BIOS 662 Fall 2018

Random Number Generation

David Couper, Ph.D.

david_couper@unc.edu

or

couper@bios.unc.edu

https://sakai.unc.edu/portal

Outline

- Uses of random numbers in statistics
- Generation of random numbers from the uniform distribution
- Generation of random numbers from the normal distribution
- Random numbers from other distributions
- Controlling the sequence in SAS

Random Numbers in Statistics

Many statistical applications rely on a stream of random numbers:

- Simulations
- Sampling
- Randomization
- Re-sampling methods (e.g. bootstrap)
- Multiple imputation
- Markov Chain Monte Carlo methods

Generation of Uniform Random Numbers

- Properties required of a sequence of independent and identically distributed (iid) uniform (0, 1) random numbers:
  - uniformity
  - independence
  - identically distributed
- Mechanical methods
- http://www.random.org/

  "RANDOM.ORG offers *true* random numbers to anyone on the Internet. The randomness comes from atmospheric noise, which for many purposes is better than the pseudo-random number algorithms typically used in computer programs."

Generation of Uniform Random Numbers

- Statistical software packages provide one or more *pseudo*-random number generators
- A sequence of pseudo-random numbers is:
  - deterministic
  - reproducible
  - finite
- uniform random numbers usually uniform on $(0, 1)$
- if $X \sim U(0, 1)$,

  let $Y = a + bX$,

  then $Y \sim U(a, a + b)$

Generation of Uniform Random Numbers

- Multiplicative congruential generator

$$x_i = ax_{i-1}(\text{mod } m),\ i = 1, 2, 3, \ldots$$

  - mod is the modulo (or modulus) operator, giving the remainder after dividing by m
  - x_0 is called the *seed*
  - the *cycle length* is the number of distinct values of x_i before the sequence starts to repeat itself
  - the sequence is of full period if the cycle contains all the numbers in $\{1, 2, 3, \ldots, (m-1)\}$
  - different values of x_0 correspond to starting at different points in the cycle

Uniform Random Number Generator in SAS

- `CALL RANUNI(seed,x);`
- `x = RANUNI(seed);`
- IML function: `x = UNIFORM(seed);`
  - `x` is the value returned by the function
  - seed is a non-negative integer $< 2^{31} - 1$
  - if seed $= 0$, time of day used as seed for the stream
  - in IML the seed can be a matrix
  - `CALL RANUNI` gives greater control of the seed and random number streams than `RANUNI`

Uniform Random Number Generator in SAS

- "x is generated from the uniform distribution on the interval (0,1), using a prime modulus multiplicative generator with modulus $2^{31} - 1$ and multiplier 397204094."
- Cycle length is $2^{31} - 2 = 2.147 \times 10^9$
- Program on next page ran for 16 minutes, 42 seconds, on a PC, September 2012

Uniform Random Number Generator in SAS

```
proc iml;
  n=1;
  seed=83763;
  x=ranuni(seed);
  print x;
  y=ranuni(seed);
  do until (x=y);
    n=n+1;
    y=ranuni(seed);
    end;
  print y;
  print n;
quit;
run;
```

```
    n

2.14748E9
```

Generation of $N(0,1)$ Random Numbers

- Box-Muller transformation: If u_1 and u_2 are independent random numbers from $U(0,1)$, set:

  $$z_1 = \sqrt{-2\ln(u_1)}\cos(2\pi u_2)$$

  $$z_2 = \sqrt{-2\ln(u_1)}\sin(2\pi u_2)$$

  then z_1 and z_2 are independent random numbers from $N(0,1)$

- If z is a random number from $N(0,1)$,

  let $x = \mu + \sigma z$,

  then x is a random number from $N(\mu, \sigma^2)$

$N(0,1)$ Random Number Generator in SAS

- `CALL RANNOR(seed,x);`
- `x = RANNOR(seed);`
- IML function: `x = NORMAL(seed);`
  - `x` is the value returned by the function
  - seed is a non-negative integer $< 2^{31} - 1$
  - if seed $= 0$, time of day used as seed for the stream
  - in IML the seed can be a matrix
  - SAS uses the Box-Muller transformation of `RANUNI` uniform variates

Random Number Generation in R

- Random numbers from the uniform distribution

  `runif(n, min=0, max=1)`

  - `n` is the number of observations to generate from $U($`min`,`max`$)$
  - Default generator is "Mersenne-Twister", the twisted generalized feedback shift register algorithm of Matsumoto and Nishimura (1998), with cycle length $2^{19937}-1 = 10^{500}$
  - `set.seed(k)`, where `k` is an integer; if the seed is not set, clock time is used to generate a seed

- Random numbers from the normal distribution

  `rnorm(n, mean = 0, sd = 1)`

Another Random Number Generator in SAS

- SAS has a random number generator that uses the "Mersenne-Twister" with cycle length $2^{19937} - 1 = 10^{500}$.
- Syntax: `RAND('dist', parm-1,...,parm-k)`

  `dist` indicates the distribution from which to generate the random number; there are around 20 distributions available to be called by the function.

  `parm-1,...,parm-k` are parameters that need to be specified for the distribution; the number of parameters depends on the distribution.
- For the Normal: `RAND('NORMAL', mean, std_dev)`
- For the standard Normal: `RAND('NORMAL', 0, 1)`

  or just `RAND('NORMAL')`

Random Numbers From Other Distributions

- Built-in functions for many generators in SAS, R, etc.
- Inverse transformation method

  If the cumulative distribution function $F(x)$ can be written in closed form, set $u = F(x)$ and solve for x. Then, generate u from $U(0,1)$ and calculate x
- Example: exponential distribution

$$f(x) = \lambda e^{-\lambda x} \quad \text{and} \quad F(x) = 1 - e^{-\lambda x}, \quad x \geq 0$$

  Set $u = F(x) = 1 - e^{-\lambda x}$.

  Solving for x: $x = -\frac{1}{\lambda} \ln(1-u)$.

  Because $u \sim U(0,1) \;\Rightarrow\; 1-u \sim U(0,1)$

  we can use $x = -\frac{1}{\lambda} \ln(u)$

Setting the Seed in SAS

- It is usually desirable to use a specified seed rather than `seed = 0` because
  - debugging a program is easier
  - results are repeatable (e.g. for an audit)
- SAS makes this difficult when random numbers are needed in multiple data steps, such as a simulation repeating a data step multiple times

```
%let seed0=97231;
%let sampsize=4;
%let nreps=2;

data begindat;
  do i=1 to &sampsize;
  output;
  end;
```

```
%macro simulation1(reps=);
   %do i=1 %to &reps;
     data sim1; set begindat;
        x=ranuni(&seed0);
     proc print data=sim1;
   %end;
%mend simulation1;

%simulation1(reps=&nreps);

Obs    i       x

 1     1    0.09563
 2     2    0.69591
 3     3    0.31711
 4     4    0.58969

Obs    i       x

 1     1    0.09563
 2     2    0.69591
 3     3    0.31711
 4     4    0.58969
```

```
%macro simulation1(reps=);
   %do i=1 %to &reps;
     data sim1; set begindat;
        x=ranuni(&seed0+&i);
     proc print data=sim1;
   %end;
%mend simulation1;

%simulation1(reps=&nreps);
```

```
Obs    i       x
 1     1    0.28059
 2     2    0.66600
 3     3    0.71693
 4     4    0.84909

Obs    i       x
 1     1    0.46555
 2     2    0.63609
 3     3    0.11675
 4     4    0.10849
```

- Although the example on the previous page works, it is unsatisfactory because it is not clear how the sequence jumps around in the cycle
- How about putting the seed in the call statement?

```
call ranuni(&seed0,x);

ERROR 135-185: Attempt to change the value of the constant 97231
in the RANUNI subroutine call.
```

- Try assigning the macro variable `&seed0` to another variable first

```
%macro simulation2(reps=);
   %do i=1 %to &reps;
     data sim2; set begindat;
        seed=&seed0;
        call ranuni(seed,x);
     proc print data=sim2;
   %end;
%mend simulation2;

%simulation2(reps=&nreps);
```

```
Obs    i       seed           x

 1     1    205356066    0.095626
 2     2    205356066    0.095626
 3     3    205356066    0.095626
 4     4    205356066    0.095626

Obs    i       seed           x

 1     1    205356066    0.095626
 2     2    205356066    0.095626
 3     3    205356066    0.095626
 4     4    205356066    0.095626
```

```
%macro simulation3(reps=);
   %do i=1 %to &reps;

data sim3;
set begindat end=eof;

   retain seed &seed0;

   call ranuni(seed,x);

   if eof then do;
      call symput('seed0',put(seed,best.));
   end;

proc print data=sim3;

%end;
%mend simulation3;

%simulation3(reps=&nreps);
```

```
Obs     i         seed           x
 1      1     205356066    0.09563
 2      2    1494458509    0.69591
 3      3     680978604    0.31711
 4      4    1266346632    0.58969
```

```
Obs     i         seed           x
 1      1     730678039    0.34025
 2      2    1730858377    0.80599
 3      3     876140027    0.40798
 4      4    1971178368    0.91790
```

```
%let sampsize=8;
```

```
Obs     i         seed           x
 1      1     205356066    0.09563
 2      2    1494458509    0.69591
 3      3     680978604    0.31711
 4      4    1266346632    0.58969
 5      5     730678039    0.34025
 6      6    1730858377    0.80599
 7      7     876140027    0.40798
 8      8    1971178368    0.91790
```