Paper P-408

Producing a Table of Seeds for Random Number Generation

F. Joseph Kelley
University of Georgia, Athens, GA

Abstract.
The need for producing a random sample of some population of interest is common and is a frequent topic of discussion in various disciplines and in the fora related to those disciplines. When using SAS® Software, one of the most straightforward (albeit inefficient) methods of producing such a sample is assigning a (uniform) random number to each observation in the data file of interest; sorting by that random number; then selecting the first ***n*** records (observations) desired. If drawing several samples from the original population, this process could be repeated as many times as might be wished; however a SAS Macro could easily be written to accomplish the same.

If researchers wished to assign different random number generation seeds to each sample, a small complication is introduced. This might be resolved by using a single seed to set up a table of seeds, but that introduces a potential trap. This paper will present a method of setting up table of seeds for random number generation and will present a small example that attempts to minimize the sorting and space requirements of the basic process.

Introduction.
A researcher has a data file of ***N*** observations. For the purposes of a study, it is necessary to generate ***k*** samples of ***n*** observations each. Both ***k*** and ***n*** may vary, but this will be at the discretion of the researcher. It is required that a separate random number seed be used in selecting each sample and this cannot be ≤ 0 (which causes the random number generator to use the system clock) as the results must be reproducible. The samples produced are independent of each other; an observation may appear in more than one sample. When working with more than one sample, it would be necessary to provide a seed to be used by the random number generator; this would require we devise a selection of seeds in advance. For 100 samples, we will need 100 seeds. It is of course necessary there be no duplicates in this selection, as they will generate identical streams of "random" numbers.

In the usual SAS program, we might code something like this:

```
data test ;
    infile fileref ;
    retain seed 999999937;
    input variable list ;
    ran = ranuni(seed) ;
    output ;
proc sort   data=test;
    by ran ;
data select ;
    set text ;
    if _N_ <= n then
      output;
```

This code assigns the random number ran (uniformly distributed on the interval (0,1)) and then sorts the data by that number. The final step simply takes the first ***n*** records.

If only one or only a few samples were desired, this would be fine (we would vary the value assigned to seed of course). When we need a number of samples, this becomes tedious and error-prone. SAS itself provides the tools needed for a more robust solution.

Solution 1.
More control over the seeds used is needed. If ***k*** samples are to be generated, then ***k*** seeds will be needed. Instead of the ranuni function, a SAS call function is used. Instead of RANUNI, we will use CALL RANUNI. The ranuni *function* as used in the program above, is replaced with

```
call ranuni(seed,ran);
```

instead. In this form, RANUNI is called almost as though it were a (Fortran) subroutine; a value of seed is passed and, on return, ran has been assigned a random value and seed has a new value for the next call (the old having been replaced). Using this, we can save the value of seed, and use it elsewhere. We could call ranuni a dozen times, save the values of seed to an array then use the array values as the seeds for our large samples. However, we must remember that if

seed1 → seed2 → seed3 → … → seed*n*

Then saving these and using "seed2" will produce the same "seed3", "seed4" and so on

that "seed1" did. So we can add a small "perturbation" to our seeds. Here is some code that does that:

```
data test ;
    retain seed 999999937 ;
    array seeds {*} seed1-seed3 ;
    do i = 1 to 3 ;
        put i= seed= ;
        call ranuni(seed,x);
        seeds{i} = seed + 64
        put _ALL_ ;
    end;
run;
```

The little seed assignment loop could be placed in a program, executed once and fill the SEEDS array with numbers for subsequent use. It might be placed in a macro, and we would almost certainly use macro variables to take care of the number of seeds we assigned (our ***k***) and the number of observations to go in each sample (our ***n***). Depending upon the values of ***N*** (the total number of records/observations in the file we wish to sample), ***n*** (the total to be in each sample) and ***k*** (the total number of samples to take), we will probably find that a substantial number of our records will not be in any sample and few will be in more than one. In some areas, ***N*** may be enormous (financial institutions, for example, may deal with millions of records). We certainly wouldn't want to create 10 copies of the same huge file, then sort it just to obtain a relative handful of records. So let's consider that our researcher may face some constraints.

Solution 2.

Rather than assign a random number to each observation, sort the data, then extract our observation, why not assign each observation all ***k*** random numbers at once? The problem is, of course, that we will use the results individually: how are we to identify the records we wish in any individual sample?

With a few modifications, the code used above will get us started, but more is needed. If, for example, we wish to draw five samples of 1000 from a file with, say, 20,000 observations, how would we determine whether any particular observation were desired at all? And if it were to be a part of one of the samples, which would it fall into? In the plan described, each observation would have been assigned 5 numbers from the uniform distribution (each of these numbers will have been generated by the seed for that sample). We cannot examine the random numbers themselves and determine whether the observation should be taken. However, the RANK procedure will allow us to order the random variables. RANK will assign an ordinal value to the variables named in the procedure call. If we have 2 random numbers in an observation:

Obs	Value	_rndm1	_rndm2
1	2045.2339	0.37313	0.73802

They might look like this after the RANK procedure:

Obs	Value	_rndm1	_rndm2
1	2045.2339	10	24

If we were using an ***n*** of 15, the observation above would be used in the 1st sample, but not in the 2nd. Easy to tell when you can see it. Proc RANK is an old procedure, and does have a drawback: identical values for a variable will either be the highest, the lowest or the average rank (this is the default). That is, if we had five numbers: 1, 2, 3, 4, 4, the default ranks would be: 1, 2, 3, 4.5, 4.5. (This last is the average of 4 and 5). We don't expect our random numbers to repeat, but it is an inherent failing of random number generators, though it is unlikely we would encounter it even in very large sampling. Additionally, although the numbers might be repeated, as long as the data values themselves are unique, this is of no consequence, though it might produce a sample slightly larger than desired.

The Macro and Example.

What follows is a macro and small example, together with selected output. A short note on style: the author attempts to make any (data step) variables or files created in his macros look generally unlike any he might create or encounter in his usual programming. So, for example, loop counters tend to be "_i_" and not "i" when inside a macro, and dataset names tend to begin (and end) with "_".

```
%let strt_seed  = 999999937 ;
```

Assign a SEED; in this case, a prime number is specified.

```
options mprint ;

%macro Assign_Rndm(dsin=,dsout=,
select=,num_reps=);
```

The SAS macro, ASSIGN_RNDM has four parameters:

- DSIN and DSOUT name the input and output files.
- SELECT is the number to be in any sample (the ***n***).
- NUM_REPS are the number of samples to create (the ***k***).

```
   data _NULL_ ;
     retain seed &strt_seed;
```

```
 do i = 1 to &num_reps ;
   call ranuni(seed,x);
   seeds = seed + 64 ;
   call symput('seed'||
      trim(left(put(i,3.))),
      seeds) ;
 end;
```

This is a macro version of the usual DATA step shown previously. Instead of writing to an array, the values are saved as macro variables.

```
  run;

  data _temp_ ;
    set &dsin ;
    retain

%do i = 1 %to &num_reps ;
    seed&i  &&seed&i
%end ;
%str(;);
```

The RETAIN statement is used to assign the seeds from the macro variables to array values.

```
  array _seeds {*}
    seed1-seed&num_reps ;
  array _rndm {&num_reps};
   do _j_ = 1 to &num_reps;
    call ranuni(_seeds{_j_},
      _rndm{_j_});
  end ;
  run;
```

At this point, each observation has been assigned "*NUM_REPS*" random numbers. Now we must determine whether the observations are wanted. This intermediate output is saved in a temporary file: *_TEMP_*

```
  proc rank  data=_temp_
     out=_temp2_ ;
  var _rndm1-_rndm&num_reps;
```

The random numbers (*_RNDM1-_RNDUM&NUM_REPS*) are now ranked from lowest to highest. Though not strictly necessary, a second intermediate file *_TEMP2_* is used for the output. This could be omitted. The random numbers in *_RNDMn* have been replaced with their ranks.

```
data &dsout ;
  set _temp2_ ;
  array _rndm {&num_reps};
  drop _rndm1-_rndm&num_reps
   _i_ _j_
   seed1-seed&num_reps ;
  do _i_=1 to dim(_rndm);
   if _rndm{_i_} <= &select
     then do;
        _k_ = _i_ ;
        _o_ = _rndm{_i_} ;
        output ;
      end ;
  end ;
```

Now, much as when we looked at the data above, the program scans the values of *_RNDMn* on each observation. If one of these is found to be less than or equal to *SELECT*, the observation is written to *DSOUT*. Most of the extra variables added by the macro are removed, however two remain:

- *_K_* which identifies the sample.
- *_O_* which identifies the order within the sample (the rank order).

These will be used by the SORT which follows.

```
  run ;
  proc sort data=&dsout ;
    by _k_ _o_ ;
```

We now have a data file containing all the samples. It may be broken into smaller files or processed with BY-group processing.

```
  proc datasets library=work;
    delete _temp_ _temp2_ ;
  run;
```

This step simply removes the temporary Library files created.

```
%mend Assign_Rndm ;
```

Here is a sample program using the macro:

```
data test ;
    infile cards;
    input id $ region $ age score ;
cards;
a23 1   19  23.8
a22 1   20  22.1
b02 2   22  26.3
d45 3   19  24.4
b22 2   18  28.2
c90 4   20  22.1
s33 2   21  23.7
a20 3   22  25.1
d14 3   24  25.2
c12 4   21  21.8
d29 4   22  22.6
s11 3   21  26.1
e09 3   22  23.2
a65 2   25  21.4
a28 1   19  22.3
c16 2   20  24.1
d11 2   20  21.8
g14 1   21  25.3
b40 4   21  24.9
b41 4   20  23.5
c10 3   22  26.2
h48 1   19  20.9
a39 2   21  24.6
```

```
b24 1   21  23.8
d19 4   22  21.6
c47 4   20  25.1
p47 3   19  24.1
a17 2   20  25.7
b52 1   21  24.8
d33 1   20  23.6
;
%Assign_Rndm(dsin=test,select=8,
   dsout=test2,num_reps=5)

run;

proc print  data=test2 ;
run;
```

Here is some selected output:

The SAS System

Obs	id	region	age	score	_k_	_o_
1	g14	1	21	25.3	1	1
2	s33	2	21	23.7	1	2
3	a20	3	22	25.1	1	3
4	d33	1	20	23.6	1	4
5	e09	3	22	23.2	1	5
6	d19	4	22	21.6	1	6
7	c12	4	21	21.8	1	7
8	a39	2	21	24.6	1	8
9	h48	1	19	20.9	2	1
10	s11	3	21	26.1	2	2
11	a22	1	20	22.1	2	3
12	b24	1	21	23.8	2	4
13	a28	1	19	22.3	2	5
14	c47	4	20	25.1	2	6
15	d14	3	24	25.2	2	7
16	a17	2	20	25.7	2	8

Conclusion.

Selecting a random sample is not (usually) a serious problem. Selecting multiple samples can be; however, a DATA step or a macro can relieve that problem as well. The macro shown above trades processor time for space. It also could result in slightly larger sample sizes because of the possibility of non-unique ranks caused by duplicate random numbers. Though the existence of these duplicates is unlikely, it is not impossible. The (possible) duplicate would not produce duplicate observations (unless the data itself had these, in which case, it is another problem altogether). At worst, it presents the possibility of a slightly larger sample size. Further checks could be made to verify the size and discard excess observations. However, it seems likely that if a researcher were selecting samples from a data file that was so large as to make duplicate ranks a possibility, anything that might reduce the space requirements would be necessary.

References.

Johnson, Robert E. and Hui Liu, "Psuedo-Random Numbers – Out of Uniform," in *Proceedings of the Twenty-Fifth Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute, Inc, pp. 1218-1220.

SAS Institute, Inc., (1990) *SAS® Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute, Inc.

SAS Institute, Inc., (1990) *SAS® Procedures Guide, Version 6, Third Edition*, Cary, NC: SAS Institute, Inc.

Trademark.

Author Contact Information.

F. Joseph Kelley
Research and Computational Science
University Computing & Networking Services
University of Georgia
Athens, GA 30602-1911

jkelley@uga.edu