Algorithm 436

Product Type Trapezoidal Integration [D1]

W. Robert Boland [Recd. 10 Dec. 1970 and 14 May 1971]
Department of Mathematics, Clemson University, Clemson, SC 29631

Key Words and Phrases: numerical integration, product type quadrature, trapezoidal integration
CR Categories: 5.16
Language: Fortran

Description

This subroutine uses the product type trapezoidal rule compounded n times to approximate the value of the integral

$$\int_a^b f(x)g(x)\,dx.$$

The approximating sum is

$$\frac{h}{6}\sum_{j=1}^{n}\left(f(a+(j-1)h), f(a+jh)\right)\begin{pmatrix}2 & 1\\ 1 & 2\end{pmatrix}\begin{pmatrix}g(a+(j-1)h)\\ g(a+jh)\end{pmatrix},$$

where $h = (b-a)/n$. Note that if $g(x) \equiv 1$ (or $f(x) \equiv 1$), the rule reduces to the regular trapezoidal rule. The procedure was proposed and discussed by Boland and Duris in [1].

The subroutine was written in Fortran using double precision arithmetic and was checked on an IBM 360 Model 50. The calling parameters for the routine are as follows. A is the name for the lower limit of integration, and B is the name for the upper limit. N is the number of times the formula is to be compounded. The basic interval $[A, B]$ is subdivided into N subintervals each of length $(B - A)/N$ and the rule is applied to each subinterval. *FN* and *GN* are names of double precision *FUNCTION* subprograms which evaluate the functions $f(x)$ and $g(x)$, respectively. These are to be supplied by the user. The result is stored in *VINT*.

There are no machine dependent parameters.

References

1. Boland, W.R., and Duris, C.S. Product type quadrature formulas. *BIT 11*, 2 (1971), 139–158.

Algorithm

```
      SUBRØUTINE  PTRAP(A, B, N, FN, GN, VINT)
C
C THIS SUBRØUTINE USES THE PRØDUCT TYPE TRAPEZØIDAL RULE
C CØMPØUNDED N TIMES TØ APPRØXIMATE THE INTEGRAL FRØM A TØ B
C ØF THE FUNCTIØN FN(X) * GN(X). FN AND GN ARE FUNCTIØN
C SUBPRØGRAMS WHICH MUST BE SUPPLIED BY THE USER. THE
C RESULT IS STØRED IN VINT.
C
      DØUBLE PRECISIØN A, AG, AM(2,2), B, F(2), FN, G(2),
     *                 GN, H, VINT, X, DBLE
      DATA AM(1,1), AM(2,2) /2 * 2.D0/, AM(1,2), AM(2,1)
     *    /2 * 1.D0/
      H = (B - A) / DBLE(FLØAT(N))
      VINT = 0.D0
      X = A
      F(2) = FN(A)
      G(2) = GN(A)
      DØ 2 I = 1, N
        F(1) = F(2)
        G(1) = G(2)
        X = X + H
        F(2) = FN(X)
        G(2) = GN(X)
        DØ 2 J = 1, 2
          AG = 0.D0
          DØ 1 K = 1, 2
    1       AG = AG + AM(J,K) * G(K)
    2     VINT = VINT + F(J) * AG
      VINT = H * VINT / 6.D0
      RETURN
      END
```

Algorithm 437

Product Type Simpson's Integration [D1]

W. Robert Boland [Recd. 10 Dec. 1970 and 14 May 1971]
Department of Mathematics, Clemson University, Clemson, SC 29631

Key Words and Phrases: numerical integration, product type quadrature, Simpson's rule
CR Categories: 5.16
Language: Fortran

Description

This subroutine uses the product type Simpson's rule compounded n times to approximate the value of the integral

$$\int_a^b f(x)g(x)\,dx.$$

The approximating sum is

$$\frac{h}{30}\sum_{j=1}^{n}\left(f(a+(j-1)h), f(a+(j-\tfrac{1}{2})h), f(a+jh)\right) \cdot \begin{pmatrix}4 & 2 & -1\\ 2 & 16 & 2\\ -1 & 2 & 4\end{pmatrix}\begin{pmatrix}g(a+(j-1)h)\\ g(a+(j-\tfrac{1}{2})h)\\ g(a+jh)\end{pmatrix},$$

where $h = (b-a)/n$. Note that if $g(x) \equiv 1$ (or $f(x) \equiv 1$), the rule reduces to the regular Simpson's rule. The procedure was proposed and discussed by Boland and Duris in [1].

The subroutine was written in Fortran using double precision arithmetic and was checked on an IBM 360 Model 50. The calling parameters for the routine are as follows. A is the name for the lower limit of integration and B is the name for the upper limit. N is the number of times the formula is to be compounded. The basic interval $[A, B]$ is subdivided into N subintervals each of length $(B - A)/N$ and the rule is applied to each subinterval. *FN* and *GN* are names of double precision *FUNCTION* subprograms which evaluate the functions $f(x)$ and $g(x)$, respectively. These are to be supplied by the user. The result is stored in *VINT*.

There are no machine dependent parameters.

References

1. Boland, W.R., and Duris, C.S. Product type quadrature formulas. *BIT 11*, 2 (1971), 139–158.

Algorithm

```
      SUBROUTINE PSIMP(A,B,N,FN,GN,VINT)
C
C THIS SUBROUTINE USES THE PRODUCT TYPE SIMPSON RULE
C COMPOUNDED N TIMES TO APPROXIMATE THE INTEGRAL FROM A TO B
C OF THE FUNCTION FN(X) * GN(X). FN AND GN ARE FUNCTION
C SUBPROGRAMS WHICH MUST BE SUPPLIED BY THE USER. THE
C RESULT IS STORED IN VINT.
C
      DOUBLE PRECISION A, AG, AM(3,3), B, F(3), FN, G(3),
     *                 GN, H, VINT, X(2), DBLE
      DATA AM(1,1), AM(3,3) /2 * 4.D0/, AM(1,2), AM(2,1),
     *     AM(2,3), AM(3,2) /4 * 2.D0/, AM(1,3), AM(3,1)
     *     /2 * -1.D0/, AM(2,2) /16.D0/
      H = (B - A) / DBLE(FLOAT(N))
      X(1) = A + H / 2.D0
      X(2) = A + H
      VINT = 0.D0
      F(3) = FN(A)
      G(3) = GN(A)
      DO 3 I = 1, N
        F(1) = F(3)
        G(1) = G(3)
        DO 1 J = 1,2
          F(J+1) = FN(X(J))
          G(J+1) = GN(X(J))
    1     X(J) = X(J) + H
        DO 3 J = 1, 3
          AG = 0.D0
          DO 2 K = 1, 3
    2       AG = AG + AM(J,K) * G(K)
    3     VINT = VINT + F(J) * AG
      VINT = H * VINT / 30.D0
      RETURN
      END
```

Algorithm 438

Product Type Two-point Gauss-Legendre-Simpson's Integration [D1]

W. Robert Boland [Recd. 10 Dec. 1970 and 14 May 1971]
Department of Mathematics, Clemson University, Clemson, SC 29631

Key Words and Phrases: numerical integration, product type quadrature, Gaussian quadrature, Simpson's rule
CR Categories: 5.16
Language: Fortran

Description

This subroutine uses the product type two-point Gauss-Legendre-Simpson's rule compounded n times to approximate the value of the integral

$$\int_a^b f(x)g(x)\,dx.$$

The approximating sum is

$$\frac{h}{12}\sum_{j=1}^{n}\left(f(a+(j-\tfrac{1}{2}-3^{1/2}/6)h), f(a+(j-\tfrac{1}{2}+3^{1/2}/6)h)\right) \cdot \begin{pmatrix} 1+3^{1/2} & 4 & 1-3^{1/2} \\ 1-3^{1/2} & 4 & 1+3^{1/2} \end{pmatrix} \begin{pmatrix} g(a+(j-1)h) \\ g(a+(j-\frac{1}{2})h) \\ g(a+jh) \end{pmatrix},$$

where $h = (b - a)/n$. Note that if $g(x) \equiv 1$, the rule reduces to the regular two-point Gauss-Legendre rule, while if $f(x) \equiv 1$, it reduces to the regular Simpson's rule. The procedure was proposed and discussed by Boland and Duris in [1].

The subroutine was written in Fortran using double precision arithmetic and was checked on an IBM 360 Model 50. The calling parameters for the routine are as follows. A is the name for the lower limit of integration and B is the name for the upper limit. N is the number of times the formula is to be compounded. The basic interval $[A, B]$ is subdivided into N subintervals each of length $(B - A)/N$ and the rule is applied to each subinterval. *FN* and *GN* are names of double precision *FUNCTION* subprograms which evaluate the functions $f(x)$ and $g(x)$, respectively. These are to be supplied by the user. The result is stored in *VINT*.

There are four machine dependent constants. These are:

(i) $1 + 3^{1/2} \approx 2.732050807568877$,
(ii) $1 - 3^{1/2} \approx -0.7320508075688773$,
(iii) $\frac{1}{2} - 3^{1/2}/6 \approx 0.2113248654051871$, and
(iv) $\frac{1}{2} + 3^{1/2}/6 \approx 0.7886751345948129$.

The first constant is assigned to $AM(1, 1)$ and $AM(2, 3)$, the second to $AM(1, 3)$ and $AM(2, 1)$, while the third and fourth are used in the calculation of $X(1)$ and $X(2)$, respectively.

References

1. Boland, W.R., and Duris, C.S. Product type quadrature formulas. *BIT 11*, 2 (1971), 139–158.

Algorithm

```
      SUBROUTINE  P2PGS(A, B, N, FN, GN, VINT)
C
C THIS SUBROUTINE USES THE PRODUCT TYPE TWO-POINT GAUSS-
C LEGENDRE-SIMPSON RULE COMPOUNDED N TIMES TO APPROXIMATE
C THE INTEGRAL FROM A TO B OF THE FUNCTION FN(X) * GN(X).
C FN AND GN ARE FUNCTION SUBPROGRAMS WHICH MUST BE SUPPLIED
C BY THE USER.  THE RESULT IS STORED IN VINT.
C
      DOUBLE PRECISION A, AG, AM(2,3), B, F(2), FN, G(3),
     *                 GN, H, VINT, X(2), Y(2), DBLE
      DATA AM(1,1), AM(2,3) /2 * 2.732050807568877D0/,
     *     AM(1,2), AM(2,2) /2 * 4.D0/, AM(1,3), AM(2,1)
     *     /2 * -.7320508075688773D0/
      H = (B - A) / DBLE(FLOAT(N))
      X(1) = A + .2113248654051871D0 * H
      X(2) = A + .7886751345948129D0 * H
      Y(1) = A + H / 2.D0
      Y(2) = A + H
      VINT = 0.D0
      G(3) = GN(A)
      DO 3 I = 1, N
        G(1) = G(3)
        DO 1 J = 1, 2
          F(J) = FN(X(J))
          G(J+1) = GN(Y(J))
          X(J) = X(J) + H
    1     Y(J) = Y(J) + H
        DO 3 J = 1, 2
          AG = 0.D0
          DO 2 K = 1, 3
    2       AG = AG + AM(J,K) * G(K)
    3     VINT = VINT + F(J) * AG
      VINT = H * VINT / 12.D0
      RETURN
      END
```

Algorithm 439

Product Type Three-point Gauss-Legendre-Simpson's Integration [D1]

W. Robert Boland [Recd. 10 Dec. 1970 and 14 May 1971]
Department of Mathematics, Clemson University, Clemson, SC 29631

Key Words and Phrases: numerical integration, product type quadrature, Gaussian quadrature, Simpson's rule
CR Categories: 5.16
Language: Fortran

Description

This subroutine uses the product type three-point Gauss-Legendre-Simpson's rule compounded n times to approximate the value of the integral

$$\int_a^b f(x)g(x)\,dx.$$

The approximating sum is

$$\frac{h}{9}\sum_{j=1}^{n}\left(f(a+(j-\tfrac{1}{2}-\tfrac{1}{2}(3/5)^{1/2})h), f(a+(j-\tfrac{1}{2})h), f(a+(j-\tfrac{1}{2}+\tfrac{1}{2}(3/5)^{1/2})h)\right) \cdot \begin{pmatrix} \tfrac{3}{4}(1+(5/3)^{1/2}) & 1 & \tfrac{3}{4}(1-(5/3)^{1/2}) \\ 0 & 4 & 0 \\ \tfrac{3}{4}(1-(5/3)^{1/2}) & 1 & \tfrac{3}{4}(1+(5/3)^{1/2}) \end{pmatrix} \begin{pmatrix} g(a+(j-1)h) \\ g(a+(j-\tfrac{1}{2})h) \\ g(a+jh) \end{pmatrix},$$

where $h = (b-a)/n$. Note that if $g(x) \equiv 1$, the rule reduces to the regular three-point Gauss-Legendre rule, while if $f(x) \equiv 1$, it reduces to the regular Simpson's rule. The procedure was proposed and discussed by Boland and Duris in [1].

The subroutine was written in Fortran using double precision arithmetic and was checked on a IBM 360 Model 50. The calling parameters for the routine are as follows. A is the name for the lower limit of integration and B is the name for the upper limit. N is the number of times the formula is to be compounded. The basic interval $[A, B]$ is subdivided into N subintervals each of length $(B-A)/N$ and the rule is applied to each subinterval. *FN* and *GN* are names of double precision *FUNCTION* subprograms which evaluate the functions $f(x)$ and $g(x)$, respectively. These are to be supplied by the user. The result is stored in *VINT*.

There are four machine dependent constants. These are:

(i) $\tfrac{3}{4}(1+(5/3)^{1/2}) \approx 1.718245836551854$,
(ii) $\tfrac{3}{4}(1-(5/3)^{1/2}) \approx -0.2182458365518542$,
(iii) $\tfrac{1}{2}(1-(3/5)^{1/2}) \approx 0.1127016653792583$, and
(iv) $\tfrac{1}{2}(1+(3/5)^{1/2}) \approx 0.8872983346207417$.

The first constant is assigned to $AM(1,1)$ and $AM(2,3)$, the second to $AM(1,3)$ and $AM(2,1)$, while the third and fourth are used in the calculation of $X(1)$ and $X(2)$, respectively.

References

1. Boland, W.R., and Duris, C.S. Product type quadrature formulas. *BIT 11*, 2 (1971), 139–158.

Algorithm

```
      SUBRØUTINE  P3PGS ( A, B, N, FN, GN, VINT)
C
C THIS SUBRØUTINE USES THE PRØDUCT TYPE THREE-PØINT GAUSS-
C LEGENDRE-SIMPSØN RULE CØMPØUNDED N TIMES TO APPRØXIMATE
C THE INTEGRAL FRØM A TØ B ØF THE FUNCTIØN FN(X) * GN(X).
C FN AND GN ARE FUNCTIØN SUBPRØGRAMS WHICH MUST BE SUPPLIED
C BY THE USER.  THE RESULT IS STØRED IN VINT.
C
      DØUBLE PRECISIØN A, AG, AM(2,3), B, F(2), FN, G(3),
     *                 GN, H, VINT, X(2) , Y(2), DBLE
      DATA AM(1,1), AM(2,3) /2 * 1.718245836551854D0/,
     *     AM(1,2), AM(2,2) /2 * 1.D0/, AM(1,3), AM(2,1)
     *     /2 * -.2182458365518542D0/
      H = (B - A) / DBLE(FLØAT(N))
      X(1) = A + .1127016653792583D0 * H
      X(2) = A + .8872983346207417D0 * H
      Y(1) = A + H / 2.D0
      Y(2) = A + H
      VINT = 0.D0
      G(3) = GN(A)
      DØ 3 I = 1, N
        AG = FN(Y(1))
        G(1) = G(3)
        DØ 1 J = 1, 2
          F(J) = FN(X(J))
          G(J+1) = GN(Y(J))
          X(J) = X(J) + H
1         Y(J) = Y(J) + H
        VINT = VINT + AG * 4.D0 * G(2)
        DØ 3 J = 1,2
          AG = 0.D0
          DØ 2 K = 1, 3
2           AG = AG + AM(J,K) * G(K)
3         VINT = VINT + F(J) * AG
      VINT = H * VINT / 9.D0
      RETURN
      END
```

Certification of Algorithm 266 [G5]

Pseudo-Random Numbers [M.C. Pike and I.D. Hill, *Comm. ACM 8* (Oct. 1965), 605]

Walter L. Sullins [Recd. 12 Feb. 1971]
School of Education, Indiana State University
Terre Haute, IN 47809

Key Words and Phrases: pseudo-random numbers, testing random number generators
CR Categories: 5.5

The Pike and Hill Algorithm 266 [2] generates pseudo-random numbers in a prescribed open interval. Pike and Hill presented favorable evidence for the serial and poker tests [1] but omitted discussion of frequency tests.

The purpose of the present certification was to test the hypothesis that the numbers generated by the algorithm are rectangularly distributed. Nine sequences of numbers in the interval (0, 1) were generated, and each was divided into 500 blocks of various lengths. In each case the distribution of numbers was tested against a uniform distribution, with .1 interval width, by computing χ^2 on nine degrees of freedom for each of the 500 blocks within the sequence. The results are given in the table below.

Run	Starting value	Block length	Sequence length	Proportion
1	32347753	400	200,000	.012
2	52142147	600	300,000	.018
3	52142123	640	320,000	.014
4	53214215	960	480,000	.008
5	23521425	1000	500,000	.006
6	42321479	1040	520,000	.006
7	20302541	1560	780,000	.006
8	32524125	1600	800,000	.010
9	42152159	2600	1,300,000	.004

The proportions reported are the proportions of the 500 blocks which produced significant chi-square values when the probability of incorrectly rejecting the hypothesis of uniformity was set at .01. Thus there is considerable assurance that the numbers generated by the algorithm are rectangularly distributed. These findings also support the algorithm with respect to Yule's [3] recommendation that block sums be compared with expectation.

References

1. Kendall, M.G., and Babington-Smith, B. Randomness and random sampling numbers. *J. Royal Statist. Soc. 101* (1938), 147–166.
2. Pike, M.C., and Hill, I.D. Algorithm 266: Pseudo-random numbers. *Comm. ACM 8* (Oct. 1965), 605.
3. Yule, G. Udny. A test of Tippett's random sampling numbers. *J. Royal Statist. Soc. 101* (1938), 167–172.

Certification of Algorithm 379 [D1]

Squank (Simpson Quadrature Used Adaptively—Noise Killed) [J.N. Lyness, *Comm. ACM 13* (Apr. 1970), 260–263]

P. Hallet and E. Mund [Recd. 18 Jan. 1971 and 27 Apr. 1971]
Service de Métrologie Nucléaire, Université Libre de Bruxelles, Brussels, Belgium

Work supported by the Belgian Fonds National de la Recherche Scientifique.

Key Words and Phrases: numerical integration, integration rule, adaptive integration, automatic integration, Simpson's rule, numerical quadrature, quadrature rule, adaptive quadrature, automatic quadrature, round-off error control
CR Categories: 5.16

The algorithm was compiled and run without corrections on a CDC-6400 with a machine accuracy parameter of 0.7×10^{-14}. Our purpose was to test *SQUANK*'s ability to integrate a function blurred by random noise, and so the function *FUN(X)* is the result of applying a random perturbation R to some regular function $f(x)$, either by adding R to x before computing f, hereafter referred to as "x-noise", or by adding R to f after having computed it, "y-noise". R is taken as

$R = C * (2. * RANF(X) - 1.)$

where C is the noise amplitude and *RANF* is a system function generating pseudorandom numbers $(0. \leq RANF(X) \leq 1.)$

Our test program called *SQUANK* 100 times in situations involving all combinations of noise amplitude $C = 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}$, required tolerance $\epsilon_1 = 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}$, both noise types and the two functions $f_1(x) = k_1\, exp(x)$ and $f_2(x) = k_2(1 + 10^4 x^2)^{-1}$ integrated on [0, 1]. The constants k_1 and k_2 were chosen to normalize unblurred integrals to unity so that errors and tolerances may be seen as absolute or relative.

A rough calculation shows that y-noise causes in both integrals a deviation D that shouldn't exceed C. For x-noise, with $f(x + R) \simeq f(x) + Rf'(x)$, D shouldn't exceed respectively C and k_2C (meaning that the second function is oversensitive to x-noise by a factor $k_2 \simeq 200/\pi$).

The test program was run five times, yielding different results because the random perturbations were irreproducible. The following quantities were kept and averaged over the five runs.

$|\epsilon_2|$ actual error (specifically, ϵ_2 is the difference $SQUANK(\cdots) - 1.0$).
ϵ_3 error estimate (specifically, ϵ_3 is the value of parameter *RUM* as returned by *SQUANK*).
N number of function evaluations.

A sample of these averaged results is given in Table I.

Table I

		$f_1(x)$, x-noise			$f_1(x)$, y-noise		
C	ϵ_1	$\lvert\epsilon_2\rvert$	ϵ_3	N	$\lvert\epsilon_2\rvert$	ϵ_3	N
10^{-8}	10^{-12}	6.3 10^{-10}	5.5 10^{-9}	11278	8.4 10^{-10}	5.9 10^{-9}	8416
10^{-8}	10^{-4}	7.9 10^{-9}	1.0 10^{-4}	9	9.2 10^{-9}	1.0 10^{-4}	9
10^{-2}	10^{-12}	7.6 10^{-4}	4.6 10^{-3}	6986	8.9 10^{-4}	5.3 10^{-3}	8747
10^{-2}	10^{-4}	1.6 10^{-2}	1.5 10^{-3}	106	1.3 10^{-3}	1.6 10^{-3}	59

		$f_2(x)$, x-noise			$f_2(x)$, y-noise		
C	ϵ_1	$\lvert\epsilon_2\rvert$	ϵ_3	N	$\lvert\epsilon_2\rvert$	ϵ_3	N
10^{-8}	10^{-12}	2.9 10^{-8}	4.4 10^{-7}	35956	1.8 10^{-10}	5.5 10^{-9}	63254
10^{-8}	10^{-4}	5.8 10^{-6}	1.0 10^{-4}	77	5.8 10^{-6}	1.0 10^{-4}	77
10^{-2}	10^{-12}	6.3 10^{-2}	3.5 10^{-1}	9127	1.3 10^{-3}	5.2 10^{-3}	8496
10^{-2}	10^{-4}	6.9 10^{-2}	4.2 10^{-1}	3896	7.2 10^{-4}	2.5 10^{-3}	205

In 487 of the 500 calls it was found that *SQUANK*'s accuracy estimate of its own result was reliable, i.e. that $\epsilon_3 > |\epsilon_2|$. For the remaining 13 calls, the ratio $|\epsilon_2|/\epsilon_3$ ranged from 1 to 20 (in the worst cases, $\epsilon_3 \simeq \epsilon_1 < C$, just as if *SQUANK* had failed to notice the presence of the noise).

In 473 of the 500 calls it was found that *SQUANK*'s estimation was as good as could be reasonably expected, i.e., that $\epsilon_3 < \max(D, \epsilon_1) = \epsilon_4$. For the remaining 27 calls (all of them for f_2) the ratio ϵ_3/ϵ_4 never exceeded 1.15 (note that the test was made on ϵ_3, not on the actual error $|\epsilon_2|$).

Remark on Algorithm 176 [E2]

Least Squares Surface Fit [T.D. Arthurs, *Comm. ACM 6* (June 1963), 313]

Ernst Schuegraf [Recd. 1 Mar. 1971]
Department of Mathematics. St. Francis Xavier University, Antigonish, Nova Scotia, Canada

Algorithm 176 contains one misprint. The line which reads:

begin $e[i] = y[i]$;

should read:

begin $e[i] = z[i]$;

Remark on Algorithm 195 [F4]
BANDSOLVE [Donald H. Thurnau, *Comm. ACM 6* (Aug. 1963), 441]

Ernst Schuegraf [Recd. 1 Mar. 1971]
Department of Mathematics, St. Francis Xavier University, Antigonish, Nova Scotia, Canada

Algorithm 195 was transliterated into Fortran IV for the IBM 360/50. Various matrices with different values of N and M were used. The execution time was recorded and the accuracy of the results was checked.

Execution time [sec]

	$M = 11$	$M = 15$	$M = 21$	$M = 25$
$N = 50$	.2	.7	1.1	1.9
$N = 100$	.6	1.6	2.5	4.2

The execution time shows the expected proportionality to $((M - 1) \div 2)^2 \cdot N$. (Note the definition of M!) When checking the results, it was found that the algorithm failed for singular and near singular matrices. To protect against this, it is recommended to introduce a tolerance *eps* for a test on singularity and a label *fail*. This requires the following changes in the procedure declaration:

procedure *BANDSOLVE* (*C,N,M,V,eps,fail*);

It is necessary to insert the following statements in the blockhead of the procedure:

real *eps*; **label** *fail*;

After the statement *piv* := *r*; insert:

if *abs* (*C*[*piv*, 1]) < *eps* **go to** *fail*;

Remark on Algorithm 394 [H]
Decision Table Translation [R.B. Dial, *Comm. ACM 13* (Sept. 1970), 570]

D.R.T. Marshall [Recd. 3 Mar. 1971]
Data Processing Department, University of Waterloo Waterloo, Ontario, Canada

Key Words and Phrases: decision table, decision table translation
CR Categories: 4.19

The first comment of procedure *split* has the words "columns" and "row" transposed in sentences four/five. It should read "If the input tables has no columns, then *split* returns zero, ... If the table is entirely dashes or has no rows, then *split*, ...

The statement in the main procedure invoking the procedure *split* uses a variable "*I*", which is not defined.

This variable should be initialized to establish the "first" column in the array to be processed. This would, of course, normally be set to one.

The writer has programmed and executed the algorithm successfully in PL/I with the above noted changes.

Remarks on:
Algorithm 352 [S22]
Characteristic Values and Associated Solutions of Mathieu's Differential Equation
[Donald S. Clemm, *Comm. ACM 12* (July 1969), 399–407]

Algorithm 385 [S13]
Exponential Integral *Ei*(x)
[Kathleen Paciorek, *Comm. ACM 13* (July 1970), 446–447]

Algorithm 392 [D3]
Systems of Hyperbolic P.D.E.
[Robert R. Smith and Dennis McCall, *Comm. ACM 13* (Sept. 1970), 567–570]

Michael J. Frisch [Recd. 27 Jan. 1971]
University Computer Center, University of Minnesota, Minneapolis, MN 55455

Key Words and Phrases: ANSI Fortran standard
CR Categories: 4.0, 4.22

The following items were found during compilation of the algorithms written in Fortran published to date in Communications. The MNF compiler written at the University of Minnesota for CDC 6000 Series machines by Lawrence A. Liddiard and E. James Mundstock was used to check the validity of the algorithms.

Algorithm 352 does not conform to the standard in subroutine *MATH* which calls subroutine *SUM* with arguments that were in an *EXTERNAL* statement but not in a type statement. The dummy argument in subroutine *SUM* has type *DOUBLE PRECISION* so a statement *DOUBLE PRECISION DS, DC, DDS, DDC, PS, PC, DPS, DPC* should be inserted before the *EXTERNAL* statement in subroutine *MATH* (Section 8.4.2).

In subroutine *JNS*, the dummy argument *M* is also in blank common, contrary to 7.2.1.3. In the same subroutine, arrays *D, G, P*, and *Q* are referenced by array name instead of array element name as required in Section 7.2.1.4. The statement should be: *EQUIVALENCE* (*A*,*G*(1)), (*D*(1),*G*(2)), (*P*(1),*G*(4)), (*Q*(1), *G*(7)), (*DM*,*G*(10)), (*B*,*G*(11)).

Algorithm 385 does not conform to the standard in that the function name *DEI* appears in a type statement (Section 8.3.1). It should not appear there, and the function statement should be *DOUBLE PRECISION FUNCTION DEI* (*X*1). The third statement (*PRINT* 500) after the statement numbered 100 is not among the statements allowed in standard Fortran. A comment card separates the initial line from the continuation line in the statement numbered 200 contrary to Section 3.2.1.

Algorithm 392 does not conform to the standard in subroutine *CHARAC* in which at six statements before the statement numbered 145, the variable dimension *M* of the array *DATA* is redefined during execution contrary to Section 7.2.1.1.2.

Remark on Algorithm 405 [F2]
Roots of Matrix Pencils: The Generalized Eigenvalue Problem [A.M. Dell, R.L. Weil, and G.L. Thompson, *Comm. ACM 14*, (Feb. 1971), 113–117]

Richard M. Heiberger [Recd. 19 May 1971, 29 July 1971, and 8 Sept. 1971]
Department of Statistics, Harvard University*

Key Words and Phrases: eigenvalues, matrix roots, pencil roots
CR Categories: 5.1, 5.3

Algorithm 405 calculates rank-reducing numbers which are similar to, but not identical to, generalized eigenvalues. An eigenvalue of A with respect to B, as defined in this Remark, satisfies the equations

$$x^T(A - \lambda B) = 0, \qquad (A - \lambda B)y = 0 \tag{1}$$

for appropriately dimensioned vectors x and y. A rank-reducing number λ_0, as defined by Thompson and Weil [3], further satisfies

$$\text{Rank } (A - \lambda_0 B) < \text{Rank } (A - \lambda B) \tag{2}$$

for some value $\lambda \neq \lambda_0$. The distinction is meaningful only if the matrices A and B are of less than full rank.

The definition (1) is the simplest generalization of the ordinary eigenvalue problem in that the only new concept is the replacement of an identity matrix with an arbitrary matrix B. This form of the problem arises in many physical contexts, usually with A and B square symmetric, and B positive definite (see [4] for examples). Dell, Weil, and Thompson find that in their context the additional condition (2) is desirable since rank-reducing numbers are always discrete, finite in number, and related to a Jordan-like canonical form.

In order to insure that all eigenvalues, as defined here by (1), are discrete, one further condition than given in Algorithm 405 must be tested. It is necessary that

$$\text{Rank } (A - \lambda B) = \min\,(m, n) \tag{3}$$

for at least one value of λ. In the special case that $m = n$ (square matrices) the condition (3) is equivalent to

$$\det\,(A - \lambda B) \neq 0 \tag{4}$$

for at least one value of λ. When this condition is violated, the spectrum of eigenvalues is continuous; that is, for every complex number λ there exist vectors x and y such that (1) is satisfied. Discrete rank-reducing numbers may exist even when the rank condition (3) is violated. Example 8 accompanying Algorithm 405 does not satisfy condition (3) and therefore does not have discrete eigenvalues although it does have discrete rank-reducing numbers.

The procedure *PENCIL* is similar to the algorithm developed by Fix and Heiberger [1] for the generalized eigenvalue problem when A and B are Hermitian matrices. We showed that the spectrum of $Ax - \lambda Bx = 0$ consists of stable and unstable eigenvalues, which undergo, respectively, small and large changes in response to small changes in A and B. We proved that our algorithm isolates and accurately computes the eigenspace associated with the stable eigenvalues. We did not attempt to extend our proof to non-Hermitian and rectangular matrices, for which Algorithm 405 may also be used. Our proof explicitly does not apply to rank-reducing numbers unless the rank condition (3) is satisfied. Instead it suggests that the computed solution may be inaccurate, as the first example in [1] shows. We programmed in APL [2] and Fortran (unpublished).

The following changes to Algorithm 405 will modify it to calculate either eigenvalues or rank-reducing numbers at the user's option. The user of rank-reducing numbers will be warned if the rank condition (3) is not satisfied, and there may be numerical inaccuracy in his solution.

* This work was supported by the Cambridge Project through the Department of Statistics, Harvard University. Author's current address; Department of Statistics, Iowa State University, Ames, IA 50010.

Page 113, column 1. Replace procedure heading with:

```
procedure PENCIL (A,B,m,n,LAMDA,Sp,Par,Tol,eigrrn);
```

Page 113, column 1, preceding first **comment** insert:

```
integer array eigrrn;
```

Page 113, column 1, first **comment**. Replace the sentence:

The input parameters of *PENCIL* must be *A*, *B*, *m*, *n*, and *Tol*.

with the following:

The input parameter *eigrrn*[1] is used to direct the program to calculate either eigenvalues or rank-reducing numbers. If *eigrrn*[1] = 0, then eigenvalues will be calculated. If *eigrrn*[1] = 1, then rank-reducing numbers will be calculated. The parameter *eigrrn*[2] must be set to 0 as an input parameter. As an output parameter *eigrrn*[2] indicates whether the rank condition (3) is satisfied. If *eigrrn*[2] = 0, the condition is satisfied. If *eigrrn*[2] = 1, the condition is violated. When the rank condition is violated and eigenvalues are being calculated, the parameter *Par* is set to 0 indicating no roots, and the procedure is terminated. When the rank condition is violated and rank-reducing numbers are being calculated, the procedure continues calculations as at present, but the user is warned that there may be numerical inaccuracy in the solution. The input parameters of *PENCIL* must be *A*, *B*, *m*, *n*, *Tol*, *eigrrn*[1], and *eigrrn*[2].

Page 116, column 1, preceding line −11. Insert:

```
if ((n-r-q-t≠0) ∧ (m-r-q-s≠0)) then
begin
comment Set parameter for continuous spectrum;
    eigrrn[2] := 1;
if eigrrn[1] = 0 then
    begin
    comment Set parameter for no solution;
      Par := 0;  go to Endp;
    end;
    comment Beware of possible numerical inaccuracy;
end;
```

Page 116, column 1, line −4. Replace with:

```
PENCIL(G,H,r-t,r-s,LAMDA,Sp,Par,Tol,eigrrn);
```

There are several typographical errors. The following lines should read as given below.

Page 115, column 2, lines −8 and −7:

```
Tol)
else t := 0;
```

Page 115, column 2, line −5:

```
REDUCE(P3,E21,P4,limr,r,s,Tol)
```

Page 116, column 1, line 1:

```
Par := 0;  go to Endp;
```

I would also suggest that the following **value** parts be added for more efficient execution.

```
Procedure PENCIL                  Procedure Matmul
  value A,B,m,n,Tol;                value u,v,w;

Procedure REDUCE                  Procedure SEARCH
  value X,m,n,Tol;                  value Lim,m,n,veci,vecj;
```

References

1. Fix, G., and Heiberger, R.M. An algorithm for the Ill-conditioned generalized eigenvalue problems. *SIAM J. Numer. Anal.* (Mar. 1972), 78–88.
2. Heiberger, R.M. APL functions for data analysis and statistics. Res. Rep. CP-5, Dep. of Statistics, Harvard U., 1971.
3. Thompson, G.L., and Weil, R.L. Reducing the rank of $(A - \lambda B)$. Proc. Amer. Math. Soc. 26, 4 (Dec. 1970), 548–54.
4. Wilkinson, J.H. *The Algebraic Eigenvalue Problem.* Oxford U. Press, Oxford, 1965.