

## JC Con't

**Pets:** No pets at home. I rather love watching animals in nature, especially birds (I usually stay away from white bears).

**Sports/hobbies:** Cross-country skiing, hiking, canoeing, fishing. But over all of this: music!

**Your ideal weekend would be:** One rainy day with a good book and Handel, Grieg, Puccini (and no boring stuff like housekeeping); one sunny day with good friends, ending in a copious dinner.

**Favorite foods:** One dry martini, charcuterie (prosciutto, dry sausages) with cheese (refined paste, blue, goat), paté French sticks and red wine (and good porto). In short, anything good for cholesterol! Can I hope for an invitation? ;o)

**If I could be anything at all (besides SAS programmer), I would be...** A musician, a composer, a chansonnier.

**When I'm not programming in SAS, I like to...** Teach SAS.

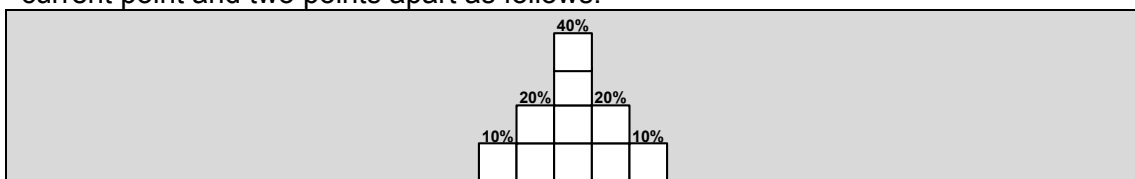
### Things every SAS Programmers should know:

I chose three tips:

1. LAG function versus NEXT.
2. Many-to-many merge.
3. Sorting arrays.

#### 1. LAG function versus NEXT.

**The situation:** A customer catches a signal from a sensor and wishes to smooth it to ease analysis. He insists on using the five points weighted average method: the current point and two points apart as follows:



**The solution:** There is no function, similar to LAG looking forward. However, we have the capability to do it:

```
/******  
/** 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. Many-to-many merge.

**The situation:** SAS merge usually takes fewer resources than an SQL join. However, in a many-to-many situation, SAS does not produce the expected Cartesian product as per generated with an SQL join.

**The solution:** Use the SET option KEY=. For the operation, we scan a Table A (which does not have to be sorted) and match observations with the Table B (the table that must be indexed). However, a problem occurs: Once we extract a series of observations from the Table B, we cannot reset the pointer at the beginning of Table B unless we search in the index with a new key value (The UNIQUE option of the SET KEY= is useless since each SET within a data step has its own pointer).

Check hereafter the resource statistics resulting from the execution:

```
/******  
/** Generate 2 tables to be match-merged (Many-to-Many). **/  
/******  
data a ;  
  input (a b) ($) ;  
datalines ;  
A B  
D E  
B A  
B B  
B C  
C D  
E A  
E B  
run ;  
data b(index=(a)) ;          /* Index b to allow SET KEY= */  
  input (a c) ($) ;  
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 ;          /* For resetting pointer purposes */  
        save = a ;  
        a = '00'X ;  
      end ;  
      do until(_iorc_) ;  
        set b key=a ;  
        if ^ iorc then /* Found */  
          output ;  
        else  
          _error_ = 0 ;          /* Avoid dumping the PDV in the Log */  
        end ;  
      if i = 1 then  
        a = save ;          /* Prepare KEY for the second pass */  
    end ;  
  run ;  
options nofullstimer ;  
/******  
/** Check resulting table is the same **/  
/******  
proc compare data=c1 compare=c2 ;  
run ;
```

### 3. Sorting Arrays.

**The situation:** It is required – sometimes – to sort the elements of an array. SCL ASORT function is not available in SAS Basic.

**The solution:** Create your own ASORT procedure. Here is an example:

```

/*****
** 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 ;

```