

J.-P. BENZÉCRI

F. BENZÉCRI

**Sources de programmes d'analyse de données
en langage PASCAL : (III) : élaboration de
tableaux divers (IIIA) : observer, ranger,
découper, copier les données**

Les cahiers de l'analyse des données, tome 22, n° 3 (1997),
p. 245-270

<http://www.numdam.org/item?id=CAD_1997__22_3_245_0>

© Les cahiers de l'analyse des données, Dunod, 1997, tous droits réservés.

L'accès aux archives de la revue « Les cahiers de l'analyse des données » implique l'accord avec les conditions générales d'utilisation (<http://www.numdam.org/conditions>). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

NUMDAM

Article numérisé dans le cadre du programme
Numérisation de documents anciens mathématiques

<http://www.numdam.org/>

**SOURCES DE PROGRAMMES
D'ANALYSE DE DONNÉES EN LANGAGE PASCAL:
(III) : ÉLABORATION DE TABLEAUX DIVERS
(IIIA) : OBSERVER, RANGER, DÉCOUPER, COPIER
LES DONNÉES**

[SOURCES PASCAL (IIIA)]

J.-P. & F. BENZÉCRI

0 Élaboration des données avant analyse: le programme 'zrang'

Nous avons dit, en IA§0, que, même en se bornant à ce que nous considérons comme indispensable pour élaborer, après un codage approprié, le tableau décrivant quelques centaines d'individus de diverses classes par un ensemble hétérogène de variables, il faut utiliser une dizaine de programmes différents. Or, pour élaborer un codage, nous passons toujours, et souvent plusieurs fois de suite, par le programme 'zrang'.

Tel qu'il est, 'zrang' n'a ni toutes les fonctions ni, encore moins, la présentation qu'on souhaiterait pour le tableur de l'avenir; mais nous y avons introduit, par étapes, des fonctions indispensables qu'on ne trouve pas dans les logiciels usuels; notamment l'affichage répété d'histogrammes, embrassant un intervalle arbitrairement choisi de la variable considérée; avec, dans les créneaux, les sigles des individus qui y sont compris.

1 Le programme principal

À la différence du programme d'analyse de correspondance (cf. IB) et de celui de classification ascendante hiérarchique (cf. IIB), le programme 'zrang' comporte, non seulement une hiérarchie de procédures, mais un programme principal, long et complexe en lui-même; que nous considérerons d'abord.

Sans prétendre que l'ordre de 'zrang' ne pourrait être montré plus clairement si on allégeait le programme principal en créant des procédures initiale et finale, nous dirons que l'observation et le codage des données est régi par un dialogue complexe, plus ou moins libre selon l'option choisie par l'utilisateur.

1.1 Préparatifs pour une exécution du programme 'zrang'

1.1.1 Initialisation et choix du tableau à traiter

Après initialisation de l'écran (y compris le choix d'un rectangle: fond, qui intervient exclusivement dans l'instruction: eraserect(fond), pour effacer

```

begin rps:='O'; setrect (fond, 0, 0, 600, 500);
Benzecri;
moveto(2,10);
{****}while not (rps='N') do begin carti:=0;rps:='N';
litab(s1j,@tlk,@nomba,@titre,repr,carti, cartj, sli);
readln(rpf);rcr:='N';unloadseg(@litab);
if (repr='O') then begin writeln('carti=',carti:4,';cartj=',cartj:3);
writeln(nomba);writeln(titre);caru:=0;
for j:=0 to cartj do klji[j]:=pti(tlk[j]);siglj:=pzgi(s1j);carm:=0;
ffli:=plng(@tampi);sgli:=pzgl(@tampi);ttri:=stringptr(@tampi);rech:='N';

```

ce qui a été affiché), 'zrang' entre dans une boucle générale: while not (rps='N')... , dont la première procédure est: 'litab', (cf. IA§2), également appelée par 'qori' (cf. IB§3.1) pour choisir et mettre en mémoire centrale le tableau à traiter. On notera que: carti, nombre total des lignes est ici mis initialement à zéro; parce que comme 'qori', mais à la différence de 'cums' (programme qui crée, par cumul, des lignes et colonnes adjointes à un tableau de base: cf. IIC), 'zrang' n'ajoute pas de ligne au tableau traité.

L'instruction: readln(rpf), immédiatement après la lecture n'a d'autre rôle que d'inviter l'utilisateur à entreprendre un dialogue. Dans la suite, le caractère: rpf, est demandé dans des circonstances analogues (par exemple, pour afficher ou effacer un graphique); et la réponse n'est prise en compte que dans la procédure 'afficher', où ce qui n'était initialement qu'une réponse vide, précédant l'affichage du croisement de deux variables, sert, présentement, à choisir la représentation des points du nuage: par leur sigle entier, ou par un caractère unique, diversement choisi (cf. *infra*, §§4.2 et 5.4).

L'ordre: rcr:='N', élimine, initialement, le croisement de deux variables; qui ne sera possible que dans l'observation sans créer de tableau; l'ordre: rcr:='C', étant introduit dans le cas: rpf='Z' (cf. *infra*, §1.1.3.2).

Si a été choisi un tableau présent sur le disque: if (repr='O')..., le programme initialise des pointeurs pour accéder au contenu de ce tableau: titre, sigles et nombres.

1.1.2 Dialogue pour choisir le traitement à effectuer

Cinq voies s'offrent d'abord: transposer(T), copier(C), ranger(R), découper(D), ou observer les données sans créer de tableau(Z).

Seul: Z, affiche les variables dans un ordre quelconque (avec des répétitions; et sans contrainte imposée par l'ordre des colonnes); fait des croisements, trace éventuellement des cartes. Avec: D, ou:R, on doit suivre l'ordre des colonnes; Z, est aussi l'option prise par défaut, pour un caractère de réponse quelconque.

Les opérations: C, et: T, ont été mises dans 'zrang' pour permettre de consulter comme des textes des tableaux codés en format binaire ('.z', ou: '.w'); ou pour mettre un tableau sous la forme requise par 'qorlsup'

```

while not (rech='O') do begin rab:='N';
  writeln('faut il transposer(T), copier(C), ranger(R), découper(D), ');
  write('ou observer les données sans créer de tableau(Z) ');
  readln(suf);
  if not (suf in ['C','D','R','T','Z']) then suf:='Z';
  if (suf in ['C','T']) then begin
    write('faut il écrire un fichier en format de texte(X) ou en binaire(B)
    readln(suv);
    if (suv='X') then begin
      if (suf='T') then suv:='Y';suf:=suv;
      write('faut-il écrire des nombres entiers(E) ou réels(R) ');
      readln(rpx);end;end;
  if (suf='D') then begin
    writeln('faut il découper les variables');
    write('          avec des bornes préétablies(P), ou non(N) ');
    readln(rab);suc:='z';
    while not (suc in ['B','M','Q','S']) do begin
      writeln('faut il créer un tableau booléen(B), barycentrique(Q) ');
      writeln('          un tableau des numéros des modalités(M) ');
      write('          ou un tableau de numéros continus(S) ');
      readln(suc);if not(suc in ['B','M','Q','S']) then
        writeln('ERREUR il faut répondre B, M, Q ou S') end end;
    write('ce choix est-il confirmé O ou N ');readln(rech) end;

```

(transposition: cf. ID§0); ou par 'soustab' et 'jxtab' (format binaire; pour créer des sous-tableaux d'un tableau, ou juxtaposer deux tableaux). Deux caractères: rpv, et: rpx, spécifient le format du tableau à créer.

Avec: suf='D', doit être choisi le mode de découpage.

Si l'on opère avec des bornes préétablies, i.e. données par un fichier de texte déjà présent sur disque, 'zrang' procède au codage sans rien afficher qui fasse voir les données: histogramme, carte... Le fichier des bornes peut avoir été créé lors d'un passage précédent de 'zrang' mais utilisé d'abord à d'autres fins; ou avoir été modifié ensuite; ou créé par un programme particulier tel que: 'pers', qui prépare le codage des réponses à un questionnaire, en forme d'échelle, suivant l'équation personnelle des sujets, sans que l'utilisateur considère la distribution des réponses aux questions, prises une par une.

De plus, par le caractère: suc, doit être spécifié, d'une part, le mode de découpage: disjonctif, en (0, 1), avec, pour les modalités de chaque variable, des intervalles disjoints; ou découpage barycentrique, avec, pour un individu compris entre deux pivots, des notes non nulles sur chacune des modalités afférentes à ces pivots. Et, d'autre part, est spécifié le type du tableau à créer: tableau de numéros de modalité, avec une seule colonne pour chacune des variables initialement données; (qu'il s'agisse de numéro entier: M, pour le format disjonctif; ou de numéro continu: S, pour le format barycentrique); ou tableau éclaté, avec, pour chacune des variables initiales, un bloc de colonnes dont chacune se rapporte à une modalité (soit: B, tableau booléen, en (0, 1); soit: Q, tableau continu, avec, en général, pour chaque individu, dans le bloc de chaque variable, deux notes consécutives non nulles dont la somme est 1).

On sait que les tableaux de numéros (quel qu'en soit le format: entier: M, ou continu: S) servent au programme 'zBurt' pour construire le tableau de BURT ou sa généralisation au cas continu. Tandis que les tableaux éclatés (de format: B, ou: Q), qui représentent le nuage des individus (lignes) peuvent être analysés directement; ou, avec des résultats équivalents, adjoints en supplément à l'analyse du tableau de BURT. L'avantage, dans cette dernière voie, est que, d'une part, on a une appréciation plus favorable de la décroissance des taux d'inertie dans la suite des facteurs; et que, d'autre part, on fait commodément des analyses mettant en supplément des modalités: par exemple les modalités centrales de réponses codées suivant trois pivots.

1.1.3 Vérifications quant à l'existence de fichiers

1.1.3.1 Fichier de bornes pour le codage des variables

Selon le mode de découpage, on parle, pour une modalité, de valeur maxima, ou de valeur pivot; cette distinction est notée dans la chaîne: mxpv.

```

if (suf='D') then mxpv:=' maxima ';
if (suc in ['Q','S']) then mxpv:=' pivot ';
if (rab='P') then begin nomf:=concat(nomba,'Dcodx');
  if not (verif(stringptr(@nomf))=2) then begin rab:='N';suf:='Z';
    writeln('ERREUR il n y a pas de fichier Dcodx') end end;
if (rab='P') then begin
  reset(ft,nomf);
  if not eof(ft) then readln(ft,chaîne);
  if not eof(ft) then readln(ft,chaîne);
  carac:=' ';lecnombre;
  if not (ffl=cartj) then begin close(ft);rab:='N';suf:='Z';
    write('ERREUR de Dcodx cartj=',cartj:3,' et non',ffl:4);
    readln(rp0);end end;
  if (rab='P') then begin j:=0;
    while not (j=cartj) do begin j:=j+1;carac:=' ';lecsig;
      if not (sigler(siglj^[j])=sigler(sig)) then begin
        rab:='N';suf:='Z';
        write('ERREUR de Dcodx siglj^[j]= ');
        write(sigler(siglj^[j]),' et non ',sigler(sig));readln(rp0);
        j:=cartj end
      else begin
        lecnombre;m2:=ffl;
        if not eof(ft) then readln(ft,chaîne);carac:=' ';
        for m1:=1 to m2 do lecsig;
        for m1:=1 to m2 do if not eof(ft) then read(ft,ret) end end;
        close(ft) end;
    if (rab='P') then begin suf:='d';reset(ft,nomf);
      writeln('le fichier Dcodx semble correct');
      readln(ft,chaîne);readln(ft,chaîne);carac:=' ';lecnombre end;
    if (suf='D') then begin rewrite(ft,concat(nomba,'Dcodx'));
      writeln(ft,titre);
      writeln(ft,nomba,'Dcodx: bornes pour le découpage des variables');
      writeln(ft,'le nombre des variables est',cartj:4) end;

```

Sous la condition: rab='P', le programme doit s'enquérir de l'existence d'un fichier de bornes préétabli; fichier de texte dont le nom est formé en ajoutant à celui du tableau de base le suffixe: 'Dcodx'. Comme on le voit sur un exemple, le fichier: 'Dcodx', a, après deux lignes de titre, une ligne où peut être lu le nombre des variables. Puis, il y a, pour chacune de celles-ci, un

```

enquête sociologique:
Disq:tabDcodx: bornes pour le découpage des variables
le nombre des variables est 24
V01 a 3 modalités dont les sigles et valeurs pivot sont
      01<          01=          01>
-1.000000000e+0 -1.264458412e-1 1.000000000e+0
V02 a 3 modalités dont les sigles et valeurs pivot sont
      02<          02=          02>
-1.000000000e+0 2.756822154e-1 1.000000000e+0

```

alinéa dont la première ligne donne, après le sigle, le nombre des modalités, et se termine par un membre de phrase dont seule la 1-ère lettre après le blanc est lue comme une commande (cette lettre est, en général: 'm'; mais peut, par exception, être: 'µ', cf. *infra*, §1.2.2). Le reste de l'alinéa énumère d'abord les sigles des modalités; puis les bornes ou pivots; avec, si le fichier: 'Dcodx', est créé par 'zrang', une disposition régulière; mais dont on peut s'affranchir si on modifie ou crée de toutes pièces un fichier de commande.

Plusieurs instructions composées servent à vérifier la conformité de: 'Dcodx', avec le tableau à traiter, quant au nombre des variables et à leurs sigles. En cas d'échec, le programme affiche un commentaire d'erreur, et suit son cours pour l'observation exclusive des variables (rab:='N'; suf:='Z'). En cas de succès, le caractère de commande: 'D', est changé en une minuscule (par l'instruction: suf:='d'); et le fichier: 'Dcodx', déjà consulté une première fois et fermé, est rouvert en lecture.

Si (suf='D') avec (rab='N'), il revient à l'utilisateur de considérer, successivement, toutes les variables, en fixant des bornes ou pivots écrits dans un fichier: 'Dcodx', créé en écriture, à cet effet, avec ses trois premières lignes.

1.1.3.2 Préparatifs pour tracer des cartes

Avec: (suf='Z'), observation des données sans créer de fichier, l'occasion est offerte de créer des croisements de variables (instruction: rcr:='C'); et éventuellement, des cartes. Il faut, pour créer des cartes, que l'ensemble I, des lignes du tableau considéré, soit un ensemble d'unités territoriales, un pays (ou

```

if (suf='Z') then begin rcr:='C';
write('les individus i sont-ils des unités territoriales O ou N ');
readln(rp2);erl:=0;
if not(rp2='N') then while not((rpl='O') or (erl=6)) do begin
write('le nom du disque utilisé est ');readln(disq);
if (copy(disq,length(disq),1)=':') then
disq:=copy(disq,1,length(disq)-1);
write('le nom du pays est ');readln(pays);
carton:=concat(disq,':',pays,'reg');
rpl:=dialof(pnf);
if (rpl='O') then begin
reset(fin,carton);read(fin,caru);close(fin);
if not(caru=carti) then begin rpl:='N';
writeln('ERREUR: le nombre des unités ne concorde pas');
writeln('avec le nombre des lignes du tableau traité');end;end;
if (rpl='N') then erl:=erl+1 else begin
write('ce choix est il confirmé O ou N ');readln(rpl);
eraserect(fond);moveto(2,10);writeln;end end end;
if (rpl='O') then begin tramer;polygoner;moveto(2,260);writeln end;

```

tout autre territoire divisé: comme une ville en arrondissements) pour lequel existe un fichier de fond de carte, caractérisé par le suffixe: 'reg'. Nous reviendrons, dans un autre article, sur le format des fichiers: 'reg' et leur création. Ici, nous dirons seulement que, dans la procédure: 'histog', une boucle: `if (rpl='O') and (caru=carti)...`, affiche une carte après chaque tracé d'histogramme, sous réserve qu'on ait trouvé un fichier: 'reg' (`rpl='O'`), pour lequel le nombre: `caru`, des unités territoriales, soit égal au nombre: `carti`, des lignes. La procédure 'tramer', prépare les trames, en gris allant du blanc au noir, afin de couvrir, sur une carte, les unités comprises dans les créneaux successifs de l'histogramme; tandis que 'polygoner' prépare le tracé rapide de chaque région, comme d'un polygone (décrit selon le format propre au MacIntosh: cf. *infra*, §§2.3 et 3.3).

1.1.4 Affectation de zones en mémoire centrale pour les tris

```
if (suf in {'D','R','Z','d'}) then begin
  ptr1:=newptr(2*carti);pla:=ptni(ptr1);
  ptr1:=newptr(2*carti);plb:=ptni(ptr1);
  ptr1:=newptr(2*carti);plc:=ptni(ptr1);
  sigli:=pzgi(sli);
  ptr1:=newptr(4*carti);plV:=pti(ptr1);
  ptr1:=newptr(4*carti);plW:=pti(ptr1);end;
```

Ces zones sont requises pour tous les traitements, excepté la simple copie ou transposition: `suf='C'`, ou: `suf='T'`.

1.2 Les divers traitements des variables par le programme 'zrang'

Cette partie consiste, essentiellement, en l'exécution de procédures données au §2.

1.2.1 Observation des variables sans créer de fichier

```
if (suf='Z') then begin rav:='O';
  while not (rav='N') do begin
    writeln('ci-dessous sont rappelés les num et sigles des variables');
    for j:=1 to cartj do begin write(j:3,sigler(siglj^[j]):5,',');
      if (j mod 9=0) then begin writeln;
        if ((j mod 207=0) and (j<cartj)) then begin
          write('pour afficher la suite des sigles entrer *');
          readln(xpf) end end end;
        if not (cartj mod 9=0) then writeln;
        write('le numéro de la variable à afficher est ');readln(j);
        if (j<1) then j:=1;if (cartj<j) then j:=cartj;
        for i:=1 to carti do plV^[i]:=klji[j]^i;
        trirs(@pla^,@plb^,@plV^,@plW^,carti);
        afficher(j);
        write('faut il afficher une autre variable O ou N ');
        readln(rav) end end;
```

Dans ce cas, l'exécution est prise dans une boucle: `while not (rav='N')`, où, après affichage des numéros et sigles des variables, offertes au choix de l'utilisateur, les valeurs de la variable sont triées, par la procédure 'trirs' (cf. *infra*, §2.5), puis affichées à l'écran, selon une procédure qui, ainsi qu'on le verra, comporte non seulement le tracé d'une suite indéfinie d'histogrammes (éventuellement accompagnés de carte), mais aussi, sur un plan et un tableau de tri, le croisement avec la variable précédemment affichée.

1.2.2 Traitement propre au découpage des variables

```

if (suf in ['D', 'R', 'd']) then begin
  for j :=1 to cartj do begin
    for i:=1 to carti do p1V^[i]:=k1ji[j]^i;
    trirs(@pla^,@plb^,@plV^,@plW^,carti);
    if (suf='d') then coupixer(j) else afficher(j);
    if ((suf='R') or ((suf in ['D', 'd']) and (suc in ['M', 'S']) and (mm=0)))
      then rangcorriger;
    if (suf='D') then begin
      write(ft, sigler(siglj^[j]):4, ' a', mm:3);
      if not(suc in ['Q', 'S']) then rbar:='z';
      if ((rbar='μ')) then write(ft, ' μ') else write(ft, ' m');
      if ((rbar='μ')) then chaine:=' limite ' else chaine:=mxxpv;
      write(ft, 'odalités dont les sigles et valeurs', chaine, 'sont');
      ct:=0;
      for c:=carm+1-mm to carm do begin
        if (ct mod 5=0) then writeln(ft); ct:=ct+1;
        write(ft, sigler(sicj[c]):16) end;
        if (ct=0) then writeln(ft); ct:=0;
        for c:=carm+1-mm to carm do begin
          if (ct mod 5=0) then writeln(ft); ct:=ct+1;
          write(ft, ' ', brm[c+mm-carm]:10) end;
          if (ct=0) then writeln(ft); writeln(ft) end;
        for i:=1 to carti do k1ji[j]^i:=p1W^[i] end end;

```

Dans ce cas, l'exécution rentre dans une boucle: for j :=1 to cartj do..., toutes les variables étant prises, successivement, dans l'ordre du numérotage.

Comme quand: suf='Z', l'instruction: $p1V^i := k1ji[j]^i$, place un pointeur sur la colonne à trier. Si l'on découpe les variables suivant des bornes préétablies: (suf='d'), la colonne considérée est, sans affichage, codée, par 'coupixer', dans l'espace affecté au pointeur: p1W. Sinon, s'exécute: 'afficher', sans présenter de carte ni de tri croisé, mais (si: suf='D') en requérant de l'utilisateur de fixer les modalités d'un codage, par sigles et bornes (ou pivots); les numéros de modalités étant gardés dans: p1W^.

Dans le cas particulier où: suf='R', ou dans le découpage suivant zéro modalité (i.e. pour une variable virtuellement exclue du découpage, mais représentée par une colonne dans le tableau de numéros des modalités), la procédure: 'rangcorriger', met dans le tableau: p1W^, les rangs multipliés par 2, de telle sorte qu'on puisse, en terme de valeurs entières, donner à un bloc de valeurs égal un rang moyen unique (cf. *infra*, §6.2).

La boucle: if (suf='D') then..., inscrit dans le fichier: 'Dcodx', l'alinéa du codage de la variable traitée (cf. *supra*, §1.1.3.1); la lettre: 'μ' (dans le mot: modalité), servant à spécifier, dans le cas du codage barycentrique, l'exception d'une variable découpée disjonctivement, en (0, 1).

Quand le traitement est achevé, les valeurs, rangs ou numéros de modalités, créées pour la variable: j, dans l'espace: p1W^, sont transférées dans la colonne: $k1ji[j]^i$, à la place des données initiales.

1.3 Sortie du tableau créé et fin de l'exécution

1.3.1 Gestion de l'espace en mémoire centrale

```

if not (suf in ['C','T','X','Y']) then unloadseg(@afficher);
if (suf in ['D','R','Z','d']) then begin dispose (plV);dispose (plW);
dispose (pla);dispose (plb);dispose (plc) end;
if (suf in ['D','d']) then begin close (ft);suf:=suc end;
if (suf in ['B','Q']) and (cartj<carM) then
  for j:= cartj+1 to carM do begin
    k1ji[j]:=pti(newptr(4*(carti+2)));t1k[j]:=@k1ji[j]^ end;

```

La procédure 'afficher', qui a fait l'essentiel du traitement des variables une par une, est déchargée de la mémoire centrale. L'espace ayant servi aux tris est libéré. Si l'on a écrit un fichier: 'Dcodx', celui-ci est fermé.

Les deux cas de découpage, par bornes préétablies: suf='d', ou en dialogue: suf='D', se rejoignent: désormais, seul doit être distingué le format du tableau à créer, noté dans le caractère: suc, avec l'ensemble des quatre modalités: {'B','M','Q','S'}; et: suf, prend la valeur de: suc.

Dans le cas où est créé un tableau booléen ou barycentrique (suf='B', ou: suf='Q'), avec, pour chacune des variables initialement données, un bloc de plusieurs colonnes, l'espace requis est affecté, en mémoire centrale, à de nouveaux pointeurs: k1ji[j].

1.3.2 Calcul des blocs de colonnes afférents aux variables

```

if (suf in ['B','Q']) then for i:=1 to carti do begin
  for ml:=1 to carM do lim[ml]:=0;
  for j:=1 to cartj do copj[j]:=round(0+k1ji[j]^i);mm:=0;
  for j:=1 to cartj do if not (nmj[j]=0)
    then begin lim[mm+copj[j]]:=1;
  if (suf='Q') then begin eps:=1;
    col:=k1ji[j]^i-copj[j];
    if (col<-0.01) then begin eps:=-1;col:=-col end;
    if (col<0.01) then col:=0;
    lim[mm+copj[j]]:=1-col;lim[mm+copj[j]+eps]:=col end;
  mm:=mm+nmj[j] end;
  for ml:=1 to carM do k1ji[ml]^i:=lim[ml] end;

```

Pour calculer un tableau où chaque variable est éclatée suivant un bloc de colonnes, on procède ligne par ligne.

La ligne auxiliaire de réels ('extended'): lim (i.e. ligne indiquée suivant les modalités) est mise à zéro.

La suite des numéros de modalités est d'abord copiée, en format entier, dans le tableau: copj. Pour toute variable effectivement éclatée (not(nmj[j]=0), la modalité désignée par copj est mise à 1. Plus précisément, dans le cas barycentrique, la masse est partagée entre deux colonnes consécutives, l'erreur d'arrondi qui affecte: copj[j], servant à calculer le partage.

Finalement, la ligne éclatée, créée dans: lim, est copiée dans les: k1ji[ml]^i, pour lesquels a été réservé l'espace approprié (cf. §1.3.1).

1.3.3 Ecriture du tableau transformé

```

cari:=carti;carj:=cartj;ct:=0;s2j:=s1j;
if (suf in ['B','Q']) then begin carj:=carm;s2j:=@sicj end;
if (suf='T') then ct:=1;
sufx:=concat(suf,'.z');
if (suf in ['Q','S','T']) then sufx:=concat(suf,'.w');
if (suf='B') then sufx:=concat(suf,'b.z');
if (suf='C') then sufx:='.w';
if not (suf in ['X','Y','Z']) then begin
  nomf:=concat(nomba,sufx);writeln(concat(nomba,sufx));
  ttri^:=concat(sufx,':',titre);writeln(ttri^);
  if (127<length(ttri^)) then ttri^:=copy(ttri^,1,127);
  critab(s1i,s2j,@t1k,@nomf,@ttri^,cari,carj,ct);
  unloadseg(@critab)
end;
if (suf in ['X','Y']) then begin if (rpx='E') then critabex else critabx;
  unloadseg(@critabx) end;
dispose(s1i);dispose(s1j);for j:=0 to cartj do dispose(k1ji[j]);
if (suf in ['B','Q']) then for j:=cartj+1 to carm do dispose(k1ji[j]);
write('faut il reprendre avec un autre fichier O ou N ');
readln(rps) end;end;{*****}
end.

```

Excepté dans le cas de l'observation sans créer de fichier: suf='Z', le programme se termine en créant un tableau transformé. Le plus souvent, l'écriture se fait en format binaire, par la procédure: 'critab' (expliquée dans IID§5)

Avant d'appeler: 'critab', il faut fixer les nombres: {cari, carj} des lignes et colonnes du tableau à créer; ainsi que le pointeur: s2j, par lequel doivent être lus les sigles des colonnes. L'écriture des nombres est, on le sait, déterminée par la fin du nom du fichier: '.z', entiers, ou:'.w', réels. La valeur: ct=1, commande une transposition.

Les procédures d'écriture de texte: 'critabex' (format entier: §7.2), et: 'critabx' (format réel, en virgule flottante: §7.1), ne sont appelées que pour copier ou transposer le tableau donné: suf='X', ou: 'Y'.

Dans tous les cas, après écriture éventuelle d'un tableau, on libère les zones de la mémoire centrale auxquelles donnent accès des pointeurs: vers les sigles: {s1i, s1j}; ou vers les colonnes du tableau: k1ji[0..cartj].

Il faut se rappeler que, comme on l'a dit au §1.1, le programme général, après deux instructions d'initialisation, entre dans une boucle: while not (rps='N') do... Celle-ci commence en appelant 'litab' pour le choix du tableau à traiter: et si ce choix a abouti à la lecture effective d'un tableau, le traitement de celui-ci est compris dans une boucle: if (repc='O') then... L'exécution de cette boucle se termine en offrant à l'utilisateur de reprendre avec un autre fichier. Après l'ordre: readln(rps), qui reçoit la réponse à cette offre, le listage lui-même se termine en répétant trois fois le mot: end; comme signe final du: if (repc='O'), puis du: while not (rps='N'); et, enfin, du programme principal lui-même.

2 Déclarations et procédures initiales

Comme dans les exposés déjà consacrés à d'autres programmes, ayant expliqué le programme principal, nous reprenons en haut du listage du programme source. Dans le cas présent, la multiplicité des procédures, nous incite à en partager l'explication entre cinq §§, numérotés de 2 à 6.

2.1 Déclarations générales du programme 'zrang'

```

program zrang;uses memtypes,quickdraw,osintf,toolintf,sane,uver,utris;
const mqmx=50;ntmax =29;umax =500;
var fin:file of integer;ft:text;
carac,repr,rech,rpf,rps,rpx,suf,suv,suc,raf,ram,
rph,rav,rab,rp0,rpl,rp2,rcr,rpm,rsc,rbar:char;
nomba,nomf,titre,chaine,carton,mxpx:string[9];
ffl:longint;brm:array[1..mqmx]of extended;
ret,res,rr2,col,co2,mxe,mne,wal,
xgch,xdr,t,yhau,ybas,gchx,drtx,hauy,basy,xh,yv:extended;
erl,c,i,il,j,jl,izl,m1,m2,m3,mm,r1,r2,r3,rz,rrr,ia,ib,eps,lxe,lne,nre,
lig,coin,droit,bas,u,ii,n,n1,n2,nlc,n2c,Pch,Pcv,hj,wj,may,nv,wl,
cartu,cartv,uz,vz,carm,cari,carj,carj,cti,ct,caru,
s,nu,x,y,xl,yl,tl,h1,v1:integer;
sgli:pzgl;ttri:stringptr;ffli:plng;ptrl:ptr;
fond:rect;P0:point;
tampi:array[1..128]of integer;trmu:array[1..umax]of integer;
zzn:array[0..26]of integer;crCb:array[0..26,0..26]of integer;
valb,valc:array[0..26]of single;
nmj,copj:array[1..jmax]of integer;lim:array[1..jmax]of extended;
klji:kji;tlk:tk;slj,s1j,s2j:ptr;
pla,plb,plc:ptni;plV,plW:pti;
sicj:array[1..imax]of zgle;
sigli,siglj:pzgi;sql,sqe,sig,sga,sgb,sgc:zgle;
hurb:array[1..umax]of polyhandle;tram:array[0..ntmax]of pattern;
disq,pays,sufx:str255;pnf:stringptr;

```

L'unité 'utris', déclarée en tête du listage contient une procédure de tri, expliquée, ci-après, au §2.5 Le nombre maximum des modalités d'une seule variable est donné par la constante: $mqmx=50$: les bornes ou pivots de celles-ci sont dans le tableau: brm. Le nombre maximum des trames pouvant servir à marquer les unités territoriales d'une carte est: $ntmx+1=30$; en fait, on n'utilise présentement que 11 trames (cf. §1.1.3); du blanc: tram[0], au noir: tram[10] en passant par 9 gris. Le nombre maximum des unités territoriales est: $umax=500$; la forme de chacune de celles-ci est un polygone, dont la description, selon le format du Macintosh, est atteinte par un: polyhandle, du tableau: hurb. Lors du tracé d'une carte, le tableau trmu donne le numéro de la trame attribuée à chaque unité territoriale. Le nombre des créneaux d'un histogramme ne pouvant dépasser 25, il lui correspond une partition, de la suite ordonnée des valeurs d'une variable, en 27 classes au plus (y compris, éventuellement, une classe initiale et une classe finale, refermant les individus non pris en compte pour le tracé d'un histogramme partiel). C'est pourquoi, la délimitation des classes, par leur élément de rang maximum, est dans: zzn[0..26]; et le croisement des découpages en classes de deux variables, affichées consécutivement, est dans le tableau: crCb[0..26,0..26].

2.2 Procédures de lecture pour un fichier de codage

```

procedure lecsig; begin sig[0]:=0;
  while (ord(carac) in [0..32]) and not eof(ft) do read(ft, carac);
  while not ((ord(carac) in [0..32]) or (sig[0]=4)) do begin
    sig[0]:=sig[0]+1;sig[sig[0]]:=ord(carac);
    if eof(ft) then carac:= ' ' else read(ft, carac) end;
  if not(carac=' ') then
    writeln('ATTENTION ',sigler(sig):4,' va au-dela de 4 caracteres');end;
procedure lecnombre;begin ffl:=0;
  while not ((ord(carac) in [48..57]) or eof(ft)) do read(ft,carac);
  while (ord(carac) in [48..57]) do begin
    ffl:=(10*ffl)+(ord(carac)-48);
    if eof(ft) then carac:= ' ' else read(ft,carac) end; end;

```

A des variantes près, ces procédures sont celles qui ont déjà servi, depuis, 'litab' (cf. IA§2), pour lire sigles ou nombres entiers.

2.3 Procédures préparant le tracé des cartes

```

procedure tramer;var w1,w2:integer; begin
  for w1:=0 to 9 do for w2:=0 to 7 do tramt[w1,w2]:=0;
  tramt[8,7]:=99;tramt[9,7]:=28;
  for w2:=0 to 7 do tramt[10,w2]:=255;
  for w2:=1 to 7 do for w1:=1 to w2 do tramt[w1,w2]:=127;end;

```

Le type: pattern (trame), est défini comme: packed array [0,7] of 0..255. Une trame décrit la couverture d'un carré [0,7][0,7] par des points (pixels) blancs ou noirs: tram[x] décrit comme un entier de 0 à 255, i.e comme une suite de huit chiffres binaires, la couverture de la x-ème ligne du carré: les valeurs: 0, et: 255, donnent, respectivement, un trait blanc et un trait noir; 127, un trait noir de longueur 7, commençant par point un blanc. Dans le tableau: tramt, la trame de rang: w1, est donnée par: tramt[w1,0..7]. La plupart de nos trames (de w1=1 à w1=7) consistent simplement en un trait horizontal (de longueur 7, et d'épaisseur w1), le reste du carré étant blanc.

```

procedure polygone;begin
  reset(fin, carton); read(fin, caru);
  s:=0;h1:=0;v1:=0;w1:=1;may:=295;nv:=1;
  for u:=1 to caru do begin
    hurb[u]:=openpoly;read(fin, nu);s:=nu-1;
    if (0<nu) then begin
      read(fin, x1);read(fin, y1);if (may<y1) then may:=y1;
      moveto(x1-10, y1-20);
      while (0<s) do begin
        read(fin, x);read(fin, y);if (may<y) then may:=y;
        lineto(x-10, y-20);s:=s-1 end;
        lineto(x1-10, y1-20) end;closepoly end;
    if (295<may) then nv:=2+((may-296) div 256);
    close(fin);end;

```

Le fichier de la carte (cf. §1.1.3.2) est: carton:=concat(disq, '.', pays, 'reg'). Le premier entier lu est le nombre: caru, des unités territoriales; suivent des blocs d'entiers afférents à chacune de celle-ci. Un bloc commence par l'entier nu, nombre des sommets du polygone; suivent les coordonnées (x,y) de ceux-ci, déjà cadrées, en x, selon la largeur de la fenêtre. L'ordre de tracé d'un polygone est, en bref, une succession d'ordre de tracé de lignes. Si le maximum: may, de y (coordonnée verticale comptée vers le bas) dépasse 295, la carte sera affichée sur plusieurs écrans superposés (du Nord au Sud).

2.4 Rappel: la procédure extérieure 'litab' de lecture des données

```
{ $$ entr } { $U ulire5 }
procedure litab (var pj:ptr;pk,pn,pt:ptr;var rc:char;
                var ic,jc:integer;var pi:ptr);external;
```

On a déjà donné, dans IA§2, le texte commenté de cette procédure.

2.5 Rappel: la procédure extérieure 'trirs' de tri par fusion d'un ensemble de nombres réels

```
UNIT utris;
INTERFACE uses memtypes,quickdraw,osintf,toolintf,sane,uver5;
procedure trirs (parp,ppra,pVr,ptVr:ptr;cr:integer);
implementation procedure trire;
var arpl,pral:ptni;Vr1,tVr1:pti;ina,la,lb,n,na,nb,nz,x:integer;
begin
arpl:=ptni (parp);pral:=ptni (ppra);Vr1:=pti (pVr);tVr1:=pti (ptVr);
.....
end.
```

La procédure 'trirs' ne diffère de 'trire', décrite dans IA§4, qu'en ce que celle-ci trie des réels en format 'extended' et celle-là des réels en format 'single'. Aussi, les pointeurs Vr1 et tVr1 sont-ils déclarés, ci-dessus, avec le type: 'pti', comme pointant chacun vers une suite de valeurs 'single' (et non avec le type: ptdri, pointeur vers une suite 'extended', comme dans 'trire').

Expliquons seulement, (dans les mêmes termes qu'en IIB§2.1.1, pour le tri des entiers) le rôle des tableaux arpl[^] et pral[^]. Initialement, arpl[^] contient la suite des entiers de 1 à cr. Ces entiers sont considérés comme les adresses initiales des réels donnés dans Vr1[^]. Au cours des itérations de la boucle générale de fusion, on effectue, entre les deux tableaux d'entiers arpl[^] et pral[^], les mêmes transferts qu'entre les deux tableaux de réels, Vr1[^] et tVr1[^]; en sorte que, après exécution de la procédure 'trire', il y a, d'une part, dans Vr1[^], (ainsi que dans tVr1[^]), l'ensemble des valeurs réelles données initialement, mais rangées en une suite non décroissante; et, d'autre part, dans arpl[^], un rappel des adresses qu'occupaient initialement ces nombres: e.g., si le réel qui est finalement dans Vr1[^][n] était, initialement, dans Vr1[^][un], on a arpl[^][n]=un. De plus, 'trirs' met, dans pral[^][un], l'adresse finale, n, du réel qui était, initialement, dans Vr1[^][un].

3 Tracé d'histogramme et de carte: la procédure 'histog'

Dans l'ensemble du texte du programme 'zrang', la procédure 'histog' n'est appelée qu'une seule fois: au sein de la procédure 'afficher'. Il va sans dire que cet appel, mis dans une boucle, se répète, dans l'exécution du programme, lors de chaque affichage d'une variable; et peut même être répété, à la demande de l'utilisateur, au cours d'un seul appel de la procédure 'afficher'.

La procédure 'histog' n'a qu'un seul argument: le numéro: vv, de la variable considérée. Mais son exécution dépend de toutes les informations qui lui sont accessibles. De façon précise, d'une part, dans 'afficher', avant d'appeler 'histog', l'histogramme est spécifié par les trois nombres: {ia, ib,

carn): [ia..ib], intervalle, défini par les rangs des valeurs (croissantes) de la variable: vv, ($1 \leq ia \leq ib \leq \text{carti}$); carn: nombre de créneaux demandés pour l'histogramme dans le dialogue de la procédure 'afficher'. Et, d'autre part, le programme principal, avant d'appeler 'afficher', appelle 'trirs':

```
for i:=1 to carti do p1V^[i]:=k1ji[vv]^[i];
trirs(@p1a^,@p1b^,@p1V^,@p1W^,carti);
```

et les valeurs de la variable: vv, sont traitées et rangées, comme l'explique le §2.5. De façon précise, 'histog' utilise les tableaux: p1a^, et: p1^V. Dans la suite ordonnée non décroissante des valeurs prises par: vv, p1^V[ii] est la valeur, ayant pour rang: ii; l'indice: p1^a[ii], permet de retrouver le sigle de l'individu dont il s'agit, et, éventuellement, le polygone qui le représente sur la carte.

3.1 Partition, en créneaux, de l'ensemble des individus

```
procedure histog(vv:integer);
var cren,cadr:rect;num:str255;
verti, hori, zut, haun, gaun, zmax, pas, nn, zi, zu, ligu, iu, deb, dec:integer;
rmin, rmax, rpas, rsup:extended;
begin coin:=2;droit:=497;bas:=250;lig:=10;
eraserect(fond);
setrect(cadr, coin, coin, droit, bas);framerect(cadr);
rmin:=p1V^[ia];rmax:=p1V^[ib];if (rmax=rmin) then carn:=1;
rpas:=(rmax-rmin)/carn;
pas:=(droit-(3*coin)) div carn;
dec:=1+(56 div pas);deb:=pas-56;if (deb<0) then deb:=0;
rsup:=rmin+rpas;n:=1;
for ii:=ia+1 to ib do while (rsup<0+p1V^[ii]) do begin
zzn[n]:=ii-1;rsup:=rsup+rpas;if (n<carn) then n:=n+1 end;
for nn:=n to carn do zzn[nn]:=ib;zzn[0]:=ia-1;zmax:=0;
for n:=1 to carn do if (zmax<zzn[n]-zzn[n-1]) then zmax:=zzn[n]-zzn[n-1];
```

La procédure commence par définir un rectangle: cadr, à encadrer sur l'écran de l'ordinateur. Les valeurs extrêmes de l'intervalle défini par les rangs: [ia..ib], sont notées: {rmin, rmax}. Cet intervalle est partagé en segments successifs égaux, de largeur: rpas, suivant le nombre: carn, des créneaux demandés. Il se peut que, du fait de la répétition de valeurs égales de: vv, on ait: rmin=rmax; en ce cas, quelle qu'ait été la demande de l'utilisateur, on ne fait qu'un seul créneau: carn:=1.

Sur l'écran, la largeur d'un créneau, mesurée en points (pixels) est: pas. Pour écrire, sous le créneau, la valeur supérieure qui le limite, précédée du signe: '≤' (cf. figure), il faut disposer de 56 pixels. Si tel est le cas, le signe '≤', a pour abscisse: deb, relativement au bord gauche du créneau. Sinon, les bornes supérieures des créneaux doivent être écrites, non sur une ligne unique, sur plusieurs lignes parallèles. En général, le nombre de lignes à utiliser est: dec. Quant aux individus, la partition en créneaux se traduit en terme de rangs: de façon précise, on calcule un tableau: zzn, tel que le n-ème créneau comprend les individus de rang ii, avec: $zzn[n-1] < ii \leq zzn[n]$. Le cas: $zzn[n-1] = zzn[n]$, correspond à un n-ème créneau qui est vide.

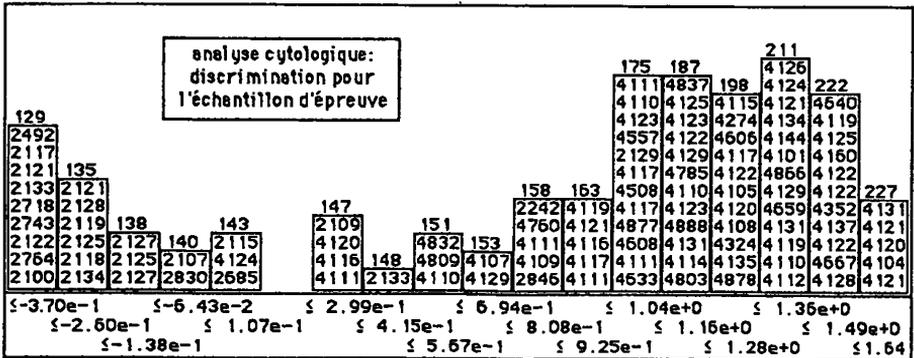
3.2 Boucle de tracé de l'histogramme par créneaux

```

for n:=1 to carn do if (0<zzn[n]-zzn[n-1]) then begin
  gaun:=(2*coin)+((n-1)*pas); zi:=zzn[n]; zu:=zzn[n-1];
  haun:=(bas-2)-round(((zi-zu)/zmax)*(bas-(12+coin)));
  setrect (cren, gaun, haun, gaun+pas, bas-2);
  framerect (cren);
  ii:=zi; verti:=haun+lig-1; hori:=gaun+pas-25;
  if (26<pas) and (verti<bas-5) then zut:=0 else zut:=1;
  while (zut=0) and (zu<ii) do begin
    moveto (hori, verti); drawstring (sigler (sigli^[pla^[ii]]));
    verti:=verti+lig; ii:=ii-1;
    if (bas-4<verti) then begin verti:=haun+lig-1; hori:= hori-25;
      if (hori<gaun+2) then zut:=1 end end;
    moveto (deb+((n-1)*pas), bas+10+(11*((n-1) mod dec));
    drawchar ('≤'); drwfl (0+plv^[zi]);
    num:=chr (48+(zi mod 10));
    if (9<zi) then num:=concat (chr (48+((zi div 10) mod 10)), num);
    if (99<zi) then if (18<carn) then
      num:=concat (chr (48+(zi div 100)), num)
    else begin
      num:=concat (chr (48+((zi div 100) mod 10)), num);
      if (999<zi) then num:=concat (chr (48+(zi div 1000)), num) end;
    if (18<carn) then moveto (gaun+pas-18, haun-1)
    else moveto (gaun+pas-24, haun-1);
    drawstring (num) end;
  moveto (coin, bas+lig+(dec*11));

```

La boucle ne trace et n'écrit que les créneaux non vides. Sur l'axe vertical orienté vers le bas, la dimension du cadre rectangulaire: cren, va de: haun, à: bas-2. On la calcule en rapportant, le cardinal du créneau considéré au maximum: zmax. Dans la mesure de l'espace disponible, on écrit les sigles des individus, celui de rang maximum étant en haut et à droite. Si une colonne de sigles ne suffit pas, et qu'il reste, à gauche, dans le créneau, une bande libre de largeur >25 pixels, on y écrit une autre colonne. Au-dessous du créneau, ainsi qu'on l'a dit, on écrit sa borne supérieure: si l'écriture des bornes exige plusieurs lignes, on calcule, modulo dec, la ligne propre au créneau considéré. Au-dessus du créneau, on écrit le rang maximum: zi, des éléments qu'il contient. Si l'espace manque pour 4 chiffres, on utilise le code ASCII pour aller au delà de 9 centaines: e.g., pour 1245, on écrit: <45 .



3.3 Tracé de la carte associée à un histogramme

```

if (rp1='0') and (caru=carti) then begin
drawstring('pour effacer l''histogramme de la var n° ');drwnum(vv);
drawstring(': ');drawstring(sigler(siglj^[vv]));
drawstring(', entrer une lettre ');lir(rp2);eraserect (fond);
moveto(5,20);
drawstring('On calcule les trames attribuées aux unités territoriales');
moveto(2,33);
for ii:=1 to ia-1 do trmu[pla^[ii]]:=0;
for ii:=ib+1 to carti do trmu[pla^[ii]]:=10;
for n:=1 to carn do begin
  if (carn=1) then s:=5 else s:=1+trunc((9*(carn-n))/carn);
  for ii:=zsn[n-1]+1 to zzn[n] do trmu[pla^[ii]]:=s end;
write('pour finir l''affichage de la carte de',sigler(siglj^[vv]):5);
writeln(' entrer la lettre N');
writeln('pour afficher la carte ou passer alternativement');
write('du Nord au Sud, entrer une autre lettre ');readln(rp2);
while not (rp2='N') do begin
  eraserect (fond);penmode (pator);
  for ii:=1 to carti do begin offsetpoly (hurb[ii],h1,v1);
    fillpoly (hurb[ii],tramt [trmu [ii]]);framepoly (hurb[ii]) end;
  moveto (2,10);penmode (patXor);
  for ii:=1 to siglj^[vv][0] do begin
    drawchar (chr (siglj^[vv][ii]));moveto (2,10*(ii+1)) end;
  lir (rp2);
  eraserect (fond);
  if (1<nv) then begin w1:=w1-(v1 div 256);
    if (w1=1) then v1:=-256;if (w1=nv) then v1:=256 end end;
  moveto (coin,bas+lig+(dec*11));end;
end;{histog}

```

A un entier: $ii < ia$, correspond une unité territoriale dont le numéro initial est: $pla^{[ii]}$; et qui, sortant (quant à la variable: vv) de l'intervalle retenu pour l'histogramme, est laissée en blanc (trame: $trmu:=0$); de même, pour: $ii > ib$, on met la trame n° 10, le noir. Entre ces deux extrêmes, aux créneaux successifs de l'histogramme, sont attribuées des trames grises, de rang 1 à 9 (cf. §2.3).

Dans la procédure 'polygoner', a été calculé le nombre: nv , d'écrans verticaux, nécessaire pour présenter la carte sur des fenêtres successives. La boucle: $for i:=1 to carti do \dots$, impose, par: $offsetpoly$, un décalage: $\{h1, v1\}$, au polygone représentant l'unité territoriale d'indice: ii . Relativement à une carte virtuelle, dont l'étendue peut dépasser la taille de l'écran, ceci équivaut à placer celui-ci comme une fenêtre, par laquelle on voit la partie, effectivement affichée, de la carte. En l'état du programme, le décalage horizontal: $h1$, est toujours nul (il n'y a pas de découpage de la carte suivant des bandes verticales parallèles); en revanche, si $(1 < nv)$, le décalage vertical: $v1$, varie de telle sorte que la fenêtre visible glisse, alternativement, de haut en bas et de bas en haut.

Les lettres du sigle: $siglj^{[vv]}$, de la variable représentée, sont affichées, au coin supérieur gauche de l'écran, de haut en bas; et, en-dessous du sigle, par un caractère: $rp2$, que saisit la procédure 'lir' (cf. IA§1.3.5), l'utilisateur écrit l'ordre de poursuivre ou d'interrompre l'affichage de la carte.

4 Procédures pour le croisement de deux variables

Le croisement n'est possible que dans l'observation des données sans créer de tableau: suf='Z'.

Le croisement est proposé si: rcr='R'. Initialement (cf. §1.1.1), le programme principal pose: rcr='N'; mais, si: suf='Z' (cf. §1.2.1), le programme pose: rcr='C'. Il n'y a de croisement possible qu'à partir du deuxième appel de la procédure 'afficher'. Celle-ci s'achève en appelant la procédure 'postcroiser', qui pose: rcr='R'. Telle est la condition nécessaire pour que, quand s'achève, pour une variable, l'affichage d'une série d'histogrammes (éventuellement suivis de cartes), l'utilisateur puisse demander un croisement.

Pour le croisement de deux variables: {hj, wj}, affichées l'une après l'autre en histogrammes (wj, après: hj), on affiche, d'abord, par 'planer', un plan (hj: axe horizontal, abscisse; wj, axe vertical, ordonnée); puis, par 'croiser', un tableau de tri croisé (modalités de: hj, en colonnes; modalités de: wj, en lignes).

4.1 Bornes pour la variable mise en ordonnée et numéro du créneau en fonction du numérotage des lignes: la procédure 'precroiser'

```

procedure precroiser;begin
  hauy:=-plV^[ib];basy:=-plV^[ia];
  n1:=1;if (1<ia) then n1:=0;n:=n1;
  n2:=carn;if (ib<carti) then begin n2:=n2+1;zzn[n2]:=carti end;
  for i:=1 to carti do begin
    while (zzn[n]<i) do n:=n+1;plb^[pla^[i]]:=n end;
  for n:=n1 to n2 do valb[n]:=plV^[zzn[n]];end;

```

La variable:wj, qu'on vient d'afficher tient le rôle de l'ordonnée: y. La variable: hj, précédemment affichée, est (s'il y a lieu) l'abscisse: x.

L'ensemble des individus à afficher est délimité, quant à la variable: wj, par un rang allant de ia à ib. D'où, compte tenu de l'orientation de l'écran, les bornes: {hauy, basy}, dès valeurs s'étalant sur la direction verticale de l'écran.

Pour l'abscisse, on a, de même, des bornes: {gchx, drtx}, héritées des bornes {basy, hauy}, propres à l'affichage précédent, par les deux instruction: gchx:=-basy ; drtx:=-hauy ; de la procédure 'postcroiser', appelée (cf. *supra*) à la fin de la procédure 'afficher'.

Au pointeur plb, correspond le tableau donnant, pour l'individu de rang initial (ou: numéro de ligne): ili, le numéro: plb^[ili] de son créneau, dans l'histogramme de: wj; (plus précisément, on a: 0, en deçà de l'histogramme; et: carn+1, au delà). Il y a, pour chaque créneau de la variable: wj, une valeur: valb[n], qui est sa borne supérieure (avec: $wj \leq valb[n]$, pour tout individu du créneau).

4.2 Affichage du nuage plan croisant deux variables: la procédure 'planer'

```

procedure planer;begin
eraserect(fond);
if (drtx=gchx) then col:=0 else col:=470/(drtx-gchx);
if (basy=hauy) then co2:=0 else co2:=-245/(basy-hauy);
if ((rsc='O') and (col<-co2)) then co2:=-col;
if ((rsc='O') and (-co2<col)) then col:=-co2;
Pch:=2+round(-col*gchx);Pcv:=10+round(co2*hauy);
if (gchx<=0) and (0<=drtx) and (-basy<=0) and (0<=-hauy)
then begin moveto(Pch,Pcv);drawstring('+') end;
if (0<=-hauy) and (-basy<=0)
then begin moveto(470,Pcv);drawstring(sigler(sgc)) end;
if (gchx<=0) and (0<=drtx)
then begin moveto(Pch,12);drawstring(sigler(sgb)) end;
xgch:=drtx;xdrt:=gchx;ybas:=-hauy;yhau:=-basy;
for i:=1 to carti do begin xh:=klji[hj]^[i];yv:=klji[wj]^[i];
if (gchx<=xh) and (xh<=drtx) and (-basy<=yv) and (yv<=-hauy)
then begin
if (xh<xgch) then xgch:=xh;if (xh>xdrt) then xdrt:=xh;
if (yv<ybas) then ybas:=yv;if (yv>yhau) then yhau:=yv;
Pch:=2+round(col*(xh-gchx));
Pcv:=10+round(co2*(yv+hauy));
sga:=zigler(rpf);if (rpf='') then sga:=sigli^[i];
if (rpf='i') then sga:=zigler(copy(sigler(sigli^[i]),1,1));
moveto(Pch,Pcv);drawstring(sigler(sga)) end end;
moveto(2,264);
drawstring('abscisse: ');drwfl(xgch);
drawstring(concat(' ≤ ',sigler(sgc),' ≤ '));drwfl(xdrt);
drawstring(' ordonnée: ');drwfl(ybas);
drawstring(concat(' ≤ ',sigler(sgb),' ≤ '));drwfl(yhau);
getpen(P0);moveto(2,P0.v+11);
drawstring('pour quitter le plan croisant ');
drawstring(concat(sigler(sgc),' (n°)'));drwnum(hj);
drawstring(';horiz) et ');
drawstring(concat(sigler(sgb),' (n°)'));drwnum(wj);
drawstring(';vertic) entrer * ');end;{planer}

```

D'après l'intervalle de variation de chaque coordonnée et les dimensions de la fenêtre, on calcule deux coefficients de changement d'échelle: {col, co2}. Si, dans 'afficher', l'utilisateur a demandé même échelle sur les deux axes (rsc='O'), les coefficients sont modifiés pour cela. (On prendra garde que l'orientation de l'axe vertical, vers le bas, introduit des signes '-').

Dans la mesure du possible, on met un '+' à l'origine des axes; et, à l'extrémité positive de chaque axe, le sigle de la variable qu'il représente. De même, pour l'ensemble I: un élément n'est marqué que si chacune de ses deux coordonnées tombe entre les bornes, calculées comme on l'a dit au §4.1. Ainsi qu'on le verra dans le dialogue de la procédure 'afficher' (cf. §5.4), un point peut être marqué par son sigle, la première lettre de son sigle, ou un caractère commun à tous les points.

Sous le graphique plan, sont rappelés les numéros et sigles des variables croisées; avec les bornes des intervalles de variation. Et la procédure 'planer' offre à l'utilisateur de quitter le plan, en entrant un caractère quelconque (pour lequel un ordre: lir, est mis dans 'afficher', après l'appel de 'planer').

4.3 Affichage du tableau du tri croisé de deux variables: la procédure 'croiser'

```

procedure croiser;begin
  for c:=n1c to n2c do for n:= n1 to n2 do crcb[c,n]:=0;
  for i:=1 to carti do crcb[plc^[i],plb^[i]]:=crcb[plc^[i],plb^[i]]+1;
  drawstring(' ValSup | ');n:=1;
  for c:=n1c to n2c do begin n:=-n;getpen(P0);
    if (n=1) then drwfl(0+valc[c]);
    moveto(P0.h+30+(18*n),P0.v) end;
  getpen(P0);moveto(53,P0.v+11);n:=-1;
  for c:=n1c to n2c do begin n:=-n;getpen(P0);
    if (n=1) then drwfl(0+valc[c]);
    moveto(P0.h+30+(18*n),P0.v) end;
  getpen(P0);moveto(5,P0.v+11);
  for c:=n1c to n2c+2 do drawstring('-----');
  getpen(P0);moveto(5,P0.v+11);
  for n:= n1 to n2 do begin getpen(P0);
    if (P0.v<272) then begin
      drwfl(0+valb[n]);moveto(P0.h+48,P0.v);drawchar('|');
      for c:=n1c to n2c do begin getpen(P0);
        drw4(crcb[c,n]);moveto(P0.h+30,P0.v);end;
      getpen(P0);moveto(5,P0.v+11);end;end;
    for c:=n1c to n2c+2 do drawstring('-----');
    getpen(P0);moveto(5,P0.v+11);
    drawstring('pour quitter le tri croisant ');
    drawstring(concat(sigler(sgc),' (n°)'));drwnum(hj);
    drawstring(';col) et ');
    drawstring(concat(sigler(sgb),' (n°)'));drwnum(wj);
    drawstring(';lignes) entrer * ');lir(rpf);end;{croiser}

```

On a dit (au §4.1) que: plb^i , est le numéro du créneau de l'individu de la ligne: i , dans l'histogramme de la variable: wj . De même, parce que 'postcroiser' recopie le tableau: $plb^i[1..carti]$, dans: $plc^i[1..carti]$, on a, dans: plc^i , le numéro du créneau de: i , dans l'histogramme de la variable: hj . C'est pourquoi l'individu: i , est compté dans la case: $crcb[plc^i[i],plb^i[i]]$, du tableau de tri croisé: $crvb[n1c..n2c, n1..n2]$. Pour ce tableau, le numérotage des lignes: $[n1..n2]$, a été fixé par 'precroiser' (cf. §4.1); et le numérotage des colonnes: $[n1c..n2c]$, a été recopié, par 'postcroiser', à la fin du traitement de la variable précédente, hj .

Après la mention: ValSup, on marque un tiret vertical: '|'; puis on écrit les bornes supérieures: $valc[n1c..nc2]$, des créneaux de hj , moitié sur une première ligne, moitié sur une seconde; et l'on trace un trait, en tireté. Les valeurs numériques du tableau: $crvb$, sont écrites sur des lignes, idicées par: n , dans: $[n1..n2]$; avec en tête de ligne, la borne: $valb[n]$. En interrompant, si la place manque, l'écriture de $crvb$, on offre à l'utilisateur de quitter le tri croisé.

ValSup	1.20e+1	1.40e+1	1.60e+1	1.80e+1	2.00e+1	4.80e+1						
	1.10e+1	1.30e+1	1.50e+1	1.70e+1	1.90e+1	2.20e+1						
2.20e+1	33	13	8	5	7	9	10	7	4	8	2	11
3.70e+1	15	12	5	6	2	8	10	8	3	4	9	24
5.00e+1	1		1	2	1			1	2		1	10
6.70e+1							1					2

pour quitter le tri croisant Ur95 (n°8;col) et Ph95 (n°7;lignes) entrer *

4.4 Sauvegarde d'informations relatives aux créneaux d'une variable à la fin de l'affichage de celle-ci: la procédure 'postcroiser'

```

procedure postcroiser;begin
  gchx:=-basy;drtx:=-hauy;
  sgc:=sgb;n1c:=n1;n2c:=n2;rcr:='R';
  for n:=n1 to n2 do valc[n]:=valb[n];
  for i:=1 to carti do plc^[i]:=plb^[i];end;{postcroiser}

```

La variable qui a été présentée par 'afficher', est gardée pour servir, éventuellement d'abscisse: x, ses créneaux devant être, sur le tableau de tri croisé, représenté chacun par une colonne: c. C'est pourquoi, on convertit les bornes: {basy, hauy}, en: {gchx, drtx}. On garde le sigle: sgb, en sgc; le numérotage: [n1..n2], des créneaux, en: [n1c.. n2c]; le tableau des valeurs supérieures: valb[n1..n2], en: valc[n1..nc2]; les rangs des créneaux des lignes quant aux valeurs de y: plb^[1..carti], comme rangs quant à x: plc^[1..carti].

Le rôle de ces sauvegardes est apparu dans les §§4.2 et 4.3.

5 Affichage d'une variable à l'écran par tableau, histogrammes et cartes; et dialogue de découpage: la procédure 'afficher'

```

procedure afficher(vj:integer);
var mec,ra,li,co,nli,paz,nco,mco:integer;P:point;
begin mec:=8;paz:=1;nco:=1;nli:=carti;raf:='O';mm:=0;
if (10<carti) then begin nli:=10;mco:=78 div mec;
  if (10*mco<carti) then begin nli:=20;
    paz:=1+((carti-1) div ((20*mco)+1)) end;
  nco:=1+((carti-paz) div (paz*nli)) end;

```

La procédure 'afficher' commence par caractériser, par quelques paramètres entiers, la disposition du tableau des valeurs de: vj; puis entre dans une boucle externe: while not (raf='N') do..., laquelle après un affichage du tableau des valeurs, entre elle-même, d'abord, dans une boucle interne: while not(rph='N') do..., celle-ci affichant autant d'histogrammes (éventuellement accompagnés d'une carte) que l'utilisateur le demande; et, ensuite, procède si: (suf='D'), au découpage de: vj, en modalités, par dialogue à l'écran. Et, à la sortie de la boucle externe, sous la condition: if (rcr in ['C','R']), s'exécutent les procédures de croisement, déjà expliquées au §4.

5.1 Affichage du tableau des valeurs de la variable traitée

Les instructions pour écrire les valeurs se comprennent bien sur un exemple. Il faudrait afficher, en une ou plusieurs colonnes, depuis la plus petite jusqu'à la plus grande (et en respectant les répétitions de valeurs égales), la suite des: carti, valeurs prises par la variable: vj, traitée. Compte tenu de la taille de la fenêtre, on ne peut, si: (160<carti), écrire toutes les valeurs. Sont donc seules conservées celles dont le rang est un multiple entier d'un pas, noté: paz. Dans l'exemple: paz=4; on lit, en marge gauche, les rangs: {4, 8, ..., 24, 28, 32, ..., 76, 80} des valeurs conservées dans la première colonne; et, de même, au-dessus de chaque colonne de valeurs, le rang (diminué de paz), de

```

tableau ordonné des valeurs de la variable #ph5 (n° = 10)
rng      0      80      160      240      320      400      480      560
 4 0.00e+0 2.00e+0 1.14e+2 2.81e+2 4.95e+2 9.37e+2 1.93e+3 5.41e+3
 8 0.00e+0 4.00e+0 1.19e+2 2.85e+2 5.36e+2 9.79e+2 2.04e+3 6.26e+3
.....
24 0.00e+0 1.90e+1 1.49e+2 3.23e+2 5.96e+2 1.14e+3 2.52e+3 8.62e+3
28 0.00e+0 2.00e+1 1.72e+2 3.32e+2 6.15e+2 1.18e+3 2.61e+3 9.17e+3
32 0.00e+0 2.70e+1 1.73e+2 3.54e+2 6.20e+2 1.19e+3 2.64e+3
.....
76 1.00e+0 9.70e+1 2.70e+2 4.67e+2 8.98e+2 1.73e+3 4.74e+3
80 1.00e+0 1.10e+2 2.78e+2 4.86e+2 9.15e+2 1.83e+3 4.94e+3

```

```
*
pour finir l'affichage du tableau, entrer deux fois * *
```

la première valeur qui y est écrite. Ainsi, à l'intersection de la ligne marquée, à gauche: 28, et de la colonne marquée, en haut: 320, on a la valeur de rang: $ra=28+320=348$, soit: $6.15e+2$, i.e.: 615.

```

while not (raf='N') do begin
  writeln('on affichera successivement, pour la variable', sigler(siglj^[vj]):5);
  writeln('le tableau ordonné des valeurs, puis des histogrammes');
  if (suf='D') then
    writeln('ATTENTION avant de finir l''affichage choisir les bornes !');
  write('tableau ordonné des valeurs de la variable ');
  writeln(sigler(siglj^[vj]):5, ' (n° =', vj:3, ')');
  write('rng'); for co:= 1 to nco do write((co-1)*nli*paz:mec); writeln;
  for li:=1 to nli do begin ra:=paz*li; write(ra:3);
    while (ra<=carti) do begin [write(round(0+p1V^[ra]):mec);]
      ret:=p1V^[ra];
      if (ret<0) then write(' ',ret:2) else write(ret:3);
      ra:=ra+(paz*nli) end;
    writeln end;
  readln(rpf); getpen(P);
  drawstring(' pour finir l''affichage du tableau, entrer deux fois * ');
  readln(rpf); moveto(2,P.v); lirr(rpf); moveto(15,P.v); writeln;

```

S'affiche à l'écran, au-dessus du tableau des valeurs, cet avertissement:

ATTENTION avant de finir l'affichage choisir les bornes !

en effet, la première version du programme 'zrang' ne présentait pas d'histogramme, et l'utilisateur n'avait que le tableau des valeurs pour choisir le codage. Le lecteur pourra, de même, s'étonner de ce que l'affichage du tableau s'achève par l'entrée de plusieurs lettres isolées. Nous avons dit, dans IA§1.3.5, que, pour les programmes compilés en TML2, la commande de copie de l'écran sur disque, ne peut s'exécuter quand le programme a envoyé un ordre de lecture: 'readln'. Dans le cas présent, le tableau des valeurs est écrit par des ordres usuels: 'write', 'writeln'; cette écriture, exécutée par blocs, n'est certainement achevée que quand le programme rencontre un premier ordre: readln(rpf), de lecture de caractère. L'affichage de la consigne: 'pour finir...', destinée à l'utilisateur, se fait alors non par un ordre: 'write', mais comme un tracé: 'drawstring'. Un nouvel ordre: readln(rpf), prépare alors le dialogue de spécification de l'histogramme; mais, avant d'entrer dans ce dialogue, l'interposition de: lirr, permet de commander la saisie de l'écran.

5.2 Boucle d'affichage d'histogramme

```
rph:='H';
while not (rph='N') do begin
  write('affichage d'histogramme pour la variable ');
  writeln(sigler(siglj^[vj]):5, ' ( n° =',vj:3,')');
  write('nombre de créneaux (de 1 a 25) = ');readln(carn);
  if (carn<1) then carn:=1;if (25<carn) then carn:=25;
  writeln('on rangera les valeurs comprises entre deux rangs ia et ib');
  write('le rang ia de la première valeur prise est ');readln(ia);
  write('le rang ib de la dernière valeur prise est ');readln(ib);
  if (ia<1) then ia:=1;if (carti<ib) then ib:=carti;
  if (ib<=ia) then begin ia:=1;ib:=carti end;
  histog(vj);
  drawstring('faut-il tracer un autre histogramme(H) pour la var n°');
  drwnum(vj);
  drawstring(concat(' ',sigler(siglj^[vj]),', ou non(N) '));lir(rph);
end;
```

La boucle d'affichage d'histogrammes: while not(rph='N') do..., n'a d'autre rôle que de guider l'utilisateur dans l'appel de la procédure: 'histog'. On a dit, au §3, qu'avec l'histogramme demandé, 'histog' trace, éventuellement, une carte.

5.3 Découpage de la variable en modalités

```
if (suf='D') then begin rpm:='N';rbar:='b';
  while not (rpm='O') do begin
    write('le nombre de modalités (de 2 à',mqmx:3,') sera ');readln(mm);
    if (mm<=0) then mm:=0;if (mqmx<mm) then mm:=mqmx;
    writeln(' nombre de modalités pour',sigler(siglj^[vj]):5, ' =',mm:3);
    write('ce nombre est-il confirmé O ou N ');readln(rpm) end;
    if ((suc in ['Q','S']) and not (mm=0)) then begin
      write('faut-il faire un codage barycentrique(B) ou un découpage en classes(D) ');
      readln(rbar); if (rbar='D') then rbar:='μ' else rbar:='z';end;
      if (rbar='μ') then mxpx:=' limite ' else mxpx:=mxpv;
      if (suc in ['B','Q']) and (jmax<carm+mm) then begin
        mm:=jmax-carm;if (mm<0) then mm:=0;if (mm=0) then
          drawstring('il n'y a plus de place pour créer des modalités!') end;
        nmj[vj]:=mm;r2:=0;
```

Dans le cas: suf='D', du découpage en dialogue, (et non suivant des bornes préétablies: suf='d'), la boucle: while not(rpm='O')..., demande à l'utilisateur de fixer, pour la variable traitée, le nombre: mm, des modalités.

La condition: (suc in ['Q','S']), caractérise le codage barycentrique: mais, on a dit, au §1.2.2, que cette forme de codage accepte des exceptions: certaines variables peuvent être codées en (0,1). Sur le listage 'Dcodx', ces variables ont leur format spécifié par un nombre de 'modalités' (avec le caractère: rbar='μ'); et, l'on parle de: valeur limite; et non de: valeur pivot.

Si: (suc in ['B','Q']), 'zrang' doit créer un tableau éclaté, i.e. avec une colonne par modalité, et non une colonne par variable: le programme vérifie que le nombre des colonnes ne dépassera pas: jmax; et, sinon: mm, est diminué, voire mis à zéro.

```

for ml:=1 to mm do begin carm:=carm+1;r1:=r2+1;
write('le sigle choisi pour la modalité',ml:3,' de',sigler(siglj^[vj]):5,' est ');
readln(chaine);
if (4<length(chaine)) then chaine:=copy(chaine,1,4);
if (carm<=imax) then sicj[carm]:=zigler(chaine);
if ((ml<mm) or ((suc in ['Q','S']) and not(xbar='μ')) then begin
write('le RANG de la valeur',mxxp,'de', chaine:5,' est ');
readln(rr2);
if (rr2<1) then rr2:=1;if (carti<rr2) then rr2:=carti;
r2:=trunc(rr2);r3:=r2+1;if (carti<r3) then r3:=carti;
res:=ret;ret:=(1+r2-rr2)*p1V^[r2]+((rr2-r2)*p1V^[r3]);
while (r2<carti) and (p1V^[r2+1]=p1V^[r2]) do r2:=r2+1;
if (r2<r1) then r2:=r1-1 end;
if (ml=mm) then r2:=carti;
brm[ml]:=p1V^[r2];if (suc in ['Q','S']) then brm[ml]:=ret;
for ra:=r1 to r2 do p1W^[pla^[ra]]:=ml;
if ((suc in ['Q','S']) and (1<ml) and (res<ret) and not(xbar='μ'))
then begin
col:=1/(ret-res);
for ra:=r1 to r2 do begin co2:=(ret-p1V^[ra]);
if (co2<0) then co2:=0;p1W^[pla^[ra]]:=ml-(col*co2) end end
end end {if (suf='D')};
erasetect (fond);moveto(2,10);
write('faut il réafficher la variable n°,vj:3,':,sigler(siglj^[vj]):5,', 0 ou N ');
readln(raf); if not(raf='N') then carm:=carm+mm end;

```

Le nombre: mm, des modalités étant fixé, on entre dans une boucle indiquée par celles-ci: for ml:=1 to mm do..., afin de coder la variable: vj. D'abord, on choisit le sigle; ensuite, la valeur de la borne (cas du codage logique), ou du pivot (codage barycentrique).

Dans le codage logique, la dernière borne ne joue aucun rôle; car tout ce qui vient après l'avant-dernière borne est compris dans la dernière modalité: on convient de poser que la dernière borne est la valeur maxima de vj: $p1V^{[carti]}$.

La distribution de la variable: vj, a été présentée sur un histogramme, d'après lequel l'utilisateur peut choisir le codage; notamment d'après des modes, ou maxima succesifs. D'après les valeurs écrites au bas des créneaux (cf. §3.2), on peut évaluer approximativement l'abscisse du point où l'on désire placer une borne (ou un pivot). Mais, l'écriture d'un nombre de forme générale est en butte à des erreurs. Nous avons donc préféré demander à l'utilisateur de définir une valeur par son rang, qui est un nombre entier; dont on suit la variation, en fonction de l'abscisse, d'après les rangs inscrits en haut de chaque créneau. En fait, tel qu'il est écrit ici, le programme 'zrang' accepte même des rangs continus; ce qui permet d'intrapoler, *ad libitum*, dans le cas d'échantillons de faible effectif (e.g.: carti≈30). De plus, on a dit (cf. §1.1.3.1) que le fichier 'Dcodx' contient les valeurs des bornes ou pivots, qu'on peut modifier, *a posteriori*, afin d'améliorer le codage.

Le codage consiste à écrire, dans: $p1W^{[ili]}$, le numéro de la modalité de vj afférente à l'individu de la ligne: ili. Dans le codage logique, en (0,1), ce

rang est un nombre entier; et par l'instruction:

```
for ra:=r1 to r2 do p1W^[p1a^[ra]]:=m1;
```

le numéro: m1, est attribué à tous les individus dont le rang rentre dans un intervalle: [r1..r2], facile à déterminer. Le codage barycentrique est plus complexe. Les individus dont le rang est dans l'intervalle: [r1..r2], compris entre les deux pivots numérotés: m1-1, et: m1, reçoivent par interpolation linéaire, un numéro continu compris dans l'intervalle: (m1-1, m1).

Le dialogue du codage est complexe. À l'utilisateur qui déplore un erreur dans le choix d'un sigle ou d'une borne, l'occasion est offerte de reprendre le traitement de la variable: vj. En ce cas, par l'instruction: carm:=carm-mm, le nombre des modalités déjà créées pour les variables, est réduit à la valeur qu'il avait avant l'affichage de vj. (En dehors du cas: (suf='D'), la valeur de: carm, n'a aucun sens, et il est inutile de mettre une condition à son calcul).

5.4 Appel des procédures de croisement, par la procédure 'afficher'

```
if (rcr in ['C','R']) then begin sgb:=siglj^[vj];wj:=vj;
precroiser;
if (rcr='R') then begin
write('faut il croiser',sigler(sgb):5,' avec',sigler(sgc):5,' O ou N ');
readln(rcr);
if not (rcr='N') then begin
writeln('NB on affichera successivement un plan et un tableau;');
writeln('pour appeler ou quitter le plan, entrer un caractère;');
writeln('pour marquer les points sur le plan par leur SIGLE entrer '*'');
writeln('pour les marquer par le 1-er caractère du sigle entrer 'i');
writeln('pour les marquer par un même caractère AUTRE que '*' ou 'i','');
write('entrer ce dernier caractère ');readln(rpf);
write('faut-il adopter la même échelle sur les deux axes O ou N ');
readln(rsc);
planer;lir(rpf);eraserect(fond);moveto(5,10);writeln;
croiser end end;
postcroiser;hj:=wj end;
end;{afficher}
```

Ainsi qu'on l'a annoncé (au début du §5), à la sortie de la boucle externe, la procédure 'afficher' offre, sous certaines conditions, de croiser la variable: vj, avec celle: hj, traitée avant celle-ci. En général, les formes les plus approfondies d'examen des données: tracé de cartes, croisement de deux variables, ne servent que pour observer les données sans créer de tableau (Z). De plus, une variable ne peut être croisée qu'avec une autre affichée avant elle.

Les procédures du croisement ont fait l'objet du §4. Nous rappelons que 'precroiser' (cf. §4.1), inscrit dans: plb^[ili], le numéro du créneau où est l'individu de la ligne: ili. Et que 'postcroiser' (cf. §4.4), sauvegarde (pour croisement ultérieur) des informations afférentes à la variable qu'on quitte. Le graphique plan et le tableau de tri croisé sont créés par 'planer' (§4.2) et 'croiser' (§4.3). Le seul choix offert ci-dessus est celui de l'étiquetage des points du nuage (par leur sigle, par un caractère unique...).

6 Élaboration, sans dialogue, du tableau à transformer

6.1 Codage d'une variable d'après des bornes préétablies: la procédure 'coupixer'

```

procedure coupixer(vj:integer);begin carac:=' ';
  lecsig;lecnombre;m3:=ffl;mm:=m3;readln(ft,chaîne);
  while ((1<length(chaîne)) and (chaîne[1]=' ')) do
    chaîne:=copy(chaîne,2,length(chaîne)-1);
  if (mm<0) then mm:=0;if (mqmx<mm) then mm:=mqmx;
  if (suc in ['B','Q']) and (jmax<car+mm) then begin mm:=jmax-car;
    if (mm<0) then mm:=0 end;
  nmj[vj]:=mm;r2:=0;carac:=' ';
  for ml:=1 to m3 do begin lecsig;m2:=car+ml;
    if (ml<mm) and (suc in ['B','Q']) then sicj[m2]:=sig end;
  for ml:=1 to mm do begin
    car:=car+1;r1:=r2+1;
    res:=ret;read(ft,ret);if (ml=1) then res:=ret;
    while (r2<carti-1) and (plV^[r2+1]<=ret) do r2:=r2+1;
    if (r2=carti-1) and (plV^[carti]<=ret) then r2:=carti;
    if (ml=mm) then r2:=carti;
    for rz:=r1 to r2 do plW^[pla^[rz]]:=ml;
    if ((suc in ['Q','S']) and (1<ml) and (res<ret) and not(chaîne[1]='μ'))
      then begin
        col:=1/(ret-res);
        for rz:=r1 to r2 do begin
          co2:=(ret-plV^[rz]);if (co2<0) then co2:=0;
          plW^[pla^[rz]]:=ml-(co2*col) end end
      end;
  for ml:=mm+1 to m3 do read(ft,ret);writeln(sigler(siglj^[vj]));
end;{coupixer}

```

La procédure 'coupixer' a un seul et unique rôle: elle est appelée par le programme principal pour coder les variables d'après des bornes préétablies. Elle fait, dans le cas (suf='d'), ce que fait, dans le cas (suf='D'), le bloc d'instructions (compris dans 'afficher'), objet du §5.3. Mais tandis qu'au §5.3 les bornes (ou pivots) proviennent d'un dialogue à l'écran, 'coupixer' lit un fichier 'Dcodx'.

La procédure 'coupixer', lit, sur le fichier: 'Dcodx', le bloc de lignes afférent à la variable: vj. Sur la 1-ère ligne du bloc, on a le sigle de la variable, le nombre: m3, des modalités demandées; comme au §5.3, ce nombre n'est accepté que s'il est \leq mqmx; et, pour créer un tableau éclaté, on s'assure que le nombre total des colonnes ne dépassera pas jmax. Après qu'on a lu m3, il reste une chaîne, dont le 1-er caractère (non blanc): chaîne[1], est pris en compte pour distinguer, dans le cas du codage barycentrique, l'exception d'une variable codée disjonctivement, en (0, 1): cf. §1.2.2.

On lit ensuite, dans une première boucle: for ml in [1..m3] les sigles de toutes les modalités demandées; puis, dans une seconde boucle: for ml in [1..mm] leurs bornes ou pivots. Sitôt lue, chaque borne (ou pivot) sert à calculer un numéro de modalité (éventuellement continu) pour les individus qui, en bref, la précèdent. Et si le nombre: mm, des modalités retenues est $<$ m3, il reste à lire une suite de nombres, indiquée par [mm+1..m3]; nombres dont on ne fera rien.

6.2 Codage d'une variable suivant les rangs: la procédure 'rangcorriger'

```

procedure rangcorriger; begin
  res:=plV^[1];r1:=1;
  for i:=1 to carti do begin ret:=plV^[i];
    if (res<ret) then begin r2:=i-1;rrr:=r1+r2;
      for rz:=r1 to r2 do plW^[pla^[rz]]:=rrr;
      res:=ret;r1:=i end end;
    rrr:=r1+carti;
    for rz:=r1 to carti do plW^[pla^[rz]]:=rrr;
  end;{rangcorriger}

```

La procédure 'rangcorriger' est appelée par le programme principal dans le cas où: suf='R', ou dans le découpage d'une variable suivant zéro modalité (i.e. pour une variable virtuellement exclue du découpage, mais représentée par une colonne du tableau de numéros des modalités). La procédure: 'rangcorriger', met dans le tableau: plW^, les rangs multipliés par 2; en les corrigeant de telle sorte que tout groupe maximal d'individus où la variable considérée est constante reçoit un rang moyen entier (cf. *supra*, §1.2.2).

7 Sortie du tableau transformé

Nous donnons ici trois procédures: 'critabx' est appelée pour copier ou transposer un tableau en créant un texte où les valeurs des variables sont écrites en virgule flottante; de même, 'critabex' crée un texte où tous les nombres sont écrits comme des entiers. Enfin, on rappelle pour mémoire, la procédure externe 'critab', qui crée des fichiers en format binaire.

7.1 Sortie du tableau transformé comme un texte où les nombres sont écrits en virgule flottante: la procédure 'critabx'

Les sigles comme les nombres sont écrits à raison de 6 par ligne. Adresses des sigles et adresses des valeurs sont changées dans le cas: suf='Y', d'une transposition.

```

{$S sortx}
procedure critabx;begin sigli:=pzqi(sli);
  cartu:=carti;cartv:=cartj;nomf:=nomba;
  if (suf='Y') then begin cartu:=cartj;cartv:=carti;
    nomf:=concat(nomba,'T') end;
  nomf:=concat(nomf,'yy');writeln(nomf);
  rewrite(ft,nomf);writeln(ft,titre);
  ffl:=cartu;ffl:=(1000*ffl)+cartv;writeln(ft,ffl);
  for vz:=1 to cartv do begin
    if (suf='Y') then sql:=sigli^[vz] else sql:=siglj^[vz];
    write(ft,sigler(sql):12);writeln(sigler(sql):4);
    if (vz mod 6=0) then writeln(ft) end;
  if not (cartv mod 6=0) then writeln(ft);
  for uz:=1 to cartu do begin
    if (suf='Y') then sql:=siglj^[uz] else sql:=sigli^[uz];
    writeln(ft,sigler(sql):4);writeln(sigler(sql):5);
    for vz:=1 to cartv do begin
      if (suf='Y') then wal:=klji[uz]^[vz] else wal:=klji[vz]^[uz];
      write(ft,' ',0+wal:6);
      if (vz mod 6=0) then writeln(ft) end;
    if not (cartv mod 6=0) then writeln(ft) end;
  close(ft) end;{critabx}

```

