A note on the estimation of the coefficients in the Chebyshev series expansion of a function having a logarithmic singularity

By M. M. Chawla*

The estimation of the coefficients in the Chebyshev series expansion of a function has been considered by Elliot (1964). In this note we extend his analysis to the case of a function having a logarithmic singularity.

Let $f(x)$ be a function defined for $-1 \leqslant x \leqslant 1$, and let $T_n(x)$ be the Chebyshev polynomial of the first kind of degree n, defined by

$$T_n(x) = \cos(n \arccos x). \tag{1}$$

If $f(x)$ is of bounded variation in $[-1, 1]$, then we have

$$f(x) = \sum_{n=0}^{\infty}{}' a_n T_n(x)$$

where the dash on the summation sign indicates that the first term is to be halved. The coefficients are given (see Elliot, eqn. (8)) by

$$a_n = \frac{1}{\pi i} \int_C \frac{f(z)dz}{\sqrt{(z^2-1)}(z \pm \sqrt{(z^2-1)})^n} \tag{2}$$

where $f(z)$ is regular within and on the contour C, enclosing the interval $-1 \leqslant x \leqslant 1$. The sign in the integrand is chosen so that $|z \pm \sqrt{(z^2-1)}| > 1$.

Let us suppose that $f(x) = g(x) \log(c - x)$, $(c > 1)$. In equation (2), we shall now choose as the contour C, a circle $\Gamma : |z| = R$ described in the positive sense, a small circle $\gamma : |z - c| = \epsilon$, described negatively, and two line segments AB and CD where AB is just above and CD just below the part of the real axis $x < c$, and joining Γ with γ.

The function $f(z)$ is regular within this contour. We assume that $f(z)$ is such that the integral around Γ tends to zero as $R \to \infty$. Since $g(z)$ is regular at $z = c$, the integral around γ will tend to zero as $\epsilon \to 0$. Thus in the limit as $R \to \infty$ and $\epsilon \to 0$, the only contributions to a_n will come from the integrals along AB and CD. Taking $f(z) = g(z) \log(c - z)$, we find that these two integrals combine to give

$$a_n = -2 \lim_{R \to \infty} \int_c^R \frac{g(x)dx}{\sqrt{(x^2-1)}(x + \sqrt{(x^2-1)})^n}. \tag{3}$$

Putting $x = \cosh\theta$, $c = \cosh\alpha$,

$$a_n = -2 \lim_{\Theta \to \infty} \int_\alpha^\Theta g(\cosh\theta) e^{-n\theta} d\theta. \tag{4}$$

Table 1

n	ESTIMATED a_n	ACTUAL a_n
2	−0·029412	−0·029437
3	+0·003362	+0·003367
4	−0·0004325	−0·0004332
5	+0·00005934	+0·00005947
6	−0·00000848	−0·00000851
7	+0·00000125	+0·00000129
8	−0·00000019	−0·00000019

Suppose n is chosen and $g(\cosh\theta)$ is such that the main contribution to the integral comes from around $\theta = \alpha$, and assuming that $\theta - \alpha$ is small,

$$a_n \simeq -2g(c) \int_\alpha^\infty e^{-n\theta} d\theta. \tag{5}$$

Thus, the estimate for the coefficients in this case, for large n, becomes

$$a_n \simeq -\frac{2g(c)}{n} \frac{1}{(c + \sqrt{(c^2-1)})^n}. \tag{6}$$

Proceeding as above, we can derive a similar estimate for the case of a function $f(z) = h(z) \log(c + z)$, where $h(z)$ is regular at $z = -c$, $(c > 1)$. We find, for large n,

$$a_n \simeq (-1)^{n+1} \frac{2h(-c)}{n} \frac{1}{(c + \sqrt{(c^2-1)})^n}. \tag{7}$$

As an example, consider the function $f(x) = \log(3 + x)$. Estimate (7) for the coefficients in this case gives,

$$a_n \simeq (-1)^{n+1} \frac{2}{n(3 + \sqrt{8})^n}. \tag{8}$$

The estimated values of the coefficients given by equation (8) are compared in Table 1 with the "actual" values obtained by the method of collocation (see Wright, 1966). We find that the estimated values compare very well with the actual values.

References

ELLIOT, D. (1964). "The Evaluation and Estimation of the Coefficients in the Chebyshev Series Expansion of a Function", *Math. Comp.*, Vol. 18, p. 274.

WRIGHT, K. (1966). "Series methods for integration", *The Computer Journal*, Vol. 9, p. 191.

* *Department of Mathematics, Indian Institute of Technology, Hauz Khas, New Delhi-29, India.*

Algorithms Supplement

Previously published algorithms

The following Algorithms have been published in the *Communications of the Association for Computing Machinery* during the period July–September 1966.

287 MATRIX TRIANGULATION WITH INTEGER ARITHMETIC

Operates on an $m \times (n + e)$ matrix, whose elements are integers, to reduce the sub-matrix consisting of the first n columns to upper triangular form and to supply the rank of this sub-matrix.

288 SOLUTION OF SIMULTANEOUS LINEAR DIOPHANTINE EQUATIONS

Seeks the smallest positive integer, d, for which an integer solution to the equation $Ax = bd$ exists, where A is an $m \times n$ matrix, x is an $n \times 1$ vector and b is an $m \times 1$ vector.

289 CONFIDENCE INTERVAL FOR A RATIO

Finds the $(1 - 2 \times a)$ confidence limits for θ/ϕ.

290 LINEAR EQUATIONS, EXACT SOLUTIONS

Solves the matrix equation $Ax = b$ for $A[1:n, 1:n]$ and $x, b[1:n]$, where the elements of A, b are small integers and the results are required as ratios of integers. The solution vector overwrites b and has values given by det $A \times x$, where det A is the determinant of A and x is the true solution vector. det A is supplied as an output parameter.

291 LOGARITHM OF GAMMA FUNCTION

Evaluates the natural logarithm of gamma (x) for all $x > 0$, accurate to 10 decimal places.

Algorithms

Note on Algorithm 2. *FIBONACCI SEARCH*, Algorithm 7. *MINX*, and the Golden Section search.

K. J. Overholt,
University of Bergen,
Norway.

In his certification (2) of Algorithm 2 Fibonacci Search (4) Mr. Boothroyd raises the question of the justification of the apparent complexity of this algorithm, as compared to his own Algorithm 7 MINX (1). The question seems appropriate in view of the basic simplicity of the Fibonacci search: having placed the first dividing point in the given interval, the following points are placed symmetrically in this and every remaining interval. The direct application of this would lead to a much shorter program than Algorithm 2. The crux of the matter is to be found in the following comment in this Algorithm: " . . . rounding error trouble may become so evident . . . that it leads to a collapse of the procedure". This is indeed true, and necessitates calculation of the exact dividing ratios F_i/F_{i+1} ($F_0 = F_1 = 1$, $F_{i+1} = F_i + F_{i-1}$) in every step. The cause of this difficulty has been analysed by the present writer (3).

With this point in mind the Fibonacci search is really very efficiently programmed in Algorithm 2. But the necessary complication of course detracts somewhat from the method's theoretical efficiency. This is given by $r = 1/F_n$ for an n-point search, where r is the ratio of the length of the remaining to the length of the original interval. The first dividing ratio should then be F_{n-1}/F_n, and the last, which should be F_1/F_2, must be changed slightly by placing the last two points a distance ϵ apart. The quantity ϵ represents the least argument difference which gives an observable difference in the function values. In actual computation this refinement is worthless, as a reliable *a priori* estimate of ϵ usually cannot be given. More realistically then, for an n-point search Algorithm 2 starts with a dividing ratio F_n/F_{n+1} and renounces on the last ϵ-separation, obtaining a reduction ratio $r = 2/F_{n+1}$.

As n grows F_n/F_{n+1} rapidly approaches the Golden Section ratio $t = 0{\cdot}618\ldots$ The algorithm working with the fixed dividing ratio t is known as Golden Section search. It is easily programmed; two versions of this algorithm are given below. Its efficiency is $r = t^{n-1}$ for an n-point search. Since $t^{n-1} < 2/F_{n+1}$, as is easily proved, it is slightly *more* effective than the "ϵ-less" Fibonacci search. If a point estimate rather than an interval estimate is required, the situation is reversed. In a Golden Section search the minimum point found is not in the centre of the resulting interval, and the distance from this point to the further of the two boundaries (that is, the maximum error) is greater than the equivalent Fibonacci distance, in spite of this interval being slightly larger. (Asymptotically $t^{n-1}F_{n+1}/2 \to 0{\cdot}9476\ldots$ while $t^nF_{n+1} \to 1{\cdot}171\ldots$) Considering the programming requirements of the two methods the Golden Section search would seem to be the most generally useful one.

The Algorithm 7 MINX is very compact. It is, however, rather inefficient in its use of the computed function values. An n-point trisection search gives a reduction ratio $r = (2/3)^{n/2}$. For this to be less than say 10^{-6} requires $n \geqslant 70$. This should be compared to the performance of a Golden Section search, which to obtain $t^{n-1} < 10^{-6}$ requires only $n \geqslant 30$.

In addition to the references found by the authors cited the book (5) could also profitably be consulted.

I am greatly indebted to the referee for valuable criticisms and suggestions, as a result of which the following two algorithms have been extensively rewritten.

References

1. BOOTHROYD, J. (1965). "*Algorithm* 7. MINX", *The Computer Bulletin*, Vol. 9, p. 104.
2. BOOTHROYD, J. (1965). "*Certification of Algorithm* 2. FIBONACCI SEARCH", *The Computer Bulletin*, Vol. 9, p. 105.
3. OVERHOLT, K. J. (1965). "*An instability in the Fibonacci and Golden Section search methods*", *BIT*, Vol. 5, p. 284.
4. PIKE, M. C., and PIXNER, J. (1965). "*Algorithm* 2. FIBONACCI SEARCH", *The Computer Bulletin*, Vol. 8, p. 147.
5. WILDE, D. J. (1964). *Optimum Seeking Methods*, New York: Prentice-Hall Inc.

Algorithm 16. *GOLD*

K. J. Overholt,
University of Bergen,
Norway.

```
real procedure Gold (a1, a2, f, delta, fmin);
value a1, a2, delta; real a1, a2, delta, fmin; real procedure f;
comment This procedure finds by means of the Golden Section
search method an approximation Gold to the minimizing
argument xmin of the real function f(x) of the real variable x
with relative accuracy delta, measured in terms of the original
interval (a1, a2) or (a2, a1), that is, abs(Gold − xmin) ⩽
delta × abs(a2 − a1). An absolute accuracy eps may be
specified by using a procedure call with eps/(a2 − a1) instead
of delta. On exit fmin is the corresponding value of f. The
function f is assumed to be unimodal, that is, f decreases
strictly as x increases towards the minimum point, and in-
creases strictly beyond this point. The procedure makes no
attempt to test the assumed unimodality or to test for noise
in the f-values due to rounding errors. If there is any un-
certainty on this point the procedure Goldsec is recommended.
The required number N of evaluations of f is given by t^N ⩽ delta
where t is the Golden Section ratio t = 0·618 . . .;
begin
real a3, a4, f3, f4, t; integer n;
t:= (sqrt(5·0) − 1·0)/2·0;
comment In a machine of fixed word-length one would sub-
stitute the actual value of t. To sixteen decimals
t = 0·618 033 988 749 894 8. The constant −0·4812 below
equals ln(t);
a4:= a1 + t × (a2 − a1); f4:= f(a4);
a3:= a2 + t × (a1 − a2); f3:= f(a3);
for n:= entier (ln(abs(delta))/(−0·4812)) − 1 step
− 1 until 1 do
if f3 < f4 then
begin
a2:= a4; a4:= a3; f4:= f3;
a3:= a2 + t × (a1 − a2); f3:= f(a3)
end else
begin
a1:= a3; a3:= a4; f3:= f4;
a4:= a1 + t × (a2 − a1); f4:= f(a4)
end;
if f3 < f4 then
begin
Gold:= a3; fmin:= f3
end else
begin
Gold:= a4; fmin:= f4
end
end Gold
```

Algorithm 17. *GOLDSEC*

K. J. Overholt,
University of Bergen,
Norway.

```
procedure Goldsec (a, b, f, delta, noise, a1, a2, fmin);
value a, b, delta; real a, b, delta, a1, a2, fmin;
real procedure f; Boolean noise;
comment This procedure finds (in the absence of noise, see
below) by means of the Golden Section search method an
interval (a1, a2) containing the minimum of the real function
f(x) of the real variable x with relative accuracy delta, measured
in terms of the original interval (a, b), that is, abs(a2 − a1) ⩽
delta × abs(b − a). An absolute accuracy eps may be
specified by using a procedure call with eps/(b − a) instead of
delta. On exit fmin is the least f-value found. The function f
is assumed to be unimodal, that is, f decreases strictly as x
increases towards the minimum point, and increases strictly
beyond this point. The procedure continuously monitors a set
of four points to detect any violation of unimodality. If this
occurs the Boolean variable noise is set to true and the pro-
cedure exits to the calling program. The interval (a1, a2) then
contains the noisy arguments. By a normal exit noise has the
value false. This method will in most cases detect the presence
of noise in the function values due to rounding errors, the non-
observation of which might give a wholly false impression of
the accuracy obtainable. This does not mean that this pro-
cedure is wholly proof against rounding error trouble. It may
still happen that the actual minimum falls outside the interval
(a1, a2) found. In such cases noise will usually have the value
true. If rounding error trouble is severe a procedure of
stochastic or least squares nature should be contemplated;
begin
    real a3, a4, f1, f2, f3, f4, t; integer n, nb;
    procedure newpoint (ar, at, am, fr);
    real ar, at, am, fr;
    begin
    ar:= am + t × (at − am); fr:= f(ar);
    if fr < fmin then fmin:= fr
    end newpoint;
    t:= (sqrt(5·0) − 1·0)/2·0;
    comment In a machine of fixed word-length one would
    substitute the actual value of t. To sixteen decimals
    t = 0·618 033 988 749 894 8. The constant −0·4812
    below equals ln(t);
    a1:= a; a2:= b; f1:= f(a); f2:= f(b);
    fmin:= if f1 < f2 then f1 else f2;
    nb:= 1; noise:= true;
    for n:= entier (ln(abs(delta))/(−0·4812)) step
        − 1 until 1 do
    begin
    if nb < 0 then newpoint (a3, a1, a2, f3) else
    if nb = 0 then newpoint (a4, a2, a1, f4) else
    begin newpoint (a3, a1, a2, f3); newpoint (a4, a2, a1, f4)
      end;
    if f3 < f4 then
    begin
        noise:= f4 > f2; if noise then goto Exit;
        a2:= a4; f2:= f4;
        a4:= a3; f4:= f3; nb:= −1
    end else
    if f4 < f3 then
    begin
        noise:= f3 > f1; if noise then goto Exit;
        a1:= a3; f1:= f3;
        a3:= a4; f3:= f4; nb:= 0
    end else
    begin
        noise:= f4 ⩾ f2 ∨ f3 ⩾ f1; if noise then goto Exit;
        a1:= a3; f1:= f3;
        a2:= a4; f2:= f4; nb:= 1; n:= n − 2
    end
    end;
Exit:
end Goldsec
```

Algorithm 18. *SUMFAC*

J. Boothroyd,
Hydro-University Computing Centre,
University of Tasmania.

Author's note:—The **procedure** *sumfac* is linked historically with the development of one of the first generation British computers. As little as twelve years ago, strange as it may seem now, computers sometimes stood idle for as much as one-half or even one hour between program testing sessions. Inasmuch as machines were then less reliable than they are today, this raised the problem of how to decide whether a fault had developed during an idle period. On one Deuce computer a time-filling, self-checking program was devised to search for amicable numbers, number pairs (a, b) such that the factors of a, including one but excluding a, sum to b and vice versa. The program, when stopped, recorded the initial value to be used on a subsequent restart. The following driver program in Elliott 503 ALGOL uses **procedure** *sumfac* in an amicable number search over the first 10,000 integers:

```
TEST AMICABLE;
begin integer a, b, n;
      comment here declare procedure sumfac;
      for n:= 1 step 1 until 10000 do
      begin b:= sumfac(n) -n;
            if b ⩾ n then
            begin a:= sumfac(b) -b;
                  if a = n then print a, sameline, b
            end
      end
end;
```

The procedure has another use today, to measure the upper bound of the relative ALGOL/machine-code efficiency of any particular implementation. The bound tends to an upper bound as the procedure employs only integer arithmetic, uses no arrays, or blocks, or other procedures either simply or recursively. On an Elliott 503 the program takes 116 seconds in ALGOL while the same logic in SAP, the 503 assembly language, is faster by a factor of almost exactly two. The following amicable and perfect numbers are generated:

6	6
28	28
220	284
496	496
1184	1210
2620	2924
5020	5564
6232	6368
8128	8128

How long does it take on your machine in (a) ALGOL, (b) machine-code? A FORTRAN version of *TEST AMICABLE* takes 93 seconds on an IBM 1130.

```
integer procedure sumfac(n); value n; integer n;
comment for n > 1 sumfac yields the sum of the factors of n
including 1 and n. For n = 1 sumfac:= 1. If the prime
decomposition of n is a^p b^q . . . g^v then the sum of the factors
of n is
(1 + a + a^2 + . . . + a^p)(1 + b + b^2 + . . . + b^q) . . .
  (1 + g + g^2 + . . . + g^v);
begin integer pi, s, d, f, q, t; Boolean newfac;
      pi:= s:= d:= 1; f:= 2; q:= n;
newf: newfac:= true;
next: if q ⩾ f then
      begin q:= n ÷ f;
            if n ≠ q × f then
            begin f:= f + d; d:= 2;
                  goto newf
            end
            else
            begin if newfac then
                  begin newfac:= false; pi:= pi × s;
                        s:= t:= 1
                  end;
                  t:= t × f; s:= s + t; n:= q;
                  goto next
            end
      end;
      pi:= pi × s;
      sumfac:= if n ≠ 1 then pi × (1 + n) else pi
end sumfac
```

Note on Algorithm 4. *TWO BY TWO*

I. D. Hill and M. C. Pike,
Medical Research Council,
Statistical Research Unit,
115 Gower Street,
London W.C.1.

If either the row totals or the column totals of a two by two table are equal, then the second tail probability is equal to the first tail probability whichever method is used to define the second tail. This situation is likely to occur frequently and the algorithm is thus much improved by adding immediately after the label *SECOND*:

```
if r1 = r2 ∨ c1 = c2 then
begin
      E:= 2·0 × sum;
      goto EXIT3
end;
```

This modification also means that there is no danger of the most significant term of the second tail being omitted through rounding error.

Note onAlgorithm 2. *FIBONACCI SEARCH*
and on Algorithm 7. *MINX*

M. C. Pike,* I. D. Hill and F. D. James,
Medical Research Council,
Statistical Research Unit,
115 Gower Street, London W.C.1, and
Department of Applied Mathematics,
University of Liverpool.

J. Boothroyd's certification of Algorithm 2 (1) shows, to our way of thinking, a misunderstanding of computer arithmetic as opposed to pure mathematics. Boothroyd desires to find the minimum of $7x^2 + 2x + 4$, which is at $x = -1/7 = -0{\cdot}142857142\ldots$ with the function taking the value $27/7 = 3{\cdot}857142857\ldots$. However, a computer with its finite word length cannot represent $7x^2 + 2x + 4$ precisely, but has to use some function $f(x)$ which will be a good approximation of $7x^2 + 2x + 4$ though not identical to it. The FIBONACCI SEARCH algorithm will then be asked to locate the minimum of $f(x)$ and provided it does so it has

* At present at Pathology Department, Makerere University College Medical School, Box 2072, Kampala, Uganda.

worked correctly. No algorithm can use the function $f(x)$ to find the minimum of $7x^2 + 2x + 4$.

It will be noticed that the value quoted by Boothroyd as the function value at all four of his "unsatisfactory" points agrees with 27/7 to the number of significant figures the Elliott 503 is working to. All these points show the same value of the function $f(x)$, and we cannot say that one point is correct and another incorrect.

FIBONACCI SEARCH allows for the case where a minimum is so flat that three adjacent function values are equal by making a premature exit instead of continuing the search. The user may wish to know when this has occurred, since it will usually mean that the minimum point of the pure mathematical function has not been found to the accuracy requested. This is the purpose of the formal parameter *prem* in the new version of Algorithm 2 suggested below.

Boothroyd also asks about the circumstances under which the apparent complexity of Algorithm 2 is justified compared with the comparative simplicity of Algorithm 7. The answer is that Algorithm 2 is to be preferred, on grounds of speed, whenever the function to be evaluated takes a long time to compute, since the Fibonacci search method *minimizes the number of function evaluations required for a desired accuracy* (2). One may demonstrate the advantage by using any simple function, but making the machine evaluate it k times at each function call and noting the effect of increasing k.

On a test, in which the quadratic $7x^2 - 14x + 5$ was evaluated k times at each function call, the following times to find a minimum, to an accuracy of 10^{-5} in the interval (0, 4), were observed:

	FIBONACCI SEARCH	MINX
k	TIME IN UNITS OF "INSTRUCTION COUNT" ON THE ICT ATLAS COMPUTER	
1	4	5
4	5	9
16	12	25
64	40	90
256	151	351
1024	598	1392

A golden section search is almost as good as a Fibonacci search in terms of time, but has the disadvantage that the minimum point found is not in the centre of the interval of uncertainty.

Algorithm 2 may be considerably simplified by omitting the inner block with its declaration of the array *fibno*. The following new version is more efficient in terms of store. In terms of time it is effectively identical to the older version.

References

1. BOOTHROYD, J. (1965). "Certification of Algorithm 2. FIBONACCI SEARCH", *The Computer Bulletin*, Vol. 9, p. 105.
2. WILDE, D. J. (1964). *Optimum Seeking Methods*, New York: Prentice-Hall Inc.

Algorithm 2 (modified).

```
real procedure Fibonacci search (a, b, eps, fval, prem, f);
value a, b, eps; real a, b, eps, fval;
Boolean prem; real procedure f;
comment this procedure finds within plus or minus eps the
position of the minimum of the function f(x) in the range
a ⩽ x ⩽ b by the optimum minimax method. f(x) must be
monotonic decreasing from x = a to the minimum position and
then monotonic increasing to x = b.
   On exit fval contains the function value at the position
found, while prem is set to true if a premature exit has been
made or false otherwise. A premature exit indicates that the
minimum is so flat that the accuracy to which the calculations
are performed is insufficient to cut the interval of uncertainty
down to plus or minus eps;

begin
real e, ff1, ff2, p1, p2;
integer n, f1, f2, c; Boolean equal;
equal:= false;
n:=1; f1:= 2; f2:= 3;
e:= (b − a)/eps;
for c:= f1 while f2 < e do
begin
    n:= n + 1; f1:= f2; f2:= c + f2
end;
p2:= (f1/f2) × (b − a) + a; p1:= a + b − p2;
ff1:= f(p1); ff2:= f(p2);
for n:= n step −1 until 2 do
begin
    c:= f1; f1:= f2 − f1; f2:= c;
    if ff2 ⩾ ff1 then
    begin
        b:= p2; p2:= p1;
        p1:= b − (f1/f2) × (b − a);
        ff2:= ff1; ff1:= f(p1)
    end else
    begin
        a:= p1; p1:= p2;
        p2:= a + (f1/f2) × (b − a);
        ff1:= ff2; ff2:= f(p2)
    end;
    prem:= equal ∧ ff1 = ff2;
    if prem then goto EXIT;
    equal:= ff1 = ff2
end;
EXIT: if ff2 < ff1 then
begin
    fval:= ff2; Fibonacci search:= p2
end else
begin
    fval:= ff1; Fibonacci search:= p1
end
end Fibonacci search
```

Correspondence

The Editor,
The Algorithms Supplement,
The Computer Journal.

Sir,

May I draw your attention to the large number of mistakes in the published versions of Algorithms 10 and 11. These show a standard of editing which is not good enough.

In addition to the many typographical blunders there is one error of content in **procedure** *equipol* for which, in spite of its escape through the meshes of the referee's net, I accept responsibility.

The statement : $arg := arg - j \times h$;

should be : $arg := arg - j \times h - xbase$;

This is perhaps a suitable opportunity to criticise more generally. Algorithm 1 appeared in *The Computer Bulletin* of September 1964. Two years later we see the publication of Algorithm 11. Does this level of achievement justify the continuation of the supplement? It hardly appears to justify the efforts of the editor and the several contributors and referees.

Assuming that 10% of submitted material is published, it would not have appeared unreasonable two years ago to expect that, by this time, each supplement would contain six useful algorithms.

To what extent is the failure to achieve this due to lack of supporting contributors? Some, no doubt, though it is clear that you, sir, are called upon, as editor, to perform many tasks which should be done by contributors and referees.

Here are a few suggestions for improving the "turn around" time in the publication of algorithms.

(*a*) *Contributors* should be required:

1. To submit two double-spaced copies of their algorithm in fully symbol-edited publication ALGOL.
2. To submit in the exact form in which these were run, two copies of a driver program and tested procedure together with test data and results.
3. To accept the necessity for re-submission (1 and 2 above) if their contribution is returned for a correction, however trivial.
4. To accept the responsibility for checking galley proofs accurately and promptly.

(*b*) *Referees* should be required:

1. To have access to the means of testing algorithms for correct syntax and content.
2. To agree to complete their task within four weeks or send back the algorithm to you by return post.

So far as the refereeing of papers is concerned, there may be sound reasons for affording referees the protection of anonymity. In the case of ALGOL procedures where the material is more a matter of fact than opinion, there is less reason for retaining this practice. Referees who consent to waive this privilege should be permitted to correspond directly with contributors provided, of course, that copies of their letters are sent to you for the record. This would materially assist in minimizing the now intolerably long delay between submission of an algorithm and its final acceptance or rejection.

J. Boothroyd,
University of Tasmania,
Hobart, Tasmania, Australia.

Editor's comment

Mr. Boothroyd's criticisms of the number of printing mistakes in Algorithms 10 and 11 are fully justified. Steps have been taken to see that this does not happen again. As an apology the algorithms are reproduced below, fully corrected and including Mr. Boothroyd's amendment.

The standard of material submitted to the Supplement is very high but the number of submissions is small and they are usually in the field either of numerical analysis or of statistics. I should like to see an increase both in the amount of work submitted and in the scope covered. Efficiency of computer use should not be a consideration. The Algorithms Supplement is just as much, if not more, a medium for communicating ideas as a medium for distributing computer programs.

The first three conditions suggested by Mr. Boothroyd, under which authors submit algorithms, are already the policy of the Supplement. Unfortunately these conditions are not always met. It has also been agreed that, where possible, galley proofs will be returned to the authors for proof checking.

I am fortunate in having a keen, though small, band of referees to assist. It is very rare that publication is held up by the referee, except when he is too conscientious and continues to suggest further improvements. Is this a bad thing? Again, more offers of help would be appreciated. As Mr. Boothroyd states, refereeing an algorithm is a matter of fact rather than opinion and so conscientiousness is more important than high-level experience.

In addition to the above, I should also welcome contributions of a more general nature concerning possibly the future and the organization of the Supplement.

Algorithm 10.

```
real procedure aitken (x, y, arg, n, m); value arg, m, n; array
x, y; real arg; integer m, n; comment array y[0:n] contains
sample values of a function at corresponding values of the
argument contained in x[0:n] which is assumed to have been
sorted in ascending order. The procedure yields an approxi-
mation to the function at the specified value arg by evaluating
an mth order polynomial (m ⩽ n). The subset of m + 1
points used in the evaluation are suitably chosen to be evenly
distributed about the value of arg. If the requested value of
m exceeds n, the assignment m:= n occurs.

For arg < x[0] the procedure extrapolates using x[0],
x[1], . . ., x[m]
For arg > x[n] the procedure extrapolates using x[n − m],
x[n − m + 1], . . ., x[n];
begin integer i, j, mless 1; real fi, zi; real array z, f[0:m];
    integer procedure setmin (L); label L;
    begin integer i;
        for i:= 0 step 1 until n do if arg ⩽ x[i] then
        goto found;
        i:= n;
    found: if arg = x[i] then
        begin
            f[m]:= y[i]; goto L
```

```
        end;
        i:= i − m ÷ 2 − 1;
        setmin:= if i < 0 then 0 else if
        i + m > n then n − m else i
    end setmin;
    if m > n then m:= n; j:= setmin (out);
    for i:= 0 step 1 until m do
    begin
        z[i]:= arg − x[j]; f[i]:= y[j]; j:= j + 1
    end;
    mless 1:= m − 1;
    for i:= 0 step 1 until mless 1 do
    begin
        fi:= f[i]; zi:=z[i];
        for j:= i + 1 step 1 until m do
        f[j]:= fi + zi × (f[j] − fi)/(zi − z[j])
    end;
out: aitken:= f[m]
end aitken
```

Algorithm 11.

```
real procedure equipol (xbase, y, arg, n, m, h); value xbase,
arg, m, n, h; real xbase, arg, h; array y; integer m, n;
comment array y[0: n] contains sample values of a function at
corresponding equal interval values of the argument xbase,
xbase + h, xbase + 2 × h, . . ., xbase + n × h. The pro-
cedure yields an approximation to the function at the specified
value arg by evaluating an mth order polynomial (m ⩽ n),
using Aitken's iterative method. The subset of m + 1 points
used in the evaluation are suitably chosen to be evenly distri-
buted about the value of arg. If the requested value of m
exceeds n the assignment m:= n occurs.
For arg < x[0] the procedure extrapolates using x[0],
x[1], . . ., x[m]
For arg > x[n] the procedure extrapolates using x[n − m],
x[n − m + 1], . . ., x[n];
begin integer i, j, mless 1; real jh, fi; array f[0: m];
    if m > n then m:= n;
    i:= entier ((arg − xbase)/h) − m ÷ 2;
    j:= if i < 0 then 0 else if i + m > n then n − m
    else i;
    for i:= 0 step 1 until m do f[i]:= y[i + j];
    arg:= arg − j × h − xbase;
    mless 1:= m − 1;
    for i:= 0 step 1 until mless 1 do
    begin fi:= f[i]; jh:= h;
        for j:= i + 1 step 1 until m do
        begin
            f[j]:= fi + arg × (f[j] − fi)/jh;
            jh:= jh + h
        end;
        arg:= arg − h
    end;
    equipol:= f[m]
end equipol
```


The Computer Journal

Published Quarterly by

The British Computer Society, 23 Dorset Square, LONDON, N.W.1, England.

Subscription price per volume £4 10s. (U.S. $12.60). Single Copies 25s. (U.S. $3.50)

All inquiries should be sent to the Secretary at the above address.

Communications: Papers submitted for publication should be sent to E. N. Mutch, The University Mathematical Laboratory, Corn Exchange Street, Cambridge. Intending authors should first apply for *Notes on the Submission of Papers*, as the onus of preparing papers in a form suitable for sending to press lies in the first place with the authors.