Algorithms

G. E. FORSYTHE, J. G. HERRIOT, Editors

ALGORITHM 255
COMPUTATION OF FOURIER COEFFICIENTS [C6]

LINDA TEIJELO (Recd. 18 Nov. 1964 and 25 Nov. 1964)
Stanford Computation Ctr., Stanford U., Calif.

```
procedure FOURIER(F, eps, subdivmax, m, cosine, sine, cint,
  sint);
  value eps, subdivmax, m, cosine, sine;  real eps, cint, sint;
  Boolean cosine, sine;  integer subdivmax, m;
    real procedure F;
comment FOURIER computes the Fourier coefficients cint =
  ∫₀¹ F(x)cos(mπx) dx (if cosine is true) and/or sint = ∫₀¹ F(x)
  sin (mπx) dx (if sine is true), where m > 0. The method is that
  of Filon (for a brief exposition see [1] and for Filon's original work
  see [2] or [3]). Computation is terminated when the number of
  times the interval [0,1] has been halved (n) has exceeded sub-
  divmax (10 is suggested), or when n > 5 and two successive ap-
  proximations of the integral agree to within eps (10⁻⁷ is sug-
  gested) times the value of the last approximation. In the former
  case, cint or sint is assigned the value of the last approximation.
  The condition n > 5 is imposed because of substantial cancella-
  tions which may take place during the early stages of sub-
  dividing;
begin real sumcos, sumsine, oddcos, oddsine, pi, a, b, g, t, h, p, k,
  c0, c1, s0, s1, int1, int2, prevint1, prevint2, tn1, t3, temp;
  integer n, i;  Boolean bool;
  bool := false;  pi := 3.14159265359;  k := m × pi;
  sumcos := (F(1.0) × cos(k)+F(0)) × .5;
  sumsine := F(1.0) × sin(k) × .5;
L0:  n := 1;  h := 0.5;  t := .5 × k;  tn1 := 1;
L1:  c0 := cos(2.0×t);  c1 := cos(t);
  s0 := sin(2.0×t);  s1 := sin(t);
  t3 := t↑3;  p := c1 × s1;
  a := (t↑2−s1↑2×2.0+t×p)/t3;
  b := (2.0×(t×(c1↑2+1.0)−2.0×p))/t3;
  g := 4.0 × (−t×c1 + s1)/t3;
  if bool then go to L2;
  if sine then
    begin
      oddsine := F(h) × s1;
      for i := 2 step 1 until tn1 do
      begin temp := c1 × c0 − s1 × s0;
        s1 := s1 × c0 + c1 × s0;
        c1 := temp;
        oddsine := F((2×i−1)×h)×s1 + oddsine
      end;
      if n = 1 then
        begin n := 2;  h := .25;  t := .25×k;  tn1 := 2;
          prevint2 := (a×(F(0)−F(1.0)×cos(k))+
            b×sumsine+g×oddsine) × .5;
          sumsine := sumsine + oddsine;  go to L1
        end
      else
        begin int2 := h × (a×(F(0)−F(1.0)×cos(k))+
          b×sumsine+g×oddsine);
          if abs(prevint2−int2)<eps×int2∧n>5 then
            begin sint := int2;  bool := true;  go to L0 end
          else
            begin n := n + 1;
              if n > subdivmax then
                begin bool := true;
                  sint := int2;  go to L0
                end;
              sumsine := sumsine + oddsine;  h := .5 × h;
                t := .5 × t;  tn1 := 2 × tn1;
              prevint2 := int2;  go to L1
            end
        end
    end of sine computations;
L2:  if cosine then
    begin
      oddcos := F(h) × c1;
      for i := 2 step 1 until tn1 do
      begin temp := c1 × c0 − s1 × s0;
        s1 := s1 × c0 + c1 × s0;
        c1 := temp;
        oddcos := F((2×i−1)×h) × c1 + oddcos
      end;
      if n = 1 then
        begin n := 2;  h := .25;  t := .25 × k;  tn1 := 2;
          prevint1 := (a×F(1.0)×sin(k)+b×sumcos+g×oddcos)
            × .5;
          sumcos := sumcos + oddcos;  bool := true;  go to L1
        end
      else
        begin int1 := h ×(a×F(1.0)×sin(k)+b×sumcos+g×
          oddcos);
          if abs(prevint1−int1) < eps × int1 ∧ n > 5 then
            begin cint := int1;  go to exit end
          else
            begin n := n + 1;
              if n > subdivmax then begin cint := int1;
                go to exit end;
              sumcos := sumcos + oddcos;  h := .5 × h;
                t := .5 × t;  tn1 := 2 × tn1;
              prevint1 := int1;  go to L1
            end
        end
    end of cosine computations;
exit:  end FOURIER
```

REFERENCES:

1. HAMMING, R. W. *Numerical Methods for Scientists and Engineers.* McGraw-Hill, 1962, pp. 319–321.
2. TRANTER, C. J. *Integral Transforms in Mathematical Physics.* Methuen & Co., Ltd., 1951, pp. 67–72.
3. FILON, L. N. G. On a quadrature formula for trigonometric integrals. Proc. Roy. Soc. Edinburgh *49*, 1928–29, 38–47.

CERTIFICATION OF ALGORITHM 243 [B3]
LOGARITHM OF A COMPLEX NUMBER [David S. Collens *Comm. ACM 7*(Nov. 1964), 660]

J. BOOTHROYD (Recd. 18 Jan. 1965)
Computing Centre, U. of Tasmania, Hobart, Tasmania

With the label parameter *FAIL* removed from the value list to accommodate a restriction of Elliott 503 ALGOL, the algorithm was successfully run on an Elliott 503, using the data test cases published with the algorithm. The constants in the algorithm were rounded to nine significant decimal digits, and this probably explains the two differences between the results obtained and those published, namely:

a	b	c	d
−1	−1	0.346574	
2	1		0.463648

(*Algorithms are continued on page 330.*)

the particular type of data being studied, the weighting functions are not worthwhile.

Programming the Algorithm. The frequencies for which the spectral densities are computed were chosen between 0.625 and 100cps, spaced at $\frac{1}{3}$ octave. This results in 23 frequencies. Since these are fixed, and for fixed $\Delta\tau$, it is possible to compute all the cos $(\omega_j i\ \Delta\tau)$ terms and store them in a table.

With such a scheme, using FORTRAN, the spectral density of a sample of EEG autocovariance data can be computed in about $1\frac{1}{2}$ minutes. A flowchart for this computation is shown in Figure 3. It the cosines must be generated, the computations take nearly 20 minutes.

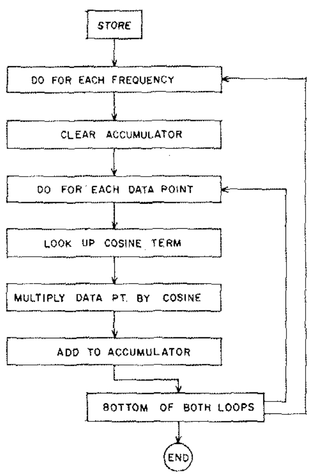

FIG. 3. Computation of spectral density (FORTRAN)

Suppose the cosines terms are stored in the following way: first a list of the M points for the first frequency, then for the second frequency, etc. Then the list subroutines may be used for this computation as shown in the flowchart of Figure 4. Execution time for this program is about 20 seconds.

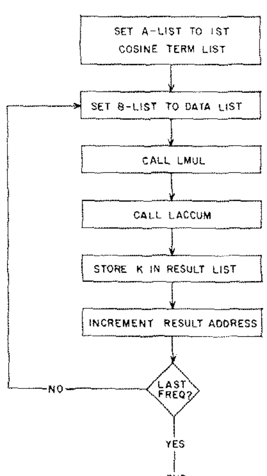

FIG. 4. Computation of spectral density

This is a speed gain of more than 4 over FORTRAN. The amount of storage space required is also much less.

4. Conclusion

Power of the Method. This programming procedure lends itself well to a certain large class of problems. For these types of problems, the routines have proven very useful. Programming in machine language or SPS is greatly simplified, yet the power and speed of machine language is essentially preserved.

It is the authors' opinion that a set of routines such as described would form a useful addition to many program libraries.

Extending the Method. Depending upon the type of work, other routines may be written to perform special types of data handling common to the particular installation. The input and output routines especially should be adapted to the particular format commonly used. Other special machine functions, such as analog/digital conversion, online plotting, special readout devices, etc. can be handled by such routines.

The Algorithm. Computation of other types of spectral densities may be possible with this method. However, convergence at high frequencies has not been investigated, so one must be careful in using it. For the type of data under study, however, the method appears quite satisfactory.

Acknowledgments. The authors wish to express thanks for the assistance and encouragement of many people, especially Dr. S. J. Dwyer, III, for his suggestions and encouragement; Dr. Roy Keller, for providing the computer time; and The Staff of the EEG project.

RECEIVED SEPTEMBER, 1964

Algorithms—cont. from page 279

CERTIFICATION OF ALGORITHM 119 [H]
EVALUATION OF A PERT NETWORK [Burton Eisenman and Martin Shapiro, *Comm. ACM 5* (Aug. 1962), 436]

L. STEPHEN COLES (Recd. 10 Nov. 1964 and 7 Dec. 1964)
Carnegie Institute of Technology, Pittsburgh, Pa.

The procedure was tested on a CDC-G20, using the ALGOL compiler developed by Carnegie Tech. Before compilation was possible, the following modifications were required in order to make it a correct ALGOL 60 procedure.

1. Insert after the end of *scan*
   switch *sw*2 := *g*1, *g*2;
2. Modify **comment** By means of the switch, *s*, ⋯
to read
   comment By means of the switches, *sw*1 and *sw*2, ⋯
3. Modify **begin switch** *s* := *b*1, *b*2;
to read
   begin switch *sw*1 := *b*1, *b*2; **go to** *sw*1 [*s*];
4. Modify **switch** *s* := *g*1, *g*2;
to read
   go to *sw*2 [*s*];

With these changes the procedure was operated successfully on a number of small test problems.