ALGORITHM 257
HAVIE INTEGRATOR [D1]

ROBERT N. KUBIK (Recd. 9 June 1964 and 21 Dec. 1964)
The Babcock & Wilcox Co. Lynchburg, Viriginia

```
real procedure havieintegrator (x, a, b, eps, integrand, m);
  value a, b, eps, m; integer m;
  real integrand, x, a, b, eps;
comment This algorithm performs numerical integration of definite integrals using an equidistant sampling of the function and repeated halving of the sampling interval. Each halving allows the calculation of a trapezium and a tangent formula on a finer grid, but also the calculation of several higher order formulas which are defined implicitly. The two families of approximate solutions will normally bracket the value of the integral and from these convergence is tested on each of the several orders of approximation. The algorithm is based on a private communication from F. Håvie of the Institutt for Atomenergi Kjeller Research Establishment, Norway. A FORTRAN version of the algorithm is in use on the Philco-2000. A few test cases have been run on the Burroughs B5000. In particular, a and b are the lower and upper limits of integration, respectively, eps is the convergence criterion, integrand is the value of the function to be integrated (sampled), and m is the maximum order approximation to be considered in attempting to satisfy the eps convergence criterion. If convergence is not gained, then the value returned is that of the nonlocal variable, mask. The parameter integrand must be an expression involving the variable of integration x. See the driver program of this algorithm for examples of the procedure call;
begin real h, endpts, sumt, sumu, d;
  integer i, j, k, n;
  real array t, u, tprev, uprev[1:m];
  x := a; endpts := integrand; x := b; endpts := 0.5 ×
    (integrand+endpts);
  sumt := 0.0; i := n := 1; h := b − a;
estimate: t[1] := h × (endpts+sumt); sumu := 0.0;
  comment t[1] = h × (0.5×f[0]+f[1]+f[2]+···+0.5×f[2^i−1]);
  x := a − h/2.0;
  for j := 1 step 1 until n do
  begin
    x := x + h; sumu := sumu + integrand
  end;
  u[1] := h × sumu; k := 1;
  comment u[1] = h × (f[1/2]+f[3/2]+···+f[(2^i−1)/2]), k
    corresponds to approximate solution with truncation error
    term of order 2k;
test: if abs(t[k]−u[k]) ≤ eps then
  begin
    havieintegrator := 0.5 × (t[k]+u[k]); go to exit
  end;
  if k ≠ i then
  begin
    d := 2↑(2×k);
    t[k+1] := (d×t[k]−tprev[k])/(d−1.0);
    tprev[k] := t[k];
    u[k+1] := (d×u[k]−uprev[k])/(d−1.0);
    uprev[k] := u[k];
    comment This implicit formulation of the higher order integration formulas is given in [ROMBERG, W. Vereinfachte Numerische Integration. Det Kong. Norske Videnskabers Selskabs Forhandl. 28, 7 (1955), Trondheim; and in STIEFEL, E. Einführung in der Numerische Mathematik. Teubner Verlagsges., Stuttgart, 1961, pp. 131–136. (English translation: An Introduction to Numerical Mathematics, Academic Press, New York, 1963, pp. 149–155)]. See also Algorithm 60 where the same implicit relationship is used to calculate t[k+1] only;
    k := k + 1;
    if k = m then
    begin
      havieintegrator := mask; go to exit
    end;
    go to test
  end;
  h := h/2.0; sumt := sumt + sumu;
  tprev[k] := t[k]; uprev[k] := u[k];
  i := i + 1; n := 2 × n;
  go to estimate;
exit: end havieintegrator
```

Following is a driver program to test havieintegrator.

```
begin comment First test case, y = ∫_0^{π/2} cos x dx = 1.0
    (0.9999999981 as executed on the B5000), is an example of the
    higher order approximations yielding fast convergence as in
    Algorithm 60; second test case, y = ∫_0^{4.3} e^{−x^2} dx = .8862269255
    (.8862269739 as executed on the B5000), is an example where
    this algorithm is superior to Algorithm 60 because the higher
    order approximations converge more slowly than the linear
    approximations; see also [THACHER, H. C., JR., Remark on
    Algorithm 60. Comm. A.C.M. 7 (July 1964), 420];
  real a, b, eps, mask, y, answer;
  a := 0.0; b := 1.5707963; eps := 0.000001; mask := 9.99;
  answer := havieintegrator (y, a, b, eps, cos(y), 12);
  outreal (1, answer);
  a := 0.0; b := 4.3;
  answer := havieintegrator (y, a, b, eps, exp(−y×y), 12);
  outreal (1, answer);
end
```

ALGORITHM 258
TRANSPORT [H]

G. BAYER (Recd. 4 May 1964 and 4 Mar. 1965);
Technische Hochschule, Braunschweig, Germany

```
procedure transport (c, x, a, b, m, n, inf, cost);
  value m, n, inf; integer m, n, inf, cost;
    integer array c, x, a, b;
comment The parameters are c[i,j] array of costs, the quantities available a[i], the quantities required b[j], i = 1, ···, m, j = 1, ···, n. Sum of a[i] = sum of b[j]. inf has to be the greatest positive integer within machine capacity, all quantities have to be integer. The flows x[i, j] are computed by the "primal-dual-algorithm," cited in [HADLEY, G. Linear Programming. Reading, London, 1962, pp. 351–367]. The procedure follows the description given on p. 357. Multiple solutions are left out of account;
begin integer i, j, p, h, k, y, t, l;
  integer array v, xsj, s, r, listv[1:n], u, xis, d, g, listu[1:m];
  Boolean array xb[1:m, 1:n];
  integer procedure sum(i, a, b, x); value a, b;
    integer i, a, b, x;
    begin integer s;
      s := 0;
      for i := a step 1 until b do s := s + x;
      sum := s
    end;
  comment Array xb for notation of "circled cells," listu and listv lists of labeled rows and columns. Other notations follow Hadley;
  for i := 1 step 1 until m do xis[i] := a[i];
  for j := 1 step 1 until n do xsj[j] := b[j];
  for i := 1 step 1 until m do
  begin h := inf; for j := 1 step 1 until n do
    begin x[i, j] := 0; p := c[i, j]; if p < h then h := p end;
    u[i] := h;
```