

Appendix III Chernikova's Localisation Algorithm

```
options notes linesize=128 mprint macrogen symbolgen ;
* -----
Canadian (Chernikova) Error Localisation Algorithm for equality/inequality linear
edits written in SAS.
SMD, Central Statistics Office, July 2000

Note: In practice this algorithm is difficult to use -
      you should discuss any proposed application with SMD first.

This algorithm finds the minimum number of variables that should be changed
so that a record that failed the edits will pass if the only those variables
identified by the algorithm are changed.

The example here has 19 variables labelled x1-x19.

Inputs required:  dataset called MATRIX containing the edit rules,
                  dataset called RHS containing the right hand side
                  of the edit rules
                  dataset called INDATA of input data to be edited
                  with variables labelled x1-xN
                  a vector called TYPE having a 1 to identify equality
                  edits and a 0 to identify inequality edit.
                  a vector of weights _W of 1's, a value greater than 1
                  can be used for a variable requiring more emphasis
* ----- ;
proc iml ;
* -----
  ----- Chernikova's Error Localisation Algorithm -----
* ----- ;
start genpat ;          * computes generalized patterns ;
  me = (_pq-p-1)/2 ;
  do i = 1 to ncol(Y) ;
    do j = p+1 to p+me ;
      _c = _c // max(Y[j,i],Y[j+me,i]) ;
      if _c[nrow(_c)] ^= 0 then _c[nrow(_c)] = 1 ;
    end ;
    _gp = _gp || _c ;
    free _c ;
  end ;
  free me ;
finish genpat ;

start cardf(_T) global(_pq, p, _w) ; * computes cardinality ;
  me = (_pq-p-1)/2 ;
  kd = repeat(0,1,ncol(_T)) ;
  do i = 1 to ncol(_T) ;
    do j = p+1 to p+me ;
      tkd=0 ;
      if _T[j,i] ^= 0 then tkd = tkd+1 ;
      if _T[j+me,i] ^= 0 then tkd = tkd+1 ;
      tkd = tkd * _w[j-p] ;
      kd[i] = kd[i] + tkd ;
    end ;
  end ;
  free me tkd ;
  return(kd) ;
finish cardf ;

start compl ;          * computes complementary condition ;
  me = (_pq-p-1)/2 ;
  cp = repeat(0,1,ncol(Y)) ;
  do i = 1 to ncol(Y) ;
    if Y[_pq,i] ^= 0 then do ;
      do j = p+1 to p+me ;
```

```

        cp[i] = cp[i] + Y[j,i]*Y[j+me,i] ;
    end ;
end ;
end ;
finish compl ;

start cherniko ;      * Chernikova's Algorithm begins here ;
p = nrow(U) ;
q = ncol(U) ;
Y = U // I(q) ;
free U ;
_pq = nrow(Y) ;
success = 0 ;
iter = 0 ;
do until (success>0) ;
    iter=iter+1 ;
    n = ncol(Y) ;
    k=maxk ;          * max cardinality stops column growth ;

    * 1.1 cardinality reset ;
    if iter > 1 then do ;
        if loc(Y[_pq,]) then
            do ;
                _U = Y[,loc(Y[_pq,])] ;
                kd = cardf(_U) ;
                kmin = min(kd) ;
                kmax = max(kd) ;
                if kmin > 0 then k = min(kmin,k) ;
                free kmin _U kd ;
            end ;
        end ;

    * 1.2 cardinality does not satisfy complementary condition - drop column ;
    run compl ;
    if any(cp) then do ;
        chkp = repeat(0,1,ncol(Y)) ;
        do i = 1 to ncol(Y) ;
            if Y[_pq,i] = 0 then chkp[i] = 1 ;
            else
                if (Y[_pq,i] > 0) & (cp[i] = 0) then chkp[i]=1 ;
            end ;
            if loc(chkp) then Y = Y[,loc(chkp)] ;
            n = ncol(Y) ;
        end ;
        free cp chkp ;

    * 1.3 cardinality too large then drop column ;
        kd = cardf(Y) ;
        if any(kd) then do ;
            Y = Y[,loc(kd<=k)] ;
            n = ncol(Y) ;
        end ;
        free kd ;

    * A2.3 same generalised pattern drop column ;
    run genpat ;
    tz = repeat(1,1,ncol(Y)) ;
    do j = 1 to ncol(Y) ;
        if Y[_pq,j] = 1 then do ;
            do kk = 1 to ncol(_gp) ;
                if (j ^= kk) & _gp[,j] = _gp[,kk] then tz[kk] = 0 ;
            end ;
        end ;
    end ;
    if any(tz) then do ;
        Y = Y[,loc(tz)] ;
        n = ncol(Y) ;
    end ;
end ;

```

```

free _gp tz;

* 2. check for UNSUCCESSFUL algorithm termination ;
do i = 1 to p ;
  if type[i]=0 & all(Y[i,] < 0 ) then success = 2 ;
  if type[i]=1 & all(Y[i,] ^= 0 ) then success = 2 ;
end ;
if success = 2 then goto lfail ;

* 3. find a pivot row ;
* get failed rows ;
if iter = 1 then _Y = Y[,loc(Y[_pq,])] ;
do i = 1 to p ;
  it = 0 ;
  do j = 1 to ncol(_Y) ;
    if type[i] = 1 & _Y[i,j]^=0 then it = i ;
    if type[i] = 0 & _Y[i,j]<0 then it = i ;
  end ;
  _f = _f // it ;
end ;
_f = _f[loc(_f)] ;
free _Y ;

if r then r=0 ;
if zr then free zr ;
do i = 1 to nrow(_f) ;
  nz=0 ; np=0 ; nq=0 ;
  do j = 1 to ncol(Y) ;
    if Y[_f[i],j] = 0 then nz = nz+1;
    if Y[_f[i],j] > 0 then np = np+1;
    if Y[_f[i],j] < 0 then nq = nq+1;
  end ;
  if nq = 0 then wq = 1 ; else wq = nq ;
  if type[i] = 0 then   zr = zr // nz + np + np*nq ;
  else                 zr = zr // nz + np*nq ;
  zr[nrow(zr)] = wq * zr[nrow(zr)] ;
end ;

zr = rank(zr) ;
do i = 1 to nrow(zr) ;
  if zr[i]=1 then do ;
    r = _f[i] ;
    i = nrow(zr)+1 ;
  end ;
end ;

btime=time() ;
n = ncol(Y) ;

* 4. create _Y (new from cols of Y with Y[r,j]>=0 (ineq) or Y[r,j]=0(eq) ;
if type[r] = 1 then _Y = Y[,loc(Y[r,]=0)] ; * equality pivot row ;
else _Y = Y[,loc(Y[r,]>=0)] ; * inequality pivot row ;

* 5.1 add extra column to _Y if Y has only 2 columns ;
if n = 2 then
  _Y = _Y || Y[,1]*abs(Y[r,2]) + Y[,2]*abs(Y[r,1]) ;

* 5.2 index set of opposite sign pairs products in pivot row ;
if n > 2 then
do s = 1 to n ;
  do t = s+1 to n ;
    if Y[r,s]*Y[r,t] < 0 then do ;
      st = s//t ;
      _S = _S || st ;
    end ;
  end ;
end ;
end ;
free s t st ;

```

```

i0 = repeat(0,_pq,1) ;
do i = 1 to _pq ;
    if all(Y[i,]>=0) then i0[i,1]=i ;
end ;
i0 = loc(i0>0) ;
* step 5.2.1,2 ;
do j = 1 to ncol(_S) ;
    if i1 then free i1 ;
    do i=1 to ncol(i0);
*set i1 = For each element (s,t) of S, find all i of I0 s.t. yis=yit=0;
        if ((Y[i0[i],_S[1,j]]=0) & (Y[i0[i],_S[2,j]]=0)) then i1=i1 || i0[i];
    end;
    if i1 then
        do ;
            do i = 1 to ncol(i1) ;
                nz=0 ;
                do lu = 1 to n ;
                    if ( (lu ^= _S[1,j]) & (lu ^= _S[2,j]) ) & (Y[i1[1,i],] = 0)
                        then nz=nz+1 ;
                end ;
                if nz = 0 then
                    _newcol = Y[_S[2,j]]*abs(Y[r,_S[1,j]]) +
                        Y[_S[1,j]]*abs(Y[r,_S[2,j]]) ;
                end ;
                _Y = _Y || _newcol ;
                free _newcol ;
            end ;
        end ;
        free Y ;
        Y = _Y ;
    end ;

*Scale the columns so that the last entry equals one;
do i=1 to ncol(Y);
    if Y[_pq,i] ^= 0 then Y[,i]=Y[,i]/Y[_pq,i];
end;

    free i0 i1 _S _Y ;

* drop processed row ;
free tz ;
tz = 1:nrow(Y) ; tz[r] = 0 ;
Y = Y[loc(tz),] ;
free tz ;
tz = 1:nrow(type) ; tz[r] = 0 ;
type = type[loc(tz)] ;
p = p-1 ;
_pq = _pq-1 ;
free tz ;

* check for SUCCESSFUL algorithm termination AND SOLUTION;
if loc(Y[_pq,]) then do ;
    _Y = Y[,loc(Y[_pq,])] ;
    schk=0 ;
    do i = 1 to p ;
        if type[i] = 1 & any(_Y[i,]^=0) then schk=1 ;
        if type[i] = 0 & any(_Y[i,]<0) then schk=1 ;
    end ;
    free _f ;
    if schk=0 then success = 1 ;
end ;
else do ;
    success = 4 ;
    goto lfail ;
end ;

time=time()-btime ;
itime = itime+time ;

```

```

        if itime > 60 then do ;
            success = 3 ;
            goto lfail ;
        end ;

end ; * until loop ;

if success = 1 then
do ;
    if loc(Y[_pq,]) then
    do ;
        Y = Y[,loc(Y[_pq,])] ;
        me = (_pq-p-1)/2 ;
        L = repeat(0,me,ncol(Y)) ;
        do i = p+1 to p+me ;
            L[i-p,] = Y[i,] - Y[i+me,] ;
        end ;
        xout = xout[,1] + L[,1] ;
        xchg = L[,1] ;
        do il = 1 to nrow(xchg) ;
            if xchg[il] ^=0 then xchg[il]=1 ;
        end ;
        tchg = iobs || xchg` ;
        chg = chg // tchg ;
        tim = iobs || k || itime ;
        utim = utim // tim ;
    end ;
end ;

lfail:
    if success = 2 then
        print iobs ' Termination condition failure - algorithm stopped ' ;
    if success = 3 then
        print iobs ' Termination condition failure - insufficient time ' ;
    if success = 4 then
        print iobs ' Termination condition failure - no solution with cardinality <= '
maxk ;

finish ;
* -----
* ----- end of Chernikova Algorithm code -----
* ----- ;

use indata ;          /* input dataset with variables labelled x1-xN ;
read all var _num_ into _inp ;

use matrix ;
read all var _num_ into A ;
use rhs ;
read all var _num_ into _rhs ;
free / A _rhs _inp ;

* set up type and _w here ;
type = repeat(0,nrow(A),1) ;    * all inequality edits by default ;
type = type // repeat(0,ncol(A),1) ;
* if edit number 5 for example is an equality then include the code ;
* type[5]=1 ;
_w = repeat(1,1,ncol(A)) ;      * all equal weights by default ;
maxk = 5 ;                      * maximum cardinality allowable ;
* ----- ;

do iobs = 1 to nrow(_inp) ;      * for each row of input dataset ;
    x = _inp[iobs,]` ;

    * 1. perform edits checks ;
    _i = A*x- b;

do it = 1 to nrow(type) ;

```

```

        if type[it] = 1 then
        do ;
            if _i[it] = 0 then _i[it] = 1 ; else _i[it]=0 ;
        end ;
        else
        do ;
            if _i[it] <= 0 then _i[it] = 1 ; else _i[it]=0 ;
        end ;
    end ;
    if all(_i>0) then status=1 ; else status=0 ;

* status = 0 is failed record so run error localisation algorithm ;
    if status = 0 then
    do ;
        ecount=ecount+1 ;
        itime = 0 ;

        * set up the upper matrix U for use by Chernikova algorithm ;
        U1 = -A || A || (b - A*x ) ;
        U2 = I(ncol(A)) || -I(ncol(A)) || x ;
        U = U1 // U2 ;
        free U1 U2 ;
        run cherniko ;
    end ;
    else do ;
        free xout ;
        xout = iobs || _inp[iobs,] ;
    end ;
    free x xout xchg _i ;
end ;
print ecount ;
print err [format=4.0] ;
cname = {_n x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19} ;
create utdat from chg [colname=cname] ;
append from chg ;
tname = {_n k itime } ;
create timdat from utim [colname=tname] ;
append from utim ;
quit ;
run ;
* ----- END ----- ;

```