALL SCATR(124,ABSC,ORDI)
   CALL EGSLG(1.4 use$, 'scatr5.d4$')
   CALL EFHPLT(1,2,2,IUNIT, ' ')
ENDIF
100 CONTINUE
END

REAL FUNCTION I(T,KURVE)
REAL T,RT,RB
COMMON / PARAMS / RT,RB,N,M,PI
X = 0.0
IF (KURVE.EQ.1) THEN
   I = RT*COS(T) - RB*COS((N+1)*T)
ELSE IF (KURVE.EQ.2) THEN
   I = RT*COS(T) - RB*COS((N+1)*T)*EXP(SIN(M*T))
ELSE IF (KURVE.EQ.3) THEN
   I = RT*COS(T)/(1.+SIN(M*T)**2) - RB*COS((N+1)*T)
ELSE IF (KURVE.EQ.4) THEN
   I = 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 z 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) \( \sin(2x) \)
(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)'
compute
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'
compute
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(2X)'
compute
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
derm

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)'
p = 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 = 1.71863999698
INTEGRATION OF HERMITE CUBIC INTERPOLANT = 1.71828281888
INTEGRATION OF CUBIC SPLINE INTERPOLANT = 1.7182517459

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
Apparently, the second program would be clearer and more succinct.

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

```fortran
REAL XDATA(21), K, F1, F2, F3, F4, H
EXTERNAL F1, F2, F3, F4
COMMON XDATA, K, H

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)'
```

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
BREAK - THE BREAK POINTS OF THE PIECEWISE CUBIC REPRESENTATION
CSCOEFF - MATRIX OF LOCAL COEFFICIENTS OF THE CUBIC PIECES
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
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,CSAKH,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 CSAKH(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=errast, err_abs=0.0d0, $err_rel=0.01d0, max_sub=1000, rule=2)
print,ans,format = '("RESULT FOR FUNCTION I = \",d20.17\")'
print, errast, format = '("ABSOLUTE ERROR ESTIMATE = \",d20.17, /\")'
anst = intfcn('f2', a, b, /double, err_est=errast, err_abs=0.0d0, $err_rel=0.01d0, max_sub=1000, rule=2)
print,anst, format = '("RESULT FOR FUNCTION II = \",d20.17\")'
print, errast, format = '("ABSOLUTE ERROR ESTIMATE = \",d20.17, /\")'
anst = intfcn('f3', a, b, /double, err_est=errast, err_abs=0.0d0, $
err_rel=0.01d0, max_sub=1000, rule=2)
print,ans,format = \texttt{\textasciitilde\textquotedquotesleft\textasciitilde RESULT FOR FUNCTION III = \textasciitilde,d20.17\textasciitilde\textasciitilde\textasciitilde\textasciitilde}\nprint,errest,format = \texttt{\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde ABSOLUTE ERROR ESTIMATE = \textasciitilde,d20.17, /\textasciitilde}\nans = intfcn('f4', a, b, /double, err_est=errest, err_abs=0.0d0, $
err_rel=0.01d0, max_sub=1000, rule=2)
print,ans,format = \texttt{\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde RESULT FOR FUNCTION IV = \textasciitilde,d20.17\textasciitilde\textasciitilde\textasciitilde\textasciitilde}\nprint,errest,format = \texttt{\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde ABSOLUTE ERROR ESTIMATE = \textasciitilde,d20.17, /\textasciitilde}\nans = intfcn('f5', a, b, /double, err_est=errest, err_abs=0.0d0, $
err_rel=0.01d0, max_sub=1000, rule=2)
print,ans,format = \texttt{\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde RESULT FOR FUNCTION V = \textasciitilde,d20.17\textasciitilde\textasciitilde\textasciitilde\textasciitilde}\nprint,errest,format = \texttt{\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde ABSOLUTE ERROR ESTIMATE = \textasciitilde,d20.17, /\textasciitilde}\nans = intfcn('f6', a, b, /double, err_est=errest, err_abs=0.0d0, $
err_rel=0.01d0, max_sub=1000, rule=2)
print,ans,format = \texttt{\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde RESULT FOR FUNCTION VI = \textasciitilde,d20.17\textasciitilde\textasciitilde\textasciitilde\textasciitilde}\nprint,errest,format = \texttt{\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde ABSOLUTE ERROR ESTIMATE = \textasciitilde,d20.17, /\textasciitilde}\nend

\begin{verbatim}
RESULTS
\end{verbatim}

RESULT FOR FUNCTION I = 2.00006692290557098
ABSOLUTE ERROR ESTIMATE = 0.0114666258600681

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 = 0.0000148998466697
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:

```
LIBRARY APPLICATION 7.3
TO EVALUATE THE SENSITIVITY OF INTEGRATION METHODS TO ROUND OFF
FOR FOLLOWING FUNCTIONS. THE INTEGRALS ON [-1.0,1.0] FOR THE
CHOSEN FUNCTIONS ARE EXACTLY KNOWN. DURING THE EVALUATION OF THE
INTEGRALS THEIR VALUES ARE PERTURBED BY MULTIPLYING BY
(1.0 + EPS) WHERE EPS IS A RANDOM NUMBER DISTRIBUTED IN [-.001,.001]

F(X) = 1 + X
F(X) = X**4 + X*X + 1
F(X) = SIN(X)
F(X) = SIN(20*X)
F(X) = 1/(1 + 40*X*X)
F(X) = X**2*SIGN(X)

A - LOWER LIMIT OF INTEGRATION
B - UPPER LIMIT OF INTEGRATION
ERRABS - ABSOLUTE ACCURACY DESIRED
ERRREL - RELATIVE ACCURACY DESIRED
IRULE - PARAMETER FOR CHOICE OF QUADRATURE
RESULT - ESTIMATE OF THE INTEGRAL
ERREST - ESTIMATE OF THE ABSOLUTE VALUE OF THE ERROR

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DOUBLE PRECISION F1,F2,F3,F4,F5,F6,RLIST(1000),ELIST(1000),A
INTEGER IRULE,MASUB,NEVAL,NSUBIN,IORD(1000)
DOUBLE PRECISION B,ERRABS,ERREST,RESULT,ALIST(1000),BLIST(1000)
EXTERNAL F1,F2,F3,F4,F5,F6,RNSET,DQ2AG

INITIALIZATIONS
A = -1.0D0
B = 1.0D0
ERRABS = 0.0D0
ERRREL = 0.01D0
IRULE = 2
MAXSUB = 1000

INITIALIZE SEED VALUE OF RANDOM NUMBER GENERATOR.
CALL RNSET(10)

COMPUTE THE INTEGRALS FOR VARIOUS FUNCTIONS.
CALL DQ2AG(F1,A,B,ERRABS,ERRREL,IRULE,RESULT,ERREST,MASUB,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,MASUB,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,ORD)
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,ORD)
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,ORD)
PRINT *, 'RESULT FOR FUNCTION VI = ',RESULT
PRINT *, 'ABSOLUTE ERROR ESTIMATE = ',ERREST

END

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
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 = 1.4899847135067D-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 = 4.64273576767110D-03

RESULT FOR FUNCTION V = 0.44713921785273
ABSOLUTE ERROR ESTIMATE = 3.8496493818691D-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:

```fortran
LIBRARY APPLICATION 8.8

THE GIVEN RATIONAL FUNCTION HAS A POSITIVE ROOT X NEAR ZERO,
AFTER PLOTTING THE FUNCTION, WE TRANSFORM IT INTO A
POLYNOMIAL AND COMPARE THE NEW ROOT WITH THE PREVIOUS ONE.

NDATA - NUMBER OF POINTS USED FOR PLOTTING
ABSC - ABSCISSAE OF POINTS USED FOR PLOTTING
ORDI - ORDINATES OF POINTS USED FOR PLOTTING
RANGE - ENDPONITS OF THE 2 AXIS
XGUESS - THE INITIAL GUESS OF THE ZERO OF THE POLYNOMIA
X - THE ZERO OF THE POLYNOMIAL
INFO - NUMBER OF ITERATIONS USED FOR FINDING THAT ZERO

REAL ABSC(100),ORDI(100,1),RANGE(4),X(1),XGUESS(1),INFO(1)
EXTERNAL PLOTP,ZREAL,F1,F2

INITIALIZATIONS

NDATA = 94
NFUN = 1
DO 10 I = 1,94
    ABSC(I) = 0.06 + I/100.0

28
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,SYMB,'X AXIS',
C 'Y AXIS','RATE OF RETURN')
C CALL SCATR(94,ABSC,ORDI)
CALL EGSGL('.! use$' 'scatr8.d1$')
CALL EFMPLT(1,1,1,IUNIT,')
XGUESS(1) = 0.1

CALL THE NONLINEAR EQUATION SOLVER
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(8,*) 'THE ZERO OF THE RATIONAL FUNCTION IS ', X(1)

CALL THE NONLINEAR EQUATION SOLVER
FOR THE POLYNOMIAL OBTAINED.
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

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 PRESSURE 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 PRESSURES 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 + DELTAX )
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 PRESSURE PRIGHT(0) = MO**2/2,
THE VELOCITIES U(J+1/2) AND PressURES P(J) ARE COMPUTED

IN THE LIMIT OF 0 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(1.75)
RETURN, YP
END
PRO L92
;
; DECLARATION AND INITIALIZATION
;
FO = 1.5
MO = 1.0
UO = 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 = [MO^-2/2.0, U0]
T = FINDGEN( MAXSTP + 1 ) / MAXSTP
;
; SOLVE THE ODE. USING DEFAULT VALUES EXCEPT TOLERANCE
;
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(0,0), SQRT( 2. * Y(0,0)),$
;
; FIND SOLUTION OF DIFFERENCE EQUATION AND PRINT RESULT
;
UDIEFE = SQRT( 2. * Y(0,0) )
FOR ISTEP = 0, MAXSTP-1 DO BEGIN
PLEFT(ISTEP) = PRIGHT(ISTEP) - FO*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), $  
+ '(" |"), 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 = 'PRESSURE', ZTITLE = 'VELOCITY', CHARSIZE = 2  
XYOUTS, 0.5, 0.8, '!6PRESSURE & VELOCITY PROJECTION', $  
/NORMAL, COLOR = 99, CHARSIZE = 1.5, ALIGN = 0.5  

END  

RESULTS

<table>
<thead>
<tr>
<th>J</th>
<th>X</th>
<th>P(X)</th>
<th>U(X)</th>
<th>-DU/DX</th>
<th>PLEFT(J)</th>
<th>U(J)</th>
<th>PRIGHT(J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>0.500</td>
<td>1.000</td>
<td>1.000</td>
<td>0.350</td>
<td>0.911</td>
<td>0.435</td>
</tr>
<tr>
<td>1</td>
<td>0.100</td>
<td>0.455</td>
<td>0.902</td>
<td>0.954</td>
<td>0.350</td>
<td>0.911</td>
<td>0.435</td>
</tr>
<tr>
<td>2</td>
<td>0.200</td>
<td>0.421</td>
<td>0.809</td>
<td>0.918</td>
<td>0.307</td>
<td>0.829</td>
<td>0.379</td>
</tr>
<tr>
<td>3</td>
<td>0.300</td>
<td>0.397</td>
<td>0.718</td>
<td>0.891</td>
<td>0.271</td>
<td>0.751</td>
<td>0.333</td>
</tr>
<tr>
<td>4</td>
<td>0.400</td>
<td>0.381</td>
<td>0.630</td>
<td>0.873</td>
<td>0.242</td>
<td>0.678</td>
<td>0.294</td>
</tr>
<tr>
<td>5</td>
<td>0.500</td>
<td>0.373</td>
<td>0.544</td>
<td>0.863</td>
<td>0.218</td>
<td>0.608</td>
<td>0.263</td>
</tr>
<tr>
<td>6</td>
<td>0.600</td>
<td>0.371</td>
<td>0.457</td>
<td>0.861</td>
<td>0.200</td>
<td>0.542</td>
<td>0.238</td>
</tr>
<tr>
<td>7</td>
<td>0.700</td>
<td>0.375</td>
<td>0.371</td>
<td>0.866</td>
<td>0.187</td>
<td>0.478</td>
<td>0.219</td>
</tr>
<tr>
<td>8</td>
<td>0.800</td>
<td>0.382</td>
<td>0.284</td>
<td>0.874</td>
<td>0.178</td>
<td>0.417</td>
<td>0.205</td>
</tr>
<tr>
<td>9</td>
<td>0.900</td>
<td>0.390</td>
<td>0.196</td>
<td>0.884</td>
<td>0.173</td>
<td>0.356</td>
<td>0.197</td>
</tr>
<tr>
<td>10</td>
<td>1.000</td>
<td>0.398</td>
<td>0.107</td>
<td>0.892</td>
<td>0.172</td>
<td>0.295</td>
<td>0.192</td>
</tr>
</tbody>
</table>
The corresponding Fortran program using the IMSL Math Library and Exponent Graphics is as follows:

```fortran
C COMPUTE PRESSURE 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 + DELTAX )
C CON2 = DELTAX**2 / 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
```
FOR INITIAL VELOCITY \( U(1/2) = 1 \) AND PRESSURE \( P(0) = \frac{Mo^2}{2} \),
the velocities \( U(J+1/2) \) and pressures \( P(J) \) are computed.

In the limit of 0 DELTAX and an infinite number of ports, the system of difference equations becomes the differential equation problem given by:

\[
\begin{align*}
\frac{DP}{DX} &= U \sqrt{2P} - FO U^*(7/4), \quad P(0) = \frac{Mo^2}{2} \\
\frac{DU}{DX} &= - \sqrt{2P}, \quad U(0) = 1
\end{align*}
\]

Where:
- \( P \) is pressure, stored in \( Y(1) \) 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.

```fortran
PARAMETER (MAXPAR = 50, NEQN = 2)
REAL MO, PARAM(MAXPAR), Y(NEQN)
REAL PRIGHT(50), PLEFT(50), U(50)
COMMON FO
EXTERNAL DERIV

SET TOLERANCE FOR IVPRK AND CONSTANTS FOR THE PROBLEM
DATA TOL / 0.0005/, FO / 1.5/, MO / 1.0/, UO / 1.0/

SET OUTPUT UNIT NUMBER
CALL UHACH( 2, NOUTPT )

SET DEFAULT VALUES OF PARAM (USED BY IVPRK)
CALL SSET( MAXPAR, 0.0, PARAM, 1 )

SET INITIAL CONDITIONS AND CONSTANTS

\[
\begin{align*}
X &= 0.0 \\
Y(1) &= M0^*2 / 2. \\
Y(2) &= U0 \\
MAXSTP &= 10 \\
DELTAX &= 1./MAXSTP \\
DELSQR &= DELTAX**2 \\
CON1 &= 2. / ( 2. + DELSQR ) \\
CON2 &= DELSQR**2/4. \\
CON3 &= DELSQR*( 2. + DELSQR ) \\
PRIGHT(1) &= Y(1) \\
U(1) &= Y(2) \\
PPLOT(1) &= Y(1) \\
UPLOT(1) &= Y(2)
\end{align*}
\]

WRITE( NOUTPT, 1000 ) 0, X, Y, SQRT( 2. * Y(1) )
1000 FORMAT( 13X,'I',5X,'DIFFERENTIAL EQUATION',3X,'I', 35
A 5X,'DIFFERENCE EQUATION',5X,'/
B 3X,'J',5X,'X',3X,'|',4X,'P(X)',6X,'U(X)',4X,'-DU/DX',
C ' | ', 'P(RIGHT(J))',4X,'U(J)',4X,'PLEFT(J) '
D I4,F8.3,' | ',F8.3,2F10.3, | ' )31X,'|

IDO = 1
DO 1030 ISTEP = 1, MAXSTEP
     XEND = ISTEP * DELTAX
     FIND SOLUTION OF DIFFERENTIAL EQUATIONS
     AT NEXT TIME STEP
     CALL IVPRK( IDO, NEQN, DERIV, X, XEND, TOL, PARAM, Y )
     FIND SOLUTION OF DIFFERENCE EQUATION AT
     NEXT TIME STEP
     PPLDT(ISTEP+1) = Y(1)
     UPLDT(ISTEP+1) = Y(2)
     PLEFT(ISTEP) = PRIGHT(ISTEP) - FO * 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) )
     A * ( U(ISTEP) + U(ISTEP+1) ) / 2.
     UDOLVE = SQRT( 2. * Y(I) )
     WRITE( NOUTPT, 1010 ) ISTEP, X, Y, UDOLVE,
     A PLEFT(ISTEP), U(ISTEP+1), PRIGHT(ISTEP+1)
1010 FORMAT(I4,F8.3,' | ',F8.3,2F10.3, | ',F8.3,2F10.3, | ' )
     IF( UDOLVE .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
<table>
<thead>
<tr>
<th>J</th>
<th>X</th>
<th>X</th>
<th>U(X)</th>
<th>X</th>
<th>U(J)</th>
<th>X</th>
<th>PLEFT(J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>0.500</td>
<td>1.000</td>
<td>1.000</td>
<td>0.350</td>
<td>0.911</td>
<td>0.435</td>
</tr>
<tr>
<td>1</td>
<td>0.100</td>
<td>0.455</td>
<td>0.902</td>
<td>0.954</td>
<td>0.307</td>
<td>0.829</td>
<td>0.379</td>
</tr>
<tr>
<td>2</td>
<td>0.200</td>
<td>0.421</td>
<td>0.809</td>
<td>0.918</td>
<td>0.271</td>
<td>0.751</td>
<td>0.333</td>
</tr>
<tr>
<td>3</td>
<td>0.300</td>
<td>0.387</td>
<td>0.718</td>
<td>0.891</td>
<td>0.242</td>
<td>0.678</td>
<td>0.294</td>
</tr>
<tr>
<td>4</td>
<td>0.400</td>
<td>0.361</td>
<td>0.630</td>
<td>0.873</td>
<td>0.218</td>
<td>0.608</td>
<td>0.263</td>
</tr>
<tr>
<td>5</td>
<td>0.500</td>
<td>0.373</td>
<td>0.544</td>
<td>0.863</td>
<td>0.200</td>
<td>0.542</td>
<td>0.238</td>
</tr>
<tr>
<td>6</td>
<td>0.600</td>
<td>0.371</td>
<td>0.457</td>
<td>0.861</td>
<td>0.187</td>
<td>0.478</td>
<td>0.219</td>
</tr>
<tr>
<td>7</td>
<td>0.700</td>
<td>0.375</td>
<td>0.371</td>
<td>0.866</td>
<td>0.178</td>
<td>0.417</td>
<td>0.205</td>
</tr>
<tr>
<td>8</td>
<td>0.800</td>
<td>0.382</td>
<td>0.284</td>
<td>0.874</td>
<td>0.178</td>
<td>0.356</td>
<td>0.197</td>
</tr>
<tr>
<td>9</td>
<td>0.900</td>
<td>0.390</td>
<td>0.196</td>
<td>0.884</td>
<td>0.172</td>
<td>0.295</td>
<td>0.192</td>
</tr>
<tr>
<td>10</td>
<td>1.000</td>
<td>0.398</td>
<td>0.107</td>
<td>0.892</td>
<td>0.172</td>
<td>0.295</td>
<td>0.192</td>
</tr>
</tbody>
</table>

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:

```fortran
SURFACE, Y, BACKGROUND = 200, COLOR = 40, XTITLE = 'X AXIS', $  
YTITLE = 'PRESSURE', 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

```fortran
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 running 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.

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

37
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 IMSLjIDL 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 12 beta version 2.0.0 of IMSLjIDL. One shortcoming of IMSLjIDL 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 STANDARD GAMMA II COLOR TABLE
  LOADCT,5
  ; FOR I = 1, MAXSTP DO BEGIN
  ; ASSIGN THE INITIAL VALUE
  U0 = 1.0*I
  W0 = 1.0*I
  Y = [W0^2/2.0, U0]
  ; SOLVE THE ODE. USING DEFAULT VALUES EXCEPT TOLERANCE
  ; 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 = 'PRESSURE', 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
  WRITE, 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)
            endwhile
        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,' VALUES ON THE GRID', $
+ '
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

1.0  0.500  0.500  0.500  0.501  0.509  0.525  0.568  0.684  1.000
   I

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

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

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

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

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

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

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

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

0.1  0.326  0.326  0.326  0.326  0.327  0.329  0.336  0.352  0.397  0.515  0.826
   I
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 $x = 0$

43
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-point star

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

execution successful

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

band ge

number of equations 81
lower bandwidth 9
upper bandwidth 9
required workspace 2349
**execution successful**

**allpack output**

```

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

**x-abscissae are**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000000E+00</td>
<td>1.000000E-01</td>
<td>2.000000E-01</td>
<td>3.000000E-01</td>
<td></td>
</tr>
<tr>
<td>4.000000E-01</td>
<td>5.000000E-01</td>
<td>6.000000E-01</td>
<td>7.000000E-01</td>
<td></td>
</tr>
<tr>
<td>8.000000E-01</td>
<td>9.000000E-01</td>
<td>1.000000E+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**y = 1.000000E+00**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.000454E-01</td>
<td>5.000700E-01</td>
<td>5.001708E-01</td>
<td>5.004570E-01</td>
<td></td>
</tr>
<tr>
<td>5.012398E-01</td>
<td>5.033692E-01</td>
<td>5.091579E-01</td>
<td>5.248935E-01</td>
<td></td>
</tr>
<tr>
<td>5.676677E-01</td>
<td>6.839397E-01</td>
<td>1.000000E+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**y = 9.000000E-01**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.644043E-01</td>
<td>4.644398E-01</td>
<td>4.645576E-01</td>
<td>4.648822E-01</td>
<td></td>
</tr>
<tr>
<td>4.657558E-01</td>
<td>4.680960E-01</td>
<td>4.743540E-01</td>
<td>4.910576E-01</td>
<td></td>
</tr>
<tr>
<td>5.355291E-01</td>
<td>6.534743E-01</td>
<td>9.643589E-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**y = 8.000000E-01**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.334106E-01</td>
<td>4.334520E-01</td>
<td>4.335796E-01</td>
<td>4.339254E-01</td>
<td></td>
</tr>
<tr>
<td>4.348457E-01</td>
<td>4.372859E-01</td>
<td>4.437430E-01</td>
<td>4.607957E-01</td>
<td></td>
</tr>
<tr>
<td>5.057303E-01</td>
<td>6.238284E-01</td>
<td>9.333652E-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**y = 7.000000E-01**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.067542E-01</td>
<td>4.067987E-01</td>
<td>4.069315E-01</td>
<td>4.072878E-01</td>
<td></td>
</tr>
<tr>
<td>4.082300E-01</td>
<td>4.107134E-01</td>
<td>4.172474E-01</td>
<td>4.344157E-01</td>
<td></td>
</tr>
<tr>
<td>4.794667E-01</td>
<td>5.975407E-01</td>
<td>9.067088E-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**y = 6.000000E-01**
```
ellpack output
-------------

contour plot of \( u \)
grid 20 by 20
execution successful
ELLPACK is designed to solve partial differential equations, it is very convenient to specify the equation, the boundary conditions, discretization methods, solvers, and the output. It has a variety 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:

```
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 LFRB AND BAND MATRIX SOLVING USING LFTIRB
C WHICH USES ITERATIVE REFINEMENT.
```
INPUT:

- FUNCTION TO CALCULATE VALUES FOR BOUNDARY CONDITIONS ON THE GRID
- FUNCTION TO CALCULATE B-VECTOR(RIGHT SIDE OF PDE) IN AX=B
- FUNCTION TO CHECK IF AN (X,Y) POINT IS ON THE GRID BOUNDARY
- BAND MATRIX OF 5-POINT STAR COEFFICIENTS
- VECTOR OF KNOWN VALUES IN AX=B
- COEFFICIENT ON FINGERS OF THE 5-POINT STAR
- COEFFICIENT OF THE MIDDLE OF THE 5-POINT STAR
- SPACING BETWEEN GRID POINTS
- COUNT OF DIAGONAL AN UPPER AN LOWER CODIAGONALS
- NUMBER OF GRID POINTS ON X AND Y AXIS
- INDEX OF EACH GRID POINT. (0,0) IS 1, (0,1) IS 2, ETC.
- X AXIS VALUE
- Y AXIS VALUE
- WORK SPACE ARRAY FOR BAND MATRIX SOLVERS

OUTPUT:

- VECTOR OF VALUES ON THE GRID. UOUT IS U IN AX=B
- RESIDUAL VECTOR AT IMPROVED SOLUTION FROM ITERATIVE REFINEMENT

PARAMETER (NGRID=11, LEN=NGRID*NGRID, LDFAC=3*NGRID+1)

REAL A(-NGRID:NGRID, 1:LEN), B(LEN), STAR, SPACE2, FAC(LDFAC,LEN)
REAL IPVT(LEN), UDUT(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

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)
  END IF
10 CONTINUE
C ENDIF
C CONTINUE
CALL LFTRB(LEN, A, NROWS, 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 cvalCncv)
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))
   &
   call grctr(ngrid,ngrid,grid,u,ngrid,s,ncv,cval)
call egsg1('.i use$', 'grctr1.d1$'
   call efspht(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

1.0  0.500  0.500  0.500  0.500  0.501  0.509  0.525  0.568  0.684  1.000
0.9  0.464  0.464  0.465  0.465  0.466  0.468  0.474  0.491  0.536  0.653  0.964
0.8  0.433  0.433  0.434  0.434  0.435  0.437  0.444  0.461  0.506  0.624  0.933
0.7  0.407  0.407  0.407  0.408  0.411  0.417  0.434  0.479  0.598  0.907
0.6  0.384  0.384  0.384  0.385  0.386  0.388  0.395  0.412  0.457  0.575  0.884
<table>
<thead>
<tr>
<th>x</th>
<th>0.0</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>0.0</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.5</td>
<td>0.6</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
<td>1.0</td>
</tr>
</tbody>
</table>
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,

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

while the Exponent Graphics uses three call statements,

```idl
call grctr(ngrid,ngrid,grid,grid,u,ngrid,5,ncv,cval)
call egsel('.1 use$', 'grctr1.dl$')
call efsp1t(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:

```idl
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,"",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 = '(6X,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,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
```

53
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)
endfor
CALL TSTEP(U,IGRID,KGRID,K,UOUT,IGRID) in the Fortran program
u(*,k-1) = uout(*)
endfor

RESULTS

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.00</td>
<td>0.182</td>
<td>0.170</td>
<td>0.156</td>
<td>0.139</td>
<td>0.120</td>
<td>0.099</td>
<td>0.076</td>
<td>0.053</td>
<td>0.028</td>
<td></td>
</tr>
<tr>
<td>1.80</td>
<td>0.230</td>
<td>0.221</td>
<td>0.209</td>
<td>0.194</td>
<td>0.176</td>
<td>0.155</td>
<td>0.132</td>
<td>0.106</td>
<td>0.079</td>
<td></td>
</tr>
<tr>
<td>1.60</td>
<td>0.281</td>
<td>0.278</td>
<td>0.270</td>
<td>0.258</td>
<td>0.243</td>
<td>0.223</td>
<td>0.200</td>
<td>0.174</td>
<td>0.145</td>
<td></td>
</tr>
<tr>
<td>1.40</td>
<td>0.333</td>
<td>0.338</td>
<td>0.337</td>
<td>0.331</td>
<td>0.320</td>
<td>0.304</td>
<td>0.283</td>
<td>0.257</td>
<td>0.228</td>
<td></td>
</tr>
<tr>
<td>1.20</td>
<td>0.382</td>
<td>0.398</td>
<td>0.407</td>
<td>0.410</td>
<td>0.406</td>
<td>0.397</td>
<td>0.381</td>
<td>0.359</td>
<td>0.331</td>
<td></td>
</tr>
<tr>
<td>1.00</td>
<td>0.421</td>
<td>0.451</td>
<td>0.475</td>
<td>0.491</td>
<td>0.499</td>
<td>0.495</td>
<td>0.492</td>
<td>0.477</td>
<td>0.455</td>
<td></td>
</tr>
<tr>
<td>0.80</td>
<td>0.437</td>
<td>0.487</td>
<td>0.529</td>
<td>0.563</td>
<td>0.588</td>
<td>0.603</td>
<td>0.610</td>
<td>0.606</td>
<td>0.594</td>
<td></td>
</tr>
<tr>
<td>0.60</td>
<td>0.415</td>
<td>0.488</td>
<td>0.552</td>
<td>0.609</td>
<td>0.655</td>
<td>0.692</td>
<td>0.718</td>
<td>0.732</td>
<td>0.735</td>
<td></td>
</tr>
<tr>
<td>0.40</td>
<td>0.336</td>
<td>0.432</td>
<td>0.522</td>
<td>0.603</td>
<td>0.675</td>
<td>0.737</td>
<td>0.787</td>
<td>0.825</td>
<td>0.850</td>
<td></td>
</tr>
</tbody>
</table>
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.

\[
\begin{align*}
\text{subx} & = \text{indgen}((\text{igrid}-1)/\text{iprn}+1) \times \text{iprn} \\
\text{suby} & = \text{indgen}((\text{kgrid}-1)/\text{kprn}+1) \times \text{kprn} \\
\text{print}, \ t, \ u(\text{subx}, \text{k}-1), \ \text{format} = \text{'(F4.2,('I'),12(X,F6.3)')}' \\
\text{print}, \ x(\text{subx}), \ \text{format} = \text{'(5X,11(X,F6.3))'} \\
\text{print}, \ \text{emax}, \ \text{format} = \text{'((" MAX ERROR ="), (F15.8))'}
\end{align*}
\]
\[ y = u(\text{subx}, \ast) \]
\[ z = y(\ast, \text{suby}) \]

surface, z, findgen(9)/8.0, findgen(11)/5.0, back = 255, color = 0, $
\quad 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 \text{TSTEP} subroutine in the Fortran program can be replaced by a single IMSL/IDL statement.

\[ u(\ast, k-1) = uout(\ast) \]

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.

\text{options.} \quad \text{iilevl} = 0

\text{declarations.}
\quad \text{real} \quad \text{table}(11, 9), \text{emax}

\text{global.}
\quad \text{common} / \text{gcommon/ t, deltat, nstep}

\text{equation.}
\quad u - \text{deltat}/2.0 * ux - \text{deltat}*t/(1+t*t)*uxx = pders(x, y)

\text{boundary.}
\quad \quad \text{u} = \sin(t)/(1+t*t) \text{ on } x = 0
\quad \quad \text{u} = \sin(1+t)/(1+t*t) \text{ on } x = 1
\quad \quad \text{uy} = 0.0 \text{ on } y = 0
\quad \quad \text{uy} = 0.0 \text{ on } y = 1

\text{grid.}
\quad 17 \times \text{points} \quad , 3 \times \text{y points}

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

\text{fortran.}
\quad \text{c}
\quad \text{c}
\quad \quad \text{find max error and set up output table}
\quad \text{c}
\quad \quad \text{do 40 i = 1, 17}
emax = max(emax,abs(u((i-1)*dspace,1)-
       sin(t+(i-1)*dspace)/(1.+t*t)))

continue
if (mod(nstep,4).eq.0) then
   do 20 i = 1, 9
      table(nstep/4+1,i) = u(dspace2*(i-1),1)
   continue
endif

print the results

do 30 j = 11, 1, -1
write (6,'(x,f4.2,"|",12(x,f6.3))')
   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*
      (cos(x+t)/(1+t*t) +
      2.0*t/(1+t*t)*(-sin(x+t)/(1+t*t)))
   else
      pders = u(x,y) + deltat/2.0*(ux(x,y)
      +2.0*t/(1+t*t)*uxx(x,y))
   endif
   t = t + deltat
   return
end

RESULTS

<table>
<thead>
<tr>
<th></th>
<th>2.00</th>
<th>1.80</th>
<th>1.60</th>
<th>1.40</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.00</td>
<td>0.182</td>
<td>0.170</td>
<td>0.156</td>
<td>0.139</td>
</tr>
<tr>
<td>1.80</td>
<td>0.230</td>
<td>0.221</td>
<td>0.209</td>
<td>0.194</td>
</tr>
<tr>
<td>1.60</td>
<td>0.281</td>
<td>0.278</td>
<td>0.270</td>
<td>0.258</td>
</tr>
<tr>
<td>1.40</td>
<td>0.333</td>
<td>0.338</td>
<td>0.337</td>
<td>0.331</td>
</tr>
</tbody>
</table>
The corresponding Fortran program using the IMSL Math Library and Exponent Graphics is as follows:

```fortran
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 EXPPLICIT DISCRETIZATION 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 COEFS FOR CRANK-NICLSON
C UFUNC - FUNCTION TO CALCULATE B-VECTOR IN AU=B
C TABLE2 - 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-----------------------------------------------------------------------------
```

max error = 0.00030294
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, LFRB, UMACH

SET PROBLEM DOMAIN RANGES
DATA RANGE / 0.0, 1.0, 0.0, 2.0 /

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)

PUT BOUNDARY AND INITIAL CONDITIONS INTO U ARRAY
CALL SETB(U, IGRID, KGRID, RANGE)

CALCULATE THE COEFFICIENT MATRIX AND B VECTOR
DO 10 K = 2, KGRID
  DO 5 I = 1, IGRID
    IF POINT ON SPACE BOUNDARY, MAKE SIMPLE ASSIGNMENT FOR U
    SPECIAL A(1,1) AND A(3,IGRID) VALUES DUE TO
    IMSL BAND MATRIX STORAGE 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
      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
    ENDIF
    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)
  ENDOF
  CONTINUE

FACTOR THE COEFFICIENT MATRIX
CALL LFTRB(IGRID, A, 3, 1, 1, FAC, 4, IPVT)

SOLVE THE SYSTEM FOR UOUT( = U ON NEXT TIME LINE)
CALL LFRB(IGRID, A, 3, 1, 1, FAC, 4, IPVT, B, 1, UOUT, RES)

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,DSPACE
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,IGRIO,IPRN)
   DO 10 I =1, IGRID
      X = RANGE(1)+(I-1.0)*SPACE
      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)+SPACE*(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

<p>| | | | | | | | | | | |
|     |     |     |     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 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 | 0.267 | 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 |</p>
<table>
<thead>
<tr>
<th>0.00</th>
<th>0.000</th>
<th>0.125</th>
<th>0.247</th>
<th>0.366</th>
<th>0.479</th>
<th>0.585</th>
<th>0.682</th>
<th>0.768</th>
<th>0.841</th>
</tr>
</thead>
</table>

MAX ERROR = 0.00023970

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