

JC (suite)

Animaux : Pas d'animaux à la maison. Je préfère voir les animaux dans la nature, les oiseaux en particulier (et d'ordinaire je me tiens loin des ours blancs).

Sports/Passe-temps : Ski de fond, randonnée, canoë, pêche, mais par-dessus tout, la musique !

Votre fin de semaine idéale : Par une journée de pluie, un bon livre en compagnie de Handel, Grieg, Puccini (et rien d'ennuyant comme le ménage); par une belle journée ensoleillée, la présence d'amis et un copieux repas.

Mets préférés : Dry martini, charcuterie (prosciutto, saucisses sèches) et fromage (pâte raffinée, bleu, chèvre), avec pâté français et vin rouge (et un bon porto). En bref, tout ce qui est bon pour le cholestérol ! Quelqu'un est prêt à m'inviter ? ;o)

Si je pouvais être quelqu'un d'autre (qu'un programmeur SAS), j'aimerais être... musicien, compositeur, chansonnier.

Lorsque je ne programme pas en SAS, j'aime ... enseigner SAS.

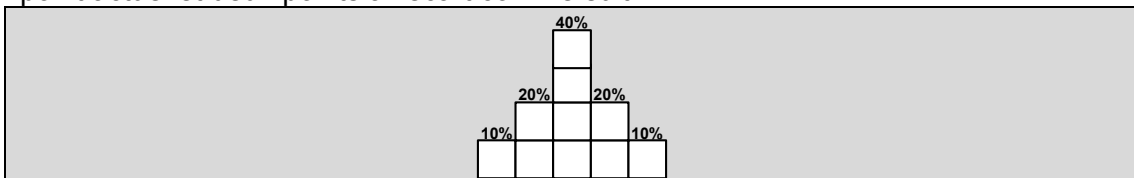
Une chose que tout programmeur SAS devrait savoir :

J'ai choisi trois trucs :

1. Fonction LAG *versus* NEXT
2. Fusion plusieurs-à-plusieurs
3. Tri de tableaux

1. Fonction LAG *versus* NEXT

La situation : Un client saisit un signal d'un capteur et désire l'atténuer pour faciliter l'analyse. Il insiste pour utiliser la méthode de moyenne pondérée à cinq points : le point actuel et deux points à l'écart comme suit :



La solution : Il n'y a pas de fonction comme LAG pour aller de l'avant. Par contre, nous avons la possibilité de le faire :

```
/******  
/** Simulate a signal (here a sine curve) with black noise (the Normal function)**/  
/** and plot */  
/******  
data raw ;  
  do x = 0 to 50 by .1 ;  
    y = sin(x) + normal(0)/10 ;  
    output ;  
  end ;  
run ;
```

```

proc gplot ;
  symbol1 i=j ;
  plot Y * x=1 ;
run ;
quit ;
/*****
** Smooth the signal **
** Lag2(y) is the value of Y 2 observations ago **
** Lag1(y) is the value of Y last observation **
** y is the value at the current observation **
** next1 is the value of Y next observation **
** next2 is the value of Y 2 observations from now **
** and plot smoothed curve **
*****/
data raw ;
  keep x y yy ;
  set raw nobs=nb obs ;
  if _n_ = 1 then
    link GETNEXT ; /* Initialize FIFO NEXT Buffer */
  link GETNEXT ; /* Put 2 obs from now in the buffer */
  yy = .1*lag2(y) + .2*lag1(y) + .4*y + .2*next1 + .1*next2 ; /* Smoothed Point */
  retain next: ;
return ;
GETNEXT:
  next1 = next2 ;
  n 2 = n + 2 ;
  if n 2 > nb obs then
    next2 = . ;
  else
    set raw(keep=y rename=(y=next2)) point=_n_2 ; /* Read 2 observations from now */
return ;
run ;
proc gplot ;
  symbol1 i=j ;
  plot YY * x=1 ;
run ;
quit ;

```

2. Fusion plusieurs-à-plusieurs

La situation : La fusion SAS prend habituellement moins de ressources qu'une jonction SQL. Toutefois, dans une situation plusieurs-à-plusieurs, SAS ne génère pas le produit cartésien prévu comme celui qui est généré par une jonction SQL.

La solution : Utilisez SET option KEY=. Pour l'opération, nous balayons la Table A (qui n'a pas à être triée) et jumelons les observations avec la Table B (la table qui doit être indexée). Mais un problème se présente : lorsque nous extrayons une série d'observations de la Table B, nous ne pouvons réinitialiser le pointeur au début de la Table B à moins de faire une recherche dans l'index avec une nouvelle valeur clé (l'option UNIQUE de SET KEY= est inutile, puisque chaque SET d'une étape DATA a son propre pointeur).

Voyez ci-dessous les statistiques de ressources résultant de l'exécution :

```
/******  
/** Generate 2 tables to be match-merged (Many-to-Many). **/  
/******  
data a ;  
  input (a b) ($) ;  
datalines ;  
A B  
D E  
B A  
B B  
B C  
C D  
E A  
E B  
run ;  
data b(index=(a)) ;  
  input (a c) ($) ;  
datalines ;  
X B  
B X  
B Y  
B Z  
Y D  
Z E  
E V  
E T  
run ;  
/******  
/** Join the tables with SQL first (check execution stats) **/  
/******  
options fullstimer ;  
proc sql ;  
  create table c1 as  
  select a.a, a.b, b.c  
  from a, b  
  where a.a = b.a  
  ;  
quit ;  
/******  
/** Join the tables with a Data Step (check execution stats) **/  
/******  
data c2 ;  
  keep a b c ;  
  set a ;  
  by notsorted a ;  
  do i = first.a + 1 to 2 ;  
    if i = 1 then  
      do ;  
        save = a ;  
        a = '00'X ;  
      end ;  
      do until(_iorc_) ;  
        set b key=a ;  
        if ^ iorc then /* Found */  
          output ;  
        else  
          _error_ = 0 ;  
        end ;  
        if i = 1 then  
          a = save ;  
      end ;  
run ;  
options nofullstimer ;  
/******  
/** Check resulting table is the same **/  
/******  
proc compare data=c1 compare=c2 ;  
run ;
```

3. Tri de tableaux

La situation : Il est nécessaire – quelquefois – de trier les éléments d'un tableau. La fonction SCL ASORT n'est pas disponible dans SAS Basic.

La solution : Créez votre propre procédure ASORT. En voici un exemple :

```

/*****
** Initialize 2 arrays with random numbers
**
*****/
data temp ;
  drop i j ;
  array a{500} ;
  array b{250} ;
  do i = 1 to 10 ;                               /* 10 observations */
    do j = 1 to 250 ;
      a[j] = ceil(ranuni(0)*10000) ;
      b[j] = ceil(ranuni(0)*10000) ;
    end ;
    do j = 251 to 500 ;
      a[j] = ceil(ranuni(0)*10000) ;
    end ;
  output ;
end ;
run ;

/*****
** Create the Macro ASORT (Simple Bubble Sort Algorithm)
** ARRAY parameter is the array to sort
** If ORDER parameter is entered, the array will be sorted Descending only if
** the parameter starts with the letter D
*****/
%macro Asort(Array,Order) ;
  %if %upcase(%substr(%str(&Order ),1,1)) = %str(D) %then
    %let Order = %str(<) ;                               /* Descending Order */
  %else
    %let Order = %str(>) ;                               /* Ascending Order */
  do __i__ = 2 to dim(&array) ;
    do j = dim(&array) to i by -1 ;
      if &array{ j -1} &order &array{ j } then
        do ;                                           /* Exchange Contents of the 2 elements */
          x = &array{ j -1} ;
          &array{__j__-1} = &array{__j__} ;
          &array{__j__} = __x__ ;
        end ;
      end ;
    end ;
  drop : ;
%mend Asort ;

/*****
** Sort Arrays
**
*****/
data temp ;
  set temp ;
  array a{*} a ;
  array b{*} b ;
  %Asort(a) ; /* Sort array a in ascending order */
  %Asort(b,d) ; /* Sort array b in descending order */
run ;

```