Using LINGO for Monte Carlo Analysis

LINDO Systems

LINGO has several features that are useful for doing analysis of systems with random or so-called Monte Carlo, features. Here we will concentrate on the @QRAND function that is available for generating quasi-random numbers.

A deterministic model can be converted to a stochastic or Monte Carlo model by the following somewhat mechanical process:

1. Write a deterministic model of the process as if all random variables were known.
2. Convert it to a stochastic scenario model by putting a scenario subscript as the first subscript of every variable that is a) random, or b) which depends upon a random variable.
3. Decide upon the number of random scenarios to generate, and generate a value for each random variable in each scenario.
4. If appropriate, optimize, that is find the best value for our stage 1 decision variables so as to strike the best compromise in the face of the uncertain stage 2 decision variables.

We illustrate with a little model for deciding how many white shirts a catalog retailer should stock in the face of random demand. The stage 1 decision variable is how many shirts to stock. The stage 2 variables are the actual demand (which is random), and the actual sales. The deterministic model looks as follows:

```
! Single period, single product inventory problem,
  Version 1, completely deterministic demand;
DATA:
  MEAN   = 100;
  SPRICE=  48;  ! The selling price/unit;
  PCOST =  22;  ! The purchase cost/unit paid by the retailer;
ENDDATA

   ! A simple model of demand, demand is
      always equal to the mean;
    DEMAND = MEAN;

  MAX = PROFIT;
    PROFIT = SPRICE*SALES - PCOST*STOCK;

 ! We cannot sell more than we stock;
      SALES <= STOCK;
 !  nor more than demand;
      SALES <= DEMAND;
```

The solution is trivial. We simply stock the mean demand.

```
    Variable           Value
      PROFIT        2600.000
       STOCK        100.0000
        MEAN        100.0000
      SPRICE        48.00000
```

```
         PCOST         22.00000
        DEMAND         100.0000
         SALES         100.0000
```

We add randomness to the model by adding a subscript to each variable whose value depends upon this randomness. Specifically, the variables that depend upon the randomness are: DEMAND, and SALES. In order to generate the random demand in LINGO it is convenient to use two additional random variables: a uniform random variable U, and a standard Normal random variable Z. We will use the @QRAND function to generate the uniform random variables.

```
! Single period, single product inventory problem,
  Version 2, demand is Normal distributed with a
  known mean and standard deviation;
SETS:
 ! There are a set of scenarios, each with a demand and sales;
  SCENE: DEMAND, SALES, U, Z;
ENDSETS
DATA:
 ! Scenario related stuff, arbitrarily use 100 scenarios;
  SCENE = 1..100;
 ! Arbitrarily choose a seed for the quasi-random number generator
   and generate a uniform random variable for each scenario;
  U = @QRAND( 314159);

! White shirt case;
  MEAN = 100;   ! Expected demand;
    SD =  25;   ! Estimated standard deviation in demand;
  SPRICE= 48;   ! The selling price/unit;
  PCOST = 22;   ! The purchase cost/unit paid by the retailer;
ENDDATA

 ! Generate the demands;
 @FOR( SCENE(s):
  ! Generate a standard Normal r.v. for each scenario;
   @FREE( Z(s)); @FREE(DEMAND(s));
    U(s)  = @PSN( Z(s)); ! This causes Z to follow a Standard Normal;
   DEMAND(s)  = @SMAX(0, MEAN + SD*Z(s)); ! No demands < 0 allowed;
     );

 ! Count number of scenarios;
    NS = @SIZE( SCENE);

 ! Maximize average profit over all scenarios;
  MAX = PROFIT;
    PROFIT = @SUM( SCENE(s): SPRICE*SALES(s) - PCOST*STOCK)/NS;
    @GIN( STOCK);  ! Can only stock an integer number;

  @FOR( SCENE(s):
 ! We cannot sell more than we stock;
      SALES(s) <= STOCK;
 !  nor more than demand;
      SALES(s) <= DEMAND(s);
      );
```

Notice the solution:

```
    Variable           Value
      PROFIT        2124.745
       STOCK        103.0000
        MEAN        100.0000
          SD        25.00000
      SPRICE        48.00000
       PCOST        22.00000
          NS        100.0000
```

Things of note: We now stock more than the mean demand, and the average or expected profit is in fact about $2125, rather than the previous, optimistic $2600. The uncertainty is costing us almost $500. The LINGO Monte Carlo model says we should stock 103 units. There is an analytical model for the single product inventory problem. For the above demand distribution and cost and selling price, this analytical model also says we should stock 103 units.

Quasi-Random Numbers

Below we display some of the uniform random variables generated by @QRAND. The function @QRAND uses stratified Latin Hypercube sampling. What this means is that the possible outcome space is partitioned into equal size cells or strata, and @QRAND makes sure that there is exactly one outcome in each stratum. Notice, that of the 100 uniform numbers, exactly one appears in the interval (0, .01), one in the interval (.01, .02), etc.

```
   Variable           Value
       U( 1)      0.006931383
       U( 2)      0.01523351
       U( 3)      0.02541635
       U( 4)      0.03120040
       U( 5)      0.04303012
       U( 6)      0.05657129
       U( 7)      0.06373309
       U( 8)      0.07156883
       U( 9)      0.08204396
      U( 10)      0.09214998
      U( 11)      0.1016653
   .
   .
   .
      U( 96)      0.9558357
      U( 97)      0.9616232
      U( 98)      0.9777526
      U( 99)      0.9804613
     U( 100)      0.9900259
```

Compared to using simple random numbers, the use of stratified Latin Hyper cube sampling can make substantial improvements in the accuracy of the simulation results for a given number of scenarios. Loosely speaking, if @QRAND is (properly) used, the number of digits of accuracy in the results is approximately equal to the number of digits in the number of scenarios. For example, if we use 99 scenarios, we might expect the results to be accurate to two digits. If we use 999 scenarios, we might expect the results to be accurate to three digits. If simple random numbers are used, we can expect the number of digits of accuracy in the results to be about half the number of digits in the number of scenarios. If we use 99 scenarios, we can expect one digit of accuracy. If we use 9999 scenarios, we can expect two digits of accuracy in the results.

Multi-dimensional Monte Carlo Models

Now let us add a bit of complexity by introducing a second shirt type. Let us suppose the first shirt type was White, while the second shirt type was Blue. Blue shirts have the same cost and selling price, but a slightly different demand distribution. First we will analyze Blue shirts by themselves as we did with White shirts.

```
! Single period, single product inventory problem,
  Version 2, demand is Normal distributed with a
  known mean and standard deviation;
SETS:
 ! There are a set of scenarios;
  SCENE: DEMAND, SALES, U, Z;
ENDSETS
DATA:
 ! Scenario related stuff;
  SCENE = 1..100;
 ! Arbitrarily choose a seed for the quasi-random number generator
   and generate a uniform random variable for each scenario;
  U = @QRAND( 314159);

! Blue shirt case;
  MEAN = 100;  ! Expected demand;
    SD =  22;  ! Estimated standard deviation in demand;
  SPRICE= 48;  ! The selling price/unit;
  PCOST = 22;  ! The purchase cost/unit paid by the retailer;
ENDDATA

 ! Generate the demands;
 @FOR( SCENE(s):
  ! Generate a standard Normal r.v. for each scenario;
   @FREE( Z(s)); @FREE(DEMAND(s));
    U(s)  = @PSN( Z(s)); ! This causes Z to follow a Standard Normal;
   DEMAND(s)  = @SMAX(0, MEAN + SD*Z(s)); ! No demands < 0 allowed;

     );

 ! Count number of scenarios;
    NS = @SIZE( SCENE);
 ! Maximize average profit over all scenarios;
  MAX = PROFIT;
    PROFIT = @SUM( SCENE(s): SPRICE*SALES(s) - PCOST*STOCK)/NS;
    @GIN( STOCK);  ! Can only stock an integer number;

  @FOR( SCENE(s):
 ! We cannot sell more than we stock;
      SALES(s) <= STOCK;
 !  nor more than demand;
      SALES(s) <= DEMAND(s);
      );
```

Because of the slight lower variability in demand, the solution below recommends stocking only 102 shirts rather than 103. Also, not surprisingly, our profit is slightly higher, although not as near as we might expect to the $2600 profit we would get if we knew in advance that demand would always be exactly 100.

```
    Variable           Value
      PROFIT        2181.795
        MEAN        100.0000
          SD        22.00000
      SPRICE        48.00000
       PCOST        22.00000
          NS        100.0000
       STOCK        102.0000
```

Now let us consider a way that we might increase profits by trying to reduce the bad effects of uncertainty. This is a catalog retailer and he is contemplating the addition of a “second choice” color on the order firm. That is, the customer can indicate a second choice color that he is willing to accept if the first choice color is out of stock. Our retailer estimates that of customers who historically ordered white, 10% would be willing to accept blue. Similarly, 20% of the customers who historically order blue would be willing to accept white. We model this demand process by saying there are three kinds of demand: customers who only want white, customers who only want blue, and customers who are willing to accept either. For customers in the third category, the retailer will of course try to give the customer his first choice, but the retailer knows he can give category three customers either color depending upon availability. There is no analytical solution to this two product model so the following Monte Carlo model is the “only game in town” for analyzing this problem.

```
! Single period, multiproduct inventory problem,
  Version 3, demand is Normal distributed with a
  known mean and standard deviation for each type
  of demand;
SETS:
   DTYPE: MEAN, SD;
 ! There are a set of scenarios;
   SCENE: SELLWW, SELLWE, SELLBB, SELLBE;
   SXD(SCENE,DTYPE): DEMAND,  U, Z;
ENDSETS
DATA:
 ! There are three types of demand:
  1) Customers who will accept only WHITE,
  2) Customers who will accept only BLUE,
  3) Customers who will accept either;
  DTYPE= WHITE BLUE EITHER;
  MEAN =  90    80   30; ! Expected demand of each type;
    SD =  24    21   10; ! Estimated standard deviation in demand;
  SPRICE= 48;  ! The selling price/unit, same for Blue & White;
  PCOST = 22;  ! The purchase cost/unit paid by the retailer;

 ! Scenario related stuff;
  SCENE = 1..100;
 ! Arbitrarily choose a seed for the quasi-random number generator
   and generate uniform random variables for each scenario;
  U = @QRAND( 314159);
ENDDATA

 ! Generate the demands;
 @FOR( SCENE(s):
   @FOR(DTYPE(p):
  !Generate standard Normal r.v. for each scenario s & demand type p;
   @FREE( Z(s,p)); @FREE(DEMAND(s,p));
   U(s,p)= @PSN( Z(s,p)); ! This causes Z to follow a Standard Normal;
   DEMAND(s,p)= @SMAX(0, MEAN(p) + SD(p)*Z(s,p)); ! No demands < 0;
      );
     );

 ! Count number of scenarios;
    NS = @SIZE( SCENE);

 ! Maximize average profit over all scenarios;
  MAX = PROFIT;
    PROFIT = @SUM( SCENE(s):
       SPRICE*(SELLWW(s) + SELLWE(s) + SELLBB(s) + SELLBE(s))
      - PCOST*(STOCKW+STOCKB))/NS;

  @GIN( STOCKW); @GIN(STOCKB); ! Can only stock an integer number;

 ! Use suggestive notation for three types of demand;
   W = 1; B = 2;  E = 3;
  @FOR( SCENE(s):
 ! SELLij = number of type i items sold to type j demand;
 ! We cannot sell more than we stock;
      SELLWW(s) + SELLWE(s) <= STOCKW;
      SELLBB(s) + SELLBE(s) <= STOCKB;
 !  nor more than demand;
      SELLWW(s) <= DEMAND(s,W);
      SELLBB(s) <= DEMAND(s,B);
      SELLWE(s) + SELLBE(s) <= DEMAND(s,E);
      );
```

The interesting part of the solution is:

```
        Variable           Value
          PROFIT        4517.732
          STOCKW        107.0000
          STOCKB        99.00000
          SPRICE        48.00000
           PCOST        22.00000
```

Notice what happened to profit. Before we allowed demand substitution, the total profit for the two products by themselves was \$2124.745 + \$2181.795 = \$4306.54. With the very modest amount of pooling of demand that we obtained by allowing customers to specify a second choice, the expected profit increased to \$4517.73.

Inventory managers may be surprised by what happened to stock levels. Ordinarily one might expect that if demands are pooled, then we need not carry as much inventory. When the two products were run independently, the total inventory stocked was 103 + 102 = 205. When we allowed a modest amount of demand substitution, the inventory stocked actually increased to 107 + 99 = 206. This is not unusual when you do this kind of risk pooling. The optimal amount to stock may in fact increase.

Quasi-Random Numbers and QRAND Revisited

When we have more that one random variable, e.g., demands for 2 or more diffirent products, we need to be aware of how @QRAND works in order to use it properly. Let us take a look at a version of SETS and DATA sections of the previous model in which we have changed the number of scenarios to 8:

```
SETS:
   DTYPE: MEAN, SD;
 ! There are a set of scenarios;
   SCENE: SELLWW, SELLWE, SELLBB, SELLBE;
   SXD(SCENE,DTYPE): DEMAND,  U, Z;
ENDSETS
DATA:
  DTYPE= WHITE BLUE EITHER;
  MEAN =  90    80   30; ! Expected demand of each type;
    SD =  24    21   10; ! Estimated standard deviation in demand;
  SPRICE= 48;  ! The selling price/unit, same for Blue & White;
  PCOST = 22;  ! The purchase cost/unit paid by the retailer;

 ! Scenario related stuff;
  SCENE = 1..8;
 ! Arbitrarily choose a seed for the quasi-random number generator
   and generate uniform random variables for each scenario;
  U = @QRAND( 314159);
ENDDATA
```

When the statement: `U = @QRAND( 314159);` is processed, the following matrix or table with 8 rows and 3 columns is generated.

```
     0.08664229   0.10996630   0.13305708
     0.53787653   0.40345019   0.37562970
     0.39000497   0.95263987   0.10519986
     0.70714114   0.56796593   0.31896174
     0.19041886   0.13841163   0.80300892
     0.79666358   0.31245544   0.90196451
     0.31770432   0.69768947   0.60454588
     0.89461034   0.75732924   0.71220632
```

The three numbers in each row are independent random variables drawn from a uniform distribution in the interval (0,1). The 8 numbers in each column are definitely not independently distributed. If you look closely at each column you will see that there is exactly one number in the column in the interval (0,.125), exactly one number in the interval (.125, .250), etc. This characteristic is known as Latin Hypercube sampling. There is a further feature to this table. If you look at all rows, there is exactly one row with the values in the cell (0,.5)x(0,.5)x(0,.5), exactly one row with values in the cell (.5,1)x(0,.5)x(0,.5), exactly one row with values in the cell (0,.5)x(.5,1)x(0,.5), …, and one row with values in the cell (.5,1)x(.5,1)x(.5,1). Thus, every combination of high and low values for each of the three random variables is encountered exactly once. This is what is known as stratified Latin Hyper cube sampling.

Summarizing, when you use @QRAND to generate random samples or scenarios, you should always make the sample or scenario set the first subscript when declaring a multidimensional or derived set in the SETS section. Thus, the declaration

```
  SETS:
     DTYPE: MEAN, SD;
   ! There are a set of scenarios;
     SCENE: SELLWW, SELLWE, SELLBB, SELLBE;
     SXD(SCENE,DTYPE): DEMAND,  U, Z;
  ENDSETS
```

is consistent with using @QRAND to generated quasi-random samples, however, the declaration:

```
  SETS:
     DTYPE: MEAN, SD;
   ! There are a set of scenarios;
     SCENE: SELLWW, SELLWE, SELLBB, SELLBE;
     DXS(DTYPE,SCENE): DEMAND,  U, Z;
  ENDSETS
```

is not consistent with the intended use of @QRAND.