ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
XIV, 1 (1974)

ALGORITHM 31

E. NEUMAN (Wrocław)

THE CALCULATION OF THE MINIMUM OF A CERTAIN FUNCTION OF SEVERAL VARIABLES

1. Procedure declaration. For given numbers m, n, ν, $a_{lj}^{(i)}$ and $b_l^{(i)}$ ($i = 1, 2, \ldots, \nu$; $l = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$), the procedure *minmaxfun* finds the vector $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ such that

(1) $$\min_x \max_{i=1,2,\ldots,\nu} \lambda_i(x) = \max_{i=1,2,\ldots,\nu} \lambda_i(\bar{x}),$$

where

(2) $$\lambda_i(x) = \sum_{l=1}^{m} \Big| \sum_{j=1}^{n} a_{lj}^{(i)} x_j + b_l^{(i)} \Big| .$$

We assume that $\nu 2^m > n$.

Data:

m — the number of terms in the exterior sum (2),
n — the number of independent variables in the function λ_i,
ni — the number of functions λ_i,
eps — the maximum positive number satisfying the machine equality *1.0* + *eps* = *1.0*,
maxr — the maximum allowed number of type **real** in the computer,
a[*1* : *ni*, *1* : *m* × *n*],
b[*1* : *ni*, *1* : *m*] — the arrays of coefficients of the function (2), where $a_{i,(l-1)n+j} \equiv a_{lj}^{(i)}$, $b_{il} \equiv b_l^{(i)}$ ($i = 1, 2, \ldots, \nu$; $l = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$),
x[*1* : *n*] — the array containing the initial approximation of the vector $\bar{x}$.

Results:

lambmin — the value of the right-hand side expression of equality (1),
x[*1* : *n*] — the array containing the components of the vector $\bar{x}$,

$g[1:ni]$ — the array of the values of the function λ_i at the point $\bar{x}$.

Non-local procedure identifiers:

sleGJ — the procedure solving the system of linear equations $Ax = c$; the procedure heading should be the following:
procedure *sleGJ*$(n, x, y, e1)$; **value** n; **integer** n; **array** x, y; **label** $e1$; where:

n — the number of equations in the system,

$x[1:n]$ — the array containing the solution of the system,

$y[1:n+1]$ — the array in which coefficients of successive equations are inserted, where $y[n+1] = c[i]$ for equation with index i,

$e1$ — the label to which it follows a jump when the system matrix is singular.

Outside of the procedure *sleGJ*, the procedure *oneequation* without parameters, which inserts in the array y the coefficients of successive equations, should be described.

matrinvra — the procedure inverting the matrix B; the procedure heading should be the following:
procedure *matrinvra* $(n, B, C, e2)$; **value** n, B; **integer** n; **array** B, C; **label** $e2$; where:

n — the degree of the matrix B,

$B[1:n, 1:n]$ — the inversed matrix,

$C[1:n, 1:n]$ — the inverse matrix of B,

$e2$ — the label to which it follows a jump when B is singular.

combination — the procedure generating all l-element combinations from the set $M = \{1, 2, \ldots, m\}$; the procedure heading should be the following:
procedure *combination*(m, l, z); **value** m; **integer** m, l; **integer array** z; where:

m — the number of elements in the set M,

l — the number of elements entering into the combination,

$z[1:l]$ — the array of type **integer** containing on entrance increasingly ordered combination of elements of the set M or zeros; on exit, it contains another combination, different from the initial one.

minmaxsol — the procedure described in [4].

```
procedure minmaxfun(m,n,ni,eps,maxr,a,b,x,g,lambmin);
 value m,n,ni;
 integer m,n,ni;
 real eps,maxr,lambmin;
 array a,b,x,g;
 begin
  integer h,h1,hk,H,i,i1,i2,ik1,ik2,j,j1,j2,k,k1,k2,l,l1,l2,
  m1,m2,n1,n2,p,p1,p3,q1,q2,q3,q4,ql;
  real lap,lopt,s,s1,s2,s3,t,t1,t2;
  Boolean b1,b2;
  n1:=n+1;
  n2:=n+2;
  m2:=2↑m;
  m1:=ni×m2;
  q2:=if m>n2 then m else n2;
  q3:=if ni>n1 then ni else n1;
  q4:=if m>n then m else n;
  begin
   integer array e1[1:m×m2],v[1:m1],yc,zc[1:q4],Z1[1:n1],
   zk[1:n],z[1:n2];
   array aa[1:n2],a1[1:n1,1:n1],c[1:n1,1:n],C,D[1:m1],
   e[1:q3,1:q2],f[1:ni,1:m],y[1:n],y1[1:(n+3)↑2/4];
   procedure oneequation;
    begin
     for i:=1 step 1 until q1 do
      aa[i]:=e[k2,i];
     k2:=k2+1
    end oneequation;
   comment insert procedure sleGJ here;
   j:=m+2;
```

```
    j1:=0;
    b2:=false;
    b1:=m=j+j;
    if b1
     then j:=j-1;
    l:=k:=0;
E1:l1:=1;
    for i:=1 step 1 until m do
     zc[i]:=0;
    k2:=m-l;
    for i:=1 step 1 until l do
     l1:=l1×(k2+i)/i;
E2:combination(m,l,zc);
    for i:=1 step 1 until m do
     yc[i]:=1;
    for i:=1 step 1 until m do
     begin
      l2:=zc[i];
      if l2≠0
       then yc[l2]:=-1
     end i;
    for i:=1 step 1 until m do
     e1[i+k]:=yc[i];
    k:=k+m;
    if b2
     then go to E3;
    for i:=1 step 1 until m do
     e1[i+k]:=-yc[i];
    k:=k+m;
E3:l1:=l1-1;
```

```
if l1>0
 then go to E2;
l:=l+1;
if l≤j
 then go to E1;
if b1∧¬b2
 then
 begin
  b2:=true;
  go to E1
 end b1∧¬b2;
if n=1
 then
 begin
  for i:=1 step 1 until ni do
   for j:=1 step 1 until m do
    begin
     e[i,j]:=a[i,j];
     f[i,j]:=b[i,j]
    end j,i;
  b2:=true;
  go to E4
 end n=1;
b1:=b2:=false;
for i:=1 step 1 until n1 do
 z[i]:=0;
for i:=1 step 1 until m1 do
 v[i]:=0;
l1:=-maxr;
E5:for i:=1 step 1 until ni do
```

```
    begin
     s1:=.0;
     l:=0;
     for j:=1 step 1 until m do
      begin
       s:=.0;
       for k:=1 step 1 until n do
        s:=s+a[i,k+l]×x[k];
       s:=abs(s+b[i,j]);
       s1:=s1+s;
       l:=l+n
      end j;
     g[i]:=s1;
     if s1>t1
      then t1:=s1
    end i;
   if b1
    then go to E6;
   lap:=t1;
   h:=p:=0;
   for i:=1 step 1 until ni do
    begin
     l:=0;
     for i1:=1 step 1 until m2 do
      begin
       s1:=.0;
       p:=p+1;
       for k:=1 step 1 until n do
        begin
         s:=.0;
```

```
      for j:=1 step 1 until m do
       s:=s+a[i,k+(j-1)×n]×e1[j+1];
      y[k]:=s
     end k;
    for j:=1 step 1 until m do
     s1:=s1+b[i,j]×e1[j+1];
    s:=.0;
    for j:=1 step 1 until n do
     s:=s+x[j]×y[j];
    s:=s+s1;
    if abs(s-lap)<eps
     then
     begin
      h:=h+1;
      v[p]:=z[h]:=p;
      for j:=1 step 1 until n do
       c[h,j]:=y[j];
      if h=n1
       then go to E7
     end abs(s-lap)<eps;
    l:=l+m
   end i1
  end i;
E8:if h=1
  then
  begin
   for j:=1 step 1 until n do
    y[j]:=-c[1,j];
   go to E9
  end h=1
```

```
     else
     begin
      H:=h-1;
      for i:=2 step 1 until h do
       begin
        i1:=i-1;
        for j:=1 step 1 until n do
         a1[i1,j]:=c[1,j]-c[i,j]
       end i;
E10: ql:=0;
E11: h1:=H-ql;
     ik1:=ik2:=1;
     j:=H-h1;
     k2:=n-h1;
     for i:=1 step 1 until h1 do
      begin
       ik1:=ik1×(j+i)/i;
       ik2:=ik2×(k2+i)/i;
       yc[i]:=zc[i]:=0
      end i;
E12: combination(H,h1,zc);
     ik1:=ik1-1;
E13: combination(n,h1,yc);
     ik2:=ik2-1;
     for i:=1 step 1 until h1 do
      begin
       s:=.0;
       k2:=zc[i];
       for j:=1 step 1 until n do
        s:=s-a1[k2,j];
```

```
          for k:=1 step 1 until h1 do
           begin
            e[i,k]:=s1:=a1[k2,yc[k]];
            s:=s+s1
           end k;
          e[i,h1+1]:=s
         end i;
        q1:=h1+1;
        k2:=1;
        sleGJ(h1,y1,aa,E14);
        for i:=1 step 1 until n do
         y[i]:=1.0;
        for i:=1 step 1 until h1 do
         y[yc[i]]:=y1[i]
       end h+1;
E9:for i:=1 step 1 until ni do
       begin
        l:=0;
        for j:=1 step 1 until m do
         begin
          s:=s1:=.0;
          for k:=1 step 1 until n do
           begin
            t1:=a[i,l+k];
            s:=s+t1×y[k];
            s1:=s1+t1×x[k]
           end k;
          e[i,j]:=s;
          f[i,j]:=s1+b[i,j];
          l:=l+n
```

```
      end j
    end i;
   b1:=false;
   p:=z[1];
E15:
   j:=p+m2;
   l:=p-m2×j;
   if l=0
    then
    begin
     i2:=j;
     k1:=m2
    end l=0
    else
    begin
     i2:=j+1;
     k1:=l
    end l≠0;
   k1:=(k1-1)×m;
   if b1
    then go to E16;
   s:=.0;
   for l:=1 step 1 until m do
    s:=s+e[i2,l]×e1[k1+l];
E17:
   b1:=s≠.0;
   if b1
    then t:=-1/s;
   t1:=maxr;
E4:p:=0;
```

```
for i:=1 step 1 until n1 do
 begin
  l:=0;
  for k:=1 step 1 until m2 do
   begin
    s:=s1:=.0;
    p:=p+1;
    for j:=1 step 1 until m do
     begin
      q1:=e1[j+1];
      s:=s+e[i,j]×q1;
      s1:=s1+f[i,j]×q1
     end j;
    if b2
     then
     begin
      C[p]:=s;
      D[p]:=s1;
      go to E18
     end b2;
    if v[p]=0
     then
     begin
      s3:=lap-s1;
      if b1
       then
       begin
        s:=s×t+1;
        if s≠.0
         then t2:=s3/s;
```

```
                    if t2≥.0∧t2<t1
                     then
                     begin
                      t1:=t2;
                      p1:=p
                     end t2≥.0∧t2<t1
                    end b1
                    else
                    begin
                     if s=.0
                      then go to E19
                      else
                      begin
                       t2:=s3/s;
                       if abs(t2)<t1
                        then
                        begin
                         t1:=t2;
                         p1:=p
                        end abs(t2)<t1
                      end s≠.0
                    end ¬b1;
E19:              end v[p]=0;
E18:             l:=l+m
                end k
              end i;
             if t1=maxr∧¬b2
              then go to E23;
             if b2
              then
```

```
begin
 minmaxsol(m1,eps,maxr,C,D,lopt,.0);
 if n=1
  then
  begin
   x[1]:=t;
   lap:=lopt;
   go to E23
  end n=1;
 for i:=1 step 1 until n1 do
  Z1[i]:=0;
 hk:=0;
 for i:=1 step 1 until m1 do
  if abs(C[i]×t+D[i]-lopt)<eps
   then
   begin
    hk:=hk+1;
    Z1[hk]:=i;
    if hk=n1
     then go to E22
   end abs(C[i]×t+D[i]-lopt)<eps,i;
E22: if abs(lap-lopt)<eps
  then go to if j2<n1 then E24 else E23
  else
  begin
   for i:=1 step 1 until m1 do
    v[i]:=0;
   h:=hk;
   lap:=lopt;
   for i:=1 step 1 until n do
```

```
              x[i]:=x[i]+t×y[i];
             i:=0;
E20:         i:=i+1;
             l:=Z1[i];
             p:=z[i]:=v[l]:=l;
             j:=p÷m2;
             l:=p-m2×j;
             i2:=if l=0 then j else j+1;
             k1:=if l=0 then m2 else l;
             k1:=(k1-1)×m;
             go to E16;
E21:         if i<hk
              then go to E20;
             b2:=h=n1;
             go to E25
            end abs(lap-lopt)≥eps
          end b2;
         if b1
          then
          begin
           lap:=lap-t1;
           t:=t1×t
          end b1
          else t:=t1;
         v[p1]:=p1;
         for i:=1 step 1 until n do
          x[i]:=x[i]+t×y[i];
         i:=1;
         for i:=i while z[i]<p1∧i≤h do
          i:=i+1;
```

```
    h:=h+1;
    i1:=i+1;
    for j:=h step -1 until i1 do
     begin
      k2:=j-1;
      z[j]:=z[k2];
      for k:=1 step 1 until n do
       c[j,k]:=c[k2,k]
     end j;
    p:=z[i]:=p1;
    b1:=true;
    go to E15;
E16:
    for k:=1 step 1 until n do
     begin
      s:=.0;
      for j:=1 step 1 until m do
       s:=s+a[i2,k+(j-1)×n]×e1[j+k1];
      y1[k+1]:=c[i,k]:=s
     end k;
    if b2
     then go to E21;
E25:
    y1[1]:=1.0;
    if h<n1
     then go to E8;
E7:for i:=1 step 1 until n1 do
     begin
      for j:=1 step 1 until n do
       a1[i,j+1]:=c[i,j];
```

```
     a1[i,1]:=1.0
    end i;
   matrinvra(n1,a1,a1,E26);
   s:=maxr;
   for i:=1 step 1 until n1 do
    begin
     s1:=a1[1,i];
     if s1<s
      then s:=s1
    end i;
   if s≥.0
    then go to E23;
E26:
   b2:=true;
   j2:=0;
   for i:=1 step 1 until n do
    zk[i]:=0;
E24:
   combination(n1,n,zk);
   l:=zk[1];
   k2:=n-1;
   for i:=1 step 1 until k2 do
    begin
     k:=zk[i+1];
     for j:=1 step 1 until n do
      a1[i,j]:=c[l,j]-c[k,j]
    end i;
   H:=n-1;
   j2:=j2+1;
   if j2≤n1
```

```
    then go to E10;
E23:
   b1:=true;
   go to E5;
E14:
   if ik2>0
    then go to E13
    else
    begin
     if ik1>0
      then go to E12;
     ql:=ql+1;
     go to if h1>1 then E11 else E23
    end ik2=0.
E6:lambmin:=lap
  end
 end minmaxfun
```

2. Method used. Let us denote by $\{e_k\}$ $(k = 1, 2, \ldots, 2^m)$ the sequence of m-element vectors whose components are all variations with repetitions of the elements -1 and 1. Let e_{k_l} $(l = 1, 2, \ldots, m)$ denote the l-th component of the vector e_k. For a fixed i $(i = 1, 2, \ldots, \nu)$, it is possible to write the function λ_i in the form

$$\lambda_i(x) = \max_{k=1,2,\ldots,2^m} \Big(\sum_{j=1}^{n} c_{kj} x_j + d_k \Big),$$

where

$$c_{kj} = \sum_{l=1}^{m} e_{k_l} a_{lj}^{(i)}, \quad d_k = \sum_{l=1}^{m} e_{k_l} b_l^{(i)} \quad (k = 1, 2, \ldots, 2^m;\ j = 1, 2, \ldots, n).$$

Further, let

$$r_k(x) = \sum_{j=1}^{n} c_{kj} x_j + d_k.$$

Now, equality (1) is equivalent to the equality

$$\min_{x} \max_{k=1,2,\ldots,s} r_k(x) = \max_{k=1,2,\ldots,s} r_k(\bar{x}), \tag{3}$$

where $s = \nu 2^m$.

The described below method of solving problem (1) is based on the descent method (see [1]) applied to problem (3); the last-mentioned method requires that every square submatrix (of dimension $n \times n$) of $C = (c_{ij})$ $(i = 1, 2, \ldots, s;\ j = 1, 2, \ldots, n)$ be non-singular. Transition from problem (1) to problem (3) causes that the above-mentioned request for the matrix C will be violated. Therefore, the description of the method is somewhat different from that given in [1].

Let us denote by $x = (x_1, x_2, \ldots, x_n)$ the initial approximation of the vector $\bar{x}$.

Define an auxiliary function

$$r(x) = \max_{k=1,2,\ldots,s} r_k(x),$$

and the set of indices $S = \{1, 2, \ldots, s\}$ $(s = \nu 2^m)$. The scheme of the method is contained in the following steps:

1. Form a set of indices $K \subset S$ such that

$$K = \{k_i \mid r_{k_1}(x) = r_{k_2}(x) = \ldots = r_{k_l}(x) > r_{k_{l+j}}(x);\ j > 0\}.$$

2. Calculate $r = r_{k_1}(x)$.

3. If $l > n$, then go to step 7.

4. Find the auxiliary vector $y = (y_1, y_2, \ldots, y_n)$; if $l = 1$, then $y_j = -c_{k_1 j}$ $(j = 1, 2, \ldots, n)$, otherwise, the numbers y_j are the solution of the system of linear equations

$$\sum_{j=1}^{n} c_{k_1 j} y_j = \sum_{j=1}^{n} c_{k_2 j} y_j = \ldots = \sum_{j=1}^{n} c_{k_l j} y_j \quad (k_i \in K, i = 1, 2, \ldots, l; 1 < l < n+1).$$

5. Let

$$s = \sum_{j=1}^{n} c_{k_1 j} y_j, \qquad s_{k_i} = \sum_{j=1}^{n} c_{k_i j} y_j \quad (k_i \notin K,\ k_i \in S).$$

Calculate

$$t_{k_{l+1}} = \min_{\substack{k_i \notin K \\ k_i \in S}} \begin{cases} \left(s\, \dfrac{r - r_{k_i}(x)}{s - s_{k_i}} > 0 \right) & \text{for } s \neq 0, \\[2ex] \dfrac{r - r_{k_i}(x)}{s_{k_i}} & \text{for } s = 0. \end{cases} \tag{4}$$

Subsequently, calculate

$$t = \begin{cases} \dfrac{-t_{k_{l+1}}}{s} & \text{for } s \neq 0, \\ \operatorname{sign} \dfrac{r - r_{k_{l+1}}(x)}{s_{k_{l+1}}} & \text{for } s = 0, \end{cases} \tag{5}$$

and substitute $r := r - t_{k_{l+1}}$ for $s \neq 0$.

6. Perform the operations $x := x + ty$, $K := K \cup \{k_{l+1}\}$ and $l := l+1$, and return to step 3.

7. Let $c_i = \{1, c_{k_i 1}, c_{k_i 2}, \ldots, c_{k_i n}\}$ $(k_i \in K;\ i = 1, 2, \ldots, n+1)$. Form the matrix c, where by c_i we denote the row of the index i. If $\det c \neq 0$, then find the matrix $c^{-1} = (p_{ij})$ $(i, j = 1, 2, \ldots, n+1)$. If the matrix c is singular, then go to step 9.

8. If, for every j $(j = 1, 2, \ldots, n+1)$, $p_{1j} \geqslant 0$ holds, then $x = \bar{x}$ (end of calculations). If, for some j, there is $p_{1j} < 0$, then go to step 9.

Remark. In the case where for every j we have $p_{1j} > 0$, the vector $\bar{x}$ is unique.

9. From the set K whose elements will be denoted now by $1, 2, \ldots, l$ $(l > n)$, for simplicity, choose an arbitrary n-element subset K', $K' = \{k_1, k_2, \ldots, k_n\} \subset K$ and perform operations like in step 4 for $l = n$. If, for certain K', $s \neq 0$ holds, then go to step 10.

10. Calculate

$$C_k = \sum_{j=1}^{n} c_{kj} y_j, \qquad D_k = \sum_{j=1}^{n} c_{kj} x_j + d_k \qquad (k = 1, 2, \ldots, s)$$

and, subsequently, find the minimum of the function $\max\limits_{k=1,2,\ldots,s} (C_k t + D_k)$. Let us denote by $\bar{t}$ the point in which this function attains the minimum. Substitute $x := x + \bar{t}y$ and return to step 1.

Equations (4) and (5) in the case $s = 0$ and steps 9 and 10 are not present in the original description of the descent method.

3. Certification. The procedure *minmaxfun* has been extensively tested on the Odra 1204 computer. The obtained results were correct. In the course of the calculation, the library procedures *sleGJ* and *matrinvra* for the Odra 1204 computer (see [5]) and a modified version of the procedure *combination* have been used. The modification of *combination* (see [2]) was that combinations of the set $\{1, 2, \ldots, m\}$ instead of the set $\{0, 1, \ldots, m-1\}$, appearing in the original description, were formed.

The time of calculation depends on the parameters m, n and v, on the form of functions (2) and on the initial approximation of the vector $\bar{x}$.

The table contains the calculation times for the procedure *minmaxfun* for different values of m, n and ν:

m	2	1	4	5	5	6	7
n	2	2	1	2	3	4	6
ν	2	6	25	12	15	10	7
time in sec.	1	3	5	183	248	623	852

4. Application. Using the procedure *minmaxfun*, the author calculated modified values of norms of certain discrete projections. Details are described in paper [3].

5. Acknowledgment. The author would like to express his gratitude to Dr. Stefan Paszkowski of the Computing Centre of the University of Wrocław for his valuable remarks during the preparation of this paper.

References

[1] E. W. Cheney, *Introduction to approximation theory*, New York 1966.
[2] J. Kurtzberg, *Algorithm 94, Combination*, Comm. ACM 5 (1962), p. 344.
[3] E. Neuman, *Projections in uniform polynomial approximations*, this fascicle, p. 99-125.
[4] — *The calculation of the minimum of a certain function of one variable*, ibidem, p. 137-142.
[5] *Algol procedures of the Odra 1204 computer*, part 1, Wrocław 1970.

MATHEMATICAL INSTITUTE
UNIVERSITY OF WROCŁAW
50-384 WROCŁAW

Received on 20. 2. 1973

ALGORYTM 31

E. NEUMAN (Wrocław)

OBLICZANIE MINIMUM PEWNEJ FUNKCJI WIELU ZMIENNYCH

STRESZCZENIE

Dla danych liczb $m, n, \nu, a_{lj}^{(i)}$ oraz $b_l^{(i)}$ $(i = 1, 2, \ldots, \nu;\ l = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n)$ procedura *minmaxfun* oblicza wektor $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ taki, że

(1) $$\min_x \max_{i=1,2,\ldots,\nu} \lambda_i(x) = \max_{i=1,2,\ldots,\nu} \lambda_i(\bar{x}),$$

gdzie

$$\lambda_i(x) = \sum_{j=1}^{m} \Big| \sum_{j=1}^{n} a_{lj}^{(i)} x_j + b_l^{(i)} \Big|. \tag{2}$$

Zakłada się, że $\nu 2^m > n$.

Dane:

m — liczba składników w sumie zewnętrznej (2),
n — liczba zmiennych niezależnych funkcji λ_i,
ni — liczba funkcji λ_i,
eps — największa liczba dodatnia spełniająca równość maszynową *1.0* + *eps* = *1.0*,
maxr — największa dopuszczalna w maszynie cyfrowej liczba typu **real**,
a [*1* : *ni*, *1* : *m* × *n*],
b [*1* : *ni*, *1* : *m*] — tablice współczynników funkcji (2), gdzie $a_{i,(l-1)n+j} \equiv a_{lj}^{(i)}$, $b_{il} \equiv b_l^{(i)}$ $(i = 1, 2, \ldots, \nu;\ l = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n)$,
x [*1* : *n*] — tablica zawierająca przybliżenie początkowe szukanego wektora $\bar{x}$.

Wyniki:

lambmin — wartość prawej strony równości (1),
x [*1* : *n*] — tablica zawierająca składowe wektora $\bar{x}$,
g [*1* : *ni*] — tablica wartości funkcji λ_i w punkcie $\bar{x}$.

Nielokalne nazwy procedur niestandardowych:

sleGJ — procedura rozwiązująca układ równań liniowych $Ax = c$; nagłówek procedurowy powinien być następujący:
procedure *sleGJ* (*n*, *x*, *y*, *e1*); **value** *n*; **integer** *n*; **array** *x*,*y*; **label** *e1*;

gdzie:

n — liczba równań układu,
x [*1* : *n*] — tablica zawierająca rozwiązanie układu,
y [*1* : *n* + *1*] — tablica, w której umieszczone są współczynniki kolejnych równań, gdzie *y* [*n* + *1*] = *c* [*i*] dla równania o numerze *i*,
e1 — etykieta, do której następuje skok, gdy macierz układu jest osobliwa.

Na zewnątrz procedury *sleGJ* powinien znajdować się opis procedury bez parametrów, o nazwie *oneequation*, umieszczającej w tablicy *y* współczynniki kolejnych równań układu.

matrinvra — procedura odwracająca macierz *B* stopnia *n*; nagłówek procedury powinien być następujący:
procedure *matrinvra* (*n*, *B*, *C*, *e2*); **value** *n*, *B*; **integer** *n*; **array** *B*, *C*; **label** *e2*;
gdzie:
n — stopień macierzy *B*,
B [*1* : *n*, *1* : *n*] — macierz odwracana,
C [*1* : *n*, *1* : *n*] — macierz odwrotna do *B*,
e2 — etykieta, do której następuje skok, gdy *B* jest macierzą osobliwą.

combination — procedura generująca wszystkie l-elementowe kombinacje ze zbioru $M = \{1, 2, \ldots, m\}$; nagłówek procedury powinien być następujący:

procedure *combination* (m, l, z); **value** m; **integer** m, l; **integer array** z;

gdzie:

m — liczba elementów wchodzących do zbioru M,

l — liczba elementów wchodzących do kombinacji,

$z[1:l]$ — tablica typu **integer**, zawierająca na wejściu pewną uporządkowaną w sposób rosnący kombinację elementów ze zbioru M lub same zera, a na wyjściu — inną kombinację, różną od początkowej,

minmaxsol — procedura, której opis znajduje się w [4].

W procedurze *minmaxfun* zastosowano metodę (opisaną w § 2) opartą na metodzie spadku. Procedura była wielokrotnie sprawdzana na maszynie cyfrowej Odra 1204. Otrzymane wyniki były poprawne. W trakcie wykonywania obliczeń korzystano z procedur bibliotecznych *sleGJ* i *matrinvra* maszyny cyfrowej Odra 1204 (patrz [5]) oraz ze zmodyfikowanej przez autora procedury *combination* (patrz [2]). Modyfikacja polegała na tym, że kombinacje wybierano ze zbioru $\{1, 2, \ldots, m\}$ zamiast ze zbioru $\{0, 1, \ldots, m-1\}$, występującego w opisie oryginalnym.