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 1967.

305 SYMMETRIC POLYNOMIALS

Expresses the symmetric sum $\sum x_{i1}^{b_1} x_{i2}^{b_2} \ldots x_{in}^{b_n}$ over n variables as a sum of determinants in the unitary symmetric functions $\sum x_{i1} x_{i2} \ldots x_{ir}$.

306 PERMUTATIONS WITH REPETITIONS

Successive calls of the algorithm generate in an array all permutations of its elements in reverse lexicographical order.

307 SYMMETRIC GROUP CHARACTERS

Produces the irreducible character of the symmetric group corresponding to the partitions of the representation and the class of the group S_n.

308 GENERATION OF PERMUTATIONS IN PSEUDO-LEXICOGRAPHIC ORDER

309 GAMMA FUNCTION WITH ARBITRARY PRECISION

Computes the value of the gamma function for any real argument for which the result can be represented within the computer, working with a given number of decimal digits. It is especially useful for variable field length computers and for multiple-precision calculations.

310 PRIME NUMBER GENERATOR 1

311 PRIME NUMBER GENERATOR 2

Algorithm 310 generates the prime numbers less than or equal to a given number. Algorithm 311 is a faster version of algorithm 310.

The following Algorithms were published in *Nordisk Tidskrift for Informationsbehandling* in the April 1967 issue.

Contribution No. 20 SMITH'S NORMAL FORM

A procedure is given for reduction of a polynomial matrix $A(\lambda)$ to Smith's normal form, all coefficients supposed to be rational numbers. In particular, the case $A(\lambda) = \lambda I\text{-}C$ is considered, where C is a constant matrix.

Paper ON THE PRACTICAL APPLICATION OF THE MODIFIED ROMBERG ALGORITHM

Algorithms

Algorithm 31. *COMPLEX FOURIER SERIES*

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

Author's Note.

This procedure implements the Cooley–Tukey algorithm (Cooley and Tukey, 1965) for computing complex Fourier series. For $n < 0$, $(N = abs(n))$ the procedure evaluates the coefficients $A(k)$ of the Fourier analysis:

$$A(k) = \frac{1}{N} \sum_{j=0}^{N-1} X(j) W^{-jk}, \; k = 0, 1, \ldots, N-1$$

of the sampled function $X(j)$. For $n > 0, (N = n)$, the function $X(j), j = 0, 1, \ldots, N-1$, may be synthesized from the coefficients by:

$$X(j) = \sum_{k=0}^{N-1} A(k) W^{+jk}$$

where, in each case, $W = e \uparrow (2\pi i/N)$.

At input, in case $n < 0$, the arrays *rea*, *ima* $[0 : N-1]$ contain, respectively, the real and imaginary parts of the function $X(j)$ at the sample points $j = 0, 1 \ldots, N-1$. At output the same arrays contain the real and imaginary parts of $A(k), k = 0, 1, 2 \ldots, N-1$. For $n > 0$ the reverse situation applies, i.e. at input $A(k)$ is provided and, for output, the procedure yields $X(j)$.

Cooley and Tukey have shown that if $N = a^p b^q c^r \ldots$ where $a, b, c \ldots$ are the prime factors of N, then the number of complex operations needed is $N(p \times a + q \times b + c \times r \ldots)$. Capitalising on the binary nature of computers, a highly efficient implementation of this algorithm is possible, in machine code, for $N = 2^m$.

This procedure places no restrictions on N, provided that $N > 1$, but is most efficient for those cases in which N is large compared with $(p \times a + q \times b + c \times r \ldots)$.

N is first decomposed into its prime factors. The Cooley–Tukey process is executed to obtain the $A(k)$ or $X(j)$ but not in the correct order. A final reordering of the elements of *rea*, *ima* is performed to achieve the desired result.

The description given above is appropriate for origin = 0. For origin = s the expansions are:

$$A(k) = \frac{1}{N} \sum_{j=-s}^{N-s-1} X(j) W^{-jk}, \; k = -s, -s+1, \ldots, N-s-1$$

and

$$X(j) = \sum_{k=-s}^{N-s-1} A(k) W^{+jk}, \; j = -s, -s+1, \ldots, N-s-1,$$

with the $A(k)$ or $X(j)$ contained, as appropriate in *rea*, *ima* $[0 : N-1]$.

[Thanks are due to the referee for his corrections and many suggested improvements.]

Reference

Cooley, J. W., and Tukey, J. W. (1965). An Algorithm for the Machine Calculation of Complex Fourier Series, *Mathematics of Computation*, Vol. 19, pp. 297–301.

```
procedure complexfourier (n, rea, ima, origin); value n, origin;
integer n, origin; array rea, ima;
begin integer nn, nb, max, wtk;
   integer array b [1 : ln (abs (n))/ ln (3·0) + 1];

   procedure primefactors (n, f, nf); value n;
   integer n, nf; integer array f;
   comment decomposes n into factors of four (and a possible
   single factor of two) and its odd prime factors. The factors
```

occupy f[1] through f[nf] of f[1 : k] where k will be sufficient if $3^k \geqslant n$;

```
begin integer i, q, p, d;
   i := 0; p := 4;
   d := − 2; q := n;
next: if q ⩾ p then
   begin q := n ÷ p;
      if n ≠ q × p then
      begin p := p + d;
         d := if d < 1 then 1 else 2
      end
      else
      begin i := i + 1;
         f[i] := p; n := q
      end;
      goto next
   end;
   if n ≠ 1 then
   begin i := i + 1;
      f[i] := n
   end;
   nf := i
end primefactors;
```

```
integer procedure rev (m, b, n, r); value m, n, r;
integer m, n, r; integer array b;
comment performs an integer transformation as follows:—
1. An integer k is expressed as (k1, k2, . . ., kn) where k1
is the most and kn the least significant digit of the n-digit
mixed radix representation of k, with ki associated with
base b[i].
2. The subset (k1, k2, . . ., kr) is converted to a second
integer, treating k1, kr as the least and most significant digits
respectively.
   The procedure identifier rev yields the result of applying
this transformation to m;
begin integer sum, i, q, d, j;
   sum := 0;
   for i := n step −1 until 2 do
   begin d := b[i];
      q := m ÷ d; j := m − q × d;
      if i ⩽ r then sum := sum × d + j;
      m := q
   end;
   rev := sum × b[1] + m
end rev;
```

```
integer procedure jtok (m, b, n); value m, n;
integer m, n; integer array b;
comment performs an integer transformation according to
the following rules:—
1. An integer j is expressed as (j1, j2, . . ., jn) where j1 is the
least significant digit and jn the most significant digit of the
n-digit mixed radix representation of j where each ji is
associated with base b[i].
2. A second integer k is formed by reversing the order of
significance of the digits, i.e. jn is now regarded as the least
significant digit.
   The procedure identifier jtok yields the result of applying
this transformation to m;
begin integer sum, i, j, d, q, nless1;
   nless1 := n − 1; sum := 0;
   for i := 1 step 1 until nless1 do
   begin d := b[i];
      q := m ÷ d; j := m − q × d;
      sum := sum × d + j;
      m := q
   end;
   jtok := sum × b[n] + m
end jtok;
```

```
procedure jkperm (rea, ima, na, origin, b, nb);
value na, nb, origin;
array rea, ima; integer array b; integer na, nb, origin;
comment rea, ima[0 : abs(na)] are the real and imaginary
parts of a complex array a.  For na > 0 the procedure
rearranges the elements so that a[j] := a[k] where k is the
j to k transformation of j defined by procedure jtok.  The
radix base vector for that procedure is b[1 : nb].
Elements a[k] are exchanged with a[j] in the order
j = 0, 1, 2, . . ., na provided k > j.  If k < j then ele-
ment a[k] is now elsewhere following previous exchanges.
In this case repeated transformations of k are made until
k > j.
   For na < 0 the rearrangement is a[j] := a[k]/(abs(na)+1).
The above description applies for origin = 0.  For origin ≠ 0
the rearrangement is a[j] := a[k] where k = jtok (j−origin)
+ origin, these sums and differences being modulo
(abs (na) + 1);
begin integer nj, j, k, n; real nn, rtemp, itemp;
   if na < 0 then
   begin n := 1 − na;
      nj := − na
   end
   else
   begin n := 1 + na;
      nj := na
   end;
   nn := n;
   for j := 0 step 1 until nj do
   begin k := j;
   test: k := k − origin;
      if k < 0 then k := k + n;
      k := jtok (k, b, nb) + origin;
      if k > nj then k := k − n;
      if k < j then goto test;
      if j ≠ k then
      begin rtemp := rea[k]; itemp := ima[k];
         if na < 0 then
         begin rtemp := rtemp / nn;
            itemp := itemp / nn
         end;
         rea[k] := rea[j]; rea[j] := rtemp;
         ima[k] := ima[j]; ima[j] := itemp
      end
      else
      if na < 0 then
      begin rea[j] := rea[j] / nn;
         ima[j] := ima[j] / nn
      end
   end j
end jkperm;
```

```
comment main program;
wtk := nn :=
primefactors (nn, b, nb) if n < 0 then − n else n;
```

```
max := b[nb];
if max < b[1] then max := 4;
begin array rw, iw, rx, ix[1 : max];
  integer nless1, wtj, j, i, r, gpstep, wg, g, kend, k, jj, kbase,
  rless1;
  real z, wrz, rwrz, iwrz, rwj, iwj, wgz, re, im, rwi, iwi,
  rsum, isum, twopi;
  nless1 := nn − 1;
  twopi := 6·283185307180;
  comment this constant is the value of 2π correct to
  12 decimal places;
  z := twopi/n; rless1 := 0;
  if n < 0 then twopi := − twopi;
  for r := 1 step 1 until nb do
  begin kbase := b[r]; gpstep := wtk;
    wtk := wtk ÷ kbase;
    wrz := twopi/kbase;
    rwrz := cos (wrz); iwrz := sin (wrz);
    for g := 0 step gpstep until nless1 do
    begin if g = 0 then
      begin rwj := 1·0;
        iwj := 0·0
      end
      else
      begin wg := rev (g, b, nb, rless1) × wtk;
        wgz := wg × z;
        rwj := rw[1] := cos (wgz);
        iwj := iw[1] := sin (wgz)
      end;
      for j := 2 step 1 until kbase do
      begin re := rwrz × rwj − iwrz × iwj;
        im := rwrz × iwj + iwrz × rwj;
        rwj := rw[j] := re;
        iwj := iw[j] := im
      end;
      kend := g + wtk − 1;
      for k := g step 1 until kend do
      begin jj := k + origin;
        for j := 1 step 1 until kbase do
        begin if jj > nless1 then jj := jj − nn;
          rx[j] := rea[jj]; ix[j] := ima[jj];
          jj := jj + wtk
        end;
        jj := k + origin;
        for i := 1 step 1 until kbase do
        begin if jj > nless1 then jj := jj − nn;
          rwi := rw[i]; iwi := iw[i];
          rsum := rx[kbase]; isum := ix[kbase];
          for j := kbase − 1 step −1 until 1 do
          if i ≠ 1 ∨ g ≠ 0 then
          begin re := rsum × rwi − isum × iwi;
            im := rsum × iwi + isum × rwi;
            rsum := re + rx[j];
            isum := im + ix[j]
          end
          else
          begin rsum := rsum + rx[j];
            isum := isum + ix[j]
          end;
          rea[jj] := rsum; ima[jj] := isum;
          jj := jj + wtk
        end i
      end k
    end g; rless1 := r
  end r;
  if n < 0 then nless1 := − nless1;
  jkperm (rea, ima, nless1, origin, b, nb)
end
end complexfourier
```

Contributions for the Algorithms Supplement should be sent to

P. Hammersley
University Mathematical Laboratory
Corn Exchange Street
Cambridge

Discussion and Correspondence

Modification of the complex method of constrained optimization

By J. A. Guin*

On the basis of some recent computational experience using the complex method of Box (1965), it has been found that the following modifications in the method increase the chances of reaching the optimum.

(1) Box has suggested that a projected trial point be moved in halfway toward the centroid of the remaining points until a new point better than the rejected one is found. If by chance all points on the line from the centroid to the projected point are worse than the original point, application of this rule causes the projected point eventually to coincide with the centroid. When this happens no further progress is possible. Considering this situation, it is recommended that if the projection factor α is found to have been reduced below a certain quantity (we have found $\alpha = 10^{-5}$ to be a satisfactory criterion) without obtaining a better function value for the projected trial point, then this trial point should be replaced to its original unprojected position and the second worst point rejected instead. This procedure tends to keep the complex moving unless the centroid is indeed near the optimum.

(2) The rule of setting an independent variable to 0·000001 inside its limit sometimes causes the method to obtain a false optimum if all points of the complex fall into this hyperplane. This happens especially when the optimum is near, but not upon the constraint. To alleviate this situation, it is recommended that the above rule be abandoned and that only the

* *Department of Chemical Engineering, The University of Texas, Austin, Texas.*