Algorithms supplement

Note by the Editor
Algorithm No. 107 is published under our agreement with the Institute of Mathematics and its Applications, and is associated with the authors' article in the *JIMA*, Volume 24, Number 1, August 1979.

Algorithm 107
A WEIGHTED SIMPLEX PROCEDURE FOR THE SOLUTION OF SIMULTANEOUS NONLINEAR EQUATIONS

W. L. Price and M. Dowson

Authors' Notes
The weighted simplex (WS) procedure provides an alternative to the Newton-Raphson (NR) procedure for the solution of N simultaneous nonlinear equations in N real variables, rapid convergence being obtained from a sufficiently good initial approximation. In contrast to the NR the WS procedure does not involve the computation of derivatives. The principle of the WS method, together with the results of comparative performance tests, are published elsewhere (Price, 1979).

Given the set of equations $f_i(x_1, \ldots, x_N) = 0$, $i = 1, \ldots N$, the procedure operates on the data associated with a SIMPLEX of $N + 1$ points in N-space. Prior to each iteration the co-ordinates of these points, x_{ij} (where $i = 1, \ldots N; j = 1, \ldots N + 1$), and the corresponding function values, f_{ij}, are held in store. For each point j of the simplex a WEIGHT, w_j, is computed by solving (by Gauss elimination) the set of $N + 1$ linear equations $\sum_j w_j = 1$; $\sum_j w_j f_{ij} = 0$, $i = 1, \ldots N$. The co-ordinates $X_i = \sum_j w_j x_{ij}$ of the weighted centroid, X, of the simplex are then determined, and each of the N functions is evaluated at X. If, at X, the magnitude of every function is less than ACC, a user-supplied measure of the required accuracy, then the procedure terminates. Otherwise one of the simplex points is discarded, its place being taken by X, and the modified simplex which results forms the basis of the next iteration. The discard point is chosen to be L, that point of the simplex with the least positive (most negative) weight, unless L happens to be the X point of the previous iteration in which case the discard point is chosen randomly from the simplex but excluding the point M, that point with the most positive weight (Price, 1979 for full explanation). The procedure is initialised by generating a simplex of $N + 1$ points randomly positioned within a hypercube of linear dimension z (the zone size) the hypercube being centred on the initial approximation supplied by the user. The choice of z is not critical, but ideally the hypercube should be large enough to embrace the exact solution sought yet small enough to exclude other possible solutions to the given system of equations.

The procedure has been programmed in ANSI FORTRAN so as to be generally applicable and to require the minimum of user coding to run a specific problem. All arrays used by the program are declared in the main program so that the user has only to change the DIMENSION statement to adapt the program for use with any particular maximum value of N. The user must supply a subroutine FUNCT(X, F, N) which calculates the values of his set of functions F, corresponding to the variables X, where X and F are both of dimension N. He must also supply, as data, the value of N, the co-ordinates of the initial approximation, the zone size z, and the required accuracy ACC. The user must code a random number generator appropriate to his computer—RANF is a standard function on the Cyber 72 used by the authors.

As an example of the operation of the program a printout is supplied for a run relating to the specific two-variable problem in which the functions are

$$F_1 = 2x_1^3 x_2 - x_2^3$$
$$F_2 = 6x_1 - x_2^2 + x_2$$

The pair of equations $F_1 = 0$ and $F_2 = 0$ have three solutions, the approximate locations at (0,0), (1.5, 3.5) and (1.5, −2.5) having been obtained by a global search procedure. The WS procedure was used to obtain a refinement of the solution for which the initial approximation is $x_1 = 1.5$, $x_2 = 3.5$. A zone size $z = 1$ was chosen so as to exclude the neighbourhoods of the other two solution points. The precision was specified by ACC $= 10^{-6}$. The WS procedure achieves the exact solution at (2,4) in six iterations, the same number as is required by the NR procedure from the same starting point and with the same precision.

Reference
PRICE, W. L. (1979). A Weighted Simplex Procedure for the Solution of Simultaneous Nonlinear Equations, *JIMA*, Vol. 24 no. 1, pp. 1-8).

```
C
C     WEIGHTED SIMPLEX PROGRAM
C     WRITTEN BY W.L.PRICE
C     TRANSLATED INTO FORTRAN BY M.DOWSON
C
      DIMENSION V(3,4),U(3,4),X(2),F(2)
C
C     DIMENSIONS ARE SET AS FOLLOWS
C     V(N+1,2N),U(N+1,N+2),X(N),F(N)
C     WHERE N=NO. OF VARIABLES AND FUNCTIONS
C
C     SET I/O STREAMS
C
      IN=1
      IOUT=2
C
C     INPUT N AND STARTING VALUES OF VARIABLES X(N)
C
      READ(IN,1000)N
      READ(IN,1001)(X(I),I=1,N)
      NP1=N+1
      NP2=N+2
      N2=N*2
C
C     CALCULATE FUNCTION VALUES AND PRINT THEM
C
      CALL FUNCT(X,F,N)
      WRITE(IOUT,2000)
      WRITE(IOUT,2001)(I,X(I),I,F(I),I=1,N)
C
C     INPUT ZONE SIZE AND ACCURACY REQUIRED
C
      READ(IN,1001)Z,ACC
      WRITE(IOUT,2002)Z,ACC
C
C     CALL SIMPLEX ROUTINE
C
      CALL WSMPLX(X,F,N,V,U,NP1,NP2,N2,Z,ACC)
C
C     OUTPUT SOLUTION VALUES
C
      WRITE(IOUT,2003)
      WRITE(IOUT,2001)(I,X(I),I,F(I),I=1,N)
      STOP
C
 1000 FORMAT(I2)
 1001 FORMAT(8E10.3)
 2000 FORMAT(25H1WEIGHTED SIMPLEX PROGRAM///16H STARTING VALUES//
     1        5X,9HVARIABLES,19X,9HFUNCTIONS/)
 2001 FORMAT(3H X(,I2,4H) = ,1PE10.3,10X,2HF(,I2,4H) = ,1PE10.3)
 2002 FORMAT(//13H ZONE SIZE = ,1PE10.3/21H ACCURACY REQUIRED = ,
     1        1PE10.3)
 2003 FORMAT(//16H SOLUTION VALUES//5X,9HVARIABLES,19X,9HFUNCTIONS/)
      END
      SUBROUTINE WSMPLX(X,F,N,V,U,NP1,NP2,N2,Z,ACC)
C
C     WEIGHTED SIMPLEX SUBROUTINE
C     WRITTEN BY W.L.PRICE
C     TRANSLATED INTO FORTRAN BY M.DOWSON
C
C     X=ARRAY OF VARIABLES.  ON ENTRY = STARTING VALUES
C                            ON EXIT  = SOLUTION VALUES
C     F=ARRAY OF FUNCTION VALUES CORRESPONDING TO X
C     N=NUMBER OF FUNCTIONS AND VARIABLES
```

```
C     V AND U ARE ARRAYS USED BY WSMPLX
C     NP1=N+1
C     NP2=N+2
C     N2=N*2
C     Z=ZONE SIZE
C     ACC=REQUIRED ACCURACY
C
      DIMENSION V(NP1,N2),U(NP1,NP2),X(N),F(N)
      FNP1=FLOAT(NP1)
C
C     GENERATE INITIAL SIMPLEX
C
      CALL FUNCT(X,F,N)
      DO 10 K=1,N
      KN=K+N
      V(1,K)=X(K)
      V(1,KN)=F(K)
   10 CONTINUE
      DO 40 J=1,N
      JP1=J+1
      DO 20 K=1,N
      X(K)=V(1,K)+Z*(0.5-RANF(0.0))
C
C     RANF IS RANDOM NUMBER GENERATOR WITH DUMMY ARGUMENT
C     RETURNING UNIFORMLY DISTRIBUTED VALUES IN RANGE 0.0 TO 1.0
C
      V(JP1,K)=X(K)
   20 CONTINUE
      CALL FUNCT(X,F,N)
      DO 30 K=1,N
      KPN=K+N
      V(JP1,KPN)=F(K)
   30 CONTINUE
   40 CONTINUE
      ID=1
C
C     COMPUTE WEIGHTED CENTROID, X
C
   50 CALL WEIGHT(V,U,L,M,N,NP1,NP2,N2)
      DO 60 K=1,N
      X(K)=0.0
      DO 60 J=1,NP1
      X(K)=X(K)+U(J,NP2)*V(J,K)
   60 CONTINUE
C
C     EVALUATE AT X AND TEST FOR CONVERGENCE
C
      CALL FUNCT(X,F,N)
      DO 70 K=1,N
      IF (ABS(F(K)).GT.ACC) GO TO 80
   70 CONTINUE
C
C     EXIT IF CONVERGED
C
      RETURN
C
C     CHOOSE DISCARD POINT
C
   80 IF (L.NE.ID) GO TO 90
      L=1+INT(FNP1*RANF(0.0))
   90 ID=L
C
C     REPLACE DISCARD POINT BY X
C
      DO 100 K=1,N
      KPN=K+N
      V(ID,K)=X(K)
      V(ID,KPN)=F(K)
  100 CONTINUE
      GOTO 50
      END
      SUBROUTINE WEIGHT(V,U,L,M,N,NP1,NP2,N2)
C
C     COMPUTE WEIGHTS AND FIND MOST POSITIVE WEIGHT, M
C                          AND LEAST POSITIVE WEIGHT, L
C
      DIMENSION V(NP1,N2),U(NP1,NP2)
C
C     INITIALISE U ARRAY
C
      DO 10 K=1,NP2
      U(1,K)=1.0
   10 CONTINUE
      DO 20 K=2,NP1
      U(K,NP2)=0.0
   20 CONTINUE
      DO 30 J=1,N
      JP1=J+1
      JPN=J+N
      DO 30 K=1,NP1
      U(JP1,K)=V(K,JPN)
   30 CONTINUE
C
C     COMPUTE WEIGHTS
C
      DO 70 I=1,N
      IP1=I+1
      KKK=NP1+IP1
      DO 50 KK=IP1,NP1
      K=KKK-KK
      IF (ABS(U(K,I)).LE.ABS(U(K-1,I))) GO TO 50
      DO 40 J=I,NP2
      B=U(K,J)
      U(K,J)=U(K-1,J)
      U(K-1,J)=B
   40 CONTINUE
   50 CONTINUE
      DO 60 K=I,N
      KP1=K+1
      DO 60 J=I,NP1
      JP1=J+1
      U(KP1,JP1)=U(KP1,JP1)-U(I,JP1)*U(KP1,I)/U(I,I)
   60 CONTINUE
   70 CONTINUE
      U(NP1,NP2)=U(NP1,NP2)/U(NP1,NP1)
      L=NP1
      M=NP1
      DO 90 KK=1,N
      K=NP1-KK
      B=0.0
      KP1=K+1
      DO 80 J=KP1,NP1
      B=B+U(K,J)*U(J,NP2)
   80 CONTINUE
      U(K,NP2)=(U(K,NP2)-B)/U(K,K)
C
C     FIND M AND L
C
      IF (U(K,NP2).LT.U(L,NP2)) L=K
      IF (U(K,NP2).GT.U(M,NP2)) M=K
   90 CONTINUE
      RETURN
      END
      SUBROUTINE FUNCT(X,F,N)
      DIMENSION X(N),F(N)
      F(1)=2.0*X(1)**3*X(2)-X(2)**3
      F(2)=6.0*X(1)-X(2)**2+X(2)
      RETURN
      END
```

Algorithm 108

EFFICIENT SOLUTION OF TRIDIAGONAL LINEAR SYSTEMS

Ole Østerby
Computer Science Department
Aarhus University

Author's Note

The solution of a tridiagonal system of linear equations of the form

$$\begin{bmatrix} b_1 & c_2 & & & \\ a_2 & b_2 & c_3 & & \\ & & \cdot & \cdot & \\ & & \cdot & \cdot & \cdot \\ & & & a_n & b_n \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \cdot \\ \cdot \\ d_n \end{bmatrix} \qquad (1)$$

is carried out efficiently by means of Gaussian elimination. In the following we assume that pivoting is not necessary. The idea behind the algorithm is not new (Sprague, 1960; Leavenworth, 1960), but we have supplied the procedure for easy reference.

The operation count is: $2n - 1$ divisions, $3n - 3$ multiplications and $3n - 3$ additions. The only new feature in our procedure is a slightly more efficient use of simply subscripted variables. The number of such references is $10n - 5$ compared to $13n - 9$ in Leavenworth (1960) and Sprague (1960). The solution is returned in array D and array B is destroyed.

The reason for this very detailed operation count is that on most computers division is slower than multiplication, and floating point addition is not so much faster that it makes the time insignificant. Subscripted variables are counted not only because of the subscript handling but also because they imply memory references. In contrast, the simple variables can (and should, if possible) stay in fast registers, of which most modern computers have sufficiently many.

If several systems with the same coefficient matrix are to be solved subsequently, we can store the intermediate results from the LU-decomposition in array A for later use when the right hand sides become available. The operation count for subsequent systems becomes: n divisions, $2n - 2$ multiplications, and $2n - 2$ additions together with $7n - 3$ references to subscripted variables.

If $|b_i| \geq |a_i| + |c_{i+1}|$ then the coefficient matrix is diagonally dominant and we can expect no difficulties with growth of round-off errors. Otherwise we may have to use pivoting, and our procedure does not apply any longer.

If the matrix is symmetric ($a_i = c_i$) we can save one array ($n - 1$ locations), say the array C, but there is no gain in the operation count. A traditional Cholesky factorisation is not competitive because of the n square roots, and a factorisation of the form $A = LDL^T$ has exactly the same operation count as Gaussian elimination.

Another important special case is when the bands are constant:

$$a_i = a, b_i = b, c_i = c.$$

If the matrix is not symmetric the operation count is not reduced although the number of references to subscripted variables drops to $6n - 1$. We must keep the B array as work space (as well as the A array if we have several systems). If the matrix is also symmetric, however, a faster algorithm can be obtained by eliminating from both ends simultaneously (Andres *et al.*, 1974). This algorithm can be generalised such that it does not require symmetry nor constant bands but only that

$$b_i = b_{n-i+1}, a_{i+1} = c_{n-i+1}, c_{i+1} = a_{n-i+1}, \quad i = 1, 2, \ldots, n \div 2.$$

We give the general algorithm in Section 5 as ALGOL procedure COUPLED. The operation count is approximately $1.5n$ divisions, $2.5n$ multiplications, $2.5n$ additions and $7.5n$ references to subscripted variables. Space can be saved by declaring A, B, C $[1: n \div 2 + 1]$.

Returning to the symmetric, constant-band case

$$b_i = b, a_i = c_i = a$$

we observe no further reduction in the operation count but there are now $5n$ references to subscripted variables. If several systems are to be solved the operation counts become almost identical for the coupled algorithm and the ordinary Gauss method. The only advantages with the more complicated coupled algorithm are its $5n$ references to subscripted variables (versus $6n$) and the saving of space ($1.5n$ locations).

2. A boundary value problem for an ordinary differential equation

The solution of

$$-\frac{d}{dx}\left(p(x)\frac{dy}{dx}\right) = f(x), 0 \leq x \leq 1 \qquad (2)$$

with suitable boundary conditions can be approximated by using central differences for the discretisation. If the mesh size is allowed to vary in $[0, 1]$, such that we seek the solution in the points $\{x_i\}$ with

$$0 = x_0 < x_1 < x_2 < \ldots < x_{n+1} = 1,$$
$$h_i = x_i - x_{i-1}, i = 1, 2, \ldots, n + 1,$$

and

$$f_i = f(x_i), y_i = y(x_i), p_i = p(x_i)$$

then a discretisation of (2) is

$$-\frac{2}{h_{i+1} + h_i} \cdot \left(\frac{y_{i+1} - y_i}{h_{i+1}} \cdot p_{i+\frac{1}{2}} - \frac{y_i - y_{i-1}}{h_i} \cdot p_{i-\frac{1}{2}}\right) = f_i, \quad i = 1, 2, \ldots, n, \quad (3)$$

and the equations for y_i become

$$-y_{i+1} \cdot \frac{p_{i+\frac{1}{2}}}{h_{i+1}} + y_i \cdot \left(\frac{p_{i+\frac{1}{2}}}{h_{i+1}} + \frac{p_{i-\frac{1}{2}}}{h_i}\right) - y_{i-1} \cdot \frac{p_{i-\frac{1}{2}}}{h_i} = f_i \cdot \frac{h_{i+1} + h_i}{2}, i = 1, 2, \ldots, n,$$

which is of the form

$$-\frac{y_{i+1}}{h^*_{i+1}} + \frac{y_i}{h^*_{i+1}} + \frac{y_i}{h^*_i} - \frac{y_{i-1}}{h^*_i} = d^*_i, i = 1, 2, \ldots, n, \qquad (4)$$

with

$$h^*_i = \frac{h_i}{p_{i-\frac{1}{2}}} \text{ and } d^*_i = f_i \cdot \frac{h_{i+1} + h_i}{2}, i = 1, 2, \ldots, n.$$

In the special case when $p(x)$ is constant, it might be preferable to define

$$h^*_i = h_i \text{ and } d^*_i = f_i \cdot \frac{h_{i+1} + h_i}{2 \cdot p}, i = 1, 2, \ldots, n,$$

again leading to (4).

The resulting linear system is thus of the form (1) with

$$a_i = c_i = -1/h^*_i, b_i = -a_i - c_{i+1}, d_i = d^*_i, i = 1, 2, \ldots, n,$$

except for d_1 and d_n which are

$$d_1 = d^*_1 + y_0/h^*_1 \text{ and } d_n = d^*_n + y_{n+1}/h^*_{n+1}$$

using the boundary conditions. If conditions other than Dirichlet-type are imposed, one or two extra equations of the same general type are added.

For this special system the algorithm by Rose (1969) is more efficient than ordinary Gaussian elimination in requiring only 1 division, $2n$ multiplications and $4n - 2$ additions in addition to $7n + 1$ references to subscripted variables. Also setting up the equations is easier since the basic values in Rose's method are the step-sizes h^*_i (and not a_i, b_i, and c_i). So for this kind of problem Rose's method is truly superior and we have supplied it as an ALGOL procedure. The parameters are the h^*_i, H$[1:n + 1]$, and the righthand sides $D[1:n]$; the latter are destroyed and the solution is returned in D.

If several systems with the same h^*_j are to be solved, we can compute and store the values $t_i = \sum_1^i h^*_j$ once (an array of n locations is needed as work space); and then compute the solution when the righthand sides become available. But only n additions can be saved in this way, so the operation count for the subsequent systems becomes: 1 division, $2n$ multiplications and $3n - 2$ additions together with $7n$ references to subscripted variables.

If the h_i^* are constant throughout the interval we do not need the array H, and the number of references to subscripted variables drops to $5n + 1$, but the operation count is unchanged.

3. The one-dimensional heat equation

Tridiagonal systems also occur in the numerical solution of the heat equation

$$\frac{\partial}{\partial x}\left(g(x, t) \cdot \frac{\partial u}{\partial x}\right) - \frac{\partial u}{\partial t} = f(x,t), 0 \leq x \leq 1, 0 \leq t$$

with suitable initial and boundary conditions. When using either the backward difference or the Crank-Nicolson method for discretisation, the resulting linear system is of the form (1). If in particular g is constant we have for Crank-Nicolson:

$$a_i = c_i = -1; b_i = 2 + \epsilon; \epsilon = \frac{2h^2}{k \cdot g}$$

where h and k are the stepsizes in the x- and t-direction, respectively.

Rose's method cannot be applied to this system, and attempts to generalise it have proved unsuccessful with respect to providing efficient algorithms (Evans, 1972; 1977).

But Gaussian elimination can be used and in this particular example we can make use of the fact that the band-elements are constant and the off-diagonal elements are -1. The operation count is: $2n - 1$ divisions, $n - 1$ multiplications and $3n - 3$ additions plus $6n - 1$ references to subscripted variables.

The coupled algorithm is even more efficient in this case since all the multiplications involving the c's can now be saved so that we end up with only $1.5n$ divisions, n multiplications, $2.5n$ additions plus $5n$ references to subscripted variables, a result which is better than the one given by Andres *et al.*, (1974).

When advancing the solution one time-step we have another system of linear equations with the same coefficient matrix, so it is a good idea to store the LU decomposition, i.e. in addition to the workspace array $B[1:n]$ we need $A[2:n]$ to store the subdiagonal elements. The operation count for the subsequent systems is: n divisions, $n - 1$ multiplications and $2n - 2$ additions plus $6n - 2$ references to subscripted variables, (or $5n$ for the coupled algorithm).

In the future we shall not expect substantial improvements on the operation counts, but we can achieve significantly better computation *times* by introducing parallel operations (Stone, 1973).

4. Computed diagonal elements

For the linear system of Section 3 we have that $\epsilon \geq 0$ imply diagonal dominance such that we have no problem with ill-condition or growth of round-off errors. In fact it is easily seen that the computed diagonal elements, b_i ($i = 1, 2, \ldots, n$ or for the coupled algorithm: $i = 1, 2, \ldots, n \div 2$) will form a monotone (decreasing) sequence, converging (as n and $i \rightarrow \infty$) towards

$$\hat{b} = 1 + \frac{\epsilon}{2} + \sqrt{\epsilon + \epsilon^2/4}.$$

5. ALGOL procedures

```
procedure TRIDIAG (n, A, B, C, D);
 value n; integer n; array A, B, C, D;
begin integer i; real bi, di, z;
 bi := B[1]; di := D[1];
 for i := 2 step 1 until n do
 begin z := A[i]/bi;
  bi := B[i] := B[i] − C[i] × z;
  di := D[i] : = D[i] − di × z
 end;
 di := D[n] := D[n]/B[n];
 for i := n − 1 step − 1 until 1 do
  di := D[i] := (D[i] − C[i + 1] × di)/B[i]
end TRIDIAG;
procedure COUPLED (n, A, B, C, D);
 value n; integer n; array A, B, C, D;
begin integer i, j, m; real bi, di, dj, z; boolean odd;
 m := n ÷ 2; odd := m + m < n;
 bi := B[1]; di := D[1]; dj := D[n];
 for i := 2 step 1 until m do
 begin z := A[i]/bi;
  bi := B[i] := B[i] − C[i] × z;
  di := D[i] := D[i] − di × z;
  j := n − i + 1;
  dj := D[j] := D[j] − dj × z
 end;
 i := m + 1; z := A[i]/bi;
 if odd then bi := B[i];
 bi := bi − C[i] × z; di := D[i] − di × z;
 if odd then
 begin bi := bi − C[i] × z; di := di − dj × z end;
 di := D[i] := di/bi; bi := B[m];
 if odd then dj := D[n − m + 1] := (dj − C[i] × di)/bi
  else dj := di;
 di := D[m] := (D[m] − C[i] × di)/bi;
 for i := m − 1 step − 1 until 1 do
 begin bi := B[i];
  di := D[i] := (D[i] − C[i + 1] × di)/bi;
  j := n − i + 1;
  dj := D[j] := (D[j] − C[i + 1] × dj)/bi
 end
end COUPLED;
procedure ROSE (n, H, D);
 value n; integer n; array H, D;
begin integer i; real s, t;
 t := H[1]; s := D[1] × t;
 for i := 2 step 1 until n do
 begin t := H[i] + t;
       s := D[i] × t + s
 end;
 t := H[n + 1] + t; s := −s/t;
 for i := n step −1 until 1 do
  s := D[i] := D[i] + s;
 s := D[1] := D[1] × H[1];
 for i := 2 step 1 until n do
  s := D[i] := D[i] × H[i] + s
end ROSE;
```

References

ANDRES, T., HOSKINS, W. D., and McMASTER, G. E. (1974). Algorithm 84. A coupled algorithm for the solution of certain tridiagonal systems of linear equations, *The Computer Journal*, vol. 17, pp. 378-379.

EVANS, D. J. (1972). An algorithm for the solution of certain tridiagonal systems of linear equations, *The Computer Journal*, vol. 15, pp. 356-359.

EVANS, D. J. (1977). On the use of fast methods for solving boundary value problems, *The Computer Journal*, vol. 20, pp. 181-184.

LEAVENWORTH, B. (1960), Algorithm 24, *CACM*, vol. 3, p. 602.

ROSE, D. J. (1969). An algorithm for solving a special class of tridiagonal systems of linear equations, *CACM*, vol. 12, pp. 234-236.

SPRAGUE, C. F. III. (1960). Algorithm 17, *CACM*, vol. 3, p. 508.

STONE, H. S. (1973). An efficient parallel algorithm for the solution of a tridiagonal linear system of equations, *JACM*, vol. 20, pp. 27-38.