Bruce McCarl's GAMS Newsletter Number 35

This newsletter covers

Updates to Expanded GAMS User Guide by McCarl et al.

I updated the Expanded User's Guide to reflect the items discussed here with a few other changes. The latest can be found at http://www.gams.com/dd/docs/bigdocs/gams2002/mccarlgamsuserguide.pdf and will be in upcoming GAMS releases.

Using Probability Distributions on models

GAMS has introduced the ability to include probability density functions and cumulative density functions plus inverse probabilities in models (only the continuous ones) and calculations via use of some provided extrinsic libraries. The continuous distributions across all of these are Beta, Cauchy, ChiSquare, Exponential, F, Gamma, Gumbel, Inverse Gaussian, Laplace, Logistic , Log Normal, Normal, Pareto, Rayleigh, Student's T, Triangular, Uniform and Weibull plus the discrete distributions Binomial, Geometric, Hypergeometric, Logarithmic, Negative Binomial, Poisson and Uniform Integer.

This involves use of the extrinsic library stodclib. When using these one prefixes the distribution name with

- PDF if the probability density function is needed
- CDF if the cumulative distribution function is needed
- ICDF if the inverse cumulative distribution function is needed

In addition when prefixing with PDF or CDF one uses a first argument which is the value of the point to be associated with the probability. When using ICDF one uses the probability as the first argument.

A list of the continuous distributions included and their parameters including a link to a Wolfram Mathworld description of the distribution follows

Beta(ALPHA,BETA)	Beta distribution with parameters ALPHA and BETA
Cauchy(MEDIAN,HALFWIDTH)	Cauchy distribution with parameters MEDIAN and HALFWIDTH
ChiSquare(DF)	Chi-squared distribution with the parameter degrees of freedom DF
Exponential(LAMBDA)	Exponential distribution with rate of change parameter LAMBDA
F(DF1,DF2)	F-distribution with parameters for degrees of freedom DF1 and DF2
Gamma(ALPHA, THETA)	Gamma distribution with parameters ALPHA and THETA
Gumbel(ALPHA,BETA)	Gumbel distribution with parameters ALPHA and BETA
InvGaussian(MU,LAMBDA)	Inverse Gaussian distribution with parameters MU and LAMBDA
Laplace(MU,BETA)	Laplace distribution with parameters MU and BETA
Logistic(MU,BETA)	Logistic distribution with parameters MU and BETA
LogNormal(MU,SIGMA)	Log Normal distribution with parameters MU and SIGMA
Normal(MEAN,STD DEV)	Normal distribution with parameters MEAN and STD DEV
Pareto(K,ALPHA)	Pareto distribution with parameters K which gives the min value of the input item and ALPHA the shape parameter
Rayleigh(SIGMA)	Rayleigh distribution with parameter SIGMA
StudentT(DF)	Student's t-distribution with parameter degrees of freedom DF
Triangular(LOW,MODE,HIGH)	Triangular distribution with parameters telling it falls between LOW and HIGH with MODE being the most probable number
Uniform(LOW,HIGH)	Uniform distribution with parameters telling it falls between LOW and HIGH
Weibull(ALPHA,BETA)	Weibull distribution with parameters ALPHA and BETA

These are used first by activating the functions and giving them a local name. For a use involving a mixture of pdfs, cdfs and icdf for the normal, beta, Cauchy and lognormal distributions this is as follows (extrinsicstoc.gms).

```
$funclibin stolib stodclib
function cdfnorm     /stolib.cdfnormal    /
         icdfnorm    /stolib.icdfnormal   /
         pdfnorm     /stolib.pdfnormal    /
```

```
cdfbeta     /stolib.cdfbeta      /
icdfbeta    /stolib.icdfbeta     /
cdfcauchy   /stolib.cdfcauchy    /
icdfcauchy  /stolib.icdfcauchy   /
cdflognorm  /stolib.cdflognormal /;
```

Where for example the function **CDFNORMAL** that gives the probability up to a specific point in a normal distribution with a given mean and standard deviation is given the local name **cdfnorm**. In turn statements like

```
normalx(i,"cdf")=cdfnorm(normalx(i,"xval"),5,2);
```

where the function is evaluated giving the cumulative probability of the normal distribution from minus infinity to the specified point **normalx(i,"xval")** in a normal distribution with a mean of 5 and a standard deviation of 2.

Examples involving alternative distributions are given below where the first case shows use in **a model equation** and the rest in calculations.

```
totcost.. totalcost=e=costtostock*inventory
       +shortfallcost
       *sum(k,(inventory+del(k)-inventory)

*(pdfnorm((inventory+del(k)),meandemand,stddevdemand)));

lognormalx(i,"cdf")
   =cdflognorm(lognormalx(i,"xval"),lognorm_m,lognorm_s);
normalx(i,"cdf")=cdfnorm(normalx(i,"xval"),10,2);
xvals(i,"val")=icdfnorm(xvals(i,"prob"),meandemand,stddevdema
nd);
shouldbepoint5=cdfbeta(xmedian,alpha,beta);
xmedian=icdfcauchy(0.5,median,halfwidth);
x=cdfcauchy(xmedian,median,halfwidth);
```

Note in this case a statement like **icdfcauchy(0.5,median,**halfwidth**)** finds the x value from a Cauchy distribution with parameters median and halfwidth that has a **cumulative probability of 50%** of the observations below it. Also pdfnorm(inventory,meandemand,stddevdemand) returns the probability of the point inventory from the normal distribution with a mean of meandemand and a standard deviation of stddevdemand**.**

When these used in model equations the model needs to be of the type DNLP. Also note the GLOBAL solvers cannot deal with models that contain such functions.

One can also use discrete distributions but only in calculations. A list of the discrete distributions included and their parameters including a link to a Wolfram Mathworld description of the distribution follows

The discrete distributions included are (Note these cannot be used in model .. equations)

Binomial(N,P)	Binomial distribution with parameters for number of trials N and success probability P in each trial
Geometric(P)	Geometric distribution with parameter giving success probability P in each trial
HyperGeo(TOTAL,GOOD,TRIALS)	Hypergeometric distribution with parameters giving total number of elements TOTAL, number of good elements GOOD and number of trials TRIALS
Logarithmic(THETA)	Logarithmic distribution with parameter THETA, also called log-series distribution
NegBinomial(FAILURES,P)	Negative Binomial distribution with parameters for the number of failures until the experiment is stopped FAILURES and the success probability P in each trial. The number generated by PDF and CDF or input to ICDF describes the number of successes before the defined number of failures
Poisson(LAMBDA)	Poisson distribution with mean LAMBDA
UniformInt(LOW,HIGH)	Integer Uniform distribution with parameters telling that the distribution falls between LOW and HIGH values.

Again these are referenced with the PDF, CDF and ICDF prefixes and the first argument being the point value of the probability. Use of these is not allowed in model equations and when attempted causes an execution error.

Including Random Numbers from extrinsic function libraries

Capabilities have been introduced to generate random numbers from a variety of **distributions** again via extrinsic functions. To do this one can use variants of the functions just discussed (those from the Stochastic Library **stocldib**) or ones from the Lindo Sampling Library (**lsadclib) although use of the Lindo ones requires a license for GAMS/Lindo**. (Without a license only a demo version is available which is restricted to the Normal and the Uniform distribution and no more than 10 sample points.)

Using the Stochastic Library stocldib

In terms of the Stochastic Library the usable continuous and discrete distributions are those listed in the tables above. When using that library one again needs to activate the library and give the function a local name. Here to generate random numbers the distribution name is prefixed with a d. Thus, for example, to generate numbers from the Cauchy, Binomial, Normal and LogNormal one would use code like the following (randnumb.gms)

```
$funclibin stolib stodclib
function randnorm     /stolib.dnormal    /
         randbin      /stolib.dbinomial  /
```

```
          randcauchy   /stolib.dcauchy    /
          randlognorm  /stolib.dlognormal /;
set i /i1*i20/
set j /norm,binomial,cauchy,lognorm/
parameter randx(i,j)    numbers from distributions;
randx(i,"norm")=randnorm(5,2);
randx(i,"binomial")=randbin(10,0.5);
randx(i,"cauchy")=randcauchy(5,1);
randx(i,"lognorm")=randlognorm(1.2,0.3);
display randx;
```

That code first activates the stoclib library then gives the desired functions local names. Subsequently the functions are used in this case generating 20 random numbers for each of the 4 distributions with the desired parameters.

One can also reset the random number seed using a function SetSeed(SEED) as follows

```
function setseedh     /stolib.SetSeed    /;
scalar x;
x=setseedh(99883);
```

where a local function name is again defined and then the seed is set by setting a scalar equal to the function with a user defined value that when this is set to the same value between runs will cause the random numbers to always follow a particular sequence.

Using the Lindo Sampling Library lsadclib

In terms of the Lindo Sampling Library, one can generate random numbers for many of the same distributions with a few additional ones as listed below. In addition one has the capability to introduce correlations and exercise control over the random number process. Note these cannot be used in model equations.

The continuous distributions included and their parameters including a link to a Wolfram Mathworld description of the distribution are listed in the table below. Here note that the parameters here also include

- the sample size (SAMSIZE) telling how many numbers to generate
- an optional parameter (VARRED)that allows one to alter the sampling procedure variance reduction method (0=none, 1=Latin Hyper Square, 2=Antithetic) with Latin Hyper Square Sampling being the default.

Beta(ALPHA,BETA,SAMSIZE[,VARRED])	Beta distribution with parameters ALPHA and BETA
Cauchy(MEDIAN,HALFWIDTH,SAMSIZE[,VARRED])	Cauchy distribution with parameters MEDIAN and HALFWIDTH

ChiSquare(DF,SAMSIZE[,VARRED])	Chi-squared distribution with the parameter degrees of freedom DF
Exponential(LAMBDA,SAMSIZE[,VARRED])	Exponential distribution with rate of change parameter LAMBDA
F(DF1,DF2,SAMSIZE[,VARRED])	F-distribution with parameters for degrees of freedom DF1 and DF2
Gamma(ALPHA, THETA,SAMSIZE[,VARRED])	Gamma distribution with parameters ALPHA and THETA
Gumbel(ALPHA,BETA,SAMSIZE[,VARRED])	Gumbel distribution with parameters ALPHA and BETA
InvGaussian(MU,LAMBDA,SAMSIZE[,VARRED])	Inverse Gaussian distribution with parameters MU and LAMBDA
Laplace(MU,BETA,SAMSIZE[,VARRED])	Laplace distribution with parameters MU and BETA
Logistic(MU,BETA,SAMSIZE[,VARRED])	Logistic distribution with parameters MU and BETA
LogNormal(MU,SIGMA,SAMSIZE[,VARRED])	Log Normal distribution with parameters MU and SIGMA
Normal(MEAN,STD DEV,SAMSIZE[,VARRED])	Normal distribution with parameters MEAN and STD DEV
Pareto(K,ALPHA,SAMSIZE[,VARRED])	Pareto distribution with parameters K which gives the min value of the input item and ALPHA the shape parameter
Rayleigh(SIGMA,SAMSIZE[,VARRED])	Rayleigh distribution with parameter SIGMA
StudentT(DF,SAMSIZE[,VARRED])	Student's t-distribution with parameter degrees of freedom DF
Triangular(LOW,MODE,HIGH,SAMSIZE[,VARRED])	Triangular distribution with parameters telling it falls between LOW and HIGH with MODE being the most probable number
Uniform(LOW,HIGH,SAMSIZE[,VARRED])	Uniform distribution with parameters telling it falls between LOW and HIGH
Weibull(ALPHA,BETA,SAMSIZE[,VARRED])	Weibull distribution with parameters ALPHA and BETA

The discrete distributions included and their parameters including a link to a Wolfram Mathworld description of the distribution are listed in the table below. Here note that the parameters also include

- the sample size (SAMSIZE) telling how many numbers to generate
- an optional parameter (VARRED)that allows one to alter the sampling procedure variance reduction method (0=none, 1=Latin Hyper Square, 2=Antithetic) with Latin Hyper Square Sampling being the default.

Binomial(N,P,SAMSIZE[,VARRED])	Binomial distribution with parameters for number of trials N and success probability P in each trial
HyperGeo(TOTAL,GOOD,TRIALS,SAMSIZE[,VARRED])	Hypergeometric distribution with parameters giving total number of elements TOTAL, number of good elements GOOD and number of trials TRIALS
Logarithmic(THETA,SAMSIZE[,VARRED])	Logarithmic distribution with parameter THETA, also called log-series distribution
NegBinomial(SUCCESS,P,SAMSIZE[,VARRED])	Negative Binomial distribution with parameters for the number of successes (SUCCESS) to be achieved and the probability of success in each trial (P). The generated random number describes the number of failures until we reached the specified number of successes. Note that the Lindo sampling library version is equivalent to the use of the one from the stochastic library when the probability there is 1-P here.
Poisson(LAMBDA)	Poisson distribution with mean LAMBDA

Table J.6: LINDO sampling functions

When using the Lindo Sampling Library one again needs to activate the library and give the functions to be used a local name where here the distribution name is prefixed with SampleLS. For example, to generate 25 numbers from the Cauchy, Binomial, Normal and LogNormal

using the default variance reduction method, one would use code like the following (randnumb.gms). In that code we first activate the library and give the items local names then in the Lindo Sampling case must set a scalar pointer to the function with the distribution arguments and sample size. This returns a pointer to a location identifying where the random numbers are stored. In turn then we use a loop statement to load in the random numbers using a local version of another function getSampleValues referencing the scalar pointer that sequentially returns each of the values in the sample when called. In turn these need to be stored into a GAMS parameter .

```
$funclibin lsalib lsadclib
function normalSample    /lsalib.SampleLSnormal /
         lognormalSample /lsalib.SampleLSlognormal /
         cauchySample    /lsalib.SampleLScauchy /
         binomialSample  /lsalib.SampleLSbinomial /
         getSampleVal    /lsalib.getSampleValues /;

scalar d,h,k,k1,k2;
set is /value01*value12/;
parameter sv(is,j);
k = normalSample(5,2,12);
loop(is, sv(is,"normal") = getSampleVal(k); );
h = lognormalSample(1.2,0.3,12);
loop(is, sv(is,"lognormal") = getSampleVal(h));
k1 = binomialSample(10,0.5,12);
loop(is, sv(is,"binomial") = getSampleVal(k1));
k2 = cauchySample(5,1,12);
loop(is, sv(is,"binomial") = getSampleVal(k2));
```

One can also cause items to be correlated using the functions **setCorrelation and induceCorrelation.** The parameters for this are

setCorrelation(POINTER1,POINTER2,COR)	Causes subsequent calls involving the samples identifies by POINTER1 and POINTER2 to have the correlation COR when next loaded using the getSampleVal function.
induceCorrelation(TYPE)	This controls the type of correlation that is being specified where TYPE is 0 for Pearson, 1 for Kendall or 2 for Spearman. It must be used after setcorrelation.

Coding to make this happen follows. Again we activate the procedures and give them local names then generate the random numbers for two same sized samples. Afterwards we reference set the correlation with arguments of the pointers to the two series then the correlation coefficient and its form. Finally the loop statement is used again to draw out the values.

```
function setCor          /lsalib.setCorrelation /
         indCor          /lsalib.induceCorrelation /;
k = normalSample(5,2,12);
h = lognormalSample(1.2,0.3,12);
d=setCor(h,k,-1);
d=indCor(1);
loop(is,sv2(is,"aftercorr","normal") = getSampleVal(k));
loop(is,sv2(is,"aftercorr","lognormal") = getSampleVal(h));
```

There are also function alternatives to set the random number seed and the type of random number generation process

setSeed(SEED)	the SEED arguments resets the random number generator according to a user defined value
setRNG(RNG)	The parameter RNG identifies the random number generator to use with possible values of -1 (free), 0 (system), 1 (lindo1), 2 (lindo2), 3 (lin1), 4 (mult1), 5 (mult2) and 6 (mersenne).

Making your own Library

Some users may wish to define their own custom extrinsic library. In doing this several steps are recommended.

- Read appendix J of the GAMS user guide to get some idea of basic functionality.
- Identify a function that you wish to be able to use from within GAMS and GAMS solves (excepting the global ones that cannot use extrinsic functions). In doing this insure that you cannot implement the function as a macro or batinclude file as those implementations would be much simpler.
- If you want to use the function in a model sent to a solver (ie within an equation) with endogenous variables as arguments, make sure you know how to compute first and second derivatives with respect to all the variables.
- Choose a programming language. Obvious candidates for a programming language are C, C++, Delphi, or Java, as GAMS makes examples available in these languages. These are available in association with the trigonometric library (trifclib) and the GAMS test library contains examples and source codes. If you download using the IDE library manager note that the manager will place the source zip code including a make file in your project file area.
- Using the demonstration models from the test library, make sure you can exercise the entire process of building and testing the extrinsic function shared library as now done for trifclib in the language of your choice. For example, if you're using C++, look at the testlib model trilib01 or cpplib01. If you're using Fortran, start with trilib03.
- Assemble and test some code that evaluates your function and its derivatives. Note when called the fist parameter tells whether to return the function evaluation, its first derivative or its second derivative.
- Integrate your code into the example code from GAMS using the source files provided for the demo library as a template. This will involve modifying the .spec file to describe the

function you are creating and creating the API files based on the .spec file, and corresponding edits to the source code actually implementing the function.
- Optionally, if this seems like a big step, start small. Implement a simple function (e.g. sqr(sum of x)) for which you know the gradients and Hessians, and get the interface to do this right. Then substitute the real function of interest.
- Do some rigorous, extensive testing of your function. Many examples exist in testlib for doing this, e.g. cpplib01 * cpplib05. The tests might involve pre-computing input args and known function values in another language and storing these values in a GDX file for comparison. Consider testing very small or very large values, and testing how errors in the function will be handled.

Courses offered

I will be teaching

- Basic to Advanced GAMS class Aug 4, 2014- Aug 8, 2014 (5 days) in the Colorado mountains at Frisco (near Breckenridge). The course spans from Basic topics to an Advanced GAMS class. Details are found at http://www.gams.com/courses/basic_and_advanced.pdf .
- Basic GAMS class Aug 4, 2014- Aug 6, 2014 (3 days) in the Colorado mountains at Frisco (near Breckenridge). The course starts assuming no GAMS background. Details are given at http://www.gams.com/courses/basic.pdf .
- Advanced GAMS class Aug 6, 2014- Aug 8, 2014 (3 days) in the Colorado mountains at Frisco (near Breckenridge). The course is for users, who have a GAMS background. Details are found at http://www.gams.com/courses/advanced.pdf .

Further information and other courses are listed on http://www.gams.com/courses.htm . Note I also give custom courses for individual groups a couple of times a year.

Unsubscribe to future issues of this newsletter

Please unsubscribe through the web form available at:
http://app.streamsend.com/public/XLmY/5eq/subscribe

This newsletter is not a product of GAMS Corporation although it is distributed with their cooperation.

July 1, 2014