SOME NOTES ON RANDOM NUMBER GENERATION WITH GAMS

ERWIN KALVELAGEN

Abstract. This document describes some issues with the generation of random numbers in GAMS.

1. Seed

The random number generators in GAMS, `uniform` and `normal`, for the Uniform and Normal distributions, are "pseudo-random": they generate a reproducable series of numbers. Every time you rerun the model, the same series is being generated.

For instance, consider the following model:

```
set i /i1*i10/;
parameter p(i);
p(i) = uniform(0,1);
display p;
```

Every time we run the model, the results are:

```
----      4 PARAMETER p

i1  0.172,    i2  0.843,    i3  0.550,    i4  0.301,    i5  0.292,    i6  0.224
i7  0.350,    i8  0.856,    i9  0.067,    i10 0.500
```

It is possible to make the series dependent on the clock, so that every time the model is run, a different set of random numbers is generated. We need to set the seed number for this:

```
execseed = 1+gmillisec(jnow);
set i /i1*i10/;
parameter p(i);
p(i) = uniform(0,1);
display p;
```

The function `gmillisec` may return zero, thus we add one to make sure `execseed` is set to a strictly positive number. It is noted that now results are no longer reproducable: every time you'll run the model, a different set of numbers is generated.

In the next sections we will concentrate on specific statistical distributions. More information can be found in references such as [2, 3].

2. Exponential distribution

An exponential distribution is defined by a density function of

$$f(x) = \lambda e^{-\lambda x} \tag{2.1}$$

Date: 11 march 2005; revised 9 april 2005.

with mean $\mu = 1/\lambda$ and variance $\sigma^2 = 1/\lambda^2$. Sometimes the probability density is written as:

$$f(x) = \frac{1}{\mu} e^{-x/\mu} \tag{2.2}$$

The cumulative distribution function (cdf) is given by

$$F(x) = 1 - e^{-\lambda x} \tag{2.3}$$

This leads to the following algorithm for generating a random number from an exponential distribution:

(1) Generate a uniform variate $U \sim U(0, 1)$
(2) Solve $F(x) = U$ for x

where $U(0, 1)$ is a uniformly distributed random number between zero and one. The inverse of F is readily available:

$$\begin{aligned} x &= -\frac{\ln(1-U)}{\lambda} \\ &= -\mu \ln(1-U) \\ &= -\mu \ln(U) \end{aligned} \tag{2.4}$$

We have used here that if $U \sim U(0, 1)$ then also $1 - U \sim U(0, 1)$.

The resulting expression

$$\text{Exponential} := -\mu \ln[U(0, 1)] \tag{2.5}$$

can be implemented in GAMS as follows:

```
scalar mu 'mean of exponential distribution' /7.5/;
set i /i1*i10/;
parameter p(i) 'exponential variate';
p(i) = -mu*log(uniform(0.001,1));
display p;
```

3. Poisson distribution

The Poisson distribution is defined by the probability function

$$P(X = n) = \frac{\lambda^n e^{-\lambda}}{n!} \tag{3.1}$$

where the mean and variance are given by

$$\begin{aligned} \mu &= \lambda \\ \sigma^2 &= \lambda \end{aligned} \tag{3.2}$$

From queueing theory it is known that for Poisson arrivals, the inter-arrival times follow an exponential distribution[6]. This relationship can be used to develop the following algorithm for generating Poisson variates with mean μ:

(1) Generate exponential variates $X_1, X_2, ...$ with mean $\mu_{\text{exp}} = 1$ and stop as soon as $\sum_{i=1}^{n} X_i \geq \mu$.
(2) Return $n - 1$

In GAMS this can look like:

```
set i /i1*i10/;

parameter p(i) 'random number with poisson distribution';
scalar mu 'mean for poisson distribution' /5/;

scalar poisson 'poisson variate';
scalar s;

loop(i,

  poisson = -1;
  s = 0;
  repeat
    s = s + [-log(uniform(0.001,1))];
    poisson = poisson + 1;
  until (s >= mu);

  p(i) = poisson;

);

display p;
```

A variant of this algorithm is[5]:

(1) Generate uniform variates $X_1, X_2, \ldots$ from $U(0,1)$ and stop as soon as $\prod_{i=1}^{n} X_i \le e^{-\mu}$.
(2) Return $n-1$

The GAMS implementation can look like:

```
set i /i1*i10/;

parameter p(i) 'random number with poisson distribution';
scalar mu 'mean for poisson distribution' /5/;

scalar poisson 'poisson variate';
scalar prd;


loop(i,

  poisson = -1;
  prd = 1;
  repeat
    prd = prd * uniform(0.001,1);
    poisson = poisson + 1;
  until (prd <= exp(-mu));

  p(i) = poisson;

);

display p;
```

This algorithm is not very suitable if μ is very large.

4. Chi-square distribution

In some cases the nonlinear solver capabilities of GAMS can be used to write quick-and-dirty random number generators. Consider the chi-square distribution. The cumulative distribution function of the chi-square distribution with ν degrees of freedom is given by[4]

$$F(x) = \gamma(\frac{x}{2}, \frac{\nu}{2}) \tag{4.1}$$

where $\gamma(.)$ is the incomplete gamma function. The algorithm

(1) Generate a uniform variate $U \sim U(0, 1)$
(2) Solve $F(x) = U$ for x

is readily implemented in GAMS:

```
$ontext

     Random number generation from Chi-Square distribution.

     Erwin Kalvelagen, march 2005

     Algorithm:
       1. generate u from U(0,1)
       2. solve F(x) = u where F(x) is the
          cdf of the chi-square distribution.

     Note: cdf of chi-square distribution with v degrees of freedom is:

          F(x) = gammareg(x/2,v/2)

     where gammareg() is the (regularized) incomplete gamma function

     For u=0 or u=1 the problem is difficult. You may want to
     replace step 1 by: generate u from U(0.001,0.999).

$offtext

scalar nu 'degrees of freedom' /5/;

set i /i1*i10/;
parameter chisquare(i) 'chi square variates';

parameter u(i);
u(i) = uniform(0.001,0.999);

variable x(i);
equation f(i);

f(i).. gammareg(x(i)/2,nu/2) =e= u(i);

model m /f/;
x.lo(i) = 0;
x.l(i) = nu;
solve m using cns;

chisquare(i) = x.l(i);

display u,chisquare;
```

The model may get into numerical problems if u_i is close to zero or close to one. In that case the distribution function $F(x)$ is either increasing steeply, or flat. A simple work-around would be to generate uniform number numbers between say 0.0001 and 0.9999.

5. Gamma distribution

A random variable $X = \sum_{i=1}^{k} Y_i$ with $Y_i \sim Exp(\lambda)$ is said to follow a gamma distribution $X \sim Gamma(k, \lambda)$. The density function is:

$$f(x) = \frac{1}{\Gamma(k)\lambda^k} x^{k-1} e^{-x/\lambda} \tag{5.1}$$

and the distribution function is:

$$F(x) = \gamma(x/\lambda, k) \tag{5.2}$$

where $\gamma(x, a)$ is the (regularized) incomplete gamma function. It has a mean and variance given by:

$$\begin{aligned} E(X) &= k\lambda \\ Var(X) &= k\lambda^2 \end{aligned} \tag{5.3}$$

The gamma distribution with integer valued k is also called the Erlang distribution. For $k = 1$ we have an exponential distribution.

The algorithm from the previous section leads to:

```
$ontext

     Random number generation from the gamma distribution.

     Erwin Kalvelagen, april 2005

     Algorithm:
       1. generate u from U(0,1)
       2. solve F(x) = u where F(x) is the
          cdf of the gamma distribution.

     Note: the cdf of the gamma distribution with parameters k,lambda is:

          F(x) =  gammareg(x/lambda,k)

     For u=0 or u=1 the problem is difficult. You may want to
     replace step 1 by: generate u from U(0.001,0.999).

$offtext

scalar k /5/;
scalar lambda /3.5/;

set i /i1*i10/;
parameter gamma(i) 'gamma variates';

parameter u(i);
u(i) = uniform(0.001,0.999);

variable x(i);
equation f(i);

f(i).. gammareg(x(i)/lambda,k) =e= u(i);

model m /f/;
x.lo(i) = 0;
x.l(i) = k*lambda;
solve m using cns;

gamma(i) = x.l(i);

display u,gamma;
```

An alternative way to generate random number with a gamma distribution is to use:

(1) Generate $u_i \sim U(0, 1)$ for $i = 1, \ldots, k$
(2) Return $-\lambda \ln \prod_i u_i$

A variant is:

(1) Generate $u_i \sim U(0, 1)$ for $i = 1, \ldots, k$
(2) Return $-\lambda \sum_i \ln u_i$

```
$ontext
```

```
     Random number generation from the gamma distribution.

     Erwin Kalvelagen, april 2005

     Algorithm:
       Generate gamma := - lambda ln prod_{i=1}^k U(0,1)


$offtext

scalar k /5/;
scalar lambda /3.5/;

set i /i1*i10/;
parameter gamma(i) 'gamma variates';

set j /j1*j100/;
abort$(k>card(j)) "increase set j";

gamma(i) = - lambda * log( prod(j$(ord(j)<=k), uniform(0,1)) );

display gamma;
```

6. Beta distribution

The approach of the previous section can be applied to the beta distribution, with density function

$$f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} \tag{6.1}$$

where $B(\alpha,\beta)$ is the beta function:

$$B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{6.2}$$

The distribution function is equal to the (regularized) incomplete beta function:

$$F(x) = I_x(\alpha,\beta) \tag{6.3}$$

The resulting generator can look like:

```
$ontext

     Random number generation from the beta distribution.

     Erwin Kalvelagen, april 2005

     Algorithm:
       1. generate u from U(0,1)
       2. solve F(x) = u where F(x) is the
          cdf of the beta distribution.

     Note: the cdf of the beta distribution with parameters a,b is:

          F(x) = betareg(x,a,b)

     For u=0 or u=1 the problem is difficult. You may want to
     replace step 1 by: generate u from U(0.001,0.999).


$offtext

scalar a /1/;
scalar b /3/;

set i /i1*i10/;
parameter beta(i) 'beta variates';
```

```
parameter u(i);
u(i) = uniform(0.001,0.999);

variable x(i);
equation f(i);

f(i).. betareg(x(i),a,b) =e= u(i);

model m /f/;
x.lo(i) = 1.0e-6;
x.l(i) = 0.5;
x.up(i) = 1-1.0e-6;
solve m using cns;

beta(i) = x.l(i);

display u,beta;
```

For a beta distribution with integer valued parameters a and b an alternative algorithm is available:

(1) Generate $\Gamma_a = \ln \prod_{i=1}^{a} U(0,1)$

(2) Generate $\Gamma_b = \ln \prod_{i=1}^{b} U(0,1)$

(3) Return $\dfrac{\Gamma_a}{\Gamma_a + \Gamma_b}$

or

```
$ontext

     Random number generation from the beta distribution.
     Assumes integer parameters a,b.

     Erwin Kalvelagen, april 2005

     Algorithm:
       1. Generate GAMA := ln prod_{i=1}^a U(0,1)
       1. Generate GAMB := ln prod_{i=1}^b U(0,1)
       2. Beta := GAMA/(GAMA+GAMB)

     You may want to replace the uniform generator call
     by  U(0.0001,1).


$offtext

*
* must be integer
*
scalar a /2/;
scalar b /3/;

set i /i1*i10/;
parameter beta(i);

set k /k1*k100/;
abort$(a>card(k)) "increase set k";
abort$(b>card(k)) "increase set k";

scalar gama,gamb;


loop(i,
   gama = log( prod(k$(ord(k)<=a), uniform(0.0001,1)) );
   gamb = log( prod(k$(ord(k)<=b), uniform(0.0001,1)) );
   beta(i) = gama/(gama+gamb);
);
```

```
display beta;
```

7. Generalized beta distribution

The *standard* beta distribution is defined over the interval $[0, 1]$. The generalized beta distribution is defined over the interval $[A, B]$. The cdf of the generalized beta distribution $F(x, A, B)$ can be expressed in terms of the cdf of the standard beta distribution $F(x)$ as follows:

$$F(x, A, B) = F\left(\frac{x - A}{B - A}\right) \tag{7.1}$$

As a result, the methods of the previous section can be used directly:

(1) Generate x from the *standard* beta distribution
(2) Return $x(B - A) + A$

8. Lognormal distribution

A random variable X has a lognormal distribution if $Y = \ln(X)$ has a normal distribution. The lognormal has two parameters m, and s, with a density function given by:

$$f(x) = \frac{1}{xs\sqrt{2\pi}} \exp\left[\frac{-\ln(x - m)^2}{2s^2}\right] \tag{8.1}$$

The cdf is given by:

$$F(x) = \Phi\left(\frac{\ln(x) - m}{s}\right) \tag{8.2}$$

where $\Phi(.)$ is the cdf of the standard normal distribution (i.e. the error function). The mean and variance are given by:

$$\begin{aligned} \mu &= \exp\left(\frac{2m + s^2}{2}\right) \\ \sigma^2 &= \exp(2m + 2s^2) - \exp(2m + s^2) \end{aligned} \tag{8.3}$$

From given a given mean and variance we can calculate m and s as follows:

$$\begin{aligned} m &= \ln\left(\frac{\mu^2}{\sqrt{\mu^2 + \sigma^2}}\right) \\ s &= \sqrt{\ln\left[\left(\frac{\sigma}{\mu}\right)^2 + 1\right]} \end{aligned} \tag{8.4}$$

If we look at our quick-and-dirty formulation:

```
$ontext

     Random number generation from the lognormal distribution.

     Erwin Kalvelagen, april 2005

     Algorithm:
       1. generate u from U(0,1)
       2. solve F(x) = u where F(x) is the
          cdf of the lognormal distribution.
```

```
     Note: the cdf of the lognormal distribution with mean mu, stderr sigma
     is:
           m = ln(mu^2 / sqrt(mu^2+sigma^2))
           s = sqrt( ln ( (sigma/mu)^2 + 1 ) )
           F(x) = errorf( (ln(x)-m)/s )

$offtext

scalar mu /8/;
scalar sigma /3.5/;

set i /i1*i10/;
parameter lognormal(i) 'lognormal variates';

parameter u(i);
u(i) = uniform(0.001,0.999);

variable x(i);
equation f(i);

scalar m,s;
m = log(sqr(mu)/sqrt(sqr(mu)+sqr(sigma)));
s = sqrt(log(sqr(sigma/mu)+1));

display m,s;

f(i).. errorf((log(x(i))-m)/s) =e= u(i);

display u;

model square /f/;
x.lo(i) = 1.0e-6;
x.l(i) = mu;
solve square using cns;

lognormal(i) = x.l(i);

display u,lognormal;
```

then we see we can also develop a simpler algorithm:

(1) Generate $x \sim N(0,1)$
(2) Return e^{xs+m}

i.e.

```
$ontext

     Random number generation from the lognormal distribution.

     Erwin Kalvelagen, april 2005

     Algorithm:
       1. generate x from N(0,1)
       2. return exp(x*s+m)
     where
           m = ln(mu^2 / sqrt(mu^2+sigma^2))
           s = sqrt( ln ( (sigma/mu)^2 + 1 ) )

$offtext

scalar mu /8/;
scalar sigma /3.5/;

set i /i1*i10/;
parameter lognormal(i) 'lognormal variates';


scalar m,s;
m = log(sqr(mu)/sqrt(sqr(mu)+sqr(sigma)));
s = sqrt(log(sqr(sigma/mu)+1));
```

```
lognormal(i) = exp(normal(0,1)*s+m);

display lognormal;
```

9. Multivariate normal distribution

To generate a multivariate normally distributed quantity with a given variance-covariance structure V, we use the following algorithm[1]:

(1) Calculate Cholesky factors: $V = LL^T$
(2) Generate $z_j \sim N(0, 1)$
(3) Return $x_i := \mu_i + \sum_j L_{i,j} z_j$

This model has been contributed by [7].

```
set i 'set indexing the jointly distributed variables' /i1*i3/;

alias (i,j,k);

parameter mu(i) 'mean values' /i1 100, i2 200, i3 300/;

table v(i,j) 'variance-covariance matrix  (symmetric)'
        i1  i2  i3
    i1  4   8   2
    i2  8   20  6
    i3  2   6   11;

variable  L(i,j) 'Cholesky decomposition of V';

equation  eqv(i,j) 'Definition of LL';

eqv(i,j)$(ord(i) le ord(j))..   V(i,j) =e= sum(k, L(i,k)*L(j,k));

model decomp /eqv/;
L.fx(i,j)$(ord(j) > ord(i)) = 0;
solve decomp using mcp;

display L.l;


*   Generate a multivariate random sample:
parameter z(i) 'Unit normal samples (IID)';

set n  'Set of random samples' /n1*n10/;

parameter x(n,i)  'Samples from a multivariate normal';


*   Change the see if you wish to produce a different pseudo-random
*   sequence:
option seed=1001;

loop(n,

*   Generate a separate IID normal for each element of Z:
    z(i) = normal(0,1);

    x(n,i) = mu(i) + sum(j, L.l(i,j) * z(j));

);

display x;
```

References

1. Peter Clifford, *Multivariate distributions*, http://home.jesus.ox.ac.uk/~clifford/a5/chap1/node16.html.

2. Merran Evans, Nicholas Hastings, and Brian Peacock, *Statistical Distributions*, 3rd ed., Wiley, 2000.
3. Norman L. Johnson, Samuel Kotz, and N. Balakrishnan, *Continuous Univariate Distributions, Volume 1*, 2nd ed., Wiley, 1994.
4. Erwin Kalvelagen, *New special functions in gams*, `http://www.amsterdamoptimization.com/pdf/specfun.pdf`, 2004.
5. Donald E. Knuth, *Seminumerical algorithms*, 3rd ed., The Art of Computer Programming, vol. 2, Addison-Wesley, 1997.
6. Averill Law and W. David Kelton, *Simulation modeling and analysis*, 3rd ed., McGraw-Hill, 1999.
7. Tom Rutherford, *Multivariate normal distributions in gams.*

GAMS Development Corp.
E-mail address: `erwin@gams.com`