mathematisch centrum

DEPARTMENT OF NUMERICAL MATHEMATICS OCTOBER

NUMAL, A LIBRARY OF NUMERICAL PROCEDURES IN ALGOL 60

VOLUME 4. ANALYTIC EVALUATIONS

amsterdam 1974

AUTHOR : J.W. DANIEL.

REVISOR : J. KOK.

INSTITUTE : MATHEMATICAL CENTRE.

RECEIVED : 730528 (EULER).
730917 (SUMPOSSERIES).

BRIEF DESCRIPTION :

THIS SECTION CONTAINS TWO PROCEDURES FOR THE SUMMATION OF CONVERGENT INFINITE SERIES:

EULER PERFORMS THE SUMMATION OF AN ALTERNATING SERIES.

SUMPOSSERIES PERFORMS THE SUMMATION OF A CONVERGENT SERIES WITH POSITIVE MONOTONICALLY DECREASING TERMS USING THE VAN WIJNGAARDEN TRANSFORMATION OF THE SERIES TO AN ALTERNATING SERIES.

KEYWORDS :

SUMMATION,
SERIES,
VAN WIJNGAARDEN TRANSFORMATION.

SUBSECTION : EULER.

CALLING SEQUENCE :

THE HEADING OF THE PROCEDURE IS :

```
"REAL""PROCEDURE" EULER(AI, I, EPS, TIM);
"VALUE" EPS, TIM; "INTEGER" I, TIM; "REAL" AI, EPS;
```

EULER : DELIVERS THE COMPUTED SUM OF THE INFINITE SERIES
A[I], I:= 0,1,... .

THE MEANING OF THE FORMAL PARAMETERS IS:

AI : <ARITHMETIC EXPRESSION>;
THE SUMMAND,
THIS EXPRESSION WILL BE DEPENDENT ON THE JENSEN PARAMETER I;
AI IS THE I-TH TERM OF THE SERIES (I >= 0).

I : <VARIABLE>;
JENSEN PARAMETER.

EPS,TIM: <ARITHMETIC EXPRESSION>;
THE SUMMATION IS CONTINUED UNTIL TIM SUCCESSIVE TERMS OF THE TRANSFORMED SERIES ARE IN ABSOLUTE VALUE LESS THAN EPS.

PROCEDURES USED : NONE.

REQUIRED CENTRAL MEMORY :

EXECUTION FIELD LENGTH : 25.

LANGUAGE : ALGOL 60.

METHOD AND PERFORMANCE :

EULER PERFORMS THE SUMMATION OF AN ALTERNATING SEQUENCE BY APPLYING EULER'S TRANSFORMATION. BY THIS TRANSFORMATION THE SEQUENCE OF TERMS IS REPLACED BY THE SEQUENCE OF MEANS OF TWO SUCCESSIVE TERMS. IF NECESSARY THE NEW SEQUENCE IS AGAIN TRANSFORMED BY EULER'S TRANSFORMATION. THE SUMMATION STOPS WHEN TIM SUCCESSIVE TERMS OF THE (ONCE OR SEVERAL TIMES TRANSFORMED) SEQUENCE ARE IN ABSOLUTE VALUE LESS THAN EPS.

REFERENCES :

P.NAUR, ED. : REVISED REPORT ON THE ALGORITHMIC LANGUAGE ALGOL 60. COPENHAGEN (1964).

EXAMPLE OF USE :

THE PROGRAM :

```
"BEGIN""INTEGER" K;
    "REAL""PROCEDURE" EULER(A, B, C, D); "CODE" 32010;

    OUTPUT(61, "("+.8D"+2D")",
        EULER((- 1) ** K / (K + 1) ** 2, K, "- 6, 100))
"END"
```

DELIVERS :

```
+.82246703"+00.
```

SUBSECTION : SUMPUSSERIES.

CALLING SEQUENCE :

THE HEADING OF THE PROCEDURE IS :

```
"REAL""PROCEDURE" SUMPOSSERIES(AI, I, MAXADDUP, MAXZERO, MAXRECURS,
                                MACHEXP, TIM);
"VALUE" MAXADDUP, MAXZERO, MAXRECURS, MACHEXP, TIM;
"REAL" AI, I, MAXZERO; "INTEGER" MAXADDUP, MAXRECURS, MACHEXP, TIM;
```

SUMPOSSERIES : DELIVERS THE COMPUTED SUM OF THE INFINITE SERIES
A[I] , I:= 1,2,.... .

THE MEANING OF THE FORMAL PARAMETERS IS:

AI : <ARITHMETIC EXPRESSION>;
THE SUMMAND,
THIS EXPRESSION SHOULD BE DEPENDENT ON THE JENSEN PARAMETER I;
AI IS THE I-TH TERM OF THE SERIES (I >= 1).

I : <VARIABLE>;
JENSEN PARAMETER.

MAXADDUP : <ARITHMETIC EXPRESSION>;
UPPER LIMIT FOR THE NUMBER OF STRAIGHTFORWARD ADDITIONS.

MAXZERO, TIM :
<ARITHMETIC EXPRESSION>;
TOLERANCES EPS AND TIM NEEDED FOR A CALL OF THE PROCEDURE EULER (THIS SECTION). MAXZERO IS ALSO USED AS A TOLERANCE FOR MAXADDUP STRAIGHTFORWARD ADDITIONS.

MAXRECURS : <ARITHMETIC EXPRESSION>;
UPPER LIMIT FOR THE RECURSION DEPTH OF THE VAN WIJNGAARDEN TRANSFORMATIONS.

MACHEXP : <ARITHMETIC EXPRESSION>;
IN ORDER TO AVOID OVERFLOW AND EVALUATION OF THOSE TERMS WHICH CAN BE NEGLECTED, MACHEXP HAS TO BE THE LARGEST ADMISSIBLE VALUE FOR WHICH TERMS WITH INDEX K * (2 ** MACHEXP) CAN BE COMPUTED (K IS SMALL). OTHERWISE OVERFLOW MIGHT OCCUR IN COMPUTING A VALUE FOR THE JENSEN PARAMETER I, WHICH CAN BE AN UNUSUALLY HIGH POWER OF 2.

PROCEDURES USED : EULER = CP32010.

REQUIRED CENTRAL MEMORY :

EXECUTION FIELD LENGTH : ABOUT 1000 * RECURSION DEPTH.

LANGUAGE : ALGOL 60.

METHOD AND PERFORMANCE :

WHEN THE TERMS AI WITH INDICES
MAXADDUP + 1, ... , MAXADDUP + TIM ARE ALL LESS THAN MAXZERO,
CONVERGENCE IS ASSUMED AND SUMPOSSERIES DELIVERS THE SUM OF THE
SERIES BY STRAIGHTFORWARD ADDITION UNTIL TIM SERIAL TERMS ARE LESS
THAN MAXZERO. OTHERWISE THE VAN WIJNGAARDEN TRANSFORMATION IS
APPLIED, YIELDING AN ALTERNATING SERIES WHICH IS SUMMED UP WITH
EULER'S METHOD. SINCE THE TERMS OF THIS ALTERNATING SERIES ARE
THEMSELVES INFINITE SERIES WITH POSITIVE TERMS, THE HERE
DESCRIBED PROCESS IS RECURSIVELY CALLED FOR THE SUMMATION OF EACH
TERM THAT IS WANTED BY EULER'S METHOD.
HOWEVER, ONLY STRAIGHTFORWARD ADDITION IS APPLIED IF THE ALLOWED
RECURSION LEVEL (SPECIFIED BY MAXRECURS) HAS BEEN REACHED.
IN THE RECURSION THE PROCESS ASKS FOR TERMS AI WITH INDICES OF
THE TYPE J * (2 ** K), IN WHICH K CAN BE VERY LARGE. IN ORDER
TO AVOID OVERFLOW AN UPPER BOUND FOR K MUST BE GIVEN IN MACHEXP.
IF K EXCEEDS THIS BOUND THE CORRESPONDING TERM IS TAKEN TO BE ZERO.

REFERENCES :

[1] DANIEL, J.W. :
SUMMATION OF A SERIES OF POSITIVE TERMS BY CONDENSATION
TRANSFORMATIONS. MATH. OF COMP. V.23, P.91-96 (1969).
[2] WIJNGAARDEN, A. VAN :
COURSE SCIENTIFIC COMPUTING B, PROCESS ANALYSIS (DUTCH)
MATHEMATISCH CENTRUM CR-18 (1965).

EXAMPLE OF USE :

THE PROGRAM :

```
"BEGIN""COMMENT" 730808, EXAMPLE OF THE USE OF SUMPOSSERIES;
   "REAL""PROCEDURE" SUMPOSSERIES(A, B, C, D, E, F, G); "CODE"
   32020;
   "INTEGER" I;
   OUTPUT(61, "("/, +.12D"+DD")",
   SUMPOSSERIES(1 / I ** 2, I, 100, "- 7, 8, 1068, 10))
"END"
```

DELIVERS :

```
+.1644934066O4"+01
NUMBER OF TERMS USED   :  462,
RECURSION DEPTH        :    1.
```

SOURCE TEXT(S) :

```
"CODE" 32010;
"REAL""PROCEDURE" EULER(AI, I, EPS, TIM);
"VALUE" EPS, TIM; "INTEGER" I, TIM; "REAL" AI, EPS;
"BEGIN""INTEGER" K, N, T; "REAL" MN, MP, DS, SUM; "ARRAY" M[0:15];
     N:= T:= 0; I:= 0; M[0]:= AI; SUM:= M[0] / 2;
NEXT TERM: I:= I + 1; MN:= AI;
     "FOR" K:= 0 "STEP" 1 "UNTIL" N "DO"
     "BEGIN" MP:= (MN + M[K]) / 2; M[K]:= MN; MN:= MP "END";
     "IF" ABS(MN) < ABS(M[N]) & N < 15 "THEN"
     "BEGIN" DS:= MN / 2; N:= N + 1; M[N]:= MN "END" "ELSE" DS:= MN;
     SUM:= SUM + DS; T:= "IF" ABS(DS) < EPS "THEN" T + 1 "ELSE" 0;
     "IF" T < TIM "THEN" "GO TO" NEXT TERM;
     EULER:= SUM
"END" EULER;
          "EOP"

"CODE" 32020;
"REAL" "PROCEDURE" SUMPOSSERIES(AI, I, MAXADDUP, MAXZERO,MAXRECURS,
                                MACHEXP, TIM);
"VALUE" MAXADDUP, MAXZERO, MAXRECURS, MACHEXP, TIM;
"REAL" AI, I, MAXZERO; "INTEGER" MAXADDUP, MAXRECURS, MACHEXP, TIM;
"BEGIN" "INTEGER" RECURS, VL, VL2, VL4;
        "REAL" "PROCEDURE" EULER(AI, I, EPS, TIM); "CODE" 32010;

        "REAL" "PROCEDURE" SUMUP(AI, I); "REAL" AI, I;
        "BEGIN" "INTEGER" J; "REAL" SUM, NEXTTERM;
           I:= MAXADDUP + 1; J:= 1;
        CHECK ADD: "IF" AI <= MAXZERO "THEN"
           "BEGIN""IF" J < TIM "THEN"
              "BEGIN" J:= J + 1; I:= I + 1; "GO TO" CHECK ADD "END"
           "END""ELSE"
           "IF" RECURS ^= MAXRECURS "THEN""GO TO" TRANSFORMSERIES;
           SUM:= 0; I:= 0; J:= 0;
        ADD LOOP: I:= I + 1; NEXTTERM:= AI;
           J:= "IF" NEXTTERM <= MAXZERO "THEN" J + 1 "ELSE" 0;
           SUM:= SUM + NEXTTERM;
           "IF" J < TIM "THEN""GO TO" ADD LOOP;
           SUMUP:= SUM; "GO TO" GOTSUM;
     TRANSFORMSERIES:
        "BEGIN""BOOLEAN" JODD; "INTEGER" J2; "ARRAY" V[1:VL];

           "REAL""PROCEDURE" BJK(J, K); "VALUE" J, K; "REAL" K;
           "INTEGER" J;
           "BEGIN""REAL" COEFF;
              "IF" K > MACHEXP "THEN" BJK:= 0 "ELSE"
              "BEGIN" COEFF:= 2 ** (K - 1); I:= J * COEFF;
                 BJK:= COEFF * AI
              "END"
           "END" BJK
```

```
;

"REAL""PROCEDURE" VJ(J); "VALUE" J; "INTEGER" J;
"BEGIN""REAL" TEMP, K;
"IF" JODD "THEN"
"BEGIN" JODD:= "FALSE"; RECURS:= RECURS + 1;
   TEMP:= VJ:= SUMUP(BJK(J, K), K);
   RECURS:= RECURS - 1;
   "IF" J <= VL "THEN" V[J]:= TEMP "ELSE"
   "IF" J <= VL2 "THEN" V[J - VL]:= TEMP
"END""ELSE"
"BEGIN" JODD:= "TRUE"; "IF" J > VL4 "THEN"
   "BEGIN" RECURS:= RECURS + 1;
      VJ:= - SUMUP(BJK(J, K), K); RECURS:= RECURS - 1
   "END""ELSE"
   "BEGIN" J2:= J2 + 1; I:= J2;
      "IF" J > VL2 "THEN" VJ:= - (V[J2 - VL] - AI) / 2
      "ELSE"
      "BEGIN" TEMP:= V["IF" J <= VL "THEN" J "ELSE"
         J - VL]:= (V[J2] - AI) / 2; VJ:= - TEMP
      "END"
   "END"
      "END"
   "END" VJ;

      J2:= 0;
      JODD:= "TRUE"; SUMUP:= EULER(VJ(J + 1), J, MAXZERO, TIM)
   "END" TRANSFORMSERIES;
GOTSUM:
"END" SUMUP;

   RECURS:= 0; VL:= 1000; VL2:= 2 * VL; VL4:= 2 * VL2;
   SUMPOSSERIES:= SUMUP(AI, I)
"END" SUMPOSSERIES;
    "EOP"
```

SECTION 4.2.1 CONTAINS TWO ALTERNATIVE PROCEDURES FOR THE COMPUTATION OF A DEFINITE INTEGRAL.

A. THE PROCEDURE QADRAT USES HIGH ORDER INTEGRATION RULES (UP TO 16-TH ORDER) AND IS APPROPRIATE FOR THE EVALUATION OVER A FINITE INTERVAL.

B. THE PROCEDURE INTEGRAL USES A 5-TH ORDER METHOD AND CAN ALSO BE USED TO CALCULATE THE INTEGRAL OVER A NUMBER OF CONSECUTIVE INTERVALS. MOREOVER THE PROCEDURE CAN BE USED FOR THE COMPUTATION OF THE DEFINITE INTEGRAL OVER AN INFINITE INTERVAL.

FOR A COMPARISION OF A NUMBER OF PROCEDURES THAT EVALUATE DEFINITE INTEGRALS : SEE REF[2].

REFERENCES :

[1] T.J.DEKKER AND C.J.ROOTHART.
INTRODUCTION TO NUMERICAL ANALYSIS. (DUTCH).
MATH. CENTRE REPORT CR 244/74, AMSTERDAM.

[2] C.J.ROOTHART AND H. FIOLET.
QUDRATURE PROCEDURES.
MATH. CENTRE REPORT MR 137/72, AMSTERDAM.

AUTHORS: C.J.ROOTHART.

CONTRIBUTORS: P.W.HEMKER.

INSTITUTE: MATHEMATICAL CENTRE.

RECEIVED: 730530.

BRIEF DESCRIPTION:

QADRAT COMPUTES THE DEFINITE INTEGRAL OF A FUNCTION OF ONE VARIABLE OVER A FINITE INTERVAL.

KEYWORDS:
INTEGRATION,
QUADRATURE,
DEFINITE INTEGRAL.

CALLING SEQUENCE:

THE HEADING OF THE PROCEDURE READS:
"REAL" "PROCEDURE" QADRAT (X, A, B, FX, E);
"VALUE" A, B; "REAL" X, A, B, FX; "ARRAY" E;

QADRAT: DELIVERS THE COMPUTED VALUE OF THE DEFINITE INTEGRAL FROM A TO B OF THE FUNCTION F(X);

THE MEANING OF THE FORMAL PARAMETERS IS:

X: <VARIABLE>;
INTEGRATION VARIABLE; X CAN BE USED AS A JENSEN-PARAMETER DURING THE EVALUATIONS OF FX;

A,B: <ARITHMETIC EXPRESSION>;
(A,B) DENOTES THE INTERVAL OF INTEGRATION;
B < A IS ALLOWED;

FX: <ARITHMETIC EXPRESSION>;
FX DENOTES THE INTEGRAND F(X). THIS EXPRESSION WILL BE DEPENDENT ON THE JENSEN-PARAMETER X.

E: <ARRAY IDENTIFIER>;
"ARRAY" E[1:3];
ENTRY: E[1]: THE RELATIVE ACCURACY REQUIRED;
E[2]: THE ABSOLUTE ACCURACY REQUIRED;
EXIT: E[3]: THE NUMBER OF ELEMENTARY INTEGRATIONS WITH H < ABS(B-A) * E[1].

PROCEDURES USED: NONE.

REQUIRED CENTRAL MEMORY:

EXECUTION FIELD LENGTH: CIRCA 16 + 9 * RECURSION DEPTH.

RUNNING TIME: DEPENDS STRONGLY ON THE DEFINITE INTEGRAL TO COMPUTE.

LANGUAGE: ALGOL 60.

METHOD AND PERFORMANCE: SEE REF[1].

REFERENCES:

[1].C.J.ROOTHART AND H.FIOLET.
QUADRATURE PROCEDURES.
MATH.CENTRE, AMSTERDAM. REPORT MR 137/72.

EXAMPLE OF USE:

```
"BEGIN"
    "REAL" "PROCEDURE" QADRAT(X,A,B,FX,E); "CODE" 32070;
    "ARRAY" E[1:3]; "REAL" T,Q;

    E[1]:= E[2]:= "-9;
    Q:= QADRAT(T,0,3.141592653589  ,SIN(T),E);
    OUTPUT(61,"("/,+.15D"+3D,3B,3ZD,/")",Q,E[3]);
"END"
```

DELIVERS:

```
+.20000000079740"+001        0
```

```
SOURCE TEXT(S);
"CODE" 32070;
"REAL" "PROCEDURE" QADRAT(X, A, B, FX, E);
"VALUE" A, B; "REAL" X, A, B, FX; "ARRAY" E;
"BEGIN" "REAL" F0, F2, F3, F5, F6, F7, F9,
     F14, V, W, HMIN, HMAX, RE, AE;

   "REAL" "PROCEDURE" LINT(X0, XN, F0, F2, F3, F5, F6, F7, F9, F14);
   "REAL" X0, XN, F0, F2, F3, F5, F6, F7, F9, F14;
   "BEGIN" "REAL" H, XM, F1, F4, F8, F10, F11, F12, F13;
      XM:= (X0 + XN) / 2; H:= (XN - X0) / 32; X:= XM + 4 * H;
      F8:= FX; X:= XN - 4 * H; F11:= FX; X:= XN - 2 * H; F12:= FX;
      V:= 0.33058017819922 6 * F7 + 0.17348511570733 8 * (F6 + F8) +
      0.32110542655997 2*(F5 + F9) + 0.13500770834104 2 * (F3 + F11)
      + 0.16571451422822 3*(F2 + F12) + 0.39397146063812 7"- 1 * (F0
      + F14); X:= X0 + H; F1:= FX; X:= XN - H; F13:= FX;
      W:= 0.26065243465697 0 * F7 + 0.23906328668476 5 * (F6 + F8) +
      0.26306263547746 7*(F5 + F9) + 0.21868193138305 7 * (F3 + F11)
      + 0.27578976466428 4"- 1 * (F2 + F12) + 0.10557501005384 6* (F1
      + F13) + 0.15711942605951 8"- 1 * (F0 + F14);
      "IF" ABS(H) < HMIN "THEN" E[3]:= E[3] + 1;
      "IF" ABS(V - W) < ABS(W) * RE + AE "OR" ABS(H) < HMIN
      "THEN" LINT:= H * W "ELSE"
      "BEGIN" X:= X0 + 6 * H; F4:= FX; X:= XN - 6 * H; F10:= FX;
         V:= 0.24567343009 3324* F7 + 0.25578625828 6921* (F6 + F8) +
         0.22852606369 0406*(F5 + F9) + 0.50055713152 5460"- 1*(F4 +
         F10) + 0.17794648773 6780*(F3 + F11)+0.58401459934 7449"- 1
         * (F2 + F12) + 0.87483094287 1331"- 1 * (F1 + F13) +
         0.18964207864 8079"- 1 * (F0 + F14);
         LINT:= "IF" ABS(V - W) < ABS(V) * RE + AE "THEN" H * V
                                                   "ELSE"
         LINT(X0, XM, F0, F1, F2, F3, F4, F5, F6, F7) - LINT(XN,
         XM, F14, F13, F12, F11, F10, F9, F8, F7)
      "END"
"END" LINT;

HMAX:= (B - A) / 16; "IF" HMAX=0 "THEN"
   "BEGIN" QADRAT:= 0; "GOTO" RETURN "END";
   RE:= E[1]; AE:= 2 * E[2] / ABS(B - A); E[3]:= 0;
   HMIN:= ABS(B - A) * RE; X:= A; F0:= FX;
   X:= A + HMAX; F2:= FX; X:= A + 2 * HMAX; F3:= FX;
   X:= A + 4 * HMAX; F5:= FX; X:= A + 6 * HMAX; F6:= FX;
   X:= A + 8 * HMAX; F7:= FX; X:= B - 4 * HMAX; F9:= FX; X:= B;
   F14:= FX;
   QADRAT:= LINT(A, B, F0, F2, F3, F5, F6, F7, F9, F14) * 16;
   RETURN;
"END" QADRAT;
     "EOP"
```

AUTHOR: H.N.GLORIE.

CONTRIBUTOR: H.FIOLET.

INSTITUTE: MATHEMATICAL CENTRE.

RECEIVED: 730606.

BRIEF DESCRIPTION:

INTEGRAL CALCULATES THE DEFINITE INTEGRAL OF A FUNCTION OF ONE VARIABLE, OVER A FINITE OR INFINITE INTERVAL OR OVER A NUMBER OF CONSECUTIVE INTERVALS.

KEYWORDS:
DEFINITE INTEGRAL,
INFINITE INTERVAL,
SIMPSON RULE,
RICHARDSON CORRECTION.

CALLING SEQUENCE:

THE HEADING OF THE PROCEDURE READS:

```
"REAL" "PROCEDURE" INTEGRAL(X,A,B,FX,E,UA,UB);
"VALUE" A,B;"REAL" X,A,B,FX;"ARRAY" E;"BOOLEAN" UA,UB;
```

INTEGRAL:

DELIVERS THE COMPUTED VALUE OF THE DEFINITE INTEGRAL OF THE FUNCTION FROM A TO B; AFTER SUCCESSIVE CALLS OF THE PROCEDURE, THE INTEGRAL OVER THE TOTAL INTERVAL IS DELIVERED, I.E. THE VALUE OF A IN THE LAST CALL WITH UA="TRUE" IS THE STARTING POINT OF THE INTERVAL.

THE MEANING OF THE FORMAL PARAMETERS IS:

X: <VARIABLE>;
INTEGRATION VARIABLE;X CAN BE USED AS JENSEN-PARAMETER FOR FX.

A,B: <ARITHMETIC EXPRESSION>;
(A,B) DENOTES THE INTERVAL OF INTEGRATION;B<A IS ALLOWED;

FX: <ARITHMETIC EXPRESSION>;
THE INTEGRAND F(X);

E: <ARRAY IDENTIFIER>;
"ARRAY" E[1:6];
ENTRY : E[1]:THE RELATIVE ACCURACY REQUIRED;
E[2]:THE ABSOLUTE ACCURACY REQUIRED;
EXIT: E[3]:THE NUMBER OF SKIPPED INTEGRATION STEPS;
E[4]:THE VALUE OF THE INTEGRAL FROM A TO B;
E[5]:"IF" UB "THEN" B "ELSE" 0;
E[6]:"IF" UB "THEN" F(B) "ELSE" 0;

UA: <BOOLEAN EXPRESSION>;
DETERMINES THE STARTING POINT OF THE INTEGRATION;
STARTING POINT:="IF" UA "THEN" A "ELSE" E[5];

UB: <BOOLEAN EXPRESSION>;
DETERMINES THE FINAL POINT OF THE INTEGRATION;
FINAL POINT:="IF" UB "THEN" B "ELSE"
"IF" B>A "THEN" +INFINITY "ELSE" -INFINITY;
IN THE CASE UB="FALSE" , THE VALUE OF B IS STILL RELEVANT (SEE METHOD AND PERFORMANCE).

PROCEDURES USED: NONE.

REQUIRED CENTRAL MEMORY:

EXECUTION FIELD LENGTH: CIRCA 16 + 5 * RECURSION LEVEL.

RUNNING TIME: DEPENDS STRONGLY ON THE INTEGRAL TO COMPUTE.

LANGUAGE: ALGOL 60.

METHOD AND PERFORMANCE:

INTEGRAL USES THE SUBPROCEDURE QAD FOR THE CALCULATION OF THE DEFINITE INTEGRAL OVER A FINITE INTERVAL. THIS IS DONE BY MEANS OF SIMPSON 'S RULE WITH RICHARDSON CORRECTION. IF THE FOURTH DERIVATIVE IS TOO LARGE (AND THUS THE CORRECTION TERM) , THE TOTAL INTERVAL IS SPLIT INTO TWO EQUAL PARTS AND THE INTEGRATION PROCESS IS INVOKED RECURSIVELY. THIS IS DONE IN SUCH A WAY THAT THE TOTAL AMOUNT OF RICHARDSON CORRECTIONS IS SLIGHTLY SMALLER THAN OR EQUAL TO
E[1] * ABS (THE INTEGRAL FROM A TO B OF (FX)) + E[2].
THE INTEGRATION OVER AN INFINITE INTERVAL REQUIRES TWO CALLS OF THE PROCEDURE QAD. IN THE FIRST CALL QAD COMPUTES THE DEFINITE INTEGRAL FROM A TO B . IN THE INTERVAL FROM B TO + OR - INFINITY THE INTEGRAND IS TRANSFORMED BY MEANS OF THE SUBSTITUTION Z=1/(X+1-B) TO THE DEFINITE INTEGRAL OF F(B-1+1/Z)/Z**2 FROM 0 TO 1. FOR THE INTEGRATION OF A DEFINITE INTEGRAL OVER A FINITE INTERVAL THE USE OF QADRAT IS RECOMMENDED ,ESPECIALLY WHEN HIGH ACCURACY IS REQUIRED.

REFERENCES:

[1] T.J.DEKKER AND C.J.ROOTHART.
INTRODUCTION TO NUMERICAL ANALYSIS.(DUTCH)
MATH.CENTR. REPORT CR 24/71, AMSTERDAM.
[2] C.J.ROOTHART AND H.FIOLET.
QUADRATURE PROCEDURES.
MATH.CENTR. REPORT MR 137/72, AMSTERDAM.

EXAMPLE OF USE:

```
"BEGIN"
    "REAL" "PROCEDURE" INTEGRAL(X,A,B,FX,E,UA,UB);"CODE" 32051;
    "ARRAY" E[1:6];"REAL" A,B,X;"BOOLEAN" UA,UB;
    UA:="TRUE";E[1]:=E[2]:="-14;
    "FOR" B:=2,4,20,100 "DO"
    "BEGIN" UB:=B<50;
        A:=INTEGRAL(X,-1,-B,10/X**2,E,UA,UB);
        OUTPUT(61,"("N,B+DDB,N,2(B+DDDB),/")",
              A,E[3],E[4],E[5],E[6]);
        UA:="FALSE"
    "END"
"END"
```

DELIVERS:

```
-4.9999999999999"+000    +00 -4.9999999999999    -002   +003
-7.4999999999998"+000    +00 -7.4999999999998    -004   +001
-9.5000000000000"+000    +00 -9.5000000000000    -020   +000
-9.9999999999998"+000    +01 -9.9999999999998    +000   +000
```

SOURCE TEXT(S);

```
"CODE" 32051;
"REAL" "PROCEDURE" INTEGRAL(X, A, B, FX, E, UA, UB);
"VALUE" A,B;"REAL" X, A, B, FX; "ARRAY" E; "BOOLEAN" UA, UB;
"BEGIN"
    "REAL" "PROCEDURE" TRANSF;
    "BEGIN" Z:= 1 / X; X:= Z + B1; TRANSF:= FX * Z * Z "END";
    "REAL" "PROCEDURE" QAD(FX); "REAL" FX;
    "BEGIN" "REAL" T, V, SUM, HMIN;
        "PROCEDURE" INT;
        "BEGIN" "REAL" X3, X4, F3, F4, H;
            X4:= X2; X2:= X1; F4:= F2; F2:= F1;
        ANEW: X:= X1:= (X0 + X2) * .5; F1:= FX;
            X:= X3:= (X2 + X4) * .5; F3:= FX; H:= X4 - X0;
            V:= (4 * (F1 + F3) +2  * F2 + F0 + F4) * 15;
            T:= 6 * F2 -4  * (F1 + F3) + F0 + F4;
            "IF" ABS(T) < ABS(V) * RE + AE "THEN"
            SUM:=SUM + (V - T) * H "ELSE"
            "IF" ABS(H) < HMIN "THEN" E[3]:= E[3] +1
            "ELSE"
            "BEGIN" INT; X2:= X3; F2:= F3; "GOTO" ANEW "END";
            X0:= X4; F0:= F4
    "END" INT;

    HMIN:= ABS(X0 - X2) * RE; X:= X1:= (X0 + X2) * .5;
    F1:=FX;SUM:= 0; INT; QAD:= SUM / 180
"END" QAD;
"REAL" X0, X1, X2, F0, F1, F2, RE, AE, B1, Z;
RE:= E[1]; "IF" UB "THEN" AE:= E[2] * 180 / ABS(B - A)
"ELSE" AE:= E[2] * 90 / ABS(B - A); "IF" UA "THEN"
"BEGIN" E[3]:= E[4]:= 0; X:= X0:= A; F0:= FX "END"
"ELSE"
"BEGIN" X:= X0:= A:= E[5]; F0:= E[6] "END";
E[5]:= X:= X2:= B; E[6]:= F2:= FX; E[4]:= E[4] + QAD(FX);
"IF"  "UB "THEN"
"BEGIN" "IF" A < B "THEN"
    "BEGIN" B1:= B -1 ; X0:= 1 "END"
        "ELSE"
        "BEGIN" B1:= B +1 ; X0:= -1  "END";
        F0:= E[6]; E[5]:= X2:= 0; E[6]:= F2:= 0;
        AE:= E[2] * 90;
        E[4]:= E[4] - QAD(TRANSF)
    "END";
    INTEGRAL:= E[4]
"END" INTEGRAL;
      "EOP"
```

AUTHOR : P.W. HEMKER.

CONTRIBUTOR : F.GROEN.

INSTITUTE : MATHEMATICAL CENTRE.

RECEIVED : 740620.

BRIEF DESCRIPTION :

TRICUB COMPUTES THE DEFINITE INTEGRAL OF A FUNCTION OF TWO VARIABLES OVER A TRIANGULAR DOMAIN.

KEYWORDS :

INTEGRATION,
QUADRATURE,
MOREDIMENSIONAL QUADRATURE,
CUBATURE,
DEFINITE INTEGRAL.

CALLING SEQJENCE :

THE HEADING OF THE PROCEDURE READS :

```
"REAL" "PROCEDURE" TRICUB ( XI, YI, XJ, YJ, XK, YK, F, RE, AE );
"VALUE" XI, YI, XJ, YJ, XK, YK, RE, AE;
"REAL"  XI, YI, XJ, YJ, XK, YK, RE, AE;
"REAL" "PROCEDURE" F;
```

TRICUB := THE COMPUTED VALUE OF THE DEFINITE INTEGRAL OF THE FUNCTION F (X, Y) OVER THE TRIANGULAR DOMAIN T WITH VERTICES (XI, YI), (XJ, YJ) AND (XK, YK).

THE MEANING OF THE FORMAL PARAMETERS IS :
XI, YI: <ARITHMETIC EXPRESSION>;
ENTRY : THE COORDINATES OF THE FIRST VERTEX OF THE TRIANGULAR DOMAIN OF INTEGRATION;
XJ, YJ: <ARITHMETIC EXPRESSION>;
ENTRY : THE COORDINATES OF THE SECOND VERTEX OF THE TRIANGULAR DOMAIN OF INTEGRATION;
XK, YK: <ARITHMETIC EXPRESSION>;
ENTRY : THE COORDINATES OF THE THIRD VERTEX OF THE TRIANGULAR DOMAIN OF INTEGRATION;
REMARK: THE ALGORITHM IS SYMMETRICAL IN THE VERTICES; THIS IMPLIES THAT THE RESULT OF THE PROCEDURE (ON ALL COUNTS) IS INVARIANT FOR ANY PERMUTATION OF THE VERTICES.
F : <PROCEDURE IDENTIFIER>;
THE HEADING OF THIS PROCEDURE READS:
"REAL" "PROCEDURE" F (X, Y); "REAL" X, Y;
THIS PROCEDURE DEFINES THE INTEGRAND;
AE, RE: <ARITHMETIC EXPRESSION>;
ENTRY: THE REQUIRED ABSOLUTE AND RELATIVE ERROR RESPECTIVELY. ONE SHOULD TO TAKE FOR "AE" AND "RE" VALUES WHICH ARE GREATER THAN THE ABSOLUTE AND RELATIVE ERROR IN THE COMPUTATION OF THE INTEGRAND F.

PROCEDURES USED : NONE.

REQUIRED CENTRAL MEMORY :

THE PROCESS IS PROGRAMMED RECURSIVELY. AT EACH RECURSION LEVEL 43 REAL NJMBERS ARE USED. HOWEVER, FOR ANY PROPERLY CHOSEN VALUES OF RE AND AE THE RECURSION DEPTH WILL NOT EXCEED THE NUMBER OF BITS IN A REAL'S MANTISSA.

RUNNING TIME : DEPENDS STRONGLY ON THE INTEGRAL TO COMPUTE.

LANGUAGE : ALGOL 60.

METHOD AND PERFORMANCE :

A NESTED SEQUENCE OF CUBATURE RULES OF ORDER 2, 3, 4 AND 5 IS APPLIED. IF THE DIFFERENCE BETWEEN THE RESULT WITH THE 4-TH DEGREE RULE AND THE RESULT WITH THE 5-TH DEGREE RULE IS TOO LARGE, THEN THE TRIANGLE IS DIVIDED INTO FOUR CONGRUENT TRIANGLES. THIS PROCESS IS APPLIED RECURSIVELY IN ORDER TO OBTAIN AN ADAPTIVE CUBATURE ALGORITHM.

REFERENCES :

[1].P.W.HEMKER,
A SEQUENCE OF NESTED CUBATURE RULES,
MATH.CENTRE, AMSTERDAM, REPORT NW 3/73.

EXAMPLE OF USE:

THE FOLLOWING PROGRAM EVALUATES THE INTEGRAL OF
F (X, Y) = COS (X) * COS (Y) OVER THE TRIANGLE T WITH
VERTICES (0, 0), (0, PI/2) AND (PI/2, PI/2).
ON EACH LINE ARE LISTED :
A: THE REQUIRED RELATIVE AND ABSOLUTE PRECISION;
B: THE COMPUTED VALUE OF THE INTEGRAL;
C: THE NUMBER OF CALLS OF THE FUNCTION F.

```
"BEGIN"
    "REAL" "PROCEDURE" TRICUB(A,B,C,D,E,F,G,H,I); "CODE" 32075;
    "INTEGER" N,C,I,K; "REAL" PI,ACC,R,S;
    "REAL" "PROCEDURE" E(X,Y); "REAL" X,Y;
    "BEGIN" C:= C+1;
            "IF" C> 20000 "THEN" "GOTO" CC;
            E:= COS(X) * COS(Y);
    "END" E;

    PI:= 3.141592653 59;
    "FOR" ACC:= "-1,"-2,"-3,"-4,"-5,"-6,"-7,"-8,"-9,"-10,"-11 "DO"
    "BEGIN" C:=0; OUTPUT(61,"("+.D"+ZD,2B,+.14D"+2ZD,2B,10ZD,/")",
              ACC, TRICUB(0,0,0,PI/2,PI/2,PI/2,E,ACC,ACC) ,C);
    "END";
    CC: OUTPUT(61,"("*")");
"END"
```

RESULTS:

```
+.1" +0   +.500639738019 70"   +0          7
+.1" -1   +.500639738019 70"   +0          7
+.1" -2   +.499991102615 04"   +0         10
+.1" -3   +.499998489592 26"   +0         13
+.1" -4   +.499998489592 26"   +0         13
+.1" -5   +.499999973782 40"   +0         43
+.1" -6   +.499999992097 92"   +0        133
+.1" -7   +.499999998931 72"   +0        313
+.1" -8   +.499999999855 71"   +0        733
+.1" -9   +.499999999986 92"   +0       1723
+.1"-10
```

SOURCE TEXT(S):

```
"CODE" 32075;
"REAL" "PROCEDURE" TRICUB(XI,YI,XJ,YJ,XK,YK,G,RE,AE);
"VALUE" XI,YI,XJ,YJ,XK,YK,RE,AE;
"REAL" XI,YI,XJ,YJ,XK,YK,RE,AE; "REAL" "PROCEDURE" G;
"BEGIN" "REAL" SURF,SURFMIN,XZ,YZ,XIJ,YIJ,XJK,YJK,XKI,YKI,GI,GJ,GK;

   "REAL" "PROCEDURE" INT(AX1,AY1,AF1,AX2,AY2,AF2,AX3,AY3,AF3,
                          BX1,BY1,BF1,BX2,BY2,BF2,BX3,BY3,BF3,
                          PX,PY,PF);
   "VALUE" BX1,BY1,BF1,BX2,BY2,BF2,BX3,BY3,BF3,PX,PY,PF;
   "REAL"  BX1,BY1,BF1,BX2,BY2,BF2,BX3,BY3,BF3,PX,PY,PF,
           AX1,AY1,AF1,AX2,AY2,AF2,AX3,AY3,AF3;
   "BEGIN" "REAL" E,I3,I4,I5,A,B,C,SX1,SY1,SX2,SY2,SX3,SY3,
       CX1,CY1,CF1,CX2,CY2,CF2,CX3,CY3,CF3,
       DX1,DY1,DF1,DX2,DY2,DF2,DX3,DY3,DF3;

       A:= AF1 + AF2 + AF3; B:= BF1 + BF2 + BF3;
       I3:= 3 * A + 27 * PF + 8 * B;
       E:= ABS(I3) * RE + AE;

       "IF" SURF < SURFMIN "OR" ABS(5 * A + 45 * PF - I3) < E
       "THEN" INT:= I3 * SURF "ELSE"
       "BEGIN" CX1:= AX1 + PX; CY1:= AY1 + PY; CF1:= G(CX1,CY1);
             CX2:= AX2 + PX; CY2:= AY2 + PY; CF2:= G(CX2,CY2);
             CX3:= AX3 + PX; CY3:= AY3 + PY; CF3:= G(CX3,CY3);
             C:= CF1 + CF2 + CF3;
             I4:= A + 9 * PF + 4 * B + 12 * C;

          "IF" ABS(I3 - I4) < E "THEN" INT:= I4 * SURF "ELSE"
          "BEGIN" SX1:= .5 * BX1; SY1:= .5 * BY1;
              DX1:= AX1 + SX1; DY1:= AY1 + SY1; DF1:= G(DX1,DY1);
              SX2:=  .5 * BX2; SY2:=  .5 * BY2;
              DX2:= AX2 + SX2; DY2:= AY2 + SY2; DF2:= G(DX2,DY2);
              SX3:=  .5 * BX3; SY3:=  .5 * BY3;
              DX3:= AX3 + SX3; DY3:= AY3 + SY3; DF3:= G(DX3,DY3);

          I5:= (51 * A + 2187 * PF + 276 * B + 972 * C -
                768 * (DF1 + DF2 + DF3))/63;
                                                    "COMMENT"
```

```
                "IF" ABS(I4 - I5) < E "THEN" INT:= I5 * SURF "ELSE"
                "BEGIN" SURF:= .25 * SURF;

                INT:=

                INT(SX1,SY1,BF1,SX2,SY2,BF2,SX3,SY3,BF3,
                    DX1,DY1,DF1,DX2,DY2,DF2,DX3,DY3,DF3,
                    PX,PY,PF) +

                INT(AX1,AY1,AF1,SX3,SY3,BF3,SX2,SY2,BF2,DX1,DY1,DF1,
                    AX1 + SX2,AY1 + SY2,G(AX1 + SX2,AY1 + SY2),
                    AX1 + SX3,AY1 + SY3,G(AX1 + SX3,AY1 + SY3),
                    .5 * CX1,.5 * CY1,CF1) +
                INT(AX2,AY2,AF2,SX3,SY3,BF3,SX1,SY1,BF1,DX2,DY2,DF2,
                    AX2 + SX1,AY2 + SY1,G(AX2 + SX1,AY2 + SY1),
                    AX2 + SX3,AY2 + SY3,G(AX2 + SX3,AY2 + SY3),
                    .5 * CX2,.5 * CY2,CF2) +
                INT(AX3,AY3,AF3,SX1,SY1,BF1,SX2,SY2,BF2,DX3,DY3,DF3,
                    AX3 + SX2,AY3 + SY2,G(AX3 + SX2,AY3 + SY2),
                    AX3 + SX1,AY3 + SY1,G(AX3 + SX1,AY3 + SY1),
                    .5 * CX3,.5 * CY3,CF3);

                    SURF:= 4 * SURF
                "END"
            "END"
        "END"
    "END" INT;

    SURF:= 0.5 * ABS(XJ * YK - XK * YJ + XI * YJ -
                     XJ * YI + XK * YI - XI * YK);
    SURFMIN:= SURF*RE; RE:= 30*RE; AE:= 30*AE/SURF;
    XZ:= (XI + XJ + XK)/3; YZ:= (YI + YJ + YK)/3;
    GI:= G(XI,YI); GJ:= G(XJ,YJ); GK:= G(XK,YK);
    XI:= XI*.5; YI:= YI*.5; XJ:= XJ*.5;
    YJ:= YJ*.5; XK:= XK*.5; YK:= YK*.5;

    TRICUB:= INT(XI,YI,GI,XJ,YJ,GJ,XK,YK,GK,
                 XJ+XK,YJ+YK,G(XJ+XK,YJ+YK),
                 XK+XI,YK+YI,G(XK+XI,YK+YI),
                 XI+XJ,YI+YJ,G(XI+XJ,YI+YJ),
                 .5 * XZ,.5 * YZ,G(XZ,YZ))/60
"END" TRICUB;
    "EOP"
```

AUTHORS: M.BAKKER.

INSTITUTE: MATHEMATICAL CENTRE, AMSTERDAM.

RECEIVED: 760131.

BRIEF DESCRIPTION:

THIS SECTION CONTAINS THE FOLLOWING PROCEDURES:

(1) GSS JAC WGHTS:

GIVEN THE TWO PARAMETERS ALFA AND BETA,THIS PROCEDURE
CALCULATES THE N ZEROS OF THE N-TH JACOBI POLYNOMIAL
AND THE CORRESPONDING GAUSS-CHRISTOFFEL NUMBERS NEEDED FOR
THE N-POINT GAUSS-JACOBI QUADRATURE OVER [-1,+1]
WITH WEIGHT FUNCTION

W(X) = (1-X)**ALFA*(1+X)**BETA;

(2) GSS LAG WGHTS:

GIVEN THE PARAMETER ALFA,THIS PROCEDURE
CALCULATES THE N ZEROS OF THE N-TH LAGUERRE POLYNOMIAL
AND THE GAUSS-CHRISTOFFEL NUMBERS NEEDED FOR THE
N-POINT GAUSS-LAGUERRE QUADRATURE OF F(X) OVER
(0, INFINITY) WITH RESPECT TO THE WEIGHT FUNCTION

W(X) = X**ALFA*EXP(-X).

THESE PROCEDURES CAN BE USED FOR GAUSSIAN QUADRATURE-
RULES OF THE JACOBI AND LAGUERRE TYPE.LET THE WEIGHT
FUNCTION W(X) AND THE INTERVAL(A,B) DETERMINE THE
SYSTEM OF POLYNOMIALS ORTHOGONAL ON (A,B) WITH RESPECT
TO W(X).THEN THE N-POINT GAUSSIAN QUADRATURE RULE
APPROXIMATES THE INTEGRAL

TO X=B
INTEGRAL F(X) W(X) DX
FROM X=A

BY THE EXPRESSION

J=N
SUM W[J] F(X[J])
J=1

WHERE THE ABSCISSAS X[J] ARE THE ZEROS OF THE N-TH
POLYNOMIAL AND W[J] ARE THE CORESPONDING GAUSS-CHRISTOFFEL
NUMBERS.

KEYWORDS:

GAUSSIAN QUADRATURE,
ZEROS OF ORTHOGONAL POLYNOMIALS,
GAUSS-CHRISTOFFEL NUMBERS,
GAUSSIAN WEIGHTS.

LANGUAGE: ALGOL 60.

REFERENCES:

[1] M.ABRAMOWITZ AND I.A. STEGUN.
HANDBOOK OF MATHEMATICAL FUNCTIONS, CH.22,

[2] J.STOER,
EINFUEHRUNG IN DIE NUMERISCHE MATHEMATIK 1,
SPRINGER VERLAG, BERLIN, HEIDELBERG, GOETTINGEN.

SUBSECTION: GSS JAC WGHTS.

CALLING SEQUENCE:

THE DECLARATION OF THE PROCEDURE IN THE CALLING PROGRAM READS:

```
"PROCEDURE" GSS JAC WGHTS(N, ALFA, BETA, X, W);
"VALUE" N, ALFA, BETA;
"INTEGER" N; "REAL" ALFA, BETA;
"CODE" 31425;
```

THE MEANING OF THE FORMAL PARAMETERS IS:

N: <ARITHMETIC EXPRESSION>;
THE UPPER BOUND OF THE ARRAYS X AND W; N>=1;

ALFA, BETA: <ARITHMETIC EXPRESSION>;
THE PARAMETERS OF THE WEIGHT FUNCTION FOR
THE JACOBI POLYNOMIALS; ALFA, BETA > -1;

X: <ARRAY IDENTIFIER>;
"ARRAY" X[1:N];
EXIT: X[I] IS THE I-TH ZERO OF THE N-TH JACOBI POLYNOMIAL;

W: <ARRAY IDENTIFIER>;
"ARRAY" W[1:N];
EXIT: W[I] IS THE GAUSS-CHRISTOFFEL NUMBER
ASSOCIATED WITH THE I-TH ZERO OF THE N-TH JACOBI POLYNOMIAL.

PROCEDURES USED:

GAMMA = CP 35061;
ALL JAC ZER = CP 31370.

REQUIRED CENTRAL MEMORY:

TWO AUXILIARY ARRAYS OF N REALS ARE USED.

RUNNING TIME: ROUGHLY PROPORTIONAL TO N CUBED.

METHOD AND PERFORMANCE:

AS IS WELL-KNOWN, THE GAUSSIAN QUADRATURE RULES ARE
BASED ON THE ZEROS OF ORTHOGONAL POLYNOMIALS.
PROCEDURES FOR THE COMPUTATION OF THESE ZEROS CAN
BE FOUND IN SECTION 3.6.2.
AFTER THE COMPUTATION OF THE ZEROS OF THE JACOBI POLYNOMIAL
THE GAUSSIAN WEIGHTS ARE COMPUTED OF THE FORMULA

$$W[I] = 1/\left(\sum_{J=0}^{J=N-1} P(J, ALFA, BETA, X[I])**2\right)$$

WHERE P(J, ALFA, BETA, X[I]) IS THE J-TH ORTHONORMAL
JACOBI POLYNOMIAL; SEE FURTHER [2], CH. III.

EXAMPLE OF USE:

THE PROGRAM

```
"BEGIN""COMMENT" EVALUATION OF THE INTEGRAL
  TO X=1
 INTEGRAL  (1+X)**2 * (1-X) * EXP(X) DX
 FROM X=-1
BY MEANS OF FIVE POINT GAUSS-JACOBI QUADRATURE.
THE EXACT VALUE IS 2*EXP(1)-10/EXP(1);
"REAL" ALFA, BETA, INT; "INTEGER" N; "ARRAY" X, W[1:5];
"REAL" "PROCEDURE" F(X); "VALUE" X; "REAL" X; F:=EXP(X);
"PROCEDURE" GSS JAC WGHTS (N, ALFA, BETA, X, W); "CODE" 31425;
ALFA:= 1; BETA:= 2; N:= 5; INT:= 0;
GSS JAC WGHTS( N, ALFA, BETA, X, W);
"FOR" N:= 1 "STEP" 1 "UNTIL" 5 "DO" INT:= INT + W[N] * F(X[N]);
OUTPUT(61, "(" /, "B+D.4D"+ZD")", INT - 2 * EXP(1) + 10 / EXP(1))
"END"
```

PRINTS THE FOLOWING RESULT:

```
 -1.5932"-10
```

SUBSECTION: GSS LAG WGHTS.

CALLING SEQUENCE:

THE DECLARATION OF THE PROCEDURE IN THE CALLING PROGRAM READS:

```
"PROCEDURE" GSS LAG WGHTS (N, ALFA, X, W);
"VALUE" N, ALFA;
"INTEGER" N; "REAL" ALFA;  "ARRAY" X, W;
"CODE" 31427;
```

THE MEANING OF THE FORMAL PARAMETERS IS:

N: <ARITHMETIC EXPRESSION>;
THE UPPER BOUND OF THE ARRAYS X AND W; N>=1;

ALFA: <ARITHMETIC EXPRESSION>;
THE PARAMETER OF THE WEIGHT FUNCTION FOR THE
LAGUERRE POLYNOMIALS;
ALFA>=1;

X: <ARRAY IDENTIFIER>;
"ARRAY" X[1: N];
EXIT: X[I] IS THE I-TH ZERO OF THE N-TH
LAGUERRE POLYNOMIAL;

W: <ARRAY IDENTIFIER>;
"ARRAY" W[1: N];
EXIT: W[I] IS THE GAUSSIAN WEIGHT CORRESPONDING
WITH THE I-TH ZERO OF THE N-TH LAGJERRE POLYNOMIAL.

PROCEDURES USED:

GAMMA = CP 35061,
ALL LAG ZER = CP 31371.

REQUIRED CENTRAL MEMORY:

TWO AUXILIARY ARRAYS OF N REALS ARE USED.

RUNNING TIME:

ROUGHLY PROPORTIONAL TO N CUBED.

METHOD AND PERFORMANCE:

THE ZEROS AND WEIGHTS ARE COMPUTED IN THE SAME
WAY AS IN THE PROCEDURE GSS JAC WGHTS.

EXAMPLE OF USE:

THE PROGRAM

```
"BEGIN" "COMMENT" COMPUTATION OF THE INTEGRAL FROM 0 TO INFINITY OF
   SIN(X)*EXP(-X) BY MEANS OF A TEN POINT GAUSS-LAGUERRE
   QUADRATURE.THE EXACT VALUE IS 0.5;
   "REAL" INT;"INTEGER" N;"ARRAY" X, W[1:10];
   "REAL""PROCEDURE" F(X); "VALUE"X; "REAL"X; F:=SIN(X);
   "PROCEDURE" GSS LAG WGHTS(N, ALFA, X, W); "CODE" 31427;
   GSS LAG WGHTS(10, 0, X, W);  INT:=0;
   "FOR" N:=10 "STEP" -1 "UNTIL" 1 "DO" INT:= INT + W[N] * F(X[N]);
   OUTPUT(61, "(-D.4D-2D)", INT-0.5)
"END"
```

PRINTS THE RESULT:

```
2.0497" -7
```

SOURCE TEXTS;

```
"CODE" 31425;
    "PROCEDURE" GSS JAC WGHTS(N, ALFA, BETA, X, W);
    "VALUE" N, ALFA, BETA; "INTEGER" N; "REAL" ALFA, BETA;
"ARRAY" X, W;
"IF" ALFA = BETA "THEN"
"BEGIN" "INTEGER" I, J, M;
    "ARRAY" B[1:N - 1]; "REAL" R0, R1, R2, S, H0, ALFA2, XI;
    "REAL" "PROCEDURE" GAMMA(X); "CODE" 35061;
    "PROCEDURE" ALL JAC ZER(N, ALFA, BETA, ZER); "CODE" 31370;

    ALL JAC ZER(N, ALFA, ALFA, X); ALFA2:= 2*ALFA;
    H0:= 2**(ALFA2 + 1)*GAMMA(1 + ALFA)**2/GAMMA(ALFA2 + 2);
    B[1]:= 1/SQRT(3 + ALFA2); M:= N - (N//2);
    "FOR" I:= 2 "STEP" 1 "UNTIL" N - 1 "DO"
    B[I]:= SQRT(I*(I + ALFA2)/(4*(I + ALFA)**2 - 1));
    "FOR" I:= 1 "STEP" 1 "UNTIL" M "DO"
    "BEGIN" XI:= ABS(X[I]); R0:= 1; R1:= XI/B[1];
        S:= 1 + R1*R1;
        "FOR" J:= 2 "STEP" 1 "UNTIL" N - 1 "DO"
        "BEGIN" R2:= (XI*R1 - B[J - 1]*R0)/B[J];
            R0:= R1; R1:= R2; S:= S + R2*R2
        "END";
        W[I]:= W[N + 1 - I]:= H0/S
    "END"
"END" "ELSE"
"BEGIN" "INTEGER" I, J; "ARRAY" A, B[0:N];
    "REAL" MIN, SUM, H0, R0, R1, R2, XI, ALFABETA;
    "PROCEDURE" ALL JAC ZER(N, ALFA, BETA, ZER); "CODE" 31370;
    "REAL" "PROCEDURE" GAMMA(X); "CODE" 35061;
    ALFABETA:= ALFA + BETA; MIN:= (BETA - ALFA)*ALFABETA;
    B[0]:= 0; SUM:= ALFABETA + 2; A[0]:= (BETA - ALFA)/SUM;
    A[1]:= MIN /SUM/(SUM + 2);
    B[1]:= 2*SQRT((1 + ALFA)*(1 + BETA)/(SUM + 1))/SUM;
    "FOR" I:= 2 "STEP" 1 "UNTIL" N - 1 "DO"
    "BEGIN" SUM:= I + I + ALFABETA;
        A[I]:= MIN/SUM/(SUM + 2);
        B[I]:= (2/SUM)*
        SQRT(I*(SUM - I)*(I + ALFA)*(I + BETA)/(SUM*SUM - 1))
    "END";
    H0:= 2**(ALFABETA + 1)*GAMMA(1 + ALFA)*GAMMA(1 + BETA)/
                                GAMMA(2 + ALFABETA);
    ALL JAC ZER(N, ALFA, BETA, X);
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" XI:= X[I]; R0:= 1; R1:= (XI - A[0])/B[1];
        SUM:= 1 + R1*R1;
        "FOR" J:= 2 "STEP" 1 "UNTIL" N - 1 "DO"
        "BEGIN" R2:= ((XI - A[J - 1])*R1 - B[J - 1]*R0)/B[J];
            SUM:= SUM + R2*R2; R0:= R1; R1:= R2
        "END";
        A[I]:= H0/SUM
    "END"
"END" GSS JAC WGHTS;
    "EOP"
```

```
"CODE" 31427;
   "PROCEDURE" GSS LAG WGHTS(N, ALFA, X, W);
"VALUE" N, ALFA; "INTEGER" N; "REAL" ALFA; "ARRAY" X, W;
"BEGIN" "INTEGER" I, J; "REAL" H0, S, R0, R1, R2, XI;
    "ARRAY" A, B[0:N];
    "PROCEDURE" ALL LAG ZER(N, ALFA, X); "CODE" 31371;
    "REAL" "PROCEDURE" GAMMA(X); "CODE" 35061;
    A[0]:= 1 + ALFA; A[1]:= 3 + ALFA; B[1]:= SQRT(A[0]);
    "FOR" I:= 2 "STEP" 1 "UNTIL" N - 1 "DO"
    "BEGIN" A[I]:= I + I + ALFA + 1;
        B[I]:= SQRT(I*(I + ALFA))
    "END";
    ALL LAG ZER(N, ALFA, X); H0:= GAMMA(1 + ALFA);
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" XI:= X[I]; R0:= 1;
        R1:= (XI - A[0])/B[1]; S:= 1 + R1*R1;
        "FOR" J:= 2 "STEP" 1 "UNTIL" N - 1 "DO"
        "BEGIN" R2:= ((XI - A[J - 1])*R1 - B[J - 1]*R0)/B[J];
            R0:= R1; R1:= R2; S:= S + R2*R2
            "END";
        W[I]:= H0/S
    "END"
"END" GSS LAG WGHTS;
    "EOP"
```

AUTHOR: J.C.P.BUS.

INSTITUTE: MATHEMATICAL CENTRE.

RECEIVED: 740218.

BRIEF DESCRIPTION:

THIS SECTION CONTAINS PROCEDURES FOR CALCULATING THE DERIVATIVES OF FUNCTIONS OF MORE VARIABLES, USING DIFFERENCE FORMULAS;
JACOBNNF CALCULATES THE JACOBIAN MATRIX OF AN N-DIMENSIONAL FUNCTION OF N VARIABLES USING FORWARD DIFFERENCES;
JACOBNMF CALCULATES THE JACOBIAN MATRIX OF AN N-DIMENSIONAL FUNCTION OF M VARIABLES USING FORWARD DIFFERENCES;
JACOBNBNDF CALCULATES THE JACOBIAN MATRIX OF AN N-DIMENSIONAL FUNCTION OF N VARIABLES, IF THIS JACOBIAN IS KNOWN TO BE A BAND MATRIX AND HAVE TO BE STORED ROW-WISE IN A ONE-DIMENSIONAL ARRAY.

KEYWORDS:

NUMERICAL DIFFERENTIATION,
FUNCTIONS OF MORE VARIABLES,
DIFFERENCE FORMULAS.

SUBSECTION: JACOBNNF.

CALLING SEQUENCE:

THE HEADING OF THIS PROCEDURE READS:
"PROCEDURE" JACOBNNF(N, X, F, JAC, I, DI, FUNCT);
"VALUE" N; "INTEGER" I, N; "REAL" DI; "ARRAY" X, F, JAC;
"PROCEDURE" FUNCT;

THE MEANING OF THE FORMAL PARAMETERS IS:

N: <ARITHMETIC EXPRESSION>;
THE NUMBER OF INDEPENDENT VARIABLES AND THE DIMENSION OF THE FUNCTION;

X: <ARRAY IDENTIFIER>;
"ARRAY" X[1:N];
ENTRY: THE POINT AT WHICH THE JACOBIAN HAS TO BE CALCULATED

F: <ARRAY IDENTIFIER>;
"ARRAY" F[1:N];
ENTRY : THE VALUES OF THE FUNCTION-COMPONENTS AT THE POINT GIVEN IN ARRAY X;

JAC: <ARRAY IDENTIFIER>;
"ARRAY" JAC[1:N, 1:N];
EXIT : THE JACOBIAN MATRIX IN SUCH A WAY THAT THE PARTIAL DERIVATIVE OF F[I] TO X[J] IS GIVEN IN
JAC[I, J], I, J = 1, ..., N;

I: <INTEGER VARIABLE>;
A JENSEN PARAMETER; DI MAY BE DEPENDENT OF I;

DI: <ARITHMETIC EXPRESSION>;
THE PARTIAL DERIVATIVES TO X[I] ARE APPROXIMATED WITH FORWARD DIFFERENCES, USING AN INCREMENT TO THE I-TH VARIABLE THAT EQUALS THE VALUE OF DI, I = 1, ..., N;

FUNCT: <PROCEDURE IDENTIFIER>;
THE HEADING OF THIS PROCEDURE SHOULD READ:
"PROCEDURE" FUNCT(N, X, F);
"VALUE" N; "INTEGER" N; "ARRAY" X, F;
THE MEANING OF THE FORMAL PARAMETERS IS:
N: <ARITHMETIC EXPRESSION>;
THE NUMBER OF INDEPENDENT VARIABLES OF THE FUNCTION F;
X: <ARRAY IDENTIFIER>;
THE INDEPENDENT VARIABLES ARE GIVEN IN X[1:N];
F: <ARRAY IDENTIFIER>;
AFTER A CALL OF FUNCT THE FUNCTION COMPONENTS SHOULD BE GIVEN IN F[1:N].

PROCEDURES USED: NONE.

REQUIRED CENTRAL MEMORY :

EXECUTION FIELD LENGTH: JACOBNNF DECLARES ONE AUXILIARY ARRAY OF ORDER N.

RUNNING TIME: PROPORTIONAL TO N ** 2.

LANGUAGE: ALGOL 60.

METHOD AND PERFORMANCE:

JACOBNNF CALCULATES THE JACOBIAN MATRIX OF AN N-DIMENSIONAL FUNCTION OF N VARIABLES; THE ELEMENTS OF THIS MATRIX, WHICH ARE THE PARTIAL DERIVATIVES OF THE FUNCTION, ARE CALCULATED USING FORWARD DIFFERENCES WITH AN INCREMENT TO THE I-TH VARIABLE OF DI, (I = 1, ..., N).

EXAMPLE OF USE:

LET F BE DEFINED BY:

```
        F[1]:= X[1] ** 3 + X[2],
        F[2]:= 10 * X[2];
```

THE JACOBIAN MATRIX AT THE POINT (2, 1) MAY BE CALCULATED AND PRINTED BY THE FOLLOWING PROGRAM:

```
"BEGIN"
    "INTEGER" I; "ARRAY" JAC[1:2, 1:2], X, F[1:2];
    "PROCEDURE" JACOBNNF(N, X, F, JAC, I, DI, FU); "CODE" 34437;
    "PROCEDURE" F1(N, X, F); "VALUE" N; "INTEGER" N; "ARRAY" X, F;
    "BEGIN" F[1]:= X[1] ** 3 + X[2]; F[2]:= X[2] * 10 "END" F1;
    X[1]:= 2; X[2]:= 1; F1(2, X, F);
    JACOBNNF(2, X, F, JAC, I, "IF" I = 1 "THEN" "-6 "ELSE" 1, F1);
    OUTPUT(71, "("*,4B,"("THE CALCULATED JACOBIAN IS:")",//,
    2(4B,2(N),/)")", JAC[1, 1], JAC[1, 2], JAC[2, 1], JAC[2, 2])
"END"
```

RESULTS:

THE CALCULATED JACOBIAN IS:

```
+1.2000005938262"+001   +1.0000000000000"+000
+0.0000000000000"+000   +1.0000000000000"+001
```

SUBSECTION: JACOBNMF.

CALLING SEQUENCE:

THE HEADING OF THIS PROCEDURE READS:
"PROCEDURE" JACOBNMF(N, M, X, F, JAC, I, DI, FUNCT);
"VALUE" N, M; "INTEGER" I, N, M; "REAL" DI; "ARRAY" X, F, JAC;
"PROCEDURE" FUNCT;

THE MEANING OF THE FORMAL PARAMETERS IS:

N: <ARITHMETIC EXPRESSION>;
THE NUMBER OF FUNCTION COMPONENTS;

M: <ARITHMETIC EXPRESSION>;
THE NUMBER OF INDEPENDENT VARIABLES;

X: <ARRAY IDENTIFIER>;
"ARRAY" X[1:M];
ENTRY: THE POINT AT WHICH THE JACOBIAN HAS TO BE CALCULATED

F: <ARRAY IDENTIFIER>;
"ARRAY" F[1:N];
ENTRY: THE VALUES OF THE FUNCTION-COMPONENTS AT THE POINT
GIVEN IN ARRAY X;

JAC: <ARRAY IDENTIFIER>;
"ARRAY" JAC[1:N, 1:M];
EXIT : THE JACOBIAN MATRIX IN SUCH A WAY THAT THE PARTIAL
DERIVATIVE OF F[I] TO X[J] IS GIVEN IN
JAC[I, J], I = 1, ..., N, J = 1, ... M;

I: <INTEGER VARIABLE>;
A JENSEN PARAMETER; DI MAY BE DEPENDENT OF I;

DI: <ARITHMETIC EXPRESSION>;
THE PARTIAL DERIVATIVES TO X[I] ARE APPROXIMATED WITH
FORWARD DIFFERENCES, USING AN INCREMENT TO THE I-TH
VARIABLE THAT EQUALS THE VALUE OF DI, I = 1, ..., M;

FUNCT: <PROCEDURE IDENTIFIER>;
THE HEADING OF THIS PROCEDURE READS :
"PROCEDURE" FUNCT(N, M, X, F);
"VALUE" N, M; "INTEGER" N, M; "ARRAY" X, F;
THE MEANING OF THE FORMAL PARAMETERS IS :

N: <ARITHMETIC EXPRESSION>;
THE NUMBER OF FUNCTION COMPONENTS;

M: <ARITHMETIC EXPRESSION>;
THE NUMBER OF INDEPENDENT VARIABLES OF THE FUNCTION F;

X: <ARRAY IDENTIFIER>;
THE INDEPENDENT VARIABLES ARE GIVEN IN X[1:M];

F: <ARRAY IDENTIFIER>;
AFTER A CALL OF FUNCT THE FUNCTION COMPONENTS SHOULD BE
GIVEN IN F[1:N].

PROCEDURES USED: NONE.

EXECUTION FIELD LENGTH: JACOBNMF DECLARES ONE AUXILIARY ARRAY OF ORDER N.

RUNNING TIME: PROPORTIONAL TO N * M.

LANGUAGE: ALGOL 60.

METHOD AND PERFORMANCE:

JACOBNMF CALCULATES THE JACOBIAN MATRIX OF AN N-DIMENSIONAL FUNCTION OF M VARIABLES; THE ELEMENTS OF THIS MATRIX, WHICH ARE THE PARTIAL DERIVATIVES OF THE FUNCTION, ARE CALCULATED USING FORWARD DIFFERENCES WITH AN INCREMENT TO THE I-TH VARIABLE OF DI,(I = 1, ..., M).

EXAMPLE OF USE:

LET F BE DEFINED BY:

```
F[1]= X[1] ** 3 + X[2],
F[2]= 10 * X[2] + X[2] * X[1],
F[3]= X[1] * X[2];
```

THE JACOBIAN MATRIX AT THE POINT (2, 1) MAY BE CALCULATED AND PRINTED BY THE FOLLOWING PROGRAM:

```
"BEGIN"
    "INTEGER" I; "ARRAY" JAC[1:3, 1:2], X[1:2], F[1:3];
    "PROCEDURE" JACOBNMF(N, X, F, JAC, I, DI, FU); "CODE" 34438;
    "PROCEDURE" F1(N, M, X, F); "VALUE" N, M; "INTEGER" N, M;
    "ARRAY" X, F;
    "BEGIN" F[1]:= X[1] ** 3 + X[2];
        F[2]:= X[2] * 10 + X[2] * X[1] ** 2; F[3]:= X[1] * X[2]
    "END" F1;
    X[1]:= 2; X[2]:= 1; F1(3, 2, X, F);
    JACOBNMF(3, 2, X, F, JAC, I, "IF" I=2 "THEN" 1 "ELSE" "-5, F1);
    OUTPUT(71, "("*,4B,"("THE CALCULATED JACOBIAN IS:")",//,
    3(4B,2(N),/)")", JAC[1, 1], JAC[1, 2], JAC[2, 1], JAC[2, 2],
    JAC[3, 1], JAC[3, 2])
"END"
```

RESULTS:

THE CALCULATED JACOBIAN IS:

```
+1.2000600002038"+001   +1.0000000000000"+000
+4.0000100000270"+000   +1.4000000000000"+001
+1.0000000003174"+000   +2.0000000000000"+000
```

SUBSECTION: JACOBNBNDF.

CALLING SEQUENCE:

THE HEADING OF THIS PROCEDURE READS :
"PROCEDURE" JACOBNBNDF(N, LW, RW, X, F, JAC, I, DI, FUNCT);
"VALUE" N, LW, RW; "INTEGER" N, I, LW, RW; "REAL" DI;
"ARRAY" X, F, JAC; "PROCEDURE" FUNCT;

THE MEANING OF THE FORMAL PARAMETERS IS:

N: <ARITHMETIC EXPRESSION>;
THE NUMBER OF INDEPENDENT VARIABLES AND THE DIMENSION OF THE FUNCTION;

LW: <ARITHMETIC EXPRESSION>;
THE NUMBER OF CODIAGONALS TO THE LEFT OF THE MAIN DIAGONAL OF THE JACOBIAN MATRIX, WHICH IS KNOWN TO BE A BAND MATRIX;

RW: <ARITHMETIC EXPRESSION>;
THE NUMBER OF CODIAGONALS TO THE RIGHT OF THE MAIN DIAGONAL OF THE JACOBIAN MATRIX;

X: <ARRAY IDENTIFIER>;
"ARRAY" X[1:N];
ENTRY: THE POINT AT WHICH THE JACOBIAN HAS TO BE CALCULATED

F: <ARRAY IDENTIFIER>;
"ARRAY" F[1:N];
ENTRY: THE VALUES OF THE FUNCTION-COMPONENTS AT THE POINT GIVEN IN ARRAY X;

JAC: <ARRAY IDENTIFIER>;
"ARRAY" JAC [1 : (LW + RW) * (N - 1) + N];
EXIT: THE JACOBIAN MATRIX IN SUCH A WAY THAT THE (I, J)-TH ELEMENT OF THE JACOBIAN, I.E. THE PARTIAL DERIVATIVE OF F[I] TO X[J], IS GIVEN IN
JAC[(LW + RW) * (I - 1) + J], FOR I = 1, ..., N
J= MAX(1, I - LW), ..., MIN(N, I + RW);

I: <INTEGER VARIABLE>;
A JENSEN PARAMETER; DI MAY BE DEPENDENT OF I;

DI: <ARITHMETIC EXPRESSION>;
THE PARTIAL DERIVATIVES TO X[I] ARE APPROXIMATED WITH FORWARD DIFFERENCES, USING AN INCREMENT TO THE I-TH VARIABLE THAT EQUALS THE VALUE OF DI, I = 1, ..., N;

FUNCT: <PROCEDURE IDENTIFIER>;
THE HEADING OF THIS PROCEDURE READS :
"PROCEDURE" FUNCT(N, L, U, X, F);
"VALUE" N, L, U; "INTEGER" N, L, U; "ARRAY" X, F;
THE MEANING OF THE FORMAL PARAMETERS IS :

N: <ARITHMETIC EXPRESSION>;
THE NUMBER OF FUNCTION COMPONENTS;

L,U: <ARITHMETIC EXPRESSION>;
LOWER AND UPPER BOUND OF THE FUNCTION COMPONENT SUBSCRIPT;

X: <ARRAY IDENTIFIER>;
THE INDEPENDENT VARIABLES ARE GIVEN IN X[1:N];

F: <ARRAY IDENTIFIER>;
AFTER A CALL OF FUNCT THE FUNCTION COMPONENTS F[I], I = L, ..., U, SHOULD BE GIVEN IN F[L:U].

PROCEDURES USED: NONE.

EXECUTION FIELD LENGTH: JACOBNMF DECLARES ONE AUXILIARY ARRAY OF MAXIMUM ORDER LW + RW + 1;

RUNNING TIME: PROPORTIONAL TO N * (LW + RW + 1).

LANGUAGE: ALGOL 60.

METHOD AND PERFORMANCE:

JACOBNBNDF CALCULATES THE JACOBIAN MATRIX OF AN N-DIMENSIONAL FUNCTION OF N VARIABLES, IF THIS JACOBIAN IS KNOWN TO BE A BAND MATRIX AND HAVE TO BE STORED ROW-WISE IN A ONE-DIMENSIONAL ARRAY; THE ELEMENTS OF THIS JACOBIAN MATRIX ARE CALCULATED USING FORWARD DIFFERENCES, WITH AN INCREMENT TO THE I-TH VARIABLE OF DI, (I = 1, ..., N).

EXAMPLE OF USE:

```
     LET F BE DEFINED BY:
F[1] = (3 - 2 * X[1]) * X[1] + 1 - 2 * X[2],
F[I] = (3 - 2 * X[I]) * X[I] + 1 - X[I-1] - 2 * X[I+1], I= 2, 3, 4,
F[5] = 4 - 2 * X[5] - X[4];
THE TRIDIAGONAL JACOBIAN MATRIX AT THE POINT X, GIVEN BY X[I] = -1,
I = 1, ..., 5,  MAY  BE  CALCULATED  AND  PRINTED  BY THE FOLLOWING
PROGRAM;

"BEGIN"
     "INTEGER" I; "ARRAY" JAC[1:13], X, F[1:5];
     "PROCEDURE" JACOBNBNDF(N, L, R, X, F, J, I, D,G); "CODE" 34439;
     "PROCEDURE" F1(N, L, U, X, F); "VALUE" N, L, U;
     "INTEGER" N, L, U; "ARRAY" X, F;
     "BEGIN" "INTEGER" I;
          "FOR" I:= L "STEP" 1 "UNTIL" ("IF" U = 5 "THEN" 4 "ELSE" U)
          "DO"
          "BEGIN" F[I]:= (3 - 2 * X[I]) * X[I] + 1 - 2 * X[I + 1];
               "IF" I ^= 1 "THEN" F[I]:= F[I] - X[I - 1]
     "END";
     "IF" U = 5 "THEN" F[5]:= 4 - X[4] - X[5] * 2
"END" F1;

"PROCEDURE" LIST(ITEM); "PROCEDURE" ITEM;
"BEGIN" "INTEGER" I;
     ITEM("("THE CALCULATED TRIDIAGONAL JACOBIAN IS:")");
     "FOR" I:= 1 "STEP" 1 "UNTIL" 13 "DO" ITEM(JAC[I])
 "END" LIST;

"PROCEDURE" LAYOUT;
FORMAT("("//,4B,40S,//,4B,2(+.5D"+D2B),/,4B,3(+.5D"+D2B),/,
16B,3(+.5D"+D2B),/,28B,3(+.5D"+D2B),/,40B,2(+.5D"+D2B),/")");
"FOR" I := 1 "STEP" 1 "UNTIL" 5 "DO" X[I]:= -1;
F1(5, 1, 5, X, F);
JACOBNBNDF(5, 1, 1, X, F, JAC, I, "IF" I = 5 "THEN" 1 "ELSE"
"-6, F1);
OUTLIST(71, LAYOUT, LIST)
"END"
```

RESULTS:

THE CALCULATED TRIDIAGONAL JACOBIAN IS:

```
+.70000"+1   -.20000"+1
-.10000"+1   +.70000"+1   -.20000"+1
             -.10000"+1   +.70000"+1   -.20000"+1
                          -.10000"+1   +.70000"+1   -.20000"+1
                                       -.10000"+1   -.20000"+1
```

SOURCE TEXT(S):

```
"CODE" 34437;
"PROCEDURE" JACOBNNF(N, X, F, JAC, I, DI, FUNCT); "VALUE" N;
"INTEGER" N, I; "REAL" DI; "ARRAY" X, F, JAC; "PROCEDURE" FUNCT;
"BEGIN" "INTEGER" J; "REAL" STEP, AID; "ARRAY" F1[1:N];
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" STEP:= DI; AID:= X[I]; X[I]:= AID + STEP;
        STEP:= 1 / STEP; FUNCT(N, X, F1);
        "FOR" J:= 1 "STEP" 1 "UNTIL" N "DO"
        JAC[J,I]:= (F1[J] - F[J]) * STEP; X[I]:= AID
    "END"
"END" JACOBNNF;
        "EOP"

"CODE" 34438;
"PROCEDURE" JACOBNMF(N, M, X, F, JAC, I, DI, FUNCT); "VALUE" N, M;
"INTEGER" N, M, I; "REAL" DI; "ARRAY" X, F, JAC; "PROCEDURE" FUNCT;
"BEGIN" "INTEGER" J; "REAL" STEP, AID; "ARRAY" F1[1:N];
    "FOR" I:= 1 "STEP" 1 "UNTIL" M "DO"
    "BEGIN" STEP:= DI; AID:= X[I]; X[I]:= AID + STEP;
        STEP:= 1 / STEP; FUNCT(N, M, X, F1);
        "FOR" J:= 1 "STEP" 1 "UNTIL" N "DO"
        JAC[J,I]:= (F1[J] - F[J]) * STEP; X[I]:= AID
    "END"
"END" JACOBNMF;
        "EOP"

"CODE" 34439;
    "PROCEDURE" JACOBNBNDF(N, LW, RW, X, F, JAC, I, DI, FUNCT);
    "VALUE" N, LW, RW; "INTEGER" I, N, LW, RW; "REAL" DI;
    "ARRAY" X, F, JAC; "PROCEDURE" FUNCT;
    "BEGIN" "INTEGER" J, K, L, U, T, B; "REAL" AID, STEPI;
        L:= 1; U:= LW + 1; T:= RW + 1; B:= LW + RW;
        "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
        "BEGIN" "ARRAY" F1[L:U];
            STEPI:= DI; AID:= X[I]; X[I]:= AID + STEPI;
            FUNCT(N, L, U, X, F1); X[I]:= AID;
            K:= I + ("IF" I <= T "THEN" 0 "ELSE" I - T) * B;
            "FOR" J:= L "STEP" 1 "UNTIL" U "DO"
            "BEGIN" JAC[K]:= (F1[J] - F[J]) / STEPI; K:=K + B "END";
            "IF" I >= T "THEN" L:= L + 1;
            "IF" U < N "THEN" U:= U + 1
    "END"
"END" JACOBNBNDF;
    "EOP"
```