SAS

STATISTICAL ANALYSIS SYSTEM

Uso di SAS per le analisi statistiche

A cura di

Laura Neri

Dip. di Economia Politica

Università degli Studi di Siena

Taken from
http://www.ats.ucla.edu/stat/SAS/library/SASTranMan_os.html

DO Blocks and DO Loops

A DO block begins with the reserved word DO and ends with the reserved word END. The statements enclosed inside DO..END are called a block. The DO..END construct is an important device to group statements inside a DATA step.

```
data two; set prova;
  if age ne . then do;
      age2 = age*age;
      age3 = age2*age;
                  end;
run;
```

If the condition in the IF statement is true, SAS executes the statements in the DO..END block. Otherwise the statements are ignored. Without the DO..END block, the DATA statement would require three IF statements. The DO statement also is part of looping constructs (iterative DO). These can be written in various ways. Here are examples:

```
do i = 1 to 4;  /* A index loop, runs from 1 to 4 in increments of 1 */
  < SAS statements>
end;

do i = 1 to 10 by 2; /* index loop, runs from 1 to 10 in increments of 2 */
  <SAS statements>
end;

/* index loop over x=10, 20, 30, 50, 55, 60, 65, ... , 100 */
do x = 10, 20, 30, 50 to 100 by 5;
  <SAS statements>
end;
```

```
do month = 'FEB', 'MAR', 'APR';
 <SAS statements>
end;

/* the statements inside the loop are executed only while the */
/* expression in parentheses is true                          */
do k =1 to 12 while(month='APR');
  <SAS statements>
end;
```

The next example generates 100 observations from a Gaussian distribution with mean 12 and variance 3. For each observation, it calculates its right-tail probability:

```
data Gauss;
  do i = 1 to 100;
     z = rannor(8923);
     p = 1 - Probnorm(z);
     x = z*Sqrt(3) + 12;
     output;
  end;
run;
```

Notice that the DO I=1 to 100; .. END; construct is executed for each observation in the data set. Since no observations are input or transferred from another SAS data set, you need the OUTPUT statement inside the DO loop to instruct SAS to write to the data set when the loop is completed.
The OUTPUT statement should be the last statement inside the loop.

Other forms of DO loops are the DO..WHILE() and DO..UNTIL() constructs. A logical expression inside the parentheses is evaluated for each repetition of the loop.

The loop continues as long as the statement is true for the DO..WHILE() loop or until the statement becomes true (DO..UNTIL()).

DO..WHILE and DO..UNTIL loops are dangerous. For example, if the logical statement in the WHILE() expression is not true, the loop will never executes. If it is not true, there must be a mechanism inside the loop that eventually makes the statement false, otherwise the loop will continue infinitely. It is important to remember that the loop is executed for each observation in the data set. Care must be exercised not to write infinite loops with DO..WHILE. For example, the following loop

```
n=0;
do while(n <= 5);
  <Statements>
  n+1;
end;
```

works, since n is changed inside the loop and the WHILE() condition eventually will become false. If you forget to increment n inside the loop, the statements will be processed indefinitely. In this case, a DO n = .. END; loop is much safer:

```
do n=1 to 5;
  <statements>
end;
```

IF .. THEN .. ELSE statements

Also known as conditional statements, these are very important when subsetting data or processing observations conditionally. The ELSE part is not necessary.

```
data age_nomiss; set prova;
  if age eq . then delete;

run;
```

In the form

IF <condition> THEN <statement1> ELSE <statement2>

SAS evaluates for each observation the logical condition. If the condition is true, it executes statement1, if it is false statement2. Important to note is that only a single statement follows the THEN and ELSE clause. For example,

```
data two; set one;
  if (x < 0) then
    y = .;
    z = sqrt(-x);
  else z = sqrt(x);
run;
```

will cause an error, since SAS expects the ELSE clause after the y=.; statement. If more than one statement is to be executed in the THEN or ELSE clause, group them into DO blocks:

```
data two; set one;
  if (x < 0) then do;
  y = .;
  z = sqrt(-x);
```

```
end; else z = sqrt(x);
run;
```

IF .. THEN .. ELSE statements can be nested:

```
data three; set prova;
  if age <= 25 then agegr = 1;
     else if age <= 40 then agegr = 2;
          else agegr = 3;
run;
```

GENERAZIONE DI NUMERI CASUALI

· Un **numero casuale** è un numero scelto da un insieme di valori egualmente probabili, cioè un numero estratto da una distribuzione uniforme;
· in una **sequenza** di numeri casuali ogni numero estratto deve essere statisticamente indipendente dagli altri.

I numeri casuali sono utili in vari casi:
· Generazione di dati cifrati e password
· Simulazione e modellazione di fenomeni complessi
· Selezione di campioni casuali

COME GENERARE NUMERI CASUALI?

È possibile generare numeri casuali con un computer?

· Sì, ci sono 2 approcci:
o Generatori di numeri *pseudo-casuali*
o Generatori di numeri casuali veri (da fenomeni fisici, p.e. gli istanti temporali in cui accade qualcosa di totalmente imprevedibile).

GENERAZIONE DI NUMERI PSEUDO-CASUALI

Gli algoritmi per la generazione di numeri pseudo-casuali utilizzano formule matematiche o tabelle pre-calcolate per produrre sequenze di numeri che *sembrano casuali*.

Gli algoritmi oggi disponibili sono buoni e i numeri generati sono come quelli realmente casuali.

GENERAZIONE DI NUMERI PSEUDO-CASUALI IN SAS

Le funzioni e le routine di SAS per la generazione di numeri casuali producono sequenze di numeri partendo da un valore iniziale *seed.*
Il *seed* deve essere un intero non negativo minore di 2^{31}-1.
È sempre possibile riottenere la successione di numeri casuali utilizzando lo stesso DATA step.
Se si usa il valore zero come *seed* è l'orologio di sistema che inizializza la sequenza, in tal caso la sequenza di numeri casuali non è replicabile.
Il *seed* può essere una costante intera o una variabile che contiene la costante intera. La variabile *seed* deve essere inizializzata prima della prima esecuzione della funzione o della CALL routine.

FUNZIONI SAS PER LA GENERAZIONE DI NUMERI PSEUDO-CASUALI

NORMAL normale standard
RANNOR normale standard
RANBIN binomiale
RANCAU Cauchy
RANEXP esponenziale standard

RANGAM gamma standard
RANPOI Poisson
RANTBL distribuzione discreta
RANTRI distribuzione triangolare
RANUNI uniforme (0,1)
UNIFORM uniforme (0,1)

Consideriamo la Distribuzione Uniforme in (0,1).
RANUNI FUNCTION
RANUNI(seed)

RANUNI ROUTINE
CALL RANUNI(seed,x);

The RANUNI routine gives greater control of the seed and random number streams than does the RANUNI function

```
data prova;
 Seed_1 =45; Seed_2= 45; Seed_3= 45;

 /*Seed_1 =44; Seed_2= 4; Seed_3= 4;*/

   do i=1 to 10;
      call ranuni(Seed_1,X1);
      call ranuni(Seed_2,X2);
      X3=ranuni(Seed_3);
      output;
   end;
run;
```

```
proc print;
    id i;
    var Seed_1-Seed_3 X1-X3;
run;

*****************************************************;
*** generazione di numeri casuali da uniforme (a,b) ***;
***      generatore moltiplicativo modulo primo     ***;
***   (Fishman e Moore, 1982, JASA, 77, 129-136)    ***;
***    modulo=2**31-1, moltiplicatore=397204094     ***;
*****************************************************;
*genera1b.sas;
*assegna un valore ai parametri della
distribuzione;
%let a=10;
%let b=20;

title "Generazione di numeri casuali da
uniforme (a=&a,b=&b)";
data dati;
a=&a; b=&b;
eu=(a+b)/2 ;
varu=((b-a)**2)/12;
label eu='valore atteso di u' varu='varianza
di u';
do i=1 to 10000;
    x=a+(b-a)*ranuni(0);
    output;
end;
run;
proc print data=dati(obs=10);
id i;
var x eu varu;
run;

proc means n mean var min max maxdec=2;
```

```
var x;
run;

title2 "distribuzione percentuale";
proc gchart;
vbar x/type=percent;
run;
```

USO DI FUNZIONI SAS PER LA GENERAZIONE DI CAMPIONI CASUALI

```
Sample1.sas
*legge tutti i dati e calcola media di voto;
data dati;
infile
'G:\documenti\didattica\CorsoSAS\2010-
11\dati\voto.txt';
input voto;
run;
proc means;
run;


title 'seleziona etichette osservazioni
campionate';
data num;
a=1;
b=897;          *num elementi popolazione;
n=100;           *numerosita campionaria;
seme=1;
do i=1 to n;
   call ranuni(seme,x);
```

```
    num=int(a+(b-a)*x);
    output;
 end;
run;

proc sort; by num;run;
proc print data=num(obs=10);
var num;
run;
/*il campione dei voti non e' selezionato,
conosco solo il numero
dell'oss da selezionare, quindi per
selezionare il campione di voti...*/

……esercizio da fare ricordando che

num=_n_;*attribuisce un identificativo
ordinato alle osservazioni;
```

```
title' seleziona direttamente il campione';
data sample;
infile
'G:\documenti\didattica\CorsoSAS\2010-
11\dati\voto.txt';
input voto;

a=1;
b=897;          *num elementi popolazione;
label=_n_;      *n. ordine (etichetta) lista
popolazione;
n=100;           *numerosita campionaria;
seme=1;

do i=1 to n;
   call ranuni(seme,x);
   num=int(a+(b-a)*x);
   if label eq int(num) then output;
end;
run;

proc print data=sample(obs=10);
var label voto num ;
run;
```

Macro Variables

Taken from a seminar

http://www.ats.ucla.edu/stat/SAS/seminars/sas_macros_introduction/default.htm

The SAS macro language is a very versatile and useful tool. It is often used to reduce the amount of regular SAS code and it facilitates passing information from one procedure to another procedure. Furthermore, we can use it to write SAS programs that are "dynamic" and flexible. Generally, we can consider macro language to be composed of macro variables and macro programs. In this session we will demonstrate how to create macro variables and how to write basic macro programs.

Macro Variables

A macro variable in SAS is a string variable that allows you to dynamically modify the text in a SAS program through symbolic substitution. The following example demonstrates how to create and use a macro variable. First we set up some system options to have a more concise output style.

```
options nodate nonumber nocenter formdlim="-";
data hsb2;
  input  id female race ses prog
         read write math scinece socst;
datalines;
 70 0 4 1 1 57 52 41 47 57
121 1 4 2 3 68 59 53 63 61
 86 0 4 3 1 44 33 54 58 31
141 0 4 3 3 63 44 47 53 56
172 0 4 2 2 47 52 57 53 61
113 1 4 2 2 44 52 51 63 61
 50 0 3 2 1 50 59 42 53 61
 11 0 1 2 2 34 46 45 39 36
 84 0 4 2 1 63 57 54 51 63
 48 1 3 2 2 57 55 52 50 51
 75 1 4 2 3 60 46 51 53 61
```

```
 60  1  4  2  2  57  65  51  63  61
 95  0  4  3  2  73  60  71  61  71
104  0  4  3  2  54  63  57  55  46
 38  0  3  1  2  45  57  50  31  56
115  0  4  1  1  42  49  43  50  56
 76  0  4  3  2  47  52  51  50  56
195  0  4  2  1  57  57  60  56  52
;
run;
```

Suppose that we want to look at the means of some variables and then do a regression analysis on the same variables.

```
proc means data = hsb2;
  var write math female socst;
run;

proc reg data = hsb2;
  model read = write math female socst;
run;

quit;
```

We can simplify the program by creating a macro variable containing all the names of the independent variables. A macro variable can be created by using the **%let** statement. All the key words in statements that are related to macro variables or macro programs are preceded by percent sign %; and when we reference a macro variable it is preceded by an ampersand sign &. When we submit our program, SAS will process the macro variables first, substituting them with the text string they were defined to be and then process the program as a standard SAS program.

```
%let indvars = write math female socst;
proc means data = hsb2;
  var &indvars;
run;

proc reg data = hsb2;
  model read = &indvars;
run;
quit;
```

We can display macro variable value as text in the log window by using **%put** statement.

```
%put my first macro variable indvars is &indvars;
```