

J.-P. BENZÉCRI

F. BENZÉCRI

**Sources de programmes d'analyse de données  
en langage PASCAL : (III) : élaboration de  
tableaux divers (IIIE) : création de lignes  
et de colonnes par cumul**

*Les cahiers de l'analyse des données*, tome 22, n° 3 (1997),  
p. 301-318

<[http://www.numdam.org/item?id=CAD\\_1997\\_\\_22\\_3\\_301\\_0](http://www.numdam.org/item?id=CAD_1997__22_3_301_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  
(III E) : CRÉATION DE LIGNES ET DE COLONNES  
PAR CUMUL**

**[SOURCES PASCAL (III E)]**

*J.-P. & F. BENZÉCRI*

**0 Introduction : adjonction, à un tableau de base, de cumuls de lignes et de cumuls de colonnes**

On sait (cf. ID) que l'objet du programme 'qorlsup' est d'adjoindre à l'analyse d'un tableau de correspondance de base,  $I \times J$ , des tableaux supplémentaires de lignes ou de colonnes.

Parfois, un tableau supplémentaire  $I \times J$  est constitué de lignes afférentes à des individus:  $ia$ , non pris en compte dans l'analyse principale; de même, un tableau:  $I \times J_b$  (présenté comme:  $J_b \times I$ , cf. ID§0), donnera, de l'ensemble  $I$  des individus de base, une description nouvelle fondée sur un ensemble  $J_b$  de variables d'un autre ordre que celles de  $J$ .

Mais il se peut, également, que des lignes et colonnes supplémentaires soient créées, par cumul, à partir du tableau de base lui-même. Par exemple, si  $I$  est un ensemble de patients, on créera une ligne:  $ia$ , cumul de celles afférentes à un sous-ensemble:  $I(ia)$  de patients; ayant reçu un certain traitement, ou traités dans un certain centre. Et le profil de  $ia$  sera la moyenne de ceux des individus:  $i$ , compris dans  $I(ia)$ .

Ou encore, si chaque ligne donne un bilan suivant un ensemble  $J$  de postes (e.g.: le bilan de la population de l'agglomération:  $i$ , suivant un ensemble:  $J$ , de professions), on créera un bilan moins détaillé suivant un ensemble  $J_b$  de postes dont chacun:  $jb$ , est le cumul d'une partie:  $J(jb)$ , de  $J$ .

Or, dans les études cliniques, l'objectif est ordinairement de découvrir et de confirmer la supériorité d'un traitement, la sensibilité d'un certain type de patients; etc. Supposons que, dans ce cas, la ligne de cumul:  $ia$ , afférente au sous-ensemble:  $I(ia)$ , recevant le traitement ou rentrant dans le type, se sépare de la moyenne générale: la question se pose, alors, d'apprécier si l'écart observé dépasse l'ordre de grandeur des fluctuations d'échantillonnage.

Dans ce but, la statistique paramétrique classique, s'est appliquée à calculer la probabilité qu'un sous-échantillon, de même effectif que celui considéré, mais tiré au hasard, s'écarte de la moyenne autant ou plus que celui-ci. Il faut toutefois prendre garde que les méthodes paramétriques ne s'appliquent que si les données ont un format déterminé; et que les calculs sont d'autant plus complexes que les effectifs en cause sont moins élevés.

L'outil informatique permet de suivre une autre voie. Soit: cardI le nombre total des sujets; cardA, le nombre de ceux compris dans I(ia); on crée par tirage aléatoire, un certain nombre, cardS, de groupes Is, pris dans I, et dont l'effectif est cardA; et pour chacun de ces groupes, on calcule une ligne de cumul, is. Dès lors, on peut apprécier l'écart de I(ia) à la moyenne générale de I en comparant le point de cumul: ia, au nuage des points de cumul: is, afférents aux cardS sous-ensembles créés par tirage. Cette comparaison obéit, certes, aux principes de la statistique classique; mais l'application en est ici facile, quand, e.g., le point ia et le nuage des {is} s'affichent dans le plan (1,2) issu d'une analyse factorielle.

Parfois, la composition de l'ensemble I(ia), par exemple quant à l'âge des sujets, pourrait, autant et plus que le traitement reçu, contribuer à écarter le point ia de la moyenne générale (ou centre du nuage I). Dans ce cas (si les effectifs restreints ne permettent pas de conduire des études partielles, par tranche d'âge), Il faut exiger que les groupes: Is, créés par tirage, aient, quant à l'âge (ou à tout autre variable que l'on voudra considérer), la même composition que I(ia). Ce qui complique le tirage aléatoire.

Les considérations ci-dessus, font comprendre le plan du présent exposé. Les §§1 et 2 donnent un listage commenté du programme 'cums', dont la fonction est d'adjoindre à un tableau de base des lignes et colonnes créées par cumul.

Au §1, est le programme principal, dont les fonctions sont: choix du tableau de base; appel de la procédure: 'cmd', de cumul proprement dite; choix des tableaux à créer (tableau de base complété, ou tableau des lignes ou colonnes créées seules,...).

Le §2, renferme l'unité: 'ucumd.p', qui contient la procédure de cumul: 'cmd'. De façon précise, comme pour les éléments supplémentaires en analyse de correspondance, cf. IB§3.2, un sous-ensemble à cumuler peut être spécifié soit par dialogue à l'écran (avec la procédure 'ensembler' de l'unité 'uens.p', expliquée dans IA§3), soit par un fichier de texte.

Le §3, est consacré au programme: 'cumhas', qui crée des listages définissant, suivant un format approprié, des sous-ensembles extraits par tirage aléatoire, d'un ensemble, I ou J, de cardinal donné.

## 1 Cumuls de lignes et de colonnes: le programme principal 'cums'

### 1.1 Déclarations du programme 'cums'

```

program cumsum;
uses memtypes, quickdraw, osintf, toolintf, sane, uver5;
var i, j, carsi, cari, carj, cari0, carj0, tepr: integer;
nomba, nomf, titre, chaine, sgt: str255;
repc, rpf: char;
t1k, t2k: tk; s1i, s2j, s3i, s4j: ptr;
sg1i, sg2j: pzgi; pt1i: pti;
{$S-entr} {$U-ulire5}
procedure litab(var pj: ptr; pk, pn, pt: ptr; var rc: char;
    var ic, jc: integer; var pi: ptr); external;
{$S-segcmd} {$U-uens5} {$U-ucumd5}
procedure cmd(pi, pj, pk, pn, ps: ptr; var cri, crj, csi, epr: integer);
external;
{$S-sort} {$U-ucrire5}
procedure critab(pi, pj, pk, pn, pt: ptr; ic, jc, tc: integer);
external;

```

Les déclarations sont semblables à celles des programmes déjà étudiés. On distingue trois segments: {\$S entr}, entrée du tableau de base, par la procédure 'litab', de l'unité: 'ulire.p', cf. IA§2; {\$S segcmd}, cumul proprement dit: cf. *infra*, §2; et: {\$S sort}, sortie de tableaux comportant lignes et colonnes créées, par la procédure 'critab', expliquée dans IID§5.

### 1.2 Choix du tableau de base

```

begin Benzecri; repc := 'O';
while not (repc = 'N') do begin
    writeln('ce programme crée des lignes et colonnes nouvelles par cumul');
    writeln('ATTENTION: si le sigle de l''élément à créer commence par le');
    writeln('signe -, les nombres calculés seront affectés de ce signe');
    writeln('          si le sigle de l''élément à créer commence par le');
    writeln('signe /, les nombres calculés seront inversés');
    writeln('          si le sigle de l''élément à créer commence par le');
    writeln('signe *, on calcule un produit (non une somme)');
    repc := 'N';
end;

```

Le programme 'cums' est constitué par une boucle unique: 'while not (repc='N')...'; laquelle s'exécute autant de fois que l'utilisateur le demande, sous réserve qu'un échec dans le choix du tableau de base n'ait donné au caractère: repc, la valeur 'N'.

Les avertissements quant à l'effet de la forme des sigles sont proposés en tête de chaque itération. Ils valent dans tous les cas: que le cumul soit défini par dialogue à l'écran, ou par un fichier de commande. Dans ce dernier cas, comme on le verra au §2, on peut aussi calculer des pourcentages, introduire des coefficients. En effet, partant des calculs de cumul, dont on a rappelé, au §0, l'intérêt pour l'analyse des données, on a, par degrés successifs, étendu les fonctions du programme 'cums'; notamment pour créer des variables combinant de diverses manières les variables de base. Le produit d'une famille vide de variables étant pris égal à la constante 1 (cf. *infra*), on peut finalement, par le jeu des signes et coefficients, créer une combinaison linéaire quelconque des variables de base.

```

while not (repc='O') do begin
  write('le nombre maximum de lignes nouvelles à créer est ');
  readln(carsi);
  if (carsi<0) then carsi:=0;if (carsi>200) then carsi:=200;
  writeln('    max =', carsi:4);
  write('faut-il éliminer des calculs une fraction des éléments O ou N ');
  readln(rpf);tepr:=0;
  if (rpf='O') then begin
    write('éliminer un demi(2), un tiers(3) ou un quart(4) ');
    readln(tepr);if (tepr<2) then tepr:=2;if (4<tepr) then tepr:=4;
    writeln('    on éliminera 1/',tepr) end;
  write('ces choix sont-ils confirmés O ou N ');readln(repc) end;

```

Pour créer des lignes nouvelles (dont le nombre maximum est, arbitrairement, fixé à 200), on doit réserver de l'espace en mémoire au delà de ce qui est requis pour le tableau de base (cf. IA§2.1): à cette fin on demande à l'utilisateur de fixer, *a priori*, le nombre carsi. Si l'on crée seulement des colonnes supplémentaires, carsi=0; mais le nombre total des colonnes, celles données avec le tableau de base et celles créées ensuite, ne peut dépasser: jmax=300 (cf. IA§1).

On a prévu le choix d'éliminer systématiquement, de tous les cumuls demandés, une certaine fraction des lignes ou des colonnes. Du point de vue de l'analyse discriminante, la fraction exclue des cumuls joue le rôle d'un échantillon d'épreuve. En effet, supposons qu'il s'agisse, e.g., de discriminer entre deux parties I1 et I2 de I. On construit, par cumul, les centres de gravité, i1 et i2, de I1 et I2. Tout individu, i, est rattaché à la classe du centre de laquelle il est le plus proche. Si, plus précisément, les individus dont le rang est un multiple de 4 ne sont pas pris en compte dans le calcul de i1 et i2, l'affectation d'un de ces individus se fait suivant une règle établie sans lui; règle dont on apprécie ainsi l'efficacité potentielle en dehors de l'échantillon de base (3/4 de I) sur lequel elle est fondée. On trouve dans [TEXTES GRECS] (§3.1; in: CAD, Vol.XVI, n°1) des comparaisons entre analyses discriminantes, faites avec ou sans échantillon d'épreuve.

```

cari:=carsi;
litab(s2j,@t1k,@nomba,@titre,repc,cari,carj,s1i);
if (repc='O') then begin
  writeln(titre);writeln('cari = ',cari:4,'    carj = ',carj:4);
  rpf:='N';cari0:=cari;carj0:=carj;
  while not (rpf='O') do begin rpf:='N';
    write('le sigle du tableau à créer est ');
    readln(chaine);
    if (4<length(chaine)) then sgt:=copy(chaine,1,4) else sgt:=chaine;
    writeln(sgt);
    write('ce sigle est-il confirmé oui(O) ou non(N) ');readln(rpf) end;

```

La procédure 'litab' est appelée en réservant l'espace pour un nombre: carsi, de lignes supplémentaires. Si l'utilisateur a pu fixer son choix sur un tableau: disq:tab, accessible sur disque, l'exécution se poursuit en demandant le sigle du tableau à créer; ce sigle: sgt, (précédé du caractère 'Σ', qui évoque une somme), figure, après le nom du tableau de base, dans celui des divers tableaux créés; ainsi que dans le listage des cumuls effectués (cf. *infra*).

### 1.3 Choix des tableaux à créer sur disque

```
cmd(sli, s2j, @t1k, @nomba, @sgt, cari, carj, carsi, tepr);
unloadseg(@cmd);
writeln('faut-il créer un tableau complet, formé du tableau initial');
write('complété par les lig et col créés O ou N ');
readln(rpf);
if not (rpf='N') then begin
  nomf:=concat (nomba, 'Σ', sgt, '.w');
  critab(sli, s2j, @t1k, @nomf, @titre, cari, carj, 0);
  unloadseg(@critab);end;
```

Après que, par la procédure 'cmd', objet du §2 (cf. *infra*), on a créé lignes et colonnes nouvelles, viennent les demandes de création de tableaux; plus précisément de transfert de la mémoire centrale à une mémoire externe (disque). Le cas du tableau complété (nommé:  $\text{disq:tab}\Sigma\text{sgt.w}$ ) est très simple: en bref, la procédure 'critab' (cf. IID§5), est appelée avec pour arguments les pointeurs mêmes afférents au tableau de base; les valeurs des nombres: cari, carj, de lignes et colonnes ayant pu croître au delà de leurs valeurs initiales, conservées dans: cari0, carj0.

```
if (carj0<carj) then begin
  writeln('faut-il créer un tableau partiel, formé seulement');
  write('des colonnes qui ont été créés, O ou N ');
  readln(rpf);
  if not (rpf='N') then begin
    nomf:=concat (nomba, 'Σ', sgt, 'c.w');
    sg2j:=pzgi (s2j); s4j:=@sg2j^[carj0+1]; t2k[0]:=t1k[0];
    for j:=1 to carj-carj0 do t2k[j]:=t1k[carj0+j];
    critab(sli, s4j, @t2k, @nomf, @titre, cari0, carj-carj0, 0);
    unloadseg (@critab);end;
```

Si un ensemble  $J_c$  de colonnes nouvelles a été effectivement créé ( $\text{carj0} < \text{carj}$ ), le choix est offert de créer un tableau  $I \times J_c$ ; avec un nom se terminant par la lettre 'c':  $\text{disq:tab}\Sigma\text{sgtc.w}$ . Mais quant aux pointeurs:  $s4j$ , pour les sigles des colonnes, et:  $t2k$ , pour le bloc des colonnes, il faut prendre garde que la lecture des sigles ou des nombres ne doit commencer qu'après le rang:  $\text{carj0}$ .

```
writeln('faut-il créer un tableau transposé, ayant pour lignes');
write('les colonnes-qui ont été créés, O ou N ');
readln(rpf);
if not (rpf='N') then begin
  nomf:=concat (nomba, 'Σ', sgt, 'b.w');
  sg2j:=pzgi (s2j); s4j:=@sg2j^[carj0+1]; t2k[0]:=t1k[0];
  for j:=1 to carj-carj0 do t2k[j]:=t1k[carj0+j];
  critab(sli, s4j, @t2k, @nomf, @titre, cari0, carj-carj0, 1);
  unloadseg (@critab);end;end;
```

On sait que pour être adjoint en supplément à l'analyse du tableau de base,  $I \times J$ , un tableau de colonnes supplémentaires:  $I \times J_c$ , doit être présenté sous forme transposée,  $J_c \times I$  (cf. ID§0). Dans ce cas, le nom se termine par 'b'; soit:  $\text{disq:tab}\Sigma\text{sgtb.w}$ . Les pointeurs sont fixés comme pour écrire le tableau non transposé; l'appel de la procédure 'critab' est seulement modifié quant à la valeur du dernier argument: 1; lequel commande la transposition (cf. IID§5, *in fine*).

```

if (cari0<cari) then begin
  writeln('faut-il créer un tableau partiel, formé seulement');
  write('des lignes qui ont été créées, O ou N ');
  readln(rpf);
  if not (rpf='N') then begin
    nomf:=concat(nomba, 'Σ', sgt, 'a.w');
    sgli:=pzqi(sli); s3i:=@sgli^[cari0+1]; t2k[0]:=t1k[0];
    for j:=1 to carj0 do begin
      ptli:=pti(t1k[j]); t2k[j]:=@ptli^[cari0+1]; end;
      critab(s3i, s2j, @t2k, @nomf, @titre, cari-cari0, carj0, 0);
      unloadseg(@critab); end; end;

```

De même, si un ensemble la de lignes nouvelles a été effectivement créé (cari0<cari), le choix est offert de créer un tableau Ia×J; avec un nom se terminant par la lettre 'a'. Et l'on procède comme pour le tableau des colonnes supplémentaires; à ceci près qu'il n'y a pas lieu de proposer une transposition.

```

for j:=0 to carj do dispose(t1k[j]); dispose(s1i); dispose(s2j);
write('faut-il reprendre avec un autre tableau O ou N ');
readln(repc); end;
end;
readln(rpf); end.

```

Comme de règle, l'exécution de la boucle générale: 'while not (repc='N')...', de traitement d'un tableau de base (cf. *supra*, §1.2), se termine en libérant l'espace réservé, en mémoire centrale, par divers pointeurs.

## 2 Création de lignes et de colonnes: la procédure 'cmd'

### 2.1 En-tête de l'unité 'ucumd' et déclarations de la procédure 'cmd'

```

UNIT ucumd;
INTERFACE uses memtypes, quickdraw, osintf, toolintf, sane, uver5;
procedure cmd(pi, pj, pk, pn, ps: ptr; var cri, crj, csi, epr: integer);
implementation
procedure ensembler (carte: integer; plmt, padj, psigt, psu: ptr);
  external;
procedure cmd;
var
i, j, q, qa, qb, carti, carsj, crq, ni, nj, nz, nc, cri0, crj0, ffc, mult: integer;
totr: real;
rpa, rpb, rpf, rpl, rpq, rpx, rpm, carac: char; ffl, fft: longint;
sgq, sig, lmt, adj, chaine: string;
psgt, pnomb: stringptr; sgli, sglj, sglq: pzqi;
pq, pg: ptr; gaq: ptni; klji: kji; ptlk: ptk;
f0, ft, f2, fu, f4: text;

```

Ainsi qu'on l'a annoncé au §1.1, la procédure 'cmd' appelle 'ensembler', pour définir, par dialogue, un sous-ensemble de lignes ou de colonnes. Le qualificatif: adj, des éléments (lmt) choisis est ici: 'à combiner'; selon les appels, on a: lmt='ligne', ou: 'colonne'.

Si les sous-ensembles ont été spécifiés, au préalable, sur un listagede commande, le nom de celui-ci doit être formé en ajoutant au nom du tableau de base, le sigle: sgt, du tableau à créer, précédé de 'Σ', et suivi de 'Σx'; soit: disq:tabΣsgtΣx. Des procédures: 'lecsig' et: 'lecnombre', analogues à celles déjà décrites en IA§2.3, servent à lire, sur le listage de spécification, les informations utiles.

```

procedure lecsig; begin sig := '';
  while (ord(carac) in [0..32]) and not eof(fu) do read(fu, carac);
  while not ((ord(carac) in [0..32]) or (length(sig)=4)) do begin
    sig := concat(sig, carac);
    if eof(fu) then carac:= ' ' else read(fu, carac) end;end;
procedure lecnombre;begin ffl:=0;
  while not ((ord(carac) in [48..57]) or eof(fu)) do read(fu,carac);
  while (ord(carac) in [48..57]) do begin
    ffl:=(10*ffl)+(ord(carac)-48);
    if eof(fu) then carac:= ' ' else read(fu,carac) end;
  ffc:=ffl mod 100000 end;

```

Plus précisément 'lecnombre', après avoir lu un entier long: ffl, calcule en fonction de celui-ci, le reste: ffc, de la division par 100000. Il faut comprendre que, dans un listage de commande (fichier: disq:tab $\Sigma$ sgt $\Sigma$ x), les entiers remplissent deux fonctions: d'une part, introduire un ordre de cumul, ou instruction composée, pour créer un élément nouveau; d'autre part, spécifier, dans un tel ordre, un par un ou par bloc, les éléments à cumuler .

exemple d'instruction de cumul  
50020021 Cinq colonne 10001 ; 10 ; 12 ;

L'entier de tête: ffl, est, en général, 20000, augmenté si l'utilisateur le désire, d'un entier donnant le rang de l'élément à créer (e.g.: 21 pour une colonne ajoutée à un tableau qui en a déjà 20). Mais, de plus, si ffl est augmenté d'un multiple de 100000: mult  $\times$  100000 (e.g., dans l'exemple: mult=500), les composantes calculées sont multipliées par: mult. Le reste: ffc, doit donc être comparé à 20000, en calculant le quotient: ffc div 20000 (cf. *infra*, §2.2).

Quant aux entiers désignant des éléments, ce sont les rangs mêmes de ceux-ci: e.g.: 12, pour la douzième colonne (ou ligne); mais si le rang est augmenté de 10000, il signale le début d'un bloc: e.g. la séquence: 10001 ; 10 , désigne, de façon concise, le bloc des dix premières colonnes du tableau en cours de traitement.

## 2.2 Début de la procédure 'cmd' et structure générale de boucle

```

begin
adj:='à combiner';nc:=0;cri0:=cri;crj0:=crj;ffl:=0;mult:=1;
ptlk:=ptk(pk);
for j:=1 to crj do klji[j]:=pti(ptlk^[j]);
sgli:=pzgi(pi);sglj:=pzgi(pj);
psgt:=stringptr(ps);pnomb:=stringptr(pn);
carsj:=jmax-crj;if (carsj<0) then carsj:=0;
carti:=cri+csi;
q:=jmax;if (q<carti) then q:=carti;
pg:=newptr(2*q);gaq:=ptni(pg);

```

Les arguments de la procédure 'cmd' sont des entiers ou des pointeurs de type général (déclarés comme: ptr). En donnant à chaque pointeur un type convenable, 'cmd' accède aux informations que le programme principal a rangées en mémoire centrale.



```

if (0=carsj) and (0=csi) then rpl:='F';
rewrite(ft,concat(pnomb^,'Σ',psgt^,'Σ'));
writeln(ft,concat(pnomb^,'Σ',psgt^,'Σ'));
chaine:=concat(pnomb^,'Σ',psgt^,'Σx');rpx:='N';
if (verif(stringptr(@chaine))=2) then begin
  rpx:='O';reset(fu,chaine);writeln(chaine);readln(fu,chaine);
  carac:=' ';lecnombre;if not(ffc div 20000=1) then rpl:='F' end;

```

S'il n'y a place ni pour ligne (csi=0) ni pour colonne (carsj=0), aucun nouvel élément ne peut être créé; ce que marque l'ordre: rpl:='F', (i.e.: Fin).

On crée un listage descriptif dont le nom: disq:tabΣsgtΣ, ne diffère de celui d'un éventuel listage de commande qu'en ce qu'il se termine par 'Σ' (et non: 'Σx'). Ce listage, donne les spécifications détaillées des éléments créés sous une forme explicite: chacun des éléments combinés étant désigné par son numéro suivi de son sigle. Tel quel, (sous réserve qu'on ajoute à son nom la lettre: 'x'), le listage descriptif peut servir de listage de commande; l'omission des sigles et la spécification par bloc n'étant qu'une tolérance admise pour abrégé l'écriture de celui-ci.

On s'enquiert d'un fichier de commande: disq:tabΣsgtΣx. Si celui-ci existe, on y lit une ligne de titre; puis un premier nombre: ffl, servant d'en-tête à un ordre de création. si le nombre: ffc, associé à ffl, ne convient pas, la procédure est arrêtée par l'ordre: rpl:='F' (cf. *supra*).

Le programme entre alors dans une boucle générale de création d'élément (ligne ou colonne); laquelle se poursuit tant que rpl ne prend pas la valeur: 'F'.

### 2.3 Entrée dans le boucle de création

```

while not(rpl='F') do begin rpl:='*';
  ffl:=ffl div 100000;mult:=ffl;if (mult<1) then mult:=1;
  if (rpx='N') then while not ((rpl in ['C','L','F']) and (rpf='O')) do begin
    write('faut-il créer ');
    if (csi>0) then write('une ligne(L) ');
    if (carsj>0) then write('une colonne(C) ');
    write('ou finir(F) ');readln(rpl);
    if (csi<=0) and (rpl='L') then rpl:='*';
    if (carsj<=0) and (rpl='C') then rpl:='*';
    rpf:='*';
    if (rpl in ['C','L','F']) then begin
      write('ce choix est il confirme O ou N ');
      readln(rpf) end end;
  if (rpx='O') then begin lecsig;sgq:=sig;lecsig;rpl:='F';
    if (sig='lign') and (0<csi) then rpl:='L';
    if (sig='colo') and (0<carsj) then rpl:='C';
    writeln(sgq:7,sig:7,' ',rpl) end;
  if (rpl='L') then begin lmt:='ligne';
    ni:=cri+1;pq:=pi;crq:=cri;sglq:=sgli end;
  if (rpl='C') then begin lmt:='colonne';
    nj:=crj+1;pq:=pj;crq:=crj;sglq:=sglj end;
  rpf:='N';

```

S'il y a un listage de commande: ...Σx', le dernier nombre lu (avant d'entrer dans la boucle, ou à la fin de l'itération précédente) comporte un

multiplicateur, mult, qui peut être supérieur à 1. S'il n'y a pas de listage: ' $\dots\Sigma x$ ', (rpx='N'), le nombre: mult, est sans objet; on offre à l'utilisateur de créer Ligne ou Colonne, selon la place disponible; et son choix se fixe entre: {'F', 'L', 'C'}; i.e.: Fin, Ligne ou Colonne.

S'il y a un listage ' $\dots\Sigma x$ ', le choix entre ligne et colonne est lu comme un sigle (formé des quatre premières lettres du mot écrit sur ' $\dots\Sigma x$ '; cf. *supra*, exemple); l'échec équivaut à une Fin. En cas de succès, on a saisi, au passage, le sigle: sqq, de l'élément à créer. Quel que soit le mode de spécification, il faut garder le compte des éléments existant (ou en projet immédiat); rpf est mis à 'N', en vue d'un retour dans la boucle, avec choix à faire.

#### 2.4 Acquisition de la spécification d'une ligne ou d'une colonne

```

if (rpl in ['C', 'L']) then begin nc:=nc+1;
  if (rpx='N') then while not (rpf='O') do begin rpf:='N';
    write('le sigle de la ',lmt,' à creer est ');
    readln(chaine);
    if (4<length(chaine)) then sqq:=copy(chaine,1,4) else sqq:=chaine;
    writeln(sqq);
    write('ce sigle est-il confirmé oui(O) ou non(N) ');
    readln(rpf) end;
  rpq:=sqq[length(sqq)];rpm:=sqq[1];
  for q:=1 to crq do gaq^[q]:=0;nz:=0;
  if (rpx='O') then begin rpa:='N';rpb:='N';lecnombre;qb:=ffc;
    if (10000<qb) then begin qb:=qb-10000;rpb:='O' end;
    while ((0<qb) and (qb<=crq)) do begin
      if (rpa='O') then for q:=qa to qb do gaq^[q]:=1;
      gaq^[qb]:=1;lecsig;lecnombre;qa:=qb;rpa:=rpb;rpb:='N';qb:=ffc;
      if (10000<qb) then begin qb:=qb-10000;rpb:='O' end end end;
  if (rpx='N') then assembler(crq,@lmt,@adj,pq,pg);
  fft:=20001+crq;if (l<mult) then fft:=fft+(100000*mult);
  writeln(fft:10,sqq:5,' ',lmt,' combine les ',lmt,' s suivantes');
  writeln(ft,fft:10,sqq:5,' ',lmt,' combine les ',lmt,' s suivantes');
  if not(epr=0) then for q:=1 to crq do if (q mod epr=0) then gaq^[q]:=0;
  for q:=1 to crq do if (gaq^[q]=1) then begin nz:=nz+1;
    write(q:4,sigler(sglq^[q]):5,' ');
    if (nz mod 6=0) then writeln;
    write(ft,q:4,sigler(sglq^[q]):5,' ');
    if (nz mod 6=0) then writeln(ft) end;
  if not(nz mod 6=0) then begin writeln(ft); writeln end;
end;

```

En dialogue, il faut d'abord demander le sigle: sqq; s'il y a un listage ' $\dots\Sigma x$ ', on sait que le sigle y a déjà été lu. Dans tous les cas, on relève, dans sqq, la 1-ère et la dernière lettre qui peuvent être des commandes; et on met à zéro la fonction caractéristique: gaq.

S'il y a un listage: ' $\dots\Sigma x$ ', on y lit la fonction gaq, en tenant compte des blocs. Il faut prendre garde que la lecture de gaq s'achève sur un nombre ffl, inadéquat pour le numérotage d'éléments à combiner; mais on s'arrête avec, en mémoire, un couple: {ffl, ffc} qui peut préparer la spécification de l'élément suivant: ce qu'on vérifie en fin de boucle, §2.5.3, par la condition de sortie: not(ffc div 20000=1).

Faute de listage ' $\dots x$ ', il faut, pour choisir les éléments à combiner, un dialogue, régi par la procédure 'ensembler'.

S'il y a un échantillon d'épreuve (condition: if not(epr=0)...; cf. *supra*, §1.2), gaq doit être modifié en éliminant les éléments dont le rang: q, est congru à zéro modulo epr: (q mod epr=0). Une fois que gaq est fixé, on affiche à l'écran l'ensemble d'éléments ainsi défini; et on le copie sur le listage explicite (avec sigles): disq:tab $\Sigma$ sgt $\Sigma$ , toujours créé.

## 2.5 Calcul d'un élément à créer en mémoire centrale; en tenant compte de toutes les lettres de commandes

### 2.5.1 Création d'une ligne

```
if (rpl='L') then begin sqli^[ni]:=zigler(sgq);
  if (rpm='*') then begin
    for j:=1 to crj do klji[j]^[ni]:=1;
    for i:=1 to cri do if (gaq^[i]=1) then
      for j:=1 to crj do klji[j]^[ni]:=klji[j]^[ni]*klji[j]^[i];end
  else begin
    for j:=1 to crj do klji[j]^[ni]:=0;
    for i:=1 to cri do if (gaq^[i]=1) then
      for j:=1 to crj do klji[j]^[ni]:=klji[j]^[ni]+klji[j]^[i];end;
  if (rpq='%') then begin totr:=0;
    for j:=1 to crj0 do totr:=totr+klji[j]^[ni];
    if '0<totr) then totr:=100/totr else totr:=0;
    for j:=1 to crj do klji[j]^[ni]:=totr*klji[j]^[ni] end;
  if (rpm='/') then for j:=1 to crj do if not(klji[j]^[ni]=0) then
    klji[j]^[ni]:=1/klji[j]^[ni];
  if (mult<1) then mult:=1;if (rpm='-') then mult:=-mult;
  if (1<mult) then for j:=1 to crj do klji[j]^[ni]:=mult*klji[j]^[ni];
  cri:=cri+1;csi:=csi-1 end;
```

Mérite seul commentaire l'effet des caractères de commande.

Si le sigle commence par une étoile (rpm='\*'), on donne d'abord la valeur 1 (unité), à tous les éléments de la ligne à créer. Puis les lignes: i, à combiner (celles pour lesquelles: gaq^[i]=1), sont introduites par multiplication. Sinon, on procède par addition, à partir d'une ligne de zéros.

Si le sigle se termine par: '%', la ligne créée est réduite à avoir pour total: 100. Et, si le sigle commence par une barre de fraction: rpq='/', toutes les composantes de la ligne créée sont inversées.

En dernier lieu, est, éventuellement, introduite une constante multiplicative: mult. S'il n'y a pas de listage de commande: ' $\dots x$ ', le nombre: ffl, est mis à zéro en tête du programme, pour ne plus varier ensuite; et: mult ne peut valoir que +1 ou -1. S'il y a un listage ' $\dots x$ ', mult est d'abord calculé comme un nombre positif, à partir du nombre: ffl, lu en tête de l'instruction de création. De plus, que le listage: ' $\dots x$ ' existe ou non, mult est affecté du signe '-', si le sigle de l'élément à créer commence par ce signe: rpm='-'. Enfin, si: mult $\neq$ 1, on le fait agir sur toute la ligne créée.

### 2.5.2 Création d'une colonne

```

if (rpl='C') then begin sglj^[nj]:=zigler(sgg);
  ptlk^[nj]:=newptr(4*carti);klji[nj]:=pti(ptlk^[nj]);
  if (rpm='*') then begin
    for i:=1 to cri do klji[nj]^i:=1;
    for j:=1 to crj do if (gaq^j)=1 then
      for i:=1 to cri do klji[nj]^i:=klji[nj]^i*klji[j]^i;end
  else begin
    for i:=1 to cri do klji[nj]^i:=0;
    for j:=1 to crj do if (gaq^j)=1 then
      for i:=1 to cri do klji[nj]^i:=klji[nj]^i+klji[j]^i;end;
  if (rpx='%') then begin totr:=0;
    for i:=1 to cri do totr:=totr+klji[nj]^i;
    if (0<totr) then totr:=100/totr else totr:=0;
    for i:=1 to cri do klji[nj]^i:=totr*klji[nj]^i end;
  if (rpm='/') then for i:=1 to cri do if not(klji[nj]^i=0) then
    klji[nj]^i:=1/klji[nj]^i;
  if (mult<1) then mult:=1;if (rpm='-') then mult:=-mult;
  if (1<mult) then for i:=1 to cri do klji[nj]^i:=mult*klji[nj]^i;
  crj:=crj+1;carsj:=carsj-1 end;

```

La création d'une colonne est en tout analogue à celle d'une ligne. C'est seulement parce que, par les pointeurs, l'accès aux colonnes diffère de l'accès aux lignes, qu'on a, pour plus de clarté, séparé deux blocs: rpl='L', et: rpl='C'.

### 2.5.3 Fin de la boucle de création et sortie de la procédure 'cmd'

```

if (rpx='O') and not(ffc div 2000=1) then begin
  rpl:='F';writeln('fini') end;
if (0=carsj) and (0=csi) then rpl:='F' end;
close(ft);if (rpx='O') then close(fu);dispose(gaq);
end;end.

```

La grande boucle de création s'achève, éventuellement, avec un ordre de sortie: rpl='F'; parce que le nombre lu sur: ' $\dots \Sigma x$ ' est inadéquat; ou qu'il ne reste plus, en mémoire centrale, d'espace réservé pour y compléter le tableau traité.

Au sortir de la boucle, on ferme le fichier: ft, utilisé pour écrire le listage: disq:tab $\Sigma$ sgt $\Sigma$ , décrivant, en détail, les éléments créés. De même, s'il y a un fichier de commande, on ferme: fu, qui servait à le lire. Le tableau: gaq, caractérisant l'ensemble des éléments à combiner, est éliminé de la mémoire centrale. C'est la fin de la procédure: 'cmd'; et de l'unité: 'ucumd'.

## 3 Définition de sous-ensembles par tirage aléatoire: le programme 'cumhas'

### 3.0 Fonction du programme 'cumhas'

Pour créer une nouvelle ligne (ou colonne), cumul d'éléments du tableau de base, l'utilisateur du programme 'cums' peut désigner l'ensemble de ces éléments soit par dialogue à l'écran, soit par un listage de commande: ' $\dots \Sigma x$ '.

Dans certains cas, le sous-ensemble des éléments à cumuler est défini par une certaine propriété; e.g.: ensemble IT des lignes afférentes aux sujets ayant

```

définition de sous-ens SE; carEns =      100; carSE =      30
l'ensemble Ens est divisé en:  3 segments
Spécification des segments S de Ens
rang de S      début      fin      pts pris dans S
      1          1         20         12
      2         21         50         15
      3         51        100          3

20001      1 ligne
1 ;   3 ;   4 ;   5 ;   6 ;   7 ;  11 ;  12 ;  13 ;  16 ;
17 ;  18 ;  21 ;  23 ;  24 ;  26 ;  27 ;  28 ;  29 ;  34 ;
36 ;  37 ;  38 ;  39 ;  42 ;  43 ;  46 ;  62 ;  63 ;  88 ;
20001      2 ligne
1 ;   2 ;   3 ;   4 ;   7 ;   8 ;   9 ;  10 ;  11 ;  13 ;
18 ;  19 ;  21 ;  24 ;  29 ;  33 ;  34 ;  35 ;  37 ;  38 ;
40 ;  42 ;  43 ;  45 ;  47 ;  49 ;  50 ;  51 ;  72 ;  81 ;
20001      3 ligne

```

reçu le traitement T; mais il se peut également qu'afin d'apprécier la validité d'une conclusion suggérée par l'ensemble des cas observés, on cumule des ensembles Is, de même cardinal que IT, créés par tirage aléatoire. Éventuellement, cf.§0, on exige que les groupes: Is, aient, quant à l'âge (ou à une autre variable), la même composition que IT.

La fonction du programme 'cumhas' est de créer un listage pour commander le cumul de tels groupes aléatoires: Is. Ainsi, le listage: Disq:ΣhasxA, publié ci-dessus, commande la création de sous-ensembles: SE, de 30 éléments, extraits d'un ensemble: Ens, de 100 éléments. De façon précise, Ens est divisé en 3 classes consécutives: {1...20}, {21...50}, {51...100}; et chaque sous-ensemble SE prend respectivement, dans ces classes: 12; 15 et 3 individus. On notera que le sigle d'un sous-ensemble est réduit à son numéro d'ordre: '1', '2', '3',...

Il va sans dire que, pour commander le traitement d'un tableau de base: Disq:tab, le listage doit recevoir un nom tel que: disq:tabΣsufΣx; le tableau de spécification des blocs ayant été éliminé.

### 3.1 Déclarations du programme 'cumhas' et structure générale de boucle

Le listage de commande étant créé sans préciser le tableau même auquel il est destiné, l'utilisateur doit seulement donner le nom du disque: Disq, où garder ce listage; tel est l'objet de la procédure: 'definir'.

La création d'un listage fait l'objet de la boucle générale: 'while (repc='O') do begin...'. Le nom du listage est: Disq:Σhasx, suivi d'une capitale qui peut aller de 'A' à 'Z'; quand le rang de l'itération croît de 1 à 26. Si l'on prétend créer plus de 26 listages de commande en un seul appel de 'cumhas', le 27-ème listage est créé avec le nom du 1-er, qu'il élimine du disque. Cette limite (d'ailleurs arbitraire, car on pourrait ajouter deux lettres au lieu d'une...) ne nous paraît pas présenter d'inconvénient.

```

program cumhas;
uses memtypes, quickdraw, osintf, toolintf, sane, uver5;
var ft:text;
repc, reps:char;
nomba, nomf:string;
izl, erl, i, id, ia, carj, j, cari, carsi, toti, cars, es, as, iter,
  meps, kies, crds, qots, mxis, pris:integer;
prisi:ptni;paux:ptr;
kis, fins, eps, mxs:array[0..12]of integer;
procedure definir; begin repc:='N';erl:=0;
while not((repc='O') or (erl=6)) do begin
  write('le nom du disque à utiliser est ');readln(nomba);
  if not(nomba[length(nomba)]=':') then nomba:=concat(nomba, ':');
  nomf:=concat(nomba, 'Σhasx');
  izl:=verif(stringptr(@nomf));
  if (izl=0) then begin erl:=erl+1;
    writeln('ERREUR le disque manque') end
  else begin id:=2;
    while not(nomf[id]=':') do id:=id+1;
    nomba:=copy(nomf, 1, id);
    nomba:=concat(nomba, 'Σhasx');
    writeln(nomba);
    write('ce nom est il confirmé O ou N ');readln(repc) end;end;end;
begin benzecri;
writeln('ce programme définit des sous-ensembles aléatoirement');
definir;iter:=0;
while (repc='O') do begin
  repc:='N';iter:=1+(iter mod 26);nomf:=concat(nomba, chr(64+iter));

```

Parmi les variables déclarées, on a plusieurs entiers, ainsi que des tableaux, dont le nom, terminé par 's', évoque la subdivision d'un ensemble en segments. L'ensemble de base étant: I, dont le cardinal est cari, le tableau: prisi<sup>^</sup>, sert à décrire, par sa fonction caractéristique en (0,1), un sous-ensemble de I, créé par tirage aléatoire. La lettre: 'I', ainsi que la présence du mot: 'ligne', sur le listage, suggèrent qu'on cumule des lignes:  $i \in I$ ; mais on peut, par un traitement de texte, remplacer le mot: 'ligne', par ':colonne'.

### 3.2 Spécification de l'ensemble de base et de ses subdivisions

```

while not(repc='O') do begin
  write('le cardinal de l'ensemble Ens dont on extrait des parties est ');
  readln(cari);if (8000<cari) then cari:=8000;if (cari<12) then cari:=12;
  writeln('cardinal de Ens =',cari:10);
  write('le cardinal des parties à créer est ');
  readln(carsi);
  if (carsi<2) then carsi:=2;if (200<carsi) then carsi:=200;
  if ((cari div 2)<carsi) then carsi:=cari div 2;
  writeln('card des parties =',carsi:10);
  write('ces nombres sont-ils confirmés O ou N ');readln(repc) end;

```

Dans une première boucle, comprise dans la boucle générale, l'utilisateur fixe le cardinal: cari, de l'ensemble de base I, et le cardinal: carsi, des parties à créer. Quant aux bornes, nous rappelons qu'en l'état du logiciel MacSAIF, 8000 est le nombre maximum des lignes; la condition  $\text{carsi} \leq 200$  est arbitraire, mais il faut se garder de créer un listage interminable, avec  $\text{carsi} = 2000 \dots$ ; la condition:  $\text{carsi} \leq \text{cari}/2$ , est naturelle pour un groupe particulier de cas.

```

paux:=newptr(2*cari);prisi:=ptni(paux);
cars:=1;fins[0]:=0;fins[1]:=cari;kis[0]:=0;kis[1]:=carsi;

write('faut-il segmenter l'ensemble Ens O ou N ');readln(reps);
if not (reps='N') then begin
  writeln('on segmentera l'ensemble Ens');
  write('ce choix est-il confirmé O ou N ');readln(reps);
  if not (reps='O') then reps:='N';end;

```

L'espace étant réservé pour le tableau:  $prisi^A$ , (cf. *supra*), on prévoit d'abord le cas où il n'y a qu'un seul segment:  $cars:=1$ ; i.e. absence de segmentation. Un segment de rang zéro, toujours vide, est décrit avec pour dernier élément:  $fins[0]:=0$  (rang précédant le début: 1, de l'ensemble de base: I). Le segment 1, qui n'est autre que I, a, de même:  $fins[1]:=cari$ . Le nombre des éléments à prendre, aléatoirement, pour un sous-ensemble, dans les segments 0 et 1 est fixé, respectivement, par:  $kis[0]:=0$ ,  $kis[1]:=carsi$ .

L'utilisateur doit ensuite dire s'il veut une segmentation effective; et si sa réponse est: 'O' (oui), il faut donner les bornes des segments; puis spécifier le nombre des éléments à prendre dans chacun des segments.

```

if (reps='O') then begin repc:='N';
  while not (repc='O') do begin
    write('le nombre Seg, des segments de Ens est ');readln(cars);
    if (cars<2) then cars:=2;if (12<cars) then cars:=12;
    if ((cari div 2)<cars) then cars:=cari div 2;
    writeln('le nombre Seg, des segments de Ens est ',cars);
    for es:=1 to cars-1 do begin
      write('le segment ',es,' doit se terminer à l'élément de rang ');
      readln(id);
      if (id<(fins[es-1]+1)) then id:=fins[es-1]+1;
      if ((cari-(cars-es))<id) then id:=cari-(cars-es);fins[es]:=id;
      writeln('le segment ',es,' se terminera à l'élément de rang ',fins[es]);end;
    fins[cars]:=cari;
    writeln('le segment ',cars,' se terminera à l'élément de rang ',fins[cars]);
    write('ces choix sont-ils confirmés O ou N ');readln(repc);end;end;

```

Le nombre:  $cars$ , des segments peut varier de 2 à 12; avec la condition complémentaire que:  $cars$  ne peut dépasser  $cari/2$ . Les fins de segments sont fixées de telle sorte que chacun de ceux-ci ait une longueur non nulle. Un segment de longueur unité peut servir à éliminer, des sous-ensembles créés par tirage aléatoire, le point singulier auquel est réduit ce segment.

En effet, après le dialogue du découpage de I en segments, vient un autre dialogue, pour fixer le nombre d'éléments:  $kis[es]$ , à prendre, dans chaque segment:  $es$ , pour créer un sous-ensemble d'éléments à cumuler. Il est clair que  $kis[es]$  ne doit pas dépasser le nombre:  $crds=fins[es]-fins[es-1]$ , cardinal du segment de rang:  $es$ .

De plus, soit:  $as$ , le nombre total des éléments à prendre dans les segments déjà considérés; il reste à prendre:  $(carsi-as)$  éléments:  $kis[es]$  ne doit pas dépasser:  $(carsi-as)$ .

```

if (reps='O') then begin repc:='N';
  while not (repc='O') do begin as:=0;
    for es:=1 to cars do begin
      if (es<cars) then begin
        write('le nombre de points à prendre du segment ',es,' est ');
        readln(id) end;
        if (es=cars) then id:=carsi-as;
        if (id<0) then id:=0;if (carsi<(as+id)) then id:=carsi-as;
        if ((fins[es]-fins[es-1])<id) then id:=(fins[es]-fins[es-1]);
        if ((cari fins[es])<(carsi-(as+id))) then id:=carsi+fins[es]-(as+cari);
        kis[es]:=id;as:=as+id;
        writeln('on prendra ',kis[es],' points du segment ',es);end;
      carsi:=as;
      write('ces choix sont-ils confirmés O ou N ');readln(repc);end;end;

```

On peut encore considérer une autre condition: au-delà du segment: es, il n'y a, dans l'ensemble de base: I, que: (cari-fins[es]) éléments. Si l'on veut que les sous-ensembles créés aient effectivement pour cardinal le nombre: carsi-demandé, il faut que: (cari-fins[es]) ne soit pas inférieur au nombre: carsi-(as+kis[es]), des éléments qui seraient à prendre au-delà du segment: es. De ce point de vue, il se peut qu'on doive borner inférieurement les nombres: kis[es]. Par exemple si l'ensemble I est partagé en trois segments consécutifs de longueur 50, avec: kis[1]=kis[2]=5, on ne pourra dépasser: carsi=60, même en donnant à: kis[3] la valeur maxima permise: 50. Dans le listage publié ici, la borne inférieure de kis[es], n'est considérée que dans une instruction fictive, mise entre accolades, comme un commentaire. Mais alors, il faut, après avoir choisi les: kis[is], donner pour valeur à: carsi, le nombre: as, des éléments pris effectivement dans les cars tranches de I.

```

for es:=1 to cars do begin
  crds:=fins[es]-fins[es-1];eps[es]:=0;
  if ((crds div 2)<kis[es]) then eps[es]:=1;
  qots:=32700 div crds;mxs[es]:=qots*crds;end;

```

Les éléments sont tirés au hasard suivant la fonction: 'random', propre au MacIntosh. Celle-ci a pour valeur un nombre entier, pseudo-aléatoire, compris entre: -32768, et :32767. Or, pour tirer un élément d'une tranche de longueur crds, il faut une source aléatoire d'entiers uniformément distribués de 1 à crds. Il revient au même de tirer des entiers: zzz, uniformément distribués de 1 à un multiple de crds; puis (cf. *infra*) d'ajouter 1 au reste de zzz modulo crds. De façon précise, on tire d'abord un entier: zzz, par 'random'. Si: zzz≤0, on considère le nombre positif: zzz=1-zzz. Si: (mxs[es]<zzz), (mxs[es] étant calculé ci-dessus,) on renonce à utiliser: zzz; sinon il s'agit bien d'un nombre uniformément distribué de 1 à mxs[es]. Le rôle de eps apparaîtra au §3.3.

Une autre particularité de 'random' est qu'à moins d'une initialisation particulière, elle fournit, à son premier appel, le nombre 1. Pour éviter cette valeur systématique, il suffit de ne pas utiliser le premier cumul spécifié par le listage  $\Sigma$ hasxA; ou même de n'utiliser que le 2-nd listage:  $\Sigma$ hasxB, produit par 'cumhas'.



### 3.3 Création d'un listage de commande par 'cumhas'

```

repc:='N';
while not(repc='O') do begin
  write('le nombre de sous-ensembles à définir est ');
  readln(carj); if (200<carj) then carj:=200;
  writeln(carj:10);
  write('ce nombre est-il confirmé O ou N ');readln(repc) end;
rewrite(ft,nomf);
writeln(ft,'définition de sous-ens SE; carEns = ',cari,', carSE = ',carsi);
if (1<cars) then begin
  writeln(ft,'l'ensemble Ens est divisé en:',cars:3,' segments');
  writeln(ft,'Spécification des segments S de Ens');
  writeln(ft,'rang de S      début      fin pts pris dans S');
  for es:=1 to cars do
    writeln(ft,es:9,fins[es-1]+1:9,fins[es]:9,kis[es]:9);
  writeln(ft);end;

```

On fixe d'abord, par un dialogue, le nombre: carj, des sous-ensembles à spécifier sur le listage de commande.

Le listage est alors ouvert. On y écrit une ligne de titre donnant le cardinal: cari, de l'ensemble de base; et celui: carsi, des sous-ensembles qu'on y définit.

Si l'ensemble de base est segmenté, on écrit, sur le listage, un tableau de spécification des segments. (cf. *supra*, §3.0).

```

for j:=1 to carj do begin
  for i:=1 to cari do prisi^[i]:=0;
  for es:=1 to cars do begin
    crds:=fins[es]-fins[es-1];mxis:=mxs[es];pris:=0;meps:=1-eps[es];
    for i:=fins[es-1]+1 to fins[es] do prisi^[i]:=eps[es];
    kies:=kis[es];if (meps=0) then kies:=crds-kies;
    while not (pris=kies) do begin
      ia:=random;if (ia<1) then ia:=1-ia;
      if (mxis<ia) then ia:=0 else ia:=fins[es-1]+1+(ia mod crds);
      if not((ia=0) or (prisi^[ia]=meps)) then begin
        prisi^[ia]:=meps;pris:=pris+1 end;end;end;

    writeln(ft,20001,j:5,' ligne ');toti:=0;writeln(20000+j);
    for i:=1 to cari do if (prisi^[i]=1) then begin toti:=toti+1;
      write(ft,i:4,' ');if (toti mod 10=0) then writeln(ft) end;
    if not(carsi mod 10=0) then writeln(ft) end;

```

Les commandes de cumul sont créées dans une boucle indiquée par: j. La fonction caractéristique du sous-ensemble est d'abord mise à zéro par l'instruction: for i:=1 to cari do prisi^[i]:=0; puis on choisit, successivement, le nombre de points à prendre dans chacun des segments de rang: es=1 ... cars.

Pour tirer, d'un segment S, dont le cardinal est: crds, un sous-ensemble de cardinal kies, on peut, effectuer une suite de tirages indépendants de nombres (uniformément distribués entre 1 et cards); en ne retenant que les nombres qui ne sont pas déjà sortis. Au début, les tirages inutiles sont rares; mais si, e.g., le nombre: pris, des éléments déjà tirés dépasse:  $(3/4).crds$ , il y a

trois chances sur quatre pour qu'un tirage soit inutile. C'est pourquoi, si kies dépasse ( $crds/2$ ), il est préférable de tirer non une partie du segment, mais le complémentaire de cette partie: en effet, le cardinal:  $crds-kies$ , du complémentaire, étant inférieur à ( $crds/2$ ), le nombre moyen des appels de 'random' pour tirer un élément est inférieur à 2.

De façon précise, selon que ( $crds-kies$ ) est  $<2$  ou  $>2$ , on a:  $esps[es]=0$ , ou:  $eps[es]=1$ . Le tirage du complémentaire s'effectue quand:  $eps[es]=1$ ; le segment considéré, de rang  $es$ , est alors mis à la valeur:  $eps[es]=1$ ; et le tirage par 'random' sert à introduire des valeurs:  $meps=0$ .

Dans le programme tel qu'il est ici publié, est redondante l'instruction initiale: `for i:=1 to cari do prisi^[i]:=0`; en effet, chaque segment est ensuite initialisé séparément. Mais l'instruction initiale est laissée, d'une part, pour plus de clarté; d'autre part, afin que le programme reste correct, si, ultérieurement, on décidait d'admettre des segmentations ne couvrant pas l'ensemble de base tout entier (i.e.: `fin[cars]<cari`).

Une autre méthode, pour borner le nombre des appels de 'random', serait de cadrer le tirage aléatoire en fonction du nombre des éléments restés inoccupés dans le segment considéré. Mais cette méthode nous paraît demander, en dehors des appels de 'random', une gestion complexe.

Une fois le tirage achevé, on écrit, sur le listage, une commande de cumul, dont le format apparaît clairement sur l'exemple du §3.0.

```
dispose(prisi);
close(ft);
```

```
write('faut-il définir un autre schéma de sous-ensembles O ou N ');
readln(repc);if not(repc='N') then repc:='O';end;
```

end.

Après spécification des carj sous-ensembles demandés, la boucle générale: `while (repc='O') do begin...`, de création d'un listage de commande s'achève (; cf. supra, §3.1). On libère l'espace réservé en mémoire centrale pour le tableau:  $prisi^{\wedge}$ , de description de sous-ensembles; on ferme le fichier de texte:  $ft$ , ouvert pour créer le listage. L'utilisateur peut demander à rentrer dans la boucle générale afin de créer un autre listage de commande, d'après un nouveau schéma de sous-ensembles.

**Références bibliographiques**

[CUM. LI. COL.]; J.-P. & F. BENZÉCRI : “Extension d'un tableau par adjonction de lignes et colonnes créées par cumul”; in *CAD*, Vol.XV, n°1; pp. 109-116; (1990);

[MacSAIF]; J.-P. & F. BENZÉCRI : “Liste des programmes du logiciel MacSAIF, Système d'Analyse des InFormations, et description de leurs fonctions”; in *CAD*, Vol.XV, n°3; pp. 359-366; (1990).

[TEXTES GRECS]; J.-P. BENZÉCRI : “Typologie de textes grecs d'après les occurrences de formes de mots outil”; in *CAD*, Vol.XVI, n°1; pp. 61-86; (1991);

[ÉTAT MacSAIF]; J.-P. & F. BENZÉCRI : “État des modifications apportées aux versions du logiciel MacSAIF”; in *CAD*, Vol.XVIII, n°3; pp. 321-344; (1993).