

Algorithms

H. J. WEGSTEIN, Editor

Contributions to this department must be in the form stated in the Algorithms Department policy statement (*Communications*, February, 1960) except that ALGOL 60 notation should be used (see *Communications*, May 1960). Contributions should be sent in duplicate to J. H. Wegstein, Computation Laboratory, National Bureau of Standards, Washington 25, D. C. Algorithms should be in the Reference form of ALGOL 60 and written in a style patterned after the most recent algorithms appearing in this department. For the convenience of the printer, please underline words that are delimiters to appear in boldface type.

Although each algorithm has been tested by its contributor, no warranty, expressed or implied, is made by the contributor, the editor, or the Association for Computing Machinery as to the accuracy and functioning of the algorithm and related algorithm material, and no responsibility is assumed by the contributor, the editor, or the association for Computing Machinery in connection therewith.

The reproduction of algorithms appearing in this department is explicitly permitted without any charge. When reproduction is for publication purposes, reference must be made to the algorithm author and to the *Communications* issue bearing the algorithm.

ALGORITHM 80 RECIPROCAL GAMMA FUNCTION OF REAL ARGUMENT

WILLIAM HOLSTEN

University of California at San Diego, La Jolla, California

real procedure RGR(x); **real** x; **real procedure** RGAM;

comment Procedure RGAM computes the real reciprocal Gamma function of real x for $-1 < x < 1$, utilizing Horner's method for polynomial evaluation of the approximation polynomial. RGR extends the range of RGAM by use of the formulae

(1) $1/\text{Gamma}(x-1) = (x-1)/\text{Gamma}(x)$ for $x < -1$,

(2) $1/\text{Gamma}(x+1) = 1/x \times \text{Gamma}(x)$ for $x < 1$;

begin **real** y;

if x = 0 **then begin** RGR := 0; **go to** EXIT **end**

if x = 1 **then begin** RGR := 1; **go to** EXIT **end**

if x < 1 **then go to** BB;

 y := 1;

AA: x := x - 1; y := y × x; **if** x > 1 **then go to** AA;

if x = 1 **then begin** RGR := 1/y; **go to** EXIT **end**

 RGR := RGAM(x)/y; **go to** EXIT;

BB: **if** x = -1 **then begin** RGR := 0; **go to** EXIT **end**

if x > -1 **then begin** RGR := RGAM(x);

go to EXIT **end**

 y := x;

CC: x := x + 1; **if** x < -1 **then begin** y := y × x;

go to CC **end**

 RGR := RGAM(x) × y;

EXIT: **end** RGR;

real procedure RGAM(x); **real** x; **integer** i;

real array B[0:13];

comment The algorithm for this routine was adapted from "University of Illinois Digital Computer, Auxiliary Library Routine B-17-328", by John Ehrman. Reference may also be made to Algorithm 34, dated February, 1961. Approximation accuracy is $\pm 2^{-35}$;

begin **real** z;

 B[0] := 1.00000 00000 00; B[1] := -.42278 43350 92;

 B[2] := -.23309 37363 65; B[3] := +.19109 11011 62;

 B[4] := -.02455 24908 87; B[5] := -.01764 52421 18;

 B[6] := +.00802 32781 13; B[7] := -.00080 43413 35;

 B[8] := -.00036 08514 96; B[9] := +.00014 56243 24;

 B[10] := -.00001 75279 17; B[11] := -.00000 26257 21;

 B[12] := +.00000 13285 54; B[13] := -.00000 01812 20;

 z := B[13];

for i := 12 **step** -1 **until** 0 **do** z := z × x + B[i];

 RGAM := z × x × (x + 1)

end RGAM;

ALGORITHM 81 ECONOMISING A SEQUENCE 1

BRIAN H. MAYOH

Digital Computer Laboratory, University of Illinois,
Urbana, Ill.

procedure ECONOMISER 1 (desired property, costs, n, C);

array costs; **integer** n;

Boolean procedure desired property;

Boolean array C;

begin **comment** Given a finite, monotonely increasing sequence of positive numbers, looked upon as prices, ECONOMISER 1 selects the cheapest subsequence with a given property. The formal parameters are: *Desired property*, a function designator to answer the question: Does the subsequence held in array C possess the required property? *n* is (number of elements in the sequence) + 1. *Costs* is an array of size [1:n]. *Costs*[1] to *costs*[n-1] hold the numbers of the sequence and *costs*[n] is any arbitrary number greater than the sum of all other elements of costs. *C* is an array of the same size and indicates a subsequence by the rule: $C[i] = \text{element } i \text{ of the original sequence is in the subsequence}$. At exit from ECONOMISER 1, *C* indicates the cheapest subsequence. It is supposed that the original sequence has the desired property;

integer d, j, k, l; **real** i;

for j := 1 **step** 1 **until** n **do** C[j] := j = 1; d := 0;

 reenter: d := d+1;

 INSIDE: **begin** **own** **real array** prices [1:d];

own **Boolean array** alternatives[1:d, 1:n];

procedure ENTER SUCCESSORS;

begin k := n-1;

 A: **if** $\neg C[k]$ **then**

begin k := k-1; **go to** A **end**; i := 0;

for j := 1 **step** 1 **until** n **do**

```

begin alternatives[ℓ,j]
  := j ≠ k ∧ j ≠ k-1 ≡ C[j];
  if alternatives[ℓ,j] then
    i := i + costs[j]
  end;
B: k := k-1;
  go to if k = 0 then find cheapest
    else if C[k] then (if k=1 then
      find cheapest else B)
    else if k=1 then E
      else if C[k-1] then D
        else find cheapest;
D: C[k-1] := false;
E: C[k] := true; go to reenter
end of ENTER SUCCESSORS;
i := 0; for j := 1 step 1 until n do
  begin alternatives[d,j] := C[j]; if C[j] then
    i := i + costs[j]
  end; prices[d] := i;
find cheapest: i := 0; for j := 1 step 1 until d do
  begin if prices[j] < i then
    begin ℓ := j; i := prices[ℓ] end
  end;
  for j := 1 step 1 until n do
    C[j] := alternatives[ℓ,j];
  if ¬ desired property then
    ENTER SUCCESSORS
  end of INSIDE;
end of ECONOMISER 1;

```

ALGORITHM 82

ECONOMISING A SEQUENCE 2

BRIAN H. MAYOH

Digital Computer Laboratory, University of Illinois,
Urbana, Ill.

procedure ECONOMISER 2 (desired property, costs, n, C, r,
Reject list); **Boolean procedure** desired property;

integer n, r; **array** costs; **Boolean array** Reject list;

begin comment In some applications of ECONOMISER 1, it
is simple to establish that some subsequences are redundant in
the sense that any sequence containing them is certainly not
the cheapest subsequence with the desired property. For such
applications ECONOMISER 2 avoids all unnecessary calls of
desired property. The new formal parameters are: *r* a variable
whose value is initially 0 and is increased by 1 every time that
desired property discovers a new redundant subsequence.
Reject list an array of size [1:r,1:n]. *Reject list* [a,b] carries the
answer to: Is element b of the original sequence in the ath
redundant subsequence found by *desired property*?

real i; **integer** d, j, k, ℓ; **Boolean** gapfilled, first time;

procedure INSIDE (entrymaker); **Boolean** entrymaker;

begin own real array prices[1:d];

own Boolean array alternatives[1:d,1:n];

procedure ENTER SUCCESSORS;

begin integer c; **Boolean array** ssq[1:n];

for j := 1 step 1 until n do ssq[j] := C[j];

c := n-1;

A: if ¬ ssq[c] then **begin** c := c-1; **go to** A **end**;

C[c] := false; C[c+1] := true;

INSIDE (true);

gapfilled := true;

B: c := c-1;

go to if c=0 **then** F **else if** ssq[c] **then**

(if c=1 **then** F **else** B) **else if** c=1 **then**

E **else if** ssq[c-1] **then** D **else** F;

D: ssq[c-1] := false;

E: for j := 1 step 1 until n do C[j] := ssq[j] ≡ j≠c;

INSIDE (true);

F: end of ENTER SUCCESSORS;

if entrymaker **then**

begin for j := 1 step 1 until r **do**

begin for k := 1 step 1 until n **do**

begin if ¬ C[k] ∧ Reject list[j,k] **then**

go to G **end**;

ENTER SUCCESSORS; **go to** H;

G: **end**;

i := 0; if gapfilled **then** d := d+1;

for j := 1 step 1 until n **do**

begin alternatives[if gapfilled **then**

d **else** ℓ, j] := C[j];

if C[j] **then** i := i + costs[j]

end; prices[if gapfilled **then** d **else** ℓ] := i

end; if first time ∨ ¬ entrymaker **then**

begin i := 0; gapfilled := first time := false;

for j := 1 step 1 until d **do**

begin if prices[j] < i **then**

begin ℓ := j; i := prices[ℓ] **end**

end;

for j := 1 step 1 until n **do**

C[j] := alternatives[ℓ,j];

if desired property **then** **go to** found;

ENTER SUCCESSORS; **go to** reenter

end;

H: **end of** INSIDE;

for j := 1 step 1 until n **do** C[j] := j=1;

d := 0; first time := gapfilled := true;

reenter: INSIDE (first time);

found:

end of ECONOMISER 2;

ALGORITHM 83

OPTIMAL CLASSIFICATION OF OBJECTS

BRIAN H. MAYOH

Digital Computer Laboratory, University of Illinois,
Urbana, Ill.

procedure OPTIMUM COVERING FINDER (Pattern, popu-
lation, set number, set prices, chosen sets, bounds, overflow);

Boolean array Pattern, chosen sets; **integer** population,
set number, bounds; **array** set prices; **label** overflow;

begin comment The number of objects in some given set is
given by *population*. The procedure is given a classification of
these objects by a collection of overlapping subsets. A cost
is assigned to each subset. Then OPTIMUM COVERING
FINDER selects the cheapest subcollection such that every
object is contained in at least one of the subsets of the sub-
collection. *set prices*[i] carries the cost of subset *i*. *Pattern*
is an array of size [1:set number,1:population] such that Pat-
tern[a,b] ≡ does subset *a* include object *b*. *chosen sets*[i] finally
carries the answer to the question: Is set *i* in the cheapest
subcollection? The programmer must restrict the amount of
space available to the procedure by setting *bounds*. From ex-
perience bounds = set number ↑ 2 suffices to avoid most alarm
exits to *overflow*;

Boolean array C[1:population], D[1:bounds, 1:population],

R, S[1:bounds,1:set number];

integer a, b, d, r, s;

Boolean procedure HAVE WE A COVERING;

begin procedure ADD to (Q,q,f); **integer** q;

real f; **Boolean array** Q;

begin if q=bounds **then** **go to** overflow **else** q := q+1;

for a := 1 step 1 until set number **do** Q[q,a] := f

end; **for** a := 1 step 1 until population **do**

```

    C[a] := false;
  for a := 1 step 1 until set number do
  begin if chosen sets[a] then
    for b := 1 step 1 until population do
      C[b] := C[b] ∨ Pattern[a,b]
    end; for a := 1 step 1 until population do
    begin if ¬ C[a] then go to E end;
    go to found;
  E: for d := 1 step 1 until s do
    begin for b := 1 step 1 until population do
      begin if C[b] ∧ ¬ D[d,b] then go to try another end;
      ADD to (R, r, chosen sets[a]);
      for b := 1 step 1 until set number do
        begin if chosen sets[b] ∧ ¬ S[d,b] then
          ADD to (R, r, S[d,a] ∨ a=b)
        end; go to F;
      try another;
    end of for statement labelled E;
    ADD to (S, s, chosen sets[a]);
    for a := 1 step 1 until population do D[s,a] := C[a];
  F: HAVE WE A COVERING := false
  end; r := s := 0;
  ECONOMISER 2 (HAVE WE A COVERING, set prices,
  set number, r, R, chosen sets);
found: end

```

CERTIFICATION OF ALGORITHM 60
 ROMBERG INTEGRATION (F. L. Bauer, *Comm.*
ACM, June, 1961)
 HENRY C. THACHER, JR.*
 Argonne National Laboratory, Argonne, Ill.

* Work supported by the U. S. Atomic Energy Commission.

This procedure was translated to the ACT III compiler language for the Royal Precision LGP-30 computer. This system provides 7+ significant decimal digits. The program was used to integrate x^n between the limits 0.01 and 1.1, and between the limits 1.1 and 0.01. The results in Table I were obtained. The pole at 0 for negative n affords a test of the reliability of the method when the higher derivatives of the integrand are large. The agreement between integrations in the forward and backward directions is an indication of the effects of round-off error.

It is apparent that the procedure gives results well within the noise level for the positive powers, and that even the effect of a closely adjacent singularity for the negative powers can be overcome.

The flexibility of the algorithm would be improved by adding to the formal parameters a procedure, check, to decide if sufficient

TABLE I. INTEGRATION OF $\int_{0.01}^{1.1} x^n dx$ AND $\int_{1.1}^{0.01} x^n dx$

n	0	+12	+12	-1
True Value	1.0900000	.26555932	-.26555932	4.7004831
Order 1	1.0899997	.57076812	-.57076842	19.641113
Order 2	1.0899997	.30614608	-.30614626	10.656923
Order 5	1.0899991	.26555693	-.26555818	4.9017590
Order 10				4.7002345
n	-1	-5	-5	-5
True Value	-4.7004831	.25000000 × 10 ⁸	-18.166667 × 10 ⁸	-.25000000 × 10 ⁸
Order 1	-19.641125	18.166655 × 10 ⁸	-8.4777719 × 10 ⁸	-8.4777766 × 10 ⁸
Order 2	-10.656929	8.4777719 × 10 ⁸	-1.0408634 × 10 ⁸	-1.0408640 × 10 ⁸
Order 5	-4.9017805	1.0408634 × 10 ⁸	-.25000715 × 10 ⁸	-.25000727 × 10 ⁸
Order 10	-4.7004402	.24999291 × 10 ⁸	-.25001311 × 10 ⁸	
Order 12				

accuracy had been obtained without carrying through the entire iteration. A possible form for this procedure would be:

```

procedure check (t1, t2, f, exit);
  real t1, t2;
  label exit;
  integer f;
begin if abs ((t2 - t1) × f) / t1 < tolerance ∧ f > minimum order
  then go to exit end.

```

The global variables tolerance, which is the maximum relative difference between approximations of increasing order, and the minimum acceptable order should be selected by the programmer for the exigencies of the problem. A check of this sort is clearly not as sound as an a priori estimate of the necessary order, but is frequently an acceptable expedient.

The Romberg quadrature algorithm is analyzed in the following references:

- Romberg, W. Vereinfachte numerische Integration. *Det Kongelige Norske Videnskaber Selskab Forhandlinger* 28, (1955), 30-36.
 Stiefel, E., and Rutishauser, H. Remarques concernant l'integration numerique. *Comptes Rendus Acad. Scil (Paris)* 252, (1961), 1899-1900.

CERTIFICATION OF ALGORITHM 78
 RATFACT (C. Perry, *Comm. ACM* 5, Feb. 1962)
 M. H. HALSTEAD
 Navy Electronics Laboratory, San Diego, Calif.

RATFACT was copied in the Navy Electronics Laboratory International ALGOL Compiler, NELIAC, and tested on the UNIVAC M-490 Countess and the CDC 1604. Polynomials of order 2 through 6 were tested. No corrections were found necessary. It was noted that a polynomial whose coefficients included a common factor would produce superfluous values of p/q , in which this fraction was indeed a root, but one in which p and q contained a common factor.

Reprints Of

"Report on the Algorithmic Language ALGOL 60"

By Peter Naur (Ed.) et al.

Communications of the ACM, Vol. 2, No 5 (May 1960), pp. 299-314

Are Now Available
 from

Association for Computing Machinery
 14 East 69 Street
 New York 21, N. Y.

* * * * *

Single copies to
 individuals: No charge.

Single copies to
 companies: 50 cts.

Multiple copies—
 First ten: 50 cts. ea.
 Next 100: 25 cts. ea.
 All over 100: 10 cts. ea.