ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
XVII, 1 (1980)

ALGORITHMS 77-78

F. PANKOWSKI (Bydgoszcz)

UNIFORM APPROXIMATION OF EMPIRICAL FUNCTIONS BEING EITHER MONOTONE OR HAVING A FIXED NUMBER OF EXTREMA

1. Procedure declarations.

(a) The procedure *prodrevsimpl* solves the following problem:

For a given number sequence $\{y_1, y_2, \ldots, y_p\}$ determine a sequence $\{z_1, z_2, \ldots, z_p\}$ with the following properties:

(i) the function

$$f = \max_{1 \leqslant k \leqslant p} |y_k - z_k|$$

reaches its minimum;

(ii) the sequence $\{z_i\}$ is monotone.

Data:

ext – integer variable with value 0 if the sequence $\{z_i\}$ is non-increasing, and with value 1 if the sequence $\{z_i\}$ is non-decreasing;

p – number of elements in the sequence $\{y_i\}$;

$d[1:3\times p]$ – data array, where $d[1] = y_1$, $d[2] = y_2, \ldots, d[p] = y_p$.

Results:

$d[1:3\times p]$ – array of results, where $z_1 = d[1]$, $z_2 = d[2], \ldots, z_p = d[p]$, and $d[2\times p]$ contains the approximation error.

(b) The procedure *prodrevsimplextr* solves problem (a) with (ii) changed to

(iii) the sequence $\{z_i\}$ has a fixed number of extrema.

Data:

lex – number of extrema in the sequence $\{z_i\}$;

ext – integer variable with value 0 if the first extremum is a minimum, and with value 1 if it is a maximum.

All remaining data and result parameters are the same as in (a).

2. Method used. The algorithms used in both procedures are derived from classical linear programming methods. They make use of special features of the considered problem; hence

1° the procedures require a small amount of computer memory, thus long sequences can be hand'ed by them;

2° the computation time is rather small and the cost of computation is in reasonable limits.

The problem which is solved by the procedure *prodrevsimpl* can be formulated as follows:

Given $y_1, y_2, \ldots, y_p$, minimize

$$\max_{1\leqslant k\leqslant p} |y_k - z_k| \tag{1}$$

to satisfy

$$z_1 \leqslant z_2 \leqslant \ldots \leqslant z_p. \tag{2}$$

The problem can be written in the following equivalent form (see [4], p. 244-245):

Minimize

$$g = z_{p+1} \tag{3}$$

under the conditions

$$\begin{gathered} y_k - z_k + z_{p+1} \geqslant 0, \qquad -(y_k - z_k) + z_{p+1} \geqslant 0 \qquad (k = 1, 2, \ldots, p), \\ z_1 \leqslant z_2 \leqslant \ldots \leqslant z_p. \end{gathered} \tag{4}$$

Problem (3)-(4) is a linear programming problem. Conditions (4) are enlarged by

$$z_k \geqslant 0 \qquad (k = 1, 2, \ldots, p). \tag{5}$$

Conditions (5) depend upon the sign of the data $y_1, y_2, \ldots, y_p$. If there are negative numbers in the data or if the difference between the maximum and minimum values in the sequence $y_1, y_2, \ldots, y_p$ is not less than the minimum value of this sequence, then, in order to fulfill (5), a sufficiently large positive number r should be added to all elements of the sequence $y_1, y_2, \ldots, y_p$; it is subtracted after the calculations from all $z_1, z_2, \ldots, z_p$.

To obtain the initial basic program we introduce artificial variables into (4) and (5) as well as the function $g = z_{3p}$ (where $z_{3p} = z_{p+1}$) as the equation with index $3p$, and also we assume that the quantities y_k for $k = 1, 2, \ldots, p$ have been already, if necessary, increased by r.

```
procedure prodrevsimpl(ext,p,d);
 value p;
 integer ext,p;
 array d;
 begin
  integer f,g,h,i,j,k,k1,l,l1,l2,m,n;
  Boolean fg,fl,fn;
  fg:=ext=1;
  n:=3×p;
  h:=n-1;
  f:=p+p;
  begin
   real r,t,x,y,y1,y2;
   integer array q[1:p+1],tb,tn,t1[1:p],s[1:n],sz[5:h];
   array b[1:n],dd[1:p];
   t:=r:=d[f-1]:=d[p];
   d[f]:=-t;
   j:=g:=0;
   y:=if fg then -1.0 else 1.0;
   for i:=f-3 step -2 until 1 do
    begin
     j:=j+1;
     x:=d[i]:=d[p-j];
     d[i+1]:=-x;
     if t<x
      then t:=x
      else
       if x<r
        then r:=x;
     tb[j]:=tn[j]:=j
```

```
 end i;
tn[p]:=tb[p]:=p;
if r>.0
 then
 begin
  x:=t-r;
  r:=if x<r then .0 else x+1.0
 end r>.0
 else
 begin
  t:=t-r;
  r:=abs(r)+t+1.0
 end r≤.0;
for i:=1 step 2 until f do
 begin
  d[i]:=d[i]+r;
  d[i+1]:=d[i+1]-r
 end i;
j:=1;
for i:=f+1 step 1 until h do
 begin
  sz[i]:=if fg then j+1 else j;
  t1[j]:=i;
  j:=j+1
 end i;
t1[p]:=n;
j:=p+1;
for i:=1 step 1 until n do
 begin
  b[i]:=if i≤f then d[i] else .0;
```

```
      s[i]:=i+j
     end i;
    l1:=p;
iter:
    if g≠l1
     then
     begin
      t:=b[2];
      m:=2;
      j:=4;
et20:for i:=j step 2 until f do
       begin
        x:=b[i];
        if x<t
         then
         begin
          t:=x;
          m:=i
         end x<t
       end i
     end g≠l1
     else
     begin
      t:=b[1];
      m:=1;
      j:=3;
      go to et20
     end g=l1;
    if t<.0
     then
```

```
    begin
iter1:
      if f≥m
       then
       begin
        j=m+2;
        k=if j=.5×m then j else l1+1
       end f≥m
       else k=sz[m];
      s[m]=k;
      g=g+1;
      q[g]=m;
      y1=y2=.0;
      t=if k≤l1 then -t else -.5×t;
      if k=l1+1
       then
       begin
        b[h+1]=-t;
        b[m]=t;
        j=m-1;
        for i=1 step 1 until j,m+1 step 1 until h do
         b[i]=b[i]-t×(if i<f∧i+2≠.5×i then -2.0 else if s[i]≤l1
              then 1.0 else .0);
        go to et1
       end k=l1+1
       else
       begin
        j=k+k;
        b[j]=b[j]+t;
        b[j-1]=b[j-1]-t;
```

```
x:=t×y;
if k=1
 then
 begin
  if tn[1]=0
   then
   begin
    y1:=b[f+1]:=b[f+1]-x;
    y2:=b[f+2]:=b[f+2]+x;
    j:=if y1<.0 then f+1 else if y2<.0 then f+2 else j
   end
   else
   begin
    y1:=b[f+1]:=b[f+1]+x;
    if y1<.0
     then j:=f+1
   end
 end k=1
 else
  if k=11
   then
   begin
    if tn[p]=0
     then
     begin
      y1:=b[h-1]:=b[h-1]-x;
      y2:=b[h]:=b[h]+x;
      j:=if y1<.0 then h-1 else if y2<.0 then h else j
     end
     else
```

```
          begin
           y1:=b[h]:=b[h]-x;
           if y1<.0
            then j:=h
          end
        end
        else
        begin
         j:=t1[k];
         y1:=b[j]:=b[j]+x;
         y2:=b[j-1]:=b[j-1]-x;
         j:=if y1<.0 then j else if y2<.0 then j-1 else j
        end;
    b[m]:=t
   end k≠l1+1;
  if y1<.0
   then
   begin
    t:=y1;
    m:=j;
    go to iter1
   end y1<.0
   else
    if y2<.0
     then
     begin
      t:=y2;
      m:=j;
      go to iter1
     end;
```

```
           go to iter
          end t<.0
          else
et1:  if b[h+1]≠.0
            then
            begin
             fn:=true;
             h:=.5×f;
             j:=q[g];
             for i:=g-1 step -1 until 1 do
              begin
               m:=q[i];
               k:=s[m];
               fl:=true;
               l1:=t1[k];
               k1:=k+k-1;
               if ¬fg
                then l1:=l1-1;
               if k≠1∧k≠h
                then
                begin
                 if j=k1∨j=l1
                  then
                  begin
et2:               j:=m;
                   fl:=false
                  end
                end
                else
                 if k=1∧(j=k1∨j=l1)
```

```
            then go to et2
            else
             if k=h∧(j=k1∨j=l1)
              then go to et2;
        if fl
         then
         else
          if fn
           then
           begin
            l:=l2:=m;
            fn:=false
           end
           else l2:=m
       end i;
      x:=b[l];
      l:=s[l];
      l2:=s[l2];
      if l>l2
       then
       begin
        i:=l;
        l:=l2;
        l2:=i
       end l>l2;
      k1:=.5×f;
      for i:=l2+1 step 1 until k1 do
       begin
        j:=tb[i];
        if tn[j]=0
```

```
                then go to et6
                else
                 if fg
                  then
                  begin
                   if d[j+j-1]<x
                    then l2:=i
                  end fg
                  else
                   if d[j+j-1]>x
                    then l2:=i
              end i;
et6:      for i:=l-1 step -1 until 1 do
              begin
               j:=tb[i];
               if tn[j]=0
                then go to et7
                else
                 if fg
                  then
                  begin
                   if d[j+j-1]>x
                    then l:=i
                  end fg
                  else
                   if d[j+j-1]<x
                    then l:=i
              end i;
et7:      for i:=l step 1 until l2 do
              begin
```

```
    j:=tb[i];
    dd[j]:=x;
    tn[j]:=0
   end i;
  j:=1;
  k:=12;
  if l≤2
   then l:=1
   else
    if tb[l]≠tb[l-1]+1
     then
     else
      if tb[l]≠tb[l-2]+2
       then l:=l-1;
  if l2≥k1-1
   then l2:=k1
   else
    if tb[l2]≠tb[l2+1]-1
     then
     else
      if tb[l2]≠tb[l2+2]-2
       then l2:=l2+1;
  if j≠1
   then
   begin
    i:=tb[l];
    dd[i]:=-d[i+i];
    tn[i]:=0
   end j≠1;
  if k≠12
```

```
 then
 begin
  i:=tb[12];
  dd[i]:=-d[i+i];
  tn[i]:=0
 end k≠12;
h:=f:=f-2×(12-l+1);
l1:=.5×f;
j:=1;
if k1≠12
 then
 for i:=1 step 1 until l1 do
  begin
   tb[i]:=tb[12+j];
   j:=j+1
  end k1≠12,i;
k:=0;
for i:=1 step 1 until p do
 if tn[i]≠0
  then
  begin
   k:=k+2;
   b[k-1]:=d[i+i-1];
   b[k]:=d[i+i]
  end tn[i]≠0;
l:=k:=j:=0;
m:=tn[1];
l2:=f+1;
fn:=¬fg∧m=0∨fg∧m≠0;
fl:=fg∧m≠0;
```

```
for i:=1 step 1 until p do
 if tn[i]=0
  then j:=j+1
  else
   if j≠0
    then
    begin
     j:=0;
     k:=k+1;
     h:=h+2;
     g:=f+k;
     l:=l+1;
     t1[l]:=if fn≡fg then g else g+1;
     if ¬fn
      then sz[g]:=l
      else
       if fg
        then sz[g]:=l+1
        else sz[g+1]:=l;
     if m=0∧k=1
      then b[f+1]:=y×dd[i-1]
      else
      begin
       x:=y×dd[i-1];
       if m≠0
        then g:=g-1;
       b[g]:=x;
       b[g-1]:=-x
      end;
     k:=k+1
```

```
                end
                else
                 if k=0
                  then
                  begin
                   k:=k+1;
                   go to et8
                  end
                  else
                  begin
et8:               g:=f+k;
                   if m≠0∧g≠12
                    then b[g-1]:=.0
                    else b[g]:=.0;
                   l:=l+1;
                   t1[l]:=if fn≡fg then g else g+1;
                   if i+1>p
                    then
                    else
                     if tn[i+1]=0∧fl
                      then g:=g+1;
                   if ¬fn
                    then sz[g]:=l
                    else
                     if fg
                      then sz[g]:=l+1
                      else sz[g+1]:=l;
                   h:=h+1;
                   k:=k+1
                  end i;
```

```
                if j=p
                 then go to et9;
                if tn[p]=0
                 then b[h]:=-y×dd[i-2]
                 else h:=h-1;
                b[h+1]:=.0;
                j:=l1+1;
                k:=h+1;
                for i:=1 step 1 until k do
                 s[i]:=i+j;
                g:=0;
                go to iter
               end b[h+1]≠.0;
            for i:=1 step 1 until l1 do
             dd[tb[i]]:=b[i+i];
         et9:
            for i:=1 step 1 until p do
             d[i]:=dd[i]-r;
            d[p+p]:=-b[n]
           end block
          end prodrevsimpl
```

Problem (3)-(5) can be written as follows:
Minimize

(6)
$$z_{3p}$$

under the conditions

(7)
$$\begin{aligned}
z_k - z_{p+1} + z_{2k-1}^d &= y_k \\
-z_k - z_{p+1} + z_{2k}^d &= -y_k
\end{aligned} \qquad (k = 1, 2, \ldots, p),$$
$$\begin{aligned}
z_1 - z_2 + z_{2p+1}^d &= 0, \\
z_2 - z_3 + z_{2p+2}^d &= 0, \\
\ldots\ldots\ldots\ldots & \\
z_{p-1} - z_p + z_{3p-1}^d &= 0, \\
-z_{p+1} + z_{3p} &= 0,
\end{aligned}$$
$$z_k \geqslant 0 \quad (k = 1, 2, \ldots, p), \qquad z_l^d \geqslant 0 \quad (l = 1, 2, \ldots, p).$$

This problem can be solved by using the dual simplex algorithm ([3], § 6.4) and the revised simplex algorithm with product form of the inverse matrix ([1], [3]). These algorithms do not, however, take advantage of the characteristic features of this problem, therefore computer time is large and there may be lack of memory for greater input sequences.

Let us take the coefficients from (7) and form the following matrix of dimension $3p \times (p+1)$:

$$\bar{A} = \begin{bmatrix} 1 & 0 & 0 & \dots & & & 0 & -1 \\ -1 & 0 & 0 & \dots & & & 0 & -1 \\ 0 & 1 & 0 & \dots & & & 0 & -1 \\ 0 & -1 & 0 & \dots & & & 0 & -1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \dots & & & 1 & -1 \\ 0 & 0 & 0 & \dots & & & -1 & -1 \\ 1 & -1 & 0 & \dots & & & 0 & 0 \\ 0 & 1 & -1 & \dots & & & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \dots & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & -1 \end{bmatrix}.$$

Denote by U the unit matrix of dimension $3p \times 3p$. The first $3p-1$ rows and columns of U form the initial basis B. Notice that the basic solution

$$d = (d_1, d_2, \dots, d_{3p-1}, d_{3p}) = (y_1, -y_1, y_2, -y_2, \dots, y_p, -y_p, 0, \dots, 0),$$

where $y_k > 0$ $(k = 1, 2, \dots, p)$, has negative components, but is a dual feasible initial solution.

Using the revised simplex algorithm, we calculate

$$\gamma_j = U_{3p}\bar{A}_j \quad (j = 1, 2, \dots, p+1), \tag{8}$$

where U_{3p} denotes the last row vector of U and $\bar{A}_j$ is the j-th column vector of $\bar{A}$. Since $\gamma_j \leqslant 0$ $(j = 1, 2, \dots, p+1)$, the optimality conditions are satisfied. Thus, using the dual and revised simplex algorithms, the problem can be solved.

Now, let us consider the structure of the matrix $\bar{A}$. Notice that the non-zero elements of $\bar{A}$ can be calculated from the following equations which follow directly from (7):

$$a'_{ik} = \begin{cases} 1 & (i = 2k-1, 2p+k), \\ -1 & (i = 2k, \ 2p+k-1), \end{cases} \tag{9}$$

where k is not equal to 1, p and $p+1$, and

(10) $$a_{i1} = \begin{cases} 1 & (i = 1,\ 2p+1), \\ -1 & (i = 2), \end{cases}$$

(11) $$a_{ip} = \begin{cases} 1 & (i = 2p-1), \\ -1 & (i = 2p,\ 3p-1), \end{cases}$$

(12) $$a_{i,p+1} = -1 \quad (i = 1, 2, \ldots, 2p, 3p).$$

There exists thus a possibility of forming the appropriate column vector of $\bar{A}$ during the calculations. This is the first group of simplifications. The second simplification group follows from an analysis of the location of the non-zero elements in $\bar{A}$. Additional simplifications are possible by the use of the product form of the inverse B'^{-1} of the newly formed basis B'.

Using the dual exit criterion

(13) $$d_l = \min_{d_i<0} \{d_i\},$$

we make the following

Agreement. If the minimum is reached for more than one d_i, choose that one whose index l is the greatest one.

It is known that the inverse matrix in the t-th iteration can be obtained from the formula

$$J_t^l J_{t-1}^l \ldots J_1^l J_0 = B_t^{-1},$$

where $J_0 = I = B^{-1}$ and J_t^l is the elementary matrix of the t-th iteration, i.e. the matrix which is obtained from the unit matrix by replacing its column l with the column V_l; the elements of V_l are determined as

(14) $$v_{il} = -\frac{w_{ik}}{w_{lk}} \quad (i \neq l), \qquad v_{ll} = \frac{1}{w_{lk}},$$

the vector W_k being calculated from

(15) $$W_k = J_t^l \big(J_{t-1}^l \ldots (J_1^l \bar{A}_k) \ldots\big),$$

where the index l indicates the column of the previous basis which is now being replaced.

Multiplication from the right of $\bar{A}_j = (a_{1j}, a_{2j}, \ldots, a_{3p,j})$ by the elementary matrix J_t^l leads to the vector W_j with components

(16) $$w_{ij} = a_{ij} + v_{il} a_{lj} \ (i \neq l), \qquad w_{lj} = v_{ll} a_{lj}.$$

Let us perform now a detailed analysis of the behaviour of the vector W_k during the iteration process. As is easily seen, in the first iteration we have $W_k = \bar{A}_k$. Using (16), let us calculate W_k' in the second iteration:

$W'_k = J_1^l \bar{A}_{k'}$. The vector $\bar{A}_{p+1}$ enters the basis in the last iteration (see [2], Theorem 3.4). From the first iteration we obtain the relation $w_{lk} = a_{lk} = -1$, thus $v_{ll} = 1/w_{lk} = -1$, where $l \leqslant 2p$ (l is even) and $k = l/2$. From (14) we know that the vector V_l for the matrix J_1^l is equal to the vector $\bar{A}_k$.

Now, calculating the vector $W_{k'}$ let us notice that $a_{lj'} = 0$ in the vector $\bar{A}_{k'}$. This is due to the fact that the vector which leaves the basis in the first iteration has an index $l \leqslant 2p$ and in the rows from 1 to $2p$, with the exception of the row $p+1$, there is only one non-zero element. Therefore, by (16) we have the identity

(17) $$W_{k'} = \bar{A}_{k'},$$

and since the pivotal element $a_{lk'}$ equals -1, due to (14) for the matrix J_2^l, we obtain

(18) $$V_l = W_{k'} = \bar{A}_{k'}.$$

From (17) we get

(19) $$J_1^l \bar{A}_j = \bar{A}_j \quad (j = 1, 2, \ldots, p).$$

Similarly, for the matrix J_2^l we have

(20) $$J_2^l \bar{A}_j = \bar{A}_j \quad (j = 1, 2, \ldots, p).$$

This follows from (16) in which

(21) $$a_{lj} = 0 \quad (j = 1, 2, \ldots, p),$$

where the index l corresponds to the matrix J_2^l.

To prove (21), consider two possibilities:

(a) $l \leqslant 2p$,

(b) $l > 2p$.

In case (a) the derivation of (21) is the same as for (17).

Consider now case (b). Analyzing the formulas for the calculation of the vector d notice that a negative element of this vector can appear for an index $l > 2p$ if the vector entering the basis in the first iteration has 1 at the l-th place. In the row l we have one more non-zero element equal to -1. The vector $W_{k'}$ which corresponds to this element is already in the matrix J_2^l.

In fact, due to (13) and the Agreement, in the second iteration the basis is leaving a vector with index $l > 2p$ if the vector which entered the basis in the first iteration has the l-th element equal to 1. It follows from (19) that in the second iteration the candidate for the basis has to be searched among the secondary vectors of $\bar{A}$.

As we know, in the row l there is still one non-zero element equal to -1 and the vector $W_{k'} = \bar{A}_{k'}$ whose l-th element is equal to -1 will enter the basis in this iteration; from (18) we conclude that this will be the vector V_l of the matrix J_2^l. Thus relation (21) must hold, and then (20) follows from (16).

Hence we have the following general statement:

THEOREM 1. *In a given iteration t the product of the actual inverse basis B^{-1} and any (but not the last) vector of $\bar{A}$ is equal to this vector, i.e.* $B^{-1}\bar{A}_j = \bar{A}_j$ $(j = 1, 2, \ldots, p)$.

From (15), Theorem 1 and the fact that the pivotal element a_{lk} of the vector $\bar{A}_k$ equals -1 we obtain

COROLLARY 1. *In any iteration (with the exception of the last one) we have* $W_k = \bar{A}_k$, *where* $\bar{A}_k$ *is the vector entering the basis in this iteration.*

To calculate the index of the vector which has to enter the basis in a given iteration, we must know, in accordance with the dual simplex algorithm, the quantity γ_j which can be calculated from (8). The actual vector U_{3p} can be evaluated from

$$\big(\ldots(e_{3p}J_t^l)J_{t-1}^l \ldots\big)J_1^l = U_{3p},$$

where $e_{3p} = (0, 0, \ldots, 0, 1)$ is a vector with $3p$ components. The elements of U_{3p} can be determined by the relation

$$u_i = e_i \ (i \neq l), \quad u_l = e_i v_{il}. \tag{22}$$

THEOREM 2. *In any iteration the vector U_{3p} is a unit vector, i.e.*

$$U_{3p} = (0, 0, \ldots, 0, 1).$$

Proof. In the matrix $\bar{A}$ the row with index $3p$ has only one non-zero element equal to -1 which lies at the place $p-1$. We know from Corollary 1 that for the matrix J_1^l we have $V_l = \bar{A}_j$ $(j = 1, 2, \ldots, p)$. The product $e_{3p}J_1^l$ gives the unit vector U_{3p} because $v_{3p,l} = a_{3p,l} = 0$, thus $u_l = 0$ in (22).

Therefore, from the identity $\gamma_j = U_{3p}\bar{A}_j$ $(j = 1, 2, \ldots, p+1)$ we get

$$\gamma_j = \begin{cases} 0 & (j = 1, 2, \ldots, p), \\ -1 & (j = p+1). \end{cases}$$

Thus in the second iteration the basis is entered by one of the vectors $\bar{A}_j$ $(j = 1, 2, \ldots, p)$ which follows directly from the dual simplex algorithm.

The next product $(e_{3p}J_2^l)J_1^l = U_{3p}$ is also a unit vector, since for the matrix J_2^l we have $v_{3p,l} = a_{3p,l} = 0$ (this holds also for the matrix J_1^l) and by (22) we know that the vector U_{3p} will have only one non-zero element $u_{3p} = 1$.

This reasoning can be repeated for all subsequent matrices J_t^l.

By Theorem 2 and by (8) we come to the following

COROLLARY 2. *In any iteration the quantities* γ_j *can be calculated by the relations* $\gamma_j = 0$ $(j = 1, 2, \ldots, p)$ *and* $\gamma_{p+1} = -1$.

THEOREM 3. *In any iteration the index* k *of the vector entering the basis can be determined by the relation*

$$k = \begin{cases} l/2 & \text{for } l \leqslant 2p \text{ and } l \text{ even}, \\ p+1 & \text{for } l < 2p \text{ and } l \text{ odd}, \\ l-2p+1 & \text{for } l > 2p, \end{cases}$$

where l *is the index of the vector which leaves the basis.*

Proof. In the case of l even and $l \leqslant 2p$ the relation $k = l/2$ follows from (9)-(11) and so does the relation $k = l-2p+1$ for $l > 2p$. If l is odd and $l < 2p$, the relation $k = p+1$ follows from (12).

In the last iteration the vector W_k can also be determined by simple relations. We have the following

THEOREM 4. *The index* k *of the vector* W_k *from the last iteration* $(k = p+1)$ *can be determined as follows:*

$$(23)\qquad k = \begin{cases} -2 & \text{for } i = 1, 3, \ldots, 2p-1, \\ 1 & \text{for } i \in I, \text{ where } I \text{ is the index set of the vectors} \\ & \qquad\qquad \text{leaving the basis in the given iteration}, \\ -1 & \text{for } i = 3p, \\ 0 & \text{for all remaining } i. \end{cases}$$

The proof of this theorem can be found in [2] (Theorem 3.6).

The actual vector d' can be calculated by the formulas

$$d'_i = d_i - \frac{d_l}{w_{lk}} w_{ik} \quad (i \neq l), \qquad d'_l = \frac{d_l}{w_{lk}},$$

where $w_{lk} = -1$ in all but the last iterations, and in the last iteration $w_{lk} = -2$.

The non-zero elements w_{ik} are equal to 1 or -1 with the exception of the last iteration in which an element equal to -2 appears.

Hence the values of the vector d' in all iterations can be determined as

$$d'_i = d_i + \delta d_l \ (i \neq l), \qquad d'_l = \delta_1 d_l,$$

where

$$\delta = \begin{cases} 0 & \text{for } w_{ik} = 0, \\ -1 & \text{for } w_{ik} = -1 \text{ not in the iteration } p+1 \text{ and for } w_{ik} = -2, \\ 1 & \text{for } w_{ik} = 1 \text{ in the iterations } t = 1, 2, \ldots, p, \\ .5 & \text{for } w_{ik} = 1 \text{ in the iteration } p+1, \\ -.5 & \text{for } w_{ik} = -1 \text{ in the iteration } p+1, \end{cases}$$

$$\delta_1 = \begin{cases} -.5 & \text{in the iteration } p+1, \\ -1 & \text{in the iterations } t = 1, 2, \ldots, p. \end{cases}$$

As follows from Corollary 1, formulas (16) can be replaced by (9)-(11), and in the last iteration — by (23).

In the algorithm realized by the procedure *prodrevsimplextr* the results of problem (1)-(2) are used, i.e. one has to remember the optimum solution of problem (1)-(2), the index set of the vectors leaving the basis in consecutive iterations and the index set of the vectors entering the basis in consecutive iterations.

The procedures take also care of the case where there is more than one optimum solution. For instance, consider the following problem: Minimize

$$\max_{1\leqslant k\leqslant 5} |y_k - z_k|$$

provided $z_1 \leqslant z_2 \leqslant z_3 \leqslant z_4 \leqslant z_5$, where $\{y_1, y_2, y_3, y_4, y_5\} = \{3, 5, 7, 6, 8\}$. The error of the optimum approximation for this problem is equal to .5, and the optimum solution is of the form

$$\{z_1, z_2, z_3, z_4, z_5\} = \{2.5, 4.5, 6.5, 6.5, 7.5\}.$$

It seems to be natural to assume $\{3, 5, 6.5, 6.5, 8\}$ as the optimum solution of this problem. This follows from the fact that we change only those values which violate the monotonicity of the function, leaving the other values unchanged. The procedure gives such a type of solution.

3. Certification. Table 1 gives the calculation times for some examples, the calculations having been carried out on the Odra 1204 computer. As is seen, the calculation times depend not only on the number of data, but also on their values.

TABLE 1

Number of data points	Number of extrema	Calculation time (in secs.)	
		prodrevsimpl	*prodrevsimplextr*
25	5	7	28
50	3	22	53
95	3	120	184
95	3	150	221
95	3	70	132

Table 2 shows the calculation results for an example and with the use of the procedure *prodrevsimplextr*. As data, sine function values with arguments $.1k$ ($k = 0, 1, \ldots, 94$) are used, perturbed by the pseudorandom number equidistributed in the interval $(-.05, .05)$. The optimum approximation error equals .025.

TABLE 2

k	Data	Results	k	Data	Results
0	.030	.030	48	−1.039	−1.039
1	.120	.120	49	−1.008	−1.008
2	.218	.218	50	−.933	−.933
3	.251	.251	51	−.882	−.888
4	.412	.412	52	−.895	−.888
5	.495	.495	53	−.833	−.833
6	.565	.565	54	−.740	−.740
7	.655	.655	55	−.728	−.728
8	.710	.710	56	−.661	−.661
9	.773	.773	57	−.530	−.530
10	.812	.812	58	−.419	−.419
11	.909	.906	59	−.363	−.363
12	.902	.906	60	−.326	−.326
13	.933	.933	61	−.210	−.210
14	1.005	.989	62	−.090	−.090
15	.974	.989	63	−.011	−.011
16	1.034	1.034	64	.166	.166
17	.943	.968	65	.166	.166
18	.993	.968	66	.334	.334
19	.970	.968	67	.421	.421
20	.954	.954	68	.481	.481
21	.885	.885	69	.606	.606
22	.759	.765	70	.685	.685
23	.771	.765	71	.740	.740
24	.713	.713	72	.804	.804
25	.622	.622	73	.868	.868
26	.501	.501	74	.930	.925
27	.441	.441	75	.920	.925
28	.320	.320	76	1.002	.990
29	.205	.205	77	.978	.990
30	.148	.148	78	1.004	1.002
31	.025	.025	79	1.000	1.002
32	−.065	−.065	80	1.038	1.038
33	−.161	−.161	81	1.011	1.011
34	−.258	−.258	82	.898	.913
35	−.326	−.326	83	.929	.913
36	−.481	−.481	84	.826	.826
37	−.514	−.514	85	.803	.803
38	−.653	−.653	86	.687	.687
39	−.664	−.664	87	.659	.659
40	−.777	−.777	88	.634	.634
41	−.782	−.782	89	.519	.519
42	−.876	−.876	90	.407	.407
43	−.924	−.924	91	.284	.284
44	−.969	−.963	92	.236	.236
45	−.970	−.963	93	.164	.164
46	−.956	−.963	94	−.019	−.019
47	−1.034	−1.034			

```
procedure prodrevsimplextr(lex,ext,p,d);
 value lex,p;
 integer lex,ext,p;
 array d;
 begin
  integer f,g,h,i,j,j1,j2,k,k1,k2,l,l1,l2,m,n;
  Boolean fg,fl,fk,fn,fj;
  n:=3×p;
  h:=n-1;
  f:=p+p;
  begin
   real r,t,x,y,y1,y2;
   integer array q[1:p+1],q1,tb,tn,t1[1:p],s,sn[1:n],s1[0:f-1];
   array b,bb[1:n];
   t:=r:=d[f-1]:=d[p];
   d[f]:=-t;
   l:=lex+2;
   fl:=l≠.5×lex;
   lex:=if fl then l+1 else l;
   fk:=false;
   fj:=ext=1;
   l1:=p;
   k2:=j:=0;
   y:=if fj then 1.0 else -1.0;
   for i:=1 step 1 until p do
    begin
     q1[i]:=0;
     t1[i]:=f+i;
     tn[i]:=tb[i]:=i
    end i;
```

```
s1[0]:=s1[f-1]:=y;
for i:=f-3 step -2 until 1 do
 begin
  s1[i]:=y;
  s1[i+1]:=-y;
  j:=j+1;
  x:=d[i]:=d[p-j];
  d[i+1]:=-x;
  if t<x
   then t:=x
   else
    if x<r
     then r:=x
 end i;
if r>.0
 then
 begin
  x:=t-r;
  r:=if x<r then .0 else x+1.0
 end r>.0
 else
 begin
  t:=t-r;
  r:=abs(r)+t+1.0
 end r≤.0;
for i:=1 step 2 until f do
 begin
  d[i]:=d[i]+r;
  d[i+1]:=d[i+1]-r
 end i;
```

```
extr:
   j:=p+1;
   for i:=1 step 1 until n do
    begin
     b[i]:=if i≤f then d[i] else .0;
     s[i]:=i+j
    end i;
   g:=0;
iter:
   if g≠11
    then
    begin
     t:=b[2];
     m:=2;
     j:=4;
et20:for i:=j step 2 until f do
       begin
        x:=b[i];
        if x<t
         then
         begin
          t:=x;
          m:=i
         end x<t
       end i
    end g≠11
    else
    begin
     t:=b[1];
     m:=1;
```

```
       j:=3;
       go to et20
      end g=11;
     if t<.0
      then
      begin
iter1:
       if f≥m
        then
        begin
         j:=m+2;
         k:=if j=.5×m then j else l1+1
        end f≥m
        else
         if fk
          then
          begin
           k:=s[q[g]];
           j:=tb[k];
           k:=if s1[j+j-1]=1 then k+1 else k-1
          end
          else
          begin
           k:=m-f;
           if k=s[q[g]]
            then k:=k+1
          end;
      s[m]:=k;
      g:=g+1;
      q[g]:=m;
```

```
y1:=y2:=.0;
t:=if k≤l1 then -t else -.5×t;
if k=l1+1
 then
 begin
  b[h+1]:=-t;
  b[m]:=t;
  j:=m-1;
  for i:=1 step 1 until j,m+1 step 1 until h do
   b[i]:=b[i]-t×(if i<f∧i+2↓.5×i then -2.0 else if s[i]≤l1
        then 1.0 else .0);
  go to et1
 end k=l1+1
 else
 begin
  l:=k+k;
  b[l]:=b[l]+t;
  b[l-1]:=b[l-1]-t;
  j:=2×tb[k];
  x:=t×s1[j-1];
  y:=t×s1[j-2];
  if k=1
   then
   begin
    if tn[1]=0
     then
     begin
      y1:=b[f+1]:=b[f+1]-y;
      y2:=b[f+2]:=b[f+2]-x;
      j:=if y1<.0 then f+1 else if y2<.0 then f+2 else j
```

```
    and tn[1]=0
    else
    begin
     y1=b[f+1]=b[f+1]-x;
     if y1<.0
      then j=f+1
    end
  end k=1
  else
   if k=11
    then
    begin
     if tn[p]=0
      then
      begin
       y1=b[h-1]=b[h-1]-y;
       y2=b[h]=b[h]-x;
       j=if y1<.0 then h-1 else if y2<.0 then h else j
      end tn[p]=0
      else
      begin
       y1=b[h]=b[h]-y;
       if y1<.0
        then j=h
      end
    end
    else
    begin
     k=t1[k];
     y1=b[k]=b[k]-x;
```

```
             y2:=b[k-1]:=b[k-1]-y;
             j:=if y1<.0 then k else if y2<.0 then k-1 else j
            end;
          b[m]:=t;
         end k⇂l1+1;
        if y1<.0
         then
         begin
          t:=y1;
          m:=j;
          go to iter1
         end y1<.0
         else
          if y2<.0
           then
           begin
            t:=y2;
            m:=j;
            go to iter1
           end;
        go to iter
       end t<.0
       else
et1: if b[h+1]⇂.0∧(k2⇂lex∨fk)
         then
         begin
          fn:=true;
          j2:=.5×f;
          j:=q[g];
          for i:=g-1 step -1 until 1 do
```

```
        begin
         m:=q[i];
         k:=s[m];
         fg:=true;
         g:=t1[k];
         k1:=k+k-1;
         j1:=tb[k];
         if j1≠1∧(s1[j1+j1-1]=-1∨k=l1)
          then g:=g-1;
         if k≠1∧k≠j2
          then
          begin
           if j=k1∨j=g
            then
            begin
et5:         j:=m;
             fg:=false
            end
          end
          else
           if k=1∧(j=k1∨j=g)
            then go to et5
            else
             if k=j2∧(j=k1∨j=g)
              then go to et5;
         if fg
          then
          else
           if fn
            then
```

```
                begin
                 l:=m;
                 fn:=false
                end
                else l2:=m
            end i;
           x:=b[l];
           l:=s[l];
           l2:=s[l2];
           if l>l2
            then
            begin
             i:=l;
             l:=l2;
             l2:=i
            end l>l2;
           if fk
            then
            begin
             k1:=.5×f;
             j1:=tb[l];
             fn:=s1[j1+j1-1]=1;
             fl:=false;
             for i:=l2+1 step 1 until k1 do
              begin
               j:=tb[i];
               if tn[j]=0Vfl
                then go to et10
                else
                 if j=q1[j]
```

```
                then
                begin
                 fl:=true;
et11:            if fn
                  then
                  begin
                   if d[j+j-1]<x
                    then l2:=i
                  end
                  else
                   if d[j+j-1]>x
                    then l2:=i
                end
                else go to et11
            end i;
et10:      fl:=false;
           for i:=l-1 step -1 until 1 do
            begin
             j:=tb[i];
             if tn[j]=0Vfl
              then go to et12
              else
               if j=q1[j]
                then
                begin
                 fl:=true;
et13:            if fn
                  then
                  begin
                   if d[j+j-1]>x
```

```
                    then l:=i
                  end
                  else
                   if d[j+j-1]<x
                    then l:=i
                end
                else go to et13
            end i;
et12:     for i:=1 step 1 until 12 do
            begin
             j:=tb[i];
             bb[j]:=x;
             tn[j]:=0
            end i;
          h:=f:=f-2×(12-l+1);
          l1:=.5×f;
          j:=1;
          if k1≠12
           then
           for i:=1 step 1 until l1 do
             begin
              tb[i]:=tb[12+j];
              j:=j+1
             end k1≠12,i;
          k:=0;
          for i:=1 step 1 until p do
           if tn[i]≠0
            then
            begin
             k:=k+2;
```

```
    b[k-1]:=d[i+i-1];
    b[k]:=d[i+i]
  end tn[i]≠0;
l:=k:=j:=0;
m:=tn[1];
fn:=m≠0;
k1:=f+1;
for i:=1 step 1 until p do
 if tn[i]=0
  then j:=j+1
  else
   if j≠0
    then
    begin
     j:=0;
     k:=k+1;
     h:=h+2;
     g:=f+k;
     l:=l+1;
     t1[l]:=if fn then g else g+1;
     if m=0∧k=1
      then b[k1]:=s1[i+i-2]×bb[i-1]
      else
      begin
       if fn
        then g:=g-1;
       k2:=i+i;
       t:=bb[i-1];
       b[g]:=s1[k2-4]×t;
       b[g-1]:=s1[k2-3]×t
```

```
                    end;
                   k:=k+1
                  end
                  else
                   if k=0
                    then
                    begin
                     k:=k+1;
                     go to et8
                    end
                    else
                    begin
et8:                 g:=f+k;
                     l:=l+1;
                     if fn∧g≠k1
                      then b[g-1]:=.0
                      else b[g]:=.0;
                     t1[l]:=if fn then g else g+1;
                     h:=h+1;
                     k:=k+1
                    end i;
              if j=p
               then go to et9;
              if tn[p]=0
               then
               begin
                j:=p-j;
                b[h]:=s1[j+j-1]×x
               end tn[p]=0
               else h:=h-1;
```

```
    b[h+1]:=.0;
    j:=l1+1;
    k:=h+1;
    for i:=1 step 1 until k do
     s[i]:=i+j;
    g:=0;
    go to iter
   end fk;
  k2:=k2+1;
  q1[l]:=l;
  q1[l2]:=l2;
  if k2≠lex
   then
   else
    if fl
     then
     begin
      for i:=1 step 1 until n do
       begin
        bb[i]:=b[i];
        sn[i]:=s[i]
       end i;
      for i:=p step -1 until 1 do
       if q1[i]≠0
        then
        begin
         q1[i]:=0;
         q1[p]:=p;
         go to et6
        end q1[i]≠0
```

```
        end fl;
et6:  x:=if fj then -1.0 else 1.0;
      l:=f-2;
      if q1[1]=1
       then
       begin
        s1[1]:=x;
        fn:=true
       end q1[1]=1
       else s1[1]:=-x;
      s1[l]:=if q1[p]=p then -x else x;
      for i:=3 step 2 until l do
       begin
        j:=i÷2+1;
        if q1[j]≠j∧fn
         then
         begin
          s1[i]:=x;
          s1[i-1]:=-x
         end q1[j]≠j∧fn
         else
          if q1[j]≠j
           then
           begin
            s[i]:=-x;
            s1[i-1]:=x
           end
           else
            if q1[j]=j∧fn
             then
```

```
              begin
               s1[i]:=s1[i-1]:=-x;
               fn:=false
              end
              else
              begin
               s1[i]:=s1[i-1]:=x;
               fn:=true
              end
         end i;
        go to extr
       end b[h+1]≠.0∧(k2≠lex∨fk);
    if fk
     then
     begin
et14:for i:=1 step 1 until l1 do
       begin
        j:=tb[i];
        bb[j]:=d[j+j-1]
       end i;
et9: for i:=1 step 1 until p do
       d[i]:=bb[i]-r;
      go to et7
     end fk;
    if b[n]<bb[n]∧fl
     then
     for i:=1 step 1 until n do
      begin
       b[i]:=bb[i];
       s[i]:=sn[i]
```

```
          and b[n]<bb[n]∧fl,i;
        if b[n]=.0
         then go to et14
         else
         begin
           fb:=true;
           go to et1
         end b[n]↓.0;
et7:
      d[p+p]:=-b[n]
    end block
  end prodrevsimplextr
```

References

[1] S. I. Gass, *Linear programming*, New York 1969.
[2] F. Pankowski, *Aproksymacja optymalna funkcji empirycznych o znanych właściwościach*, Doctoral thesis, University of Wrocław, 1975.
[3] M. Simonnard, *Programmation linéaire*, Paris 1962.
[4] С. И. Зухович и Л. И. Авдеева, *Линейное и выпуклое программирование*, Москва 1964.

INSTITUTE OF MATHEMATICS AND PHYSICS
ACADEMY OF TECHNOLOGY AND AGRICULTURE
BYDGOSZCZ, POLAND

Received on 14. 12. 1976

ALGORYTMY 77-78

F. PANKOWSKI (Bydgoszcz)

APROKSYMACJA JEDNOSTAJNA FUNKCJI EMPIRYCZNYCH, MONOTONICZNYCH I O DANEJ LICZBIE EKSTREMÓW

STRESZCZENIE

Procedura *prodrevsimpl* rozwiązuje następujące zadanie:

Dla danego ciągu liczbowego $\{y_1, y_2, \ldots, y_p\}$ wyznaczyć taki ciąg $\{z_1, z_2, \ldots, z_p\}$, który

(i) realizuje minimum funkcji $f = \max_{1 \leqslant k \leqslant p} |y_k - z_k|$,

(ii) jest monotoniczny.

Dane:

ext – liczba całkowita o wartości 0, gdy rozwiązujemy zadanie przy założeniu, że badana funkcja jest nierosnąca, oraz 1, gdy zakładamy, że badana funkcja jest niemalejąca;

p – liczba danych $y_1, y_2, \ldots, y_p$;

$d[1:3\times p]$ – tablica danych $y_1, y_2, \ldots, y_p$ ($d[1] = y_1$, $d[2] = y_2, \ldots, d[p] = y_p$).

Wyniki:

$d[1:3\times p]$ – tablica wyników $z_1, z_2, \ldots, z_p$ ($z_1 = d[1]$, $z_2 = d[2], \ldots, z_p = d[p]$);

$d[2\times p]$ – błąd aproksymacji jednostajnej.

Procedura *prodrevsimplextr* rozwiązuje zadanie takie jak procedura *prodrevsimpl*, przy czym punkt (ii) zamieniony jest następującym:

(iii) ma daną liczbę ekstremów.

Dane:

lex – liczba ekstremów aproksymowanej funkcji;

ext – liczba całkowita o wartości 0, gdy pierwsze ekstremum jest minimum, oraz 1, gdy pierwsze ekstremum jest maksimum.

Pozostałe dane i wyniki są takie jak w procedurze *prodrevsimpl*.