ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
XIV, 3 (1974)

ALGORITHM 37

F. PANKOWSKI (Bydgoszcz)

SOLUTION OF SPARSE LINEAR EQUATION SYSTEMS

1. Procedure declaration. Procedure *sparsesystem* solves sparse linear equation systems and produces reduced Crout's formulas (see Remark 1). During the calculation, the auxiliary procedures *M01*, *subst1*, *bitword*, *bit1*, *size*, *forgen*, *decompose* and *solve* are used. All of them are described internally.

The procedure is most effective if one solves the system $Ax = b$ with the matrix A of identical structure, i.e. with identical displacement of the non-zero elements of this matrix and different values, or with the constant matrix A and varying the right-hand side b.

Data:

N — number of equations and unknowns in the system,

$N1$ — number of non-zero elements of the matrix A, not counting the main diagonal elements which are always assumed to be non-zero,

$R[1:N+N1]$ — array of column numbers of the non-zero elements of the matrix A (not counting the main diagonal elements) arranged by rows in the increasing order; the column numbers of a given row should be followed by a zero entry,

E — integer indicator with value either zero or equal to the number of repetitions of using the matrix A of identical structure but with different non-zero elements; if $E \neq 0$, then $L = 0$ (see below),

L — integer indicator with value either zero or equal to the number of repetitions of using the same matrix A and different right-hand sides b; if $L \neq 0$, then $E = 0$,

wl — wordlength in bits of the computer used,

$A[1:N+N1]$ — array of the elements of the matrix A; it should contain the non-zero elements of matrix A chosen from the sequence

$$a_{11}, a_{21}, \ldots, a_{n1},$$
$$a_{12}, a_{13}, \ldots, a_{1n},$$
$$a_{22}, a_{32}, \ldots, a_{n2},$$
$$a_{23}, a_{24}, \ldots, a_{2n}, \ldots$$

without changing their order,

$B[1:N]$ — array of the right-hand sides of the system,

$X[1:N]$ — array of solutions,

readb — procedure identifier of the procedure without parameters the call of which should place appropriate elements into the array $B[1:N]$,

readm — procedure identifier of the procedure without parameters the call of which should place appropriate elements into the array $A[1:N+N1]$,

printx — procedure identifier of the procedure without parameters the call of which should make available to the user the solutions contained in array $X[1:N]$ at every repetition.

Results:

the results are available through procedure *printx* which must be provided by the user.

Remark 1. The procedure is most effective when used with E or L positive. The auxiliary procedures *M01*, *size* and *forgen* are called only once in every call of the procedure *sparsesystem*. As follows from numerical experiments performed on the ODRA 1204 computer, the calculation times of the second and all next repetitions while changing only the values of the non-zero elements of the matrix A and leaving its structure unchanged are about N times smaller than the calculation time of the first repetition. This calculation time is still smaller when only right-hand sides of the system are changed. In the experiments, the matrix A contained about 15% non-zero elements.

2. Procedure used. The algorithm used in the *sparsesystem* procedure is based on the Crout method which is also called the *compact method* (see, e.g., [1], p. 168-171).

Let the system of equations

$$Ax = b \tag{1}$$

be given, where $A = [a_{ij}]$ is the square matrix of the n-th degree composed of the coefficients of the system, and $x^{\mathrm{T}} = (x_1, x_2, \ldots, x_n)$ and $b^{\mathrm{T}} = (b_1, b_2, \ldots, b_n)$ are the n-element column vectors.

```
procedure sparsesystem(N,N1,R,E,L,wl,A,B,X,readb,readm,printx);
 value N,N1,E,L,wl;
 integer N,N1,E,L,wl;
 integer array R;
 array A,B,X;
 procedure readb,readm,printx;
 begin
  integer h,b,j1,j2,j3,j4,j5,j6,j7;
  integer array M[1:N×N+wl+1],owl,owp,okg,okd[1:N];
  procedure M01(N,M,R,owl,owp,okg,okd);
   value N;
   integer N;
   integer array M,R,owl,owp,okg,okd;
   begin
    integer m,j,b,s,l;
    j:=0;
    for m:=1 step 1 until N do
     okg[m]:=okd[m]:=0;
    for m:=1 step 1 until N do
     begin
      bitword(m,m,b,s,N);
      subst1(b+1,M[s]);
      l:=R[j+1];
      owl[m]:=if l=0 then 0 else if l<m then l else 0;
et2:  j:=j+1;
      l:=R[j];
      if l≠0
       then
       begin
        bitword(m,l,b,s,N);
```

```
          subst1(b+1,M[s]);
          if l>m
           then
           begin
            if okg[l]=0
             then okg[l]:=m
           end l>m
           else okd[l]:=m-l;
          go to et2
         end l≠0;
        l:=R[j-1];
        owp[m]:=if l>m then l-m else 0
       end m
      end MO1;
     procedure subst1(K,P);
      value K;
      integer K,P;
      begin
       comment this procedure should insert a one in bit K
 of the word representing variable P;
      end subst1;
     procedure bitword(r,c,b,s,N);
      value r,c;
      integer r,c,b,s,N;
      begin
       b:=(r-1)×N+c;
       s:=b÷wl;
       b:=b-s×wl-1;
       if b=-1
        then b:=wl-1
```

```
      else s:=s+1
    end bitword;
   Boolean procedure bit1(K,P);
    value K,P;
    integer K,P;
    begin
     comment this procedure should have assigned value true
if bit K of the word representing variable P is a one,
and value false otherwise;
    end bit1;
   procedure forgen(h,N,M,B1,D,F,F1,F2,F3,G1,G2,G3,U,owl,owp,okg,okd);
    value N;
    integer h,N;
    integer array M,B1,D,F,F1,F2,F3,G1,G2,G3,U,owl,owp,okg,okd;
    begin
     integer I,J,m,j1,j2,j3,j4,j5,j6,j7,g,z,z1,w,a,t,l,K,P,e;
     integer array wa,ka,ks,ws,kp,wp,v[1:N];
     j2:=I:=D[1]:=1;
     m:=okd[1]+1;
     for w:=2 step 1 until m do
      begin
       bitword(w,1,K,P,N);
       if bit1(K,M[P])
        then
        begin
         j2:=j2+2;
         F1[j2]:=F1[j2+1]:=I:=I+1
        end
      end w;
```

```
wa[1]:=F1[1]:=F1[2]:=1;
ka[1]:=F[1]:=I;
h:=0;
j2:=j2+1;
m:=owp[1]+1;
for w:=2 step 1 until m do
 begin
  bitword(1,w,K,P,N);
  if bit1(K,M[P])
   then
   begin
    h:=h+2;
    G1[h-1]:=G1[h]:=U[h-1]:=I:=I+1;
    U[h]:=w
   end
 end w;
D[2×N+1]:=F[3×N+1]:=h+2;
J:=I;
j6:=h;
D[N+1]:=F[N+1]:=F[2×N+1]:=F[4×N+1]:=F[5×N+1]:=g:=j1:=j3:=j4:=
        j5:=j7:=0;
for m:=2 step 1 until N do
 begin
  F[m]:=F[N+m]:=F[2×N+m]:=F[3×N+m]:=F[4×N+m]:=F[5×N+m]:=D[2×N+m]
        :=z:=0;
  a:=okg[m];
  if a≠0
   then
   for w:=s step 1 until m−1 do
    begin
```

```
      bitword(w,m,K,P,N);
      if bit1(K,M[P])
       then
       begin
        z:=z+1;
        v[z]:=w;
        ks[z]:=ka[w]:=ka[w]+1;
        ws[z]:=wa[w]
       end
    end w;
  wa[m]:=D[m]:=J+1;
  if z≠0
   then
   begin
    w:=N-m+1;
    for a:=1 step 1 until w do
     begin
      for z1:=1 step 1 until z do
       begin
        bitword(m+a-1,v[z1],K,P,N);
        if bit1(K,M[P])
         then
         begin
          g:=g+1;
          wp[g]:=ws[z1]:=ws[z1]+1;
          kp[g]:=ks[z1]
         end
       end z1;
      bitword(m+a-1,m,K,P,N);
      if bit1(K,M[P])
```

```
then
begin
 if g≠0
  then
  begin
   t:=2×N+m;
   F[t]:=F[t]+1;
   j1:=j1+2;
   F3[j1-1]:=g;
   F3[j1]:=I:=I+1;
   for l:=1 step 1 until g do
    begin
     j1:=j1+2;
     F3[j1-1]:=wp[l];
     F3[j1]:=kp[l]
    end l;
   j1:=j1+1;
   F3[j1]:=J:=J+1;
   g:=0
  end g≠0
  else
  begin
   F[m]:=F[m]+1;
   j2:=j2+2;
   F1[j2-1]:=J:=J+1;
   F1[j2]:=I:=I+1
  end g=0
end
else
 if g≠0
```

```
        then
        begin
         F[N+m]:=F[N+m]+1;
         j3:=j3+1;
         F2[j3]:=g;
         subst1(K+1,M[P]);
         for l:=1 step 1 until g do
          begin
           j3:=j3+2;
           F2[j3-1]:=wp[l];
           F2[j3]:=kp[l]
          end l;
         j3:=j3+1;
         g:=0;
         F2[j3]:=J:=J+1
        end
  end a
end z≠0
else
begin
 w:=okd[m]+1;
 t:=m-1;
 for a:=1 step 1 until w do
  begin
   t:=t+1;
   bitword(t,m,K,P,N);
   if bit1(K,M[P])
    then
    begin
     F[m]:=F[m]+1;
```

```
          j2:=j2+2;
          F1[j2-1]:=J:=J+1;
          F1[j2]:=I:=I+1
        end
      end a
    end z=0;
  z:=0;
  a:=owl[m];
  if a≠0
   then
   for w:=a step 1 until m-1 do
    begin
     bitword(m,w,K,P,N);
     if bit1(K,M[P])
      then
      begin
       z:=z+1;
       ws[z]:=wp[z]:=wa[w]:=wa[w]+1;
       ks[z]:=ka[w];
       v[z]:=kp[z]:=w
      end
    end w;
  if z=0
   then D[N+m]:=0
   else
   begin
    D[N+m]:=z;
    for l:=1 step 1 until z do
     begin
      j4:=j4+2;
```

```
      B1[j4-1]:=wp[l];
      B1[j4]:=kp[l]
    end l
  end z≠0;
ka[m]:=J;
if z≠0
  then
  begin
   w:=N-m;
   for a:=1 step 1 until w do
    begin
     for z1:=1 step 1 until z do
      begin
       bitword(v[z1],m+a,K,P,N);
       if bit1(K,M[P])
        then
        begin
         g:=g+1;
         wp[g]:=ws[z1];
         kp[g]:=ks[z1]:=ks[z1]+1
        end
      end z1;
     bitword(m,m+a,K,P,N);
     if bit1(K,M[P])
      then
      begin
       if g≠0
        then
        begin
         t:=5×N+m;
```

```
        F[t]:=F[t]+1;
        j5:=j5+2;
        G3[j5-1]:=g;
        G3[j5]:=I:=I+1;
        for l:=1 step 1 until g do
         begin
          j5:=j5+2;
          G3[j5-1]:=wp[l];
          G3[j5]:=kp[l]
         end l;
        j5:=j5+1;
        G3[j5]:=J:=J+1;
        g:=0
       end g≠0
       else
       begin
        t:=3×N+m;
        F[t]:=F[t]+1;
        j6:=j6+2;
        G1[j6-1]:=J:=J+1;
        G1[j6]:=I:=I+1
       end g=0;
     h:=h+2;
     U[h-1]:=J;
     U[h]:=a+m;
     l:=2×N+m;
     D[l]:=D[l]+1
    end
    else
     if g≠0
```

```
        then
        begin
         t:=4×N+m;
         F[t]:=F[t]+1;
         j7:=j7+1;
         G2[j7]:=g;
         subst1(K+1,M[P]);
         for l:=1 step 1 until g do
          begin
           j7:=j7+2;
           G2[j7-1]:=wp[l];
           G2[j7]:=kp[l]
          end l;
         g:=0;
         j7:=j7+1;
         G2[j7]:=J:=J+1;
         h:=h+2;
         U[h-1]:=J;
         U[h]:=a+m;
         l:=2×N+m;
         D[l]:=D[l]+1
        end
    end a
  end z≠0
  else
  begin
   w:=owp[m];
   a:=0;
   t:=m;
   for z1:=1 step 1 until w do
```

```
          begin
           a:=a+1;
           t:=t+1;
           bitword(m, t, K, P, N);
          if bit1(K,M[P])
           then
           begin
            l:=3×N+m;
            F[l]:=F[l]+1;
            j6:=j6+2;
            h:=h+2;
            G1[j6-1]:=U[h-1]:=J:=J+1;
            G1[j6]:=I:=I+1;
            U[h]:=a+m;
            l:=2×N+m;
            D[l]:=D[l]+1
           end
          end z1
        end z=0
      end m
    end forgen;
   procedure size(N,N1,h,j1,j2,j3,j4,j5,j6,j7,M,b,owl,owp,okg,okd);
    value N,N1,M;
    integer N,N1,h,j1,j2,j3,j4,j5,j6,j7,b;
    integer array M,owl,owp,okg,okd;
    begin
     integer K,P,m,z,z1,a,a1,g,l;
     integer array v[1:N];
     j2:=2;
     m:=okd[1]+1;
```

```
for l:=2 step 1 until m do
 begin
  bitword(l,1,K,P,N);
  if bit1(K,M[P])
   then j2:=j2+2
 end l;
h:=0;
m:=owp[1]+1;
for l:=2 step 1 until m do
 begin
  bitword(1,l,K,P,N);
  if bit1(K,M[P])
   then h:=h+2
 end l;
g:=j1:=j3:=j4:=j5:=j7:=0;
j6:=h;
b:=N+N1;
for m:=2 step 1 until N do
 begin
  z:=0;
  a:=okg[m];
  if a≠0
   then
   for a1:=a step 1 until m-1 do
    begin
     bitword(a1,m,K,P,N);
     if bit1(K,M[P])
      then
      begin
       z:=z+1;
```

```
        v[z]:=a1
       end
      end a1;
    if z≠0
     then
     begin
      a1:=N-m+1;
      for a:=1 step 1 until a1 do
       begin
        for z1:=1 step 1 until z do
         begin
          bitword(m+a-1,v[z1],K,P,N);
          if bit1(K,M[P])
           then g:=g+1
         end z1;
        bitword(m+a-1,m,K,P,N);
        if bit1(K,M[P])
         then
         begin
          if g≠0
           then
           begin
            j1:=j1+3;
            for l:=1 step 1 until g do
             j1:=j1+2;
            g:=0
           end g≠0
           else j2:=j2+2
         end
         else
```

```
      if g≠0
       then
       begin
        j3:=j3+2;
        subst1(K+1,M[P]);
        b:=b+1;
        for l:=1 step 1 until g do
         j3:=j3+2;
        g:=0
       end g≠0
   end a
  end z≠0
  else
  begin
   a1:=okd[m]+1;
   l:=m-1;
   for a:=1 step 1 until a1 do
    begin
     l:=l+1;
     bitword(l,m,K,P,N);
     if bit1(K,M[P])
      then j2:=j2+2
    end a
 end z =0;
z:=0;
a:=owl[m];
if a≠0
 then
 for a1:=a step 1 until m-1 do
  begin
```

```
        bitword(m,a1,K,P,N);
        if bit1(K,M[P])
         then
         begin
          z:=z+1;
          v[z]:=a1
         end
       end a1;
    if z≠0
     then
     begin
       for l:=1 step 1 until z do
        j4:=j4+2;
       a1:=N-m;
       for a:=1 step 1 until a1 do
        begin
         for z1:=1 step 1 until z do
          begin
           bitword(v[z1],m+a,K,P,N);
           if bit1(K,M[P])
            then g:=g+1
          end z1;
         bitword(n,m+a,K,P,N);
         if bit1(K,M[P])
          then
          begin
           if g≠0
            then
            begin
             j5:=j5+3;
```

```
            for l:=1 step 1 until g do
             j5:=j5+2;
            g:=0
           end g≠0
           else j6:=j6+2;
          h:=h+2
         end
         else
          if g≠0
           then
           begin
            j7:=j7+2;
            subst1(K+1,M[P]);
            b:=b+1;
            for l:=1 step 1 until g do
             j7:=j7+2;
            g:=0;
            h:=h+2
           end g≠0
      end a
    end z≠0
    else
    begin
     a1:=owp[m];
     l:=m;
     for a:=1 step 1 until a1 do
      begin
       l:=l+1;
       bitword(m,l,K,P,N);
       if bit1(K,M[P])
```

```
            then
            begin
             j6:=j6+2;
             h:=h+2
            end
          end a
        end z=0
      end m
    end size;
   procedure decompose(N,A,C,D,F,F1,F2,F3,G1,G2,G3);
    value N;
    integer N;
    integer array D,F,F1,F2,F3,G1,G2,G3;
    array A,C;
    begin
     integer j1,j2,j3,j4,j5,j6,j7,m,f,f1,l,l1;
     real s;
     j1:=j2:=j3:=j5:=j6:=j7:=0;
     for m:=1 step 1 until N do
      begin
       f:=F[m];
       for l:=1 step 1 until f do
        begin
         j2:=j2+2;
         C[F1[j2-1]]:=A[F1[j2]]
        end l;
       f:=F[N+m];
       for l:=1 step 1 until f do
        begin
         s:=0;
```

```
   j3:=j3+1;
   f1:=F2[j3];
   for l1:=1 step 1 until f1 do
    begin
     j3:=j3+2;
     s:=s-C[F2[j3-1]]×C[F2[j3]]
    end l1;
   j3:=j3+1;
   C[F2[j3]]:=s
  end l;
f:=F[2×N+m];
for l:=1 step 1 until f do
  begin
   j1:=j1+2;
   f1:=F3[j1-1];
   s:=A[F3[j1]];
   for l1:=1 step 1 until f1 do
    begin
     j1:=j1+2;
     s:=s-C[F3[j1-1]]×C[F3[j1]]
    end l1;
   j1:=j1+1;
   C[F3[j1]]:=s
  end l;
f:=F[3×N+m];
j4:=D[m];
for l:=1 step 1 until f do
  begin
   j6:=j6+2;
   C[G1[j6-1]]:=A[G1[j6]]/C[j4]
```

```
 end l;
f:=F[4×N+m];
for l:=1 step 1 until f do
 begin
  s:=0;
  j7:=j7+1;
  f1:=G2[j7];
  for l1:=1 step 1 until f1 do
   begin
    j7:=j7+2;
    s:=s-C[G2[j7-1]]×C[G2[j7]]
   end l1;
  j7:=j7+1;
  C[G2[j7]]:=s/C[j4]
 end l;
f:=F[5×N+m];
for l:=1 step 1 until f do
 begin
  j5:=j5+2;
  f1:=G3[j5-1];
  s:=A[G3[j5]];
  for l1:=1 step 1 until f1 do
   begin
    j5:=j5+2;
    s:=s-C[G3[j5-1]]×C[G3[j5]]
   end l1;
  j5:=j5+1;
  C[G3[j5]]:=s/C[j4]
 end l
end m
```

```
 end decompose;
procedure solve(h,N,B,B1,C,D,U,X);
 value h,N;
 integer h,N;
 integer array B1,D,U;
 array B,C,X;
 begin
  integer m,j4,j5,f,l;
  real s;
  array Y[1:N];
  j4:=0;
  for m:=1 step 1 until N do
   begin
    j5:=D[m];
    if D[N+m]=0
     then Y[m]:=B[m]/C[j5]
     else
     begin
      s:=B[m];
      f:=D[N+m];
      for l:=1 step 1 until f do
       begin
        j4:=j4+2;
        s:=s-C[B1[j4-1]]×Y[B1[j4]]
       end l;
      Y[m]:=s/C[j5]
     end D[N+m]≠0
   end m;
  X[N]:=Y[N];
  for l:=N-1 step -1 until 1 do
```

```
    begin
     f:=D[2×N+1];
     s:=Y[1];
     for m:=1 step 1 until f do
      begin
       s:=s-C[U[h-1]]×X[U[h]];
       h:=h-2
      end m;
     X[1]:=s
    end 1
  end solve;
j1:=N×N÷wl+1;
for j2:=1 step 1 until j1 do
 M[j2]:=0;
MO1(N,M,R,owl,owp,okg,okd);
size(N,N1,h,j1,j2,j3,j4,j5,j6,j7,M,b,owl,owp,okg,okd);
begin
 integer array D[1:3×N],B1[0:j4],F[1:6×N],F1[0:j2],F2[0:j3],F3[0:
 j1],G1[0:j6],G2[0:j7],G3[0:j5],U[0:h];
 array C[1:b];
 forgen(h,N,M,B1,D,F,F1,F2,F3,G1,G2,G3,U,owl,owp,okg,okd);
 if E≠0
  then
  begin
   readb;
   for L:=1 step 1 until E do
    begin
     readm;
     decompose(N,A,C,D,F,F1,F2,F3,G1,G2,G3);
     solve(h,N,B,B1,C,D,U,X);
```

```
      printx
     end L
   end E≠0
   else
   begin
    readm;
    decompose(N,A,C,D,F,F1,F2,F3,G1,G2,G3);
    for E:=1 step 1 until L do
     begin
      readb;
      solve(h,N,B,B1,C,D,U,X);
      printx
     end E
   end E=0
  end
end sparsesystem
```

First we decompose the matrix A into the product $A = LU$, where L is the low triangle matrix, and U the upper one.

The solution of system (1) will be found after solving the system

$$Ly = b, \tag{2}$$

and then the system

$$Ux = y. \tag{3}$$

Elements of matrices L and U will be successively evaluated from the formulas

$$l_{im} = a_{im} - \sum_{k=1}^{m-1} l_{ik} u_{km} \quad (i = m, m+1, \ldots, N), \tag{4}$$

$$u_{mj} = \Big(a_{mj} - \sum_{k=1}^{m-1} l_{mk} u_{kj}\Big) / l_{kk} \quad (j = m+1, m+2, \ldots, N), \tag{5}$$

where $m = 1, 2, \ldots, N$, $l_{ij} = 0$ for $i < j$, $u_{ii} = 1$ for $i = 1, 2, \ldots, N$, $u_{ij} = 0$ for $i > j$, and y and x in (2) and (3) are defined by

$$y_i = \Big(b_i - \sum_{k=1}^{i-1} l_{ik} y_k\Big) / l_{ii} \quad (i = 1, 2, \ldots, N), \tag{6}$$

$$x_i = y_i - \sum_{k=i+1}^{N} u_{ik} x_k \quad (i = N, N-1, \ldots, 1). \tag{7}$$

From formulas (4) and (5) we obtain the reduced Crout formulas which concern operations on non-zero elements only. Those formulas are formed in the order of their applying in the procedure. This order is shown in the following scheme:

TABLE 1

l_{11}	u_{12}	u_{13}	u_{14}	u_{15}	first step ($m = 1$)
l_{21}	l_{22}	u_{23}	u_{24}	u_{25}	2-nd step ($m = 2$)
l_{31}	l_{32}	l_{33}	u_{34}	u_{35}	3-rd step ($m = 3$)
l_{41}	l_{42}	l_{43}	l_{44}	u_{45}	4-th step ($m = 4$)
l_{51}	l_{52}	l_{53}	l_{54}		

Every step we begin with evaluating the elements of the matrix L.

Any element is obtained as the difference of the corresponding element of the matrix A and the sum of products of pairs of elements of L which are not on the diagonal and occur in the given row (on the left-hand side) and of elements of U which occur in the given column (above). Evaluating elements of U, we have still to perform the division by the corresponding elements of L which occur on the diagonal.

Reduction is made in two analogous stages, the first concerning elements of L, the other concerning elements of U. The first stage (for $m = 1$) of reduction is simple since the components of sums of formulas (4) and (5) do not occur. For the suitable i, we have $l_{im} \neq 0$ if and only if $a_{im} \neq 0$ and, for the suitable j, we have $u_{mj} \neq 0$ if and only if $a_{mj} \neq 0$. For $m \geqslant 2$, the simplified formula for l_{im} is formed if and only if $a_{im} \neq 0$ or if at least one of the components of the sum

$$\sum_{k=1}^{m-1} l_{ik} u_{km}$$

is different from zero. The similar result can be obtained for elements of the matrix U.

Elimination of the vanishing components of the sum $\sum l_{ik} u_{km}$ is performed in two steps. First we eliminate the components for which $u_{km} = 0$. Then from the remaining components we eliminate those for which $l_{ik} = 0$.

We analogously proceed with the sum $\sum l_{mk} u_{kj}$, the reduction, however, beginning with the examination of elements l_{mk}.

Example 1. Let $m = 4$ (see Table 1). We begin the reduction with the examination of elements u_{14}, u_{24} and u_{34}. If $u_{14} \neq 0$, $u_{24} = 0$ and $u_{34} \neq 0$, then in the second step we only examine the elements l_{41} and l_{43}.

Suppose $l_{41} \neq 0$ and $l_{43} \neq 0$. Then the reduced formula for l_{44} is obtained after the examination of the element a_{44}. If $a_{44} = 0$, then we finally obtain

$$l_{44} = -l_{41}u_{14} - l_{43}u_{34}.$$

Next we examine the elements l_{51} and l_{53}. Let $l_{51} = 0$, $l_{53} \neq 0$ and $a_{54} \neq 0$. In this case we have

$$l_{54} = a_{54} - l_{53}u_{34}.$$

We begin the discussion of formula (5) with the examination of the elements l_{41}, l_{42} and l_{43}. Let e. g. $l_{41} = 0$, $l_{42} \neq 0$ and $l_{43} = 0$. We examine then only the elements u_{25} and a_{45}. If e. g. $u_{25} = 0$ and $a_{45} \neq 0$, then for the element u_{45} we have the formula $u_{45} = a_{45}/l_{44}$.

3. Realization of the method in procedures. On account of economy of the fast memory we perform the reduction on the zero-one matrix M placed on bits of the computer words. Elements of this matrix are given by the formula

$$m_{ij} = \begin{cases} 1 & \text{for } a_{ij} \neq 0, \\ 0 & \text{for } a_{ij} = 0, \end{cases}$$

where a_{ij} are elements of the matrix of system (1) occurring in the array A.

To every element of this table we linearly assign the number of place, successively, according to the way of placing (see Section 1).

When reducing, instead of the matrix M we obtain the new zero-one matrix M' which corresponds to matrices L and U, i.e. the non-zero elements of these matrices are ones in M'. The matrix M' has, in general, more ones than the matrix M, i.e. L and U have more non-zero elements than the initial matrix of system (1). In view of this, numbers of places assigned to non-zero elements of L and U are, in general, different from corresponding numbers in the array A.

Example 2. Let the matrix of system (1) have the form

$$A = \begin{bmatrix} 3 & 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & -5 & 0 \\ 1 & 0 & -3 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 & 3 \end{bmatrix}.$$

Hence the zero-one matrix is given by

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Numbers of places of the elements of the array A can be written in the following form:

TABLE 2

1	3			
	4		6	
2		7		
			8	
	5			9

We see that they are consistent with the fixed way of placing the coefficients of system (1) in the array A.

We have

$$M' = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & \emptyset & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & \emptyset & 1 \end{bmatrix},$$

where $\emptyset$ denotes that the zero element was changed in the course of the reduction into the non-zero element. (No respect is paid to the case where a non-zero element is getting the zero one.) To the matrix M' there correspond new numbers of places of elements of matrices L and U which are given in the following table:

TABLE 3

1	3			
	4		7	
2	5	8		
			9	
	6		10	11

The tables given show that, beginning with number 4, we have the inconsistency of places of non-zero elements.

Numbers of places of the elements in Tables 2 and 3 are used in establishing the corresponding one-dimensional arrays containing the reduced Crout formulas.

Descriptions of the arrays:

F1 – array containing numbers of places of the elements of matrices L and A in the substitution $l_{im} = a_{im}$ (all components of the sum $\sum l_{ik}u_{km}$ are equal to zero).

F2 – array containing information about formula (4) for $a_{im} = 0$, i.e. successively, the number of non-zero components in the sum $\sum l_{ik}u_{km}$, the numbers of places of the elements l_{ik} and u_{km} occurring in non-zero components and the number of place of the element l_{im}.

F3 – array containing the corresponding numbers in formula (4), i.e., successively, the number of non-zero components in the sum $\sum l_{ik}u_{km}$, the number of place of the element a_{im}, the numbers of places of the elements l_{ik} and u_{km} in the non-zero components and the number of place of the element l_{im}.

G1, G2, G3 – arrays containing the analogous informations, respectively, concerning only to formula (5) (without the number of place of the element of the diagonal l_{kk} – see array $D[1:N]$). This way of placing the information in mentioned arrays has an essential influence on the abbreviation of the operation time of the procedure decompose.

F – array containing for every step, i.e. for $m = 1, 2, \ldots, N$, the number of repeated applications of formulas given in arrays *F1*, *F2*, *F3*, *G1*, *G2* and *G3*.

D – array containing the successive informations: $D[1:N]$ – numbers of places of the elements l_{kk} occurring on the diagonal; $D[N+1:2\times N]$ – zeros if we apply the simplified formula $y_i = b_i/l_{ii}$ (all components of the sum $\sum l_{ik}y_k$ are equal to zero, see (6)) or numbers equal to the number of non-zero components in the sum $\sum l_{ik}y_k$; $D[2\times N+1:3\times N]$ – numbers equal to the number of non-zero components in the sum $\sum u_{ik}x_k$ (see formula (7)).

B1 – array containing the numbers of places of elements l_{ik} and y_k, respectively, occurring in the non-zero components of the sum $\sum l_{ik}y_k$ in formula (6).

U – array containing the numbers of places of the elements u_{ik} and x_k in non-zero components of the sum $\sum u_{ik}x_k$ in formula (7).

These arrays allow us, without any additional examinations, to perform the operations on the simplified Crout formulas, which makes short the operation time of the procedures *decompose* and *solve.*

Arrays *owl*, *owp*, *okg* and *okd* contain the numbers determining the distance of non-zero elements occurring at the farthest from the diagonal in rows and columns, respectively, of the matrix of system (1). The procedures *size* and *forgen* using those arrays do not examine the whole zero-one matrix M, which considerably gains time.

4. Application of auxiliary procedures.

M01 — determines, in virtue of the information in the array R, the zero-one matrix M and places it on bits of computer words; it produces also the arrays *owl*, *owp*, *okg* and *okd*,

subst1 — changes into 1 the K-th bit of variable P,

bitword — evaluates the number of bit and the address of the words for a non-zero element,

bit1 — has the value **true** if the K-th bit of variable P is 1, and, in the contrary case, the value **false**,

size — evaluates dimensions of arrays occurring in the procedure *forgen*,

forgen — produces the reduced Crout formulas,

decompose — decomposes the matrix A into triangle matrices L and U,

solve — solves the system using simplified formulas (6) and (7).

References

[1] D. K. Faddeev and V. N. Faddeeva (Д. К. Фаддеев и В. Н. Фаддеева), *Вычислительные методы линейной алгебры*, Москва 1963.

[2] F. G. Gustavson, W. Liniger and R. Willoughby, *Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations*, J. ACM 1 (1970), p. 87-109.

[3] K. Jerzykiewicz and J. Szczepkowicz, *ALGOL 1204*, Warszawa 1972.

DEPARTMENT OF MATHEMATICS
GRADUATE SCHOOL OF ENGINEERING
BYDGOSZCZ

Received on 26. 6. 1972;
revised version on 13. 10. 1973

ALGORYTM 37

F. PANKOWSKI (Bydgoszcz)

ROZWIĄZYWANIE RZADKICH UKŁADÓW RÓWNAŃ LINIOWYCH

STRESZCZENIE

Procedura *sparsesystem* wytwarza zredukowane wzory Crouta i następnie rozwiązuje rzadkie układy algebraicznych równań liniowych postaci $Ax = b$.

Dane:

N – liczba równań i niewiadomych,

$N1$ – liczba elementów niezerowych macierzy A układu bez elementów przekątnej, które z założenia powinny być zawsze różne od zera,

$R[1 : N+N1]$ – tablica numerów kolumn elementów niezerowych macierzy A (bez elementów przekątnej) uporządkowanych rosnąco w wierszu i podanych wierszami; numery kolumn każdego wiersza należy zakończyć liczbą zero,

E – liczba o wartości równej zeru lub liczbie powtórzeń przy wielokrotnym rozwiązywaniu układu, przy czym zmieniają się wartości elementów niezerowych macierzy A; jeśli $E \neq 0$, to $L = 0$ (patrz niżej),

L – liczba o wartości równej zeru lub liczbie powtórzeń przy wielokrotnym rozwiązywaniu układu, przy czym zmienia się tylko jego prawa strona; jeśli $L \neq 0$, to $E = 0$,

wl – długość słowa maszynowego w bitach,

$A[1 : N+N1]$ – tablica elementów niezerowych macierzy A układu; powinny one być wybrane bez zmiany porządku z ciągu

$$a_{11}, a_{21}, \ldots, a_{n1},$$
$$a_{12}, a_{13}, \ldots, a_{1n},$$
$$a_{22}, a_{32}, \ldots, a_{n2},$$
$$a_{23}, a_{24}, \ldots, a_{2n}, \ldots,$$

$B[1 : N]$ – tablica współczynników prawej strony układu,

$X[1 : N]$ – tablica rozwiązań układu,

readb – nazwa procedury bez parametru, której wywołanie powinno umieścić odpowiednie elementy w tablicy $B[1 : N]$,

readm – nazwa procedury bez parametru, której wywołanie powinno umieścić odpowiednie elementy w tablicy $A[1 : N+N1]$,

printx – nazwa procedury bez parametru, której wywołanie powinno udostępnić użytkownikowi aktualne wartości tablicy $X[1 : N]$.

Wyniki:

otrzymuje się za pośrednictwem procedury *printx*, która musi być opisana przez użytkownika.

Przeprowadzone doświadczenia numeryczne na m.c. ODRA 1204 wykazały, że procedura *sparsesystem* jest najefektywniejsza wówczas, gdy albo zmieniają się wartości niezerowych elementów macierzy A przy nie zmienionej strukturze tej macierzy, albo gdy macierz A jest ustalona, a zmieniają się tylko prawe strony układu.