Creation of Synthetic Discrete Response Regression Models

Joseph Hilbe, *Arizona State University*

The Stata Journal, Volume 10, Nu 1: 104-124

Creation of Synthetic Discrete Response Regression Models

Joseph M. Hilbe

Statisticians employ synthetic data sets to evaluate the appropriateness of fit statistics as well as to determine the effect of modeling the data after making specific alterations to the data. Models based on synthetically created data sets have proved to be extremely useful in this respect, and appear to be used with increasing frequency in texts on statistical modeling.

In this article I demonstrate how to construct synthetic data sets that are appropriate for various popular discrete response regression models. The same methods may be used to create data specific to a wide variety of alternative models. In particular I show how to create synthetic data sets for given types of binomial, Poisson, negative binomial, proportional odds, multinomial, and hurdle models using Stata's random number generators. Demonstrated are standard models, models with an offset, models with a cluster or longitudinal effect, and models having user-defined binary, factor, or non-random continuous predictors. Typically, synthetic models have predictors with values distributed as pseudo-random uniform or pseudo-random normal. This will be our paradigm case, but synthetic data sets do not have to be established in such a manner – as we demonstrate.

In 1995, Walter Linde-Zwirble and I developed a number of (pseudo) random number generators using Stata's programming language (1995, 1998, Hilbe and Linde-Zwirble), including the binomial, Poisson, negative binomial, gamma, inverse Gaussian, beta binomial and others. Based on the rejection method, random numbers that were based on distributions belonging to the one-parameter exponential family of distributions could rather easily be manipulated to generate full synthetic data sets. A synthetic binomial data set could be created, for example, having randomly generated predictors with corresponding user-specified parameters and denominator. One could also specify whether the data was to be logit, probit, or any other appropriate binomial link function.

Stata's random number generators are not only based on a different method from those used in the earlier rnd* suite of generators, but in general they employ different parameters. The examples used in this article all rely on the new Stata functions, and are therefore unlike model creation using the older programs. This is particularly the case for the negative binomial.

I divide this article into four sections. First, I shall discuss creation of synthetic count response models – specifically, Poisson, NB2, and NB-C. Second, I develop code for binomial models, which include both Bernoulli or binary and binomial or grouped logit and probit models. Since the logic of creating and extending such models was developed in the preceding section on count models, I do not need to spend much time explaining how these models work. A third section provides a relatively brief overview of creating synthetic proportional slopes models, including the proportional odds model, and code for constructing synthetic categorical response models, e.g, the multinomial logit. Finally, I present code on how to develop synthetic hurdle models, which are examples of combining binary and count models under a single covering algorithm. Statisticians should find it relatively easy to adjust the code that is provided to construct synthetic data and models for other discrete response regression models.

1: SYNTHETIC COUNT MODELS

I shall first create a simple Poisson model since Stata's *rpoisson()* function is similar to my original *rndpoi* (used to create a single vector of Poisson distributed numbers with a specified mean) and *rndpoix* (used to create a Poisson data set) commands.

SYNTHETIC POISSON DATA
[With predictors *x1* and *x2*, having respective parameters of 0.75 and -1.25 and an intercept of 2]

```
* Joseph Hilbe  22Jan2009 : poi_rng.do
clear
set obs 50000
set seed 4744
gen x1 = invnorm(runiform())   // normally distributed: values between ~ -4.5 - 4.5
gen x2 = invnorm(runiform())   // normally distributed: values between ~ -4.5 - 4.5
gen xb = 2 + 0.75*x1 - 1.25*x2 // linear predictor; define parameters
gen exb = exp(xb)              // inverse link; define Poisson mean
gen py = rpoisson(exb)         // generate random Poisson variate with mean=exb
glm py x1 x2, nolog fam(poi)   // model resultant data
```

The model output is given as:

```
Generalized linear models                          No. of obs      =     50000
Optimization     : ML                              Residual df     =     49997
                                                   Scale parameter =         1
Deviance         =  52295.46204                    (1/df) Deviance =  1.045972
Pearson          =  50078.33993                    (1/df) Pearson  =  1.001627

Variance function: V(u) = u                        [Poisson]
Link function    : g(u) = ln(u)                    [Log]

                                                   AIC             =  4.783693
Log likelihood   = -119589.3262                    BIC             =   -488661
------------------------------------------------------------------------------
             |                 OIM
          py |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          x1 |   .7488765   .0009798   764.35   0.000     .7469562    .7507967
          x2 |  -1.246898   .0009878 -1262.27   0.000    -1.248834   -1.244962
       _cons |   2.002672   .0017386  1151.91   0.000     1.999265     2.00608
------------------------------------------------------------------------------
```

Notice that the parameter estimates approximate the user defined values. If we delete the seed line, add code to store each parameter estimate, and convert the do file to an *rclass ado* program, it is possible to perform a Monte Carlo simulation of the synthetic model parameters.

The above synthetic Poisson data and model code may be amended to do a simple Monte Carlo simulation as follows:

```
* MONTE CARLO SIMULATION OF SYNTHETIC POISSON DATA
* Joseph Hilbe 9Feb2009
program define poi_sim, rclass
version 10
drop _all
set obs 50000
gen x1 = invnorm(runiform())
gen x2 = invnorm(runiform())
gen xb = 2 + 0.75*x1 - 1.25*x2
gen exb = exp(xb)
gen py = rpoisson(exb)
glm py x1 x2, nolog fam(poi)
return scalar sx1 = _b[x1]
return scalar sx2 = _b[x2]
return scalar sc  = _b[_cons]
end
```

Once the model parameter estimates are stored in *sx1*, *sx2,* and *sc* respectively, the following simple **simulate** command can be used for a Monte Carlo simulation involving 100 repetitions. Essentially, what we are doing is performing 100 runs of the **poi_rng** do-file program, and averaging the values of the three resultant parameter estimates.

```
. simulate mx1 = r(sx1)  mx2 = r(sx2) mcon = r(sc), reps(100) : poi_sim

  . . .


. su

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         mx1 |       100    .7499347     .001024   .7476258    .752877
         mx2 |       100   -1.250061    .0010646  -1.252795  -1.246607
        mcon |       100    1.999881    .0016697   1.996198   2.003531
```

Employing a greater number of repetitions will result in mean values closer to the user specified values of .75 and -1.25. Standard errors may also be included in the above simulation, as well as values of the Pearson-dispersion statistic, which will have a value of 1.0 when the model is Poisson. The value of the heterogeneity parameter, *alpha*, may also be simulated for negative binomial models. In fact, any statistic which is stored as a return code may be simulated, as well as any other statistic for which we provide the appropriate storage code.

It should be noted that the Pearson-dispersion statistic displayed in the model output for the generated synthetic Poisson data is 1.001627. This value indicates a Poisson model with no extradispersion. That is, the model is Poisson. Values of the Pearson dispersion greater than 1.0 indicate possible overdispersion in a Poisson model. See Hilbe (2007) for a discussion of count model overdispersion, and Hilbe (2009) for a comprehensive discussion of binomial extradisperson. A brief overview of overdispersion may be found in Hardin and Hilbe (2007).

Poisson models are commonly parameterized as rate models. As such they employ an offset, which reflects the area or time over which the count response is generated. Since the natural log is the canonical link of the Poisson model, the offset must be logged prior to entry into the estimating algorithm.

A synthetic offset may be randomly generated, or may be specified by the user. For this example I will create an area offset having increasing values of 100 for each 10,000 observations in the 50,000 observation data set. The shortcut code used to create this variable is given in the first line below. We assume the same *clear*, *set obs* and *set seed* commands as in the earlier algorithm. I have commented code that can be used to generate the same offset as in the single line command that is used in this algorithm. It better shows what is being done, and can be used by those who are uncomfortable using the shortcut.

SYNTHETIC RATE POISSON DATA

```
 *  Joseph Hilbe  22Jan2009  : poio_rng.do
< clear, set obs and set seed commands>

. gen off = 100+100*int((_n-1)/10000)  // creation of offset

. * gen off=100 in 1/10000             // These lines duplicate the single line above
. * replace off=200 in 10001/20000
. * replace off=300 in 20001/30000
. * replace off=400 in 30001/40000
. * replace off=500 in 40001/50000

. gen loff = ln(off)                      // log offset prior to entry into model
. gen x1 = invnorm(runiform())
. gen x2 = invnorm(runiform())
. gen xb = 2 + 0.75*x1 - 1.25*x2 + loff   // offset added to linear predictor
. gen exb = exp(xb)
. gen py = rpoisson(exb)
. glm py x1 x2, nolog fam(poi) off(loff)  // added offset option
```

We expect that the resultant model will have approximately the same parameter values as the earlier model, but with different standard errors. Modeling the data without using the offset option results in similar parameter estimates to those produced when an offset is employed, but highly inflated intercept.

The results of the rate parameterized Poisson algorithm above is displayed below:

```
Generalized linear models                          No. of obs      =     50000
Optimization     : ML                              Residual df     =     49997
                                                   Scale parameter =         1
Deviance         =  49847.73593                    (1/df) Deviance =  .9970145
Pearson          =  49835.24046                    (1/df) Pearson  =  .9967646

Variance function: V(u) = u                        [Poisson]
Link function    : g(u) = ln(u)                    [Log]

                                                   AIC             =  10.39765
Log likelihood   = -259938.1809                    BIC             = -491108.7
------------------------------------------------------------------------------
             |                 OIM
          py |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
```

```
          x1 |   .7500656   .0000562 13346.71   0.000     .7499555    .7501758
          x2 |  -1.250067   .0000576 -2.2e+04   0.000     -1.25018   -1.249954
       _cons |   1.999832   .0001009 19827.16   0.000     1.999635     2.00003
        loff |   (offset)
------------------------------------------------------------------------------
```

I mentioned earlier that Poisson models having a Pearson dispersion greater than 1.0 indicates possible overdispersion. The negative binomial (NB2) model is commonly used in such situations to accommodate the extra dispersion.

The NB2 parameterization of the negative binomial can be generated as a Poisson-gamma mixture model, with a gamma scale parameter of 1. We use this method to create synthetic NB2 data. The negative binomial random number generator in Stata is not parameterized as NB2, but rather derives directly from the canonical negative binomial (see 2007, Hilbe). *rnbinomial()* may be used to create a synthetic canonical negative binomial (NB-C) model, but not NB2 or NB1. Below is code that can be used to construct NB2 model data. The same parameters are used here as for the above Poisson models.

SYNTHETIC NEGATIVE BINOMIAL (NB2) DATA

```
*  Joseph Hilbe  22Jan2009  : nb2_rng.do
clear
set obs 50000
set seed 4744
gen x1 = invnorm(runiform())
gen x2 = invnorm(runiform())
gen xb = 2 + 0.75*x1 - 1.25*x2  // same linear predictor as Poisson above
gen a = .5                      // value of alpha, the NB2 heterogeneity parameter
gen ia = 1/a                    // inverse alpha
gen exb = exp(xb)               // NB2 mean
gen xg = rgamma(ia, a)          // generate random gamma variate given alpha
gen xbg = exb * xg              // gamma variate parameterized by linear predictor
gen nby = rpoisson(xbg)         // generate mixture of gamma and Poisson
nbreg nby x1 x2, nolog          // model as negative binomial (NB2)
```

Model output is given as:

```
Negative binomial regression                      Number of obs   =      50000
                                                  LR chi2(2)      =   74048.24
Dispersion     = mean                             Prob > chi2     =     0.0000
Log likelihood = -153736.53                       Pseudo R2       =     0.1941
------------------------------------------------------------------------------
         nby |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          x1 |    .747614   .0038335   195.02   0.000     .7401004    .7551277
          x2 |  -1.248845   .0040646  -307.25   0.000    -1.256811   -1.240878
       _cons |     2.0027   .0039333   509.16   0.000     1.994991    2.010409
-------------+----------------------------------------------------------------
    /lnalpha |  -.6967115   .0083518                     -.7130807   -.6803423
-------------+----------------------------------------------------------------
       alpha |    .498221    .004161                      .4901319    .5064436
------------------------------------------------------------------------------
Likelihood-ratio test of alpha=0:  chibar2(01) = 4.1e+05 Prob>=chibar2 = 0.000
```

Note that the values of the parameters and of *alpha* approximate the values specified in the algorithm. These values may of course be altered by the user. To verify the appropriateness of the model I estimate the same data using the **glm** command below, with the value of *alpha* given by the maximum likelihood model. Observe the Pearson dispersion; it approximates 1.0. This same data estimated using a Poisson model yields a dispersion value of 11.67703 (not shown). The data is therefore Poisson overdispersed, but NB2 equidispersed, as we expect. See Hilbe (2007) for a discussion of NB2 overdispersion and how it compares with Poisson overdispersion.

```
. glm nby x1 x2, nolog fam(nb .498221)

Generalized linear models                          No. of obs      =     50000
Optimization     : ML                              Residual df     =     49997
                                                   Scale parameter =         1
Deviance         =    54228.123                    (1/df) Deviance =  1.084628
Pearson          =  49817.63954                    (1/df) Pearson  =  .9964126

Variance function: V(u) = u+(.498221)u^2           [Neg. Binomial]
Link function    : g(u) = ln(u)                    [Log]

                                                   AIC             =  6.149581
Log likelihood   = -153736.5309                    BIC             = -486728.3
------------------------------------------------------------------------------
             |                 OIM
         nby |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          x1 |    .747614   .0038335   195.02   0.000     .7401004    .7551277
          x2 |  -1.248845   .0040646  -307.25   0.000    -1.256811   -1.240878
       _cons |     2.0027   .0039333   509.16   0.000     1.994991    2.010409
------------------------------------------------------------------------------
```

Performing a Monte Carlo simulation of the NB2 model requires that the algorithm first estimate a maximum likelihood model, estimating *alpha*. *alpha* is then passed to a **glm** command which provides estimation of the dispersion statistic as well as parameter estimates. The value of *alpha* is entered as a constant into the **glm** algorithm by use of the option, fam(nb `=e(alpha)'). Note how the statistics we wish to use in the Monte Carlo simulation program are stored.

[Note: When Stata's **glm** command is amended so that the negative binomial family option allows maximum likelihood estimation of *alpha*, the following code can bypass the **nbreg** command.]

```
* SIMULATION OF SYNTHETIC NB2 DATA
* Joseph Hilbe Jan 2009
* x1=.75, x2=-1.25, _cons=2, alpha=0.5
program define nb2_sim, rclass
version 10
clear
set obs 50000
gen x1 = invnorm(runiform())
gen x2 = invnorm(runiform())
gen xb = 2 + 0.75*x1 - 1.25*x2
gen a = .5
gen ia = 1/a
gen exb = exp(xb)
gen xg = rgamma(ia, a)
gen xbg = exb * xg
gen nby = rpoisson(xbg)
nbreg nby x1 x2, nolog                     // model specified synthetic NB2 data
glm nby x1 x2, nolog fam(nb `=e(alpha)')  // glm with alpha from nbreg
return scalar sx1 = _b[x1]             // synthetic model value of x1
return scalar sx2 = _b[x2]             // synthetic model value of x2
return scalar sxc = _b[_cons]          // synthetic model value of intercept (_cons)
return scalar pd  = e(dispers_p)       // synthetic model value Pearson dispersion
return scalar _a = `e(a)'              // synthetic model value of alpha
end
```

In order to obtain the Monte Carlo averaged statistics we desire, we use the following options with the **simulate** command.

```
. simulate mx1= r(sx1) mx2= r(sx2) mxc= r(_cons) pdis= r(pd) alpha= r(__a), reps(100)
   : nb2_sim

. su

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         mx1 |       100    .7495019    .0036051   .7406389   .7586843
         mx2 |       100   -1.250206    .0043555  -1.258774  -1.239371
         mxc |       100    1.999624    .0036468   1.989279   2.007741
        pdis |       100      .99991    .0047734    .989656   1.011759
       alpha |       100    .5003388    .0041727   .4910004   .5130489
```

Note the range of parameter and dispersion values. The code for constructing synthetic data sets produce quite good values; i.e. they have a narrow range of values. This is exactly what we want from an algorithm that creates synthetic data.

We may employ an offset into the NB2 algorithm in the same manner as we did for the Poisson. Since the mean of the Poisson and NB2 are both exp(xb), we may use the same method. The synthetic NB2 data and model with offset is in the **nb2o_rng.do** file.

Incorporating a cluster or longitudinal effect into the algorithm takes a different tactic. For simplicity I used the same variable for a cluster effect that was used for an offset. Note that the cluster variable is not logged, nor is it added to the linear predictor. It simply adjusts the standard errors of the predictors. The command to model the cluster effect is done using the following command code:

```
 . nbreg nby x1 x2 x3, nolog cluster(off)
```

The code is in **nb2re_rng.do**.

The linear negative binomial model, NB1, is also based on a Poisson-gamma mixture distribution. Space limitations prohibit me from describing it further in this article, but construction of synthetic data and models for the NB1 is done using close to the same code as used for NB2.

The canonical negative binomial (NB-C), however, must be constructed in an entirely different manner from NB2, NB1, or from Poisson. NB-C is not a Poisson-gamma mixture, and is based on the negative binomial PDF. Stata's *rnbinomial(a,b)* function can be used to construct NB-C data. Other options such as offsets, non-random variance adjusters, and so forth, are easily adaptable for the *nbc_rng.do* function.

SYNTHETIC CANONICAL NEGATIVE BINOMIAL (NB-C) DATA

```
* Joseph Hilbe  22Jan2009  : nbc_rng.do
clear
set obs 50000
set seed 7787
gen x1 = runiform()
gen x2 = runiform()
gen xb = 1.25*x1 + .1*x2 -1.5
gen a =  1.15
gen mu = 1/((exp(-xb)-1)*a)         // inverse link function
gen p = 1/(1+a*mu)                  // probability
gen r = 1/a
gen y = rnbinomial(r, p)
cnbreg y x1 x2, nolog
```

I wrote a maximum likelihood canonical negative binomial command in 2005, which was posted to the SSC site, and have posted an amendment in late February, 2009. The statistical results are the same in the original and amended version, but the amendment is more efficient, and pedagogically easier to understand. Rather than simply inserting the NB-C inverse link function in terms of xb for each instance of μ in the log-likelihood function, I have reduced the formula for the NB-C log-likelihood to

$$LL_{NB\text{-}C} = \Sigma \{y(xb) + (1/\alpha)\ln(1-\exp(xb)) + \ln\Gamma(y+1/\alpha) - \ln\Gamma(y+1) - \ln\Gamma(1/\alpha) \}$$

Also posted to the site is a heterogeneous NB-C regression command that allows parameterization of the heterogeneity parameter, *alpha*. Stata calls the NB2 version of this a generalized negative binomial. However, as I discuss in Hilbe (2007), there are previously implemented generalized negative binomial models with entirely different parameterizations. Some are discussed in that source. Moreover, Limdep has offered a heterogeneous negative binomial for many years, which is the same model as is the generalized NB in Stata. For these reasons I prefer labeling Stata's **gnbreg** command a heterogeneous model. A **hcnbreg** command was also posted to SSC in 2005.

The synthetic NB-C model of the above created data is displayed below. Note that I had specified values of *x1* and *x2* as 1.25 and .1 respectively, and an intercept value of -1.5. *alpha* was given as 1.15. The model closely reflects the user specified parameters.

```
Canonical Negative Binomial Regression            Number of obs   =      50000
                                                  Wald chi2(2)    =    6386.70
Log likelihood = -62715.384                       Prob > chi2     =     0.0000
------------------------------------------------------------------------------
           y |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          x1 |   1.252674    .015776    79.40   0.000     1.221754    1.283595
          x2 |   .1009038   .0091313    11.05   0.000     .0830067    .1188008
       _cons |  -1.504659   .0177159   -84.93   0.000    -1.539381   -1.469936
-------------+----------------------------------------------------------------
    /lnalpha |   .1336433   .0153947     8.68   0.000     .1034702    .1638164
-------------+----------------------------------------------------------------
       alpha |   1.142985   .0175959                      1.109013    1.177998
------------------------------------------------------------------------------
AIC Statistic   =        2.509
```

2: SYNTHETIC BINOMIAL MODELS

Synthetic binomial models are constructed in the same manner as synthetic Poisson data and models. The key lines are those that generate pseudo-random variates, a line creating the linear predictor with user defined parameters, a line using the inverse link function to generate the mean, and a line using the mean to generate random variates appropriate to the distribution.

A Bernoulli distribution consists entirely of binary values, 1/0. *y* is binary and is considered here to be the response variable which is explained by the values of *x1* and *x2*. Data such as this is typically modeled using a logistic regression. A probit or complementary loglog model can also be used to model the data.

	y	x1	x2
1:	1	1	1
2:	0	1	1
3:	1	0	1
4:	1	1	0
5:	1	0	1
6:	0	0	1

The above data may be grouped by covariate patterns. The covariates here are, of course, *x1* and *x2*. With *y* now the number of successes, i.e. a count of 1's, and *m* the number of observations having the same covariate pattern, the above data may be grouped as:

	y	m	x1	x2
1:	1	2	1	1
2:	2	3	0	1
3:	1	1	1	0

The distribution of *y/m* is binomial. *y* is a count of observations having a value of *y=1* for a specific covariate pattern, and *m* is the number of observations having the same covariate pattern. One can see that the Bernoulli distribution is a subset of the binomial, i.e. a binomial distribution where *m=1*. In actuality, a logistic regression models the top data as if there were no m, regardless of the number of separate covariate patterns. Grouped logistic, or binomial-logit, regression assumes appropriate values of *y* and *m*. In Stata, grouped data such as the above can be modeled as a logistic regression using the **blogit** or **glm** commands. I recommend using the **glm** command since **glm** is accompanied with a wide variety of test statistics.

Algorithms for constructing synthetic Bernoulli models differ little from creating synthetic binomial models. The only difference is that for the binomial, *m* needs to be accommodated. I shall demonstrate the difference – and similarity – of the Bernoulli and binomial by generating data using the same parameters. First the Bernoilli-logit model, or logistic regression:

SYNTHETIC BERNOULLI-LOGIT DATA

```
* Joseph Hilbe 5Feb2009 berl_rng.do
* x1=.75, x2=-1.25, _cons=2
clear
set obs 50000
set seed 13579
gen x1 = invnorm(runiform())
gen x2 = invnorm(runiform())
gen xb = 2 + 0.75*x1 - 1.25*x2
gen exb = 1/(1+exp(-xb))                   // inverse logit link
gen by = rbinomial(1, exb)                 // specify m=1 in function
logit by x1 x2, nolog
```

The output is displayed as:

```
Logistic regression                               Number of obs   =      50000
                                                  LR chi2(2)      =   10861.44
                                                  Prob > chi2     =     0.0000
Log likelihood =   -18533.1                       Pseudo R2       =     0.2266
------------------------------------------------------------------------------
          by |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          x1 |   .7555715   .0143315    52.72   0.000     .7274822    .7836608
          x2 |  -1.256906    .016125   -77.95   0.000     -1.28851   -1.225301
       _cons |   2.018775   .0168125   120.08   0.000     1.985823    2.051727
------------------------------------------------------------------------------
```

Secondly, the code for constructing a synthetic binomial model:

SYNTHETIC BINOMIAL-LOGIT DATA

```
* Joseph Hilbe 5feb2009  binl_rng.do
* x1=.75, x2=-1.25, _cons=2
clear
set obs 50000
set seed 13579
gen x1 = invnorm(runiform())
gen x2 = invnorm(runiform())
* =================================
* Select either User Specified or Random
*  denominator.
* gen d = 100+100*int((_n-1)/10000)
gen d = ceil(10*runiform())           // integers 1-10, mean of ~5.5
* =================================
gen xb = 2 + 0.75*x1 - 1.25*x2
gen exb = 1/(1+exp(-xb))
gen by = rbinomial(d, exb)
glm by x1 x2, nolog fam(bin d)
```

The final line calculates and displays the output below:

```
Generalized linear models                          No. of obs      =     50000
Optimization     : ML                              Residual df     =     49997
                                                   Scale parameter =         1
Deviance         =  47203.16385                    (1/df) Deviance =  .9441199
Pearson          =   50135.2416                    (1/df) Pearson  =  1.002765

Variance function: V(u) = u*(1-u/d)                [Binomial]
Link function    : g(u) = ln(u/(d-u))              [Logit]

                                                   AIC             =  1.854676
Log likelihood   = -46363.90908                    BIC             = -493753.3
------------------------------------------------------------------------------
             |                 OIM
          by |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          x1 |   .7519113   .0060948   123.37   0.000     .7399657    .7638569
          x2 |  -1.246277   .0068415  -182.16   0.000    -1.259686   -1.232868
       _cons |    2.00618   .0071318   281.30   0.000     1.992202    2.020158
------------------------------------------------------------------------------
```

The only difference between the two is the code between the lines, and the use of *d* rather than *1* in the *rbinomial()* function. I show code for generating a random denominator, and code for specifying the same values as were earlier used for the Poisson and negative binomial offsets. Cameron & Trivedi (2009) have a nice discussion of generating binomial data. Their focus, however, differs from the one taken here. I nevertheless recommend reading Chapter 4 of their book.

Note the similarity of parameter values. Use of Monte Carlo simulation shows that both produce identical results. I should mention that the dispersion statistic is only appropriate for binomial models, not for Bernoulli. The binomial-logit model above has a dispersion of 1.002765, which is as we would expect. This relationship is discussed in detail in Hilbe (2009).

It is easy to amend the above code to construct synthetic probit or complementary loglog data.

Assuming the identical first six lines of the Bernoulli-logit code, the synthetic binary probit data may be generated using the following:

```
gen double exb = normprob(xb)
* replace exb=.99999999 if exb>.99999999  // add if need 50000 obs
gen double py = rbinomial(1, exb)
```

The *normprob()* function is the inverse probit link, and replaces the inverse logit link. The problem is that the function typically drops 100-400 observations in a 50000 observation data set. This occurs when *exb=1*. I have created a partial fix so that the full number of user specified synthetic observations are created, but it does bias the data slightly – very slightly, as seen from the results of a 100 repetition Monte Carlo simulation.

```
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         mx1 |       100    .7480579    .0107474   .7219242    .777306
         mx2 |       100   -1.250415    .0149592  -1.278488  -1.212393
        mcon |       100    2.001031    .0103345   1.980775   2.021287
```

If you wish to keep the full 50,000 (or whatever number you desire) synthetic probit observations, be aware of the slight bias.

3: SYNTHETIC CATEGORICAL RESPONSE MODELS

I have previously discussed in detail the creation of synthetic ordered logit, or proportional odds, data in Hilbe (2009), and refer to that source for a more thorough examination of the subject. Multinomial logit data is also examined in the same source. Because of the complexity of the model, the generated data is a bit more variable than with synthetic logit, Poisson, or negative binomial models. However, Monte Carlo simulation (not shown) proves that the mean values closely approximate the user supplied parameters and cut points.

I display code for generating synthetic ordered probit data below.

```
* SYNTHETIC ORDERED PROBIT DATA AND MODEL
* Hilbe, Joseph 19Feb 2008  : oprobit_rng.do
di in ye "b1 = .75; b2 = 1.25"
di in ye "Cut1=2; Cut2=3,; Cut3=4"
drop _all
set obs 50000
set seed 12345
gen double x1 = 3*uniform()+1
gen double x2 = 2*uniform()-1
gen double y = .75*x1 + 1.25*x2 + invnorm(uniform())
gen int ys = 1 if y<=2
replace ys=2 if y<=3 & y>2
replace ys=3 if y<=4 & y>3
replace ys=4 if  y>4
oprobit ys x1 x2, nolog
* predict double (olpr1 olpr2 olpr3 olpr4), pr
```

The modeled data appears as:

```
Ordered probit regression                         Number of obs   =      50000
                                                  LR chi2(2)      =   24276.71
                                                  Prob > chi2     =     0.0000
Log likelihood = -44938.779                       Pseudo R2       =     0.2127
------------------------------------------------------------------------------
          ys |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          x1 |   .7461112    .006961   107.18   0.000     .7324679    .7597544
          x2 |   1.254821   .0107035   117.23   0.000     1.233842    1.275799
-------------+----------------------------------------------------------------
       /cut1 |   1.994369   .0191205                      1.956894    2.031845
       /cut2 |   2.998502   .0210979                      2.957151    3.039853
       /cut3 |   3.996582   .0239883                      3.949566    4.043599
------------------------------------------------------------------------------
```

The user specified slopes were .75 and 1.25, which are closely approximated above. Likewise, the specified cuts of 2, 3, and 4 are nearly identical to the synthetic values, which are the same to the thousandths place.

The proportional slopes code is created by adjusting the linear predictor. Unlike the ordered probit, we need to generate pseudo-random uniform variates, called *err*, which are then used in the logistic link function, as attached to the end of the linear predictor. The remainder of the code is the same for both algorithms. The lines required to create synthetic proportional odds data are the following:

```
gen err = runiform()
gen y = .75*x1 + 1.25*x2 + log(err/(1-err))
```

Synthetic multinomial logit data may be constructed using the following code:

SYNTHETIC MULTINOMIAL LOGIT DATA AND MODEL

```
. Joseph Hilbe 15Feb2008  mlogit_rng.do
. y=2: x1= 0.4, x2=-0.5, _cons=1.0
. y=3: x1=-3.0, x2=0.25, _cons=2.0
. qui {
. clear
. set mem 50m
. set seed 111322
. set obs 100000
. gen x1 = runiform()
. gen x2 = runiform()
. gen denom = 1+exp(.4*x1 - .5*x2 +1 ) + exp(-.3*x1+.25*x2 +2)
. gen p1 = 1/denom
. gen p2 = exp(.4*x1-.5*x2 + 1) / denom
. gen p3 = exp(-.3*x1+.25*x2 + 2) / denom
. gen u = runiform()
. gen y = 1 if u <= p1
. gen p12 = p1 + p2
. replace y = 2 if y==. & u<=p12
. replace y = 3 if y==.
. }
. mlogit y x1 x2,  baseoutcome(1) nolog
```

Note that I have amended the *uniform()* function that was in the original code to *runiform()*, which is Stata's newest version of the pseudo-random uniform generator. The logic of the code is examined in Hilbe (2009), to which I refer the reader. However, given the nature of the multinomial probability function, the above code is rather self-explanatory. The code may be expanded to have more than three levels. New coefficients need to be defined and the probability levels expanded. See the above reference for advice on coding for more levels.

The output of the above **mlogit_rng** *do* file is displayed as:

```
Multinomial logistic regression                   Number of obs   =     100000
                                                  LR chi2(4)      =    1652.17
                                                  Prob > chi2     =     0.0000
Log likelihood = -82511.593                       Pseudo R2       =     0.0099
------------------------------------------------------------------------------
           y |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
2            |
          x1 |   .4245588   .0427772     9.92   0.000     .3407171    .5084005
          x2 |  -.5387675   .0426714   -12.63   0.000    -.6224019    -.455133
       _cons |   1.002834   .0325909    30.77   0.000     .9389565    1.066711
-------------+----------------------------------------------------------------
3            |
          x1 |  -.2953721    .038767    -7.62   0.000     -.371354   -.2193902
          x2 |   .2470191   .0386521     6.39   0.000     .1712625    .3227757
       _cons |   2.003673   .0295736    67.75   0.000      1.94571    2.061637
------------------------------------------------------------------------------
(y==1 is the base outcome)
```

By amending the **mlogit_rng.do** code to an *rclass ado* program, with the following lines added to the end:

```
return scalar x1_2 = [2]_b[x1]
return scalar x2_2 = [2]_b[x2]
return scalar _c_2 = [2]_b[_cons]
return scalar x1_3 = [3]_b[x1]
return scalar x2_3 = [3]_b[x2]
return scalar _c_3 = [3]_b[_cons]
end
```

the following Monte Carlo simulation may be run, verifying the parameters displayed from the *do* file. The ado file is named ***mlogit_sim***.

```
. simulate mx12 = r(x1_2)  mx22 = r(x2_2)  mc2 = r(_c_2) mx13 = r(x1_3) mx23 = r(x2_3)
mc3 = r(_c_3) , reps(100) : mlogit_sim

.   .   .

. su

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        mx12 |       100    .4012335    .0389845   .2992371   .4943814
        mx22 |       100   -.4972758    .0449005  -.6211451  -.4045792
         mc2 |       100    .9965573    .0300015    .917221     1.0979
        mx13 |       100   -.2989224    .0383149  -.3889697  -.2115128
        mx23 |       100    .2503969    .0397617   .1393684   .3484274
         mc3 |       100    1.998332    .0277434   1.924436   2.087736
```

It is observed that the user specified values are reproduced by the synthetic multinomial program.

4: SYNTHETIC HURDLE MODELS

Lastly, I show an example of how one can expand the above synthetic data generators to construct synthetic negative binomial-logit hurdle data. The code may be easily amended to construct Poisson-logit, Poisson-probit, Poisson-cloglog, NB2—probit, and NB2-cloglog models. In 2005 I published a number of hurdle models, which are currently on the SSC site. I show this example to demonstrate how similar synthetic models may be created for zero-truncated and zero-inflated models, as well as a variety of different types of panel models. Synthetic models and correlation structures are found in Hardin & Hilbe (2003) for GEE models.

Hurdle models are discussed in Hilbe (2007), Cameron & Trivedi (2009), and Long & Freese (2006). The traditional method of parameterizing hurdle models is to have both binary and count models be of equal length. Hurdle models having constituent models of differing predictors is discussed in Cameron & Trivedi (2009), For reasons to be discussed elsewhere, I believe equal length hurdle models are preferred.

Note that a hurdle model is a combination of a 1/0 binary model and a zero-truncated count model. There is no estimation overlap in response values of 1, as is the case for zero-inflated models.

SYNTHETIC NB2-LOGIT HURDLE DATA

```
* Joseph Hilbe  26Sep2005; Mod 4Feb2009.
* nb2logit_hurdle.do
* LOGIT: x1=-.9, x2=-.1, _c=-.2
* NB2  : x1=.75, n2=-1.25, _c=2, alpha=.5
clear
set obs 50000
set seed 1000
gen x1 = invnorm(runiform())
gen x2 = invnorm(runiform())
* NEGATIVE BINOMIAL- NB2
gen xb = 2 + 0.75*x1 - 1.25*x2
gen a = .5
gen ia = 1/a
gen exb = exp(xb)
gen xg = rgamma(ia, a)
gen xbg = exb * xg
gen nby = rpoisson(xbg)
* BERNOULLI
drop if nby==0
gen pi =1/(1+exp(-(.9*x1 + .1*x2+.2)))
gen bernoulli = runiform()>pi
replace nby=0 if bernoulli==0
rename nby y
* logit bernoulli x1 x2, nolog /// test
* ztnb y x1 x2 if y>0, nolog   /// test
* NB2-LOGIT HURDLE
hnblogit y x1 x2, nolog
```

Output for the above synthetic NB2-logit hurdle model is displayed as

```
Negative Binomial-Logit  Hurdle Regression        Number of obs   =      43443
                                                  Wald chi2(2)    =    5374.14
Log likelihood = -84654.938                       Prob > chi2     =     0.0000
------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
logit        |
          x1 |  -.8987393   .0124338   -72.28   0.000    -.9231091   -.8743695
          x2 |  -.0904395    .011286    -8.01   0.000    -.1125597   -.0683194
       _cons |  -.2096805   .0106156   -19.75   0.000    -.2304867   -.1888742
-------------+----------------------------------------------------------------
negbinomial  |
          x1 |    .743936   .0069378   107.23   0.000     .7303381    .7575339
          x2 |  -1.252363   .0071147  -176.02   0.000    -1.266307   -1.238418
       _cons |   2.003677   .0070987   282.26   0.000     1.989764     2.01759
-------------+----------------------------------------------------------------
    /lnalpha |  -.6758358   .0155149   -43.56   0.000    -.7062443   -.6454272
-------------+----------------------------------------------------------------
       alpha |   .5087311   .0078929                      .4934941    .5244384
------------------------------------------------------------------------------
AIC Statistic =     3.897
```

SUMMARY REMARKS

Synthetic data can be used with substantial efficacy for the evaluation of of statistical models. In this article I have presented algorithm code that can be used to create a number of different types of synthetic models. The code may be extended to use for the generation of yet other synthetic models.

I am a strong advocate of using these types of models to better understand the models we apply to real data. With computers gaining in memory and speed, it is possible to construct for more complex synthetic data than we have here. This has been accomplished in a number of different disciplines. I hope that the ones discussed in this article will encourage further use and construction of artificial data.

References:

Cameron, A.C. and P.K. Trivedi, *Microeconometrics Using Stata*, College Station, TX: Stata Press.

Hardin, JW and J.M Hilbe (2003), *Generalized Estimating Equations* Boca Raton, FL: Chapman & Hall/CRC.

Hardin, JW and J.M Hilbe (2007), *Generalized Linear Models and Extensions,* second edition, College Station, TX: Stata Press.

Hilbe, J.M. (2007), *Negative Binomial Regression*, Cambridge: Cambridge University Press

Hilbe, J.M (2009), *Logistic Regression Models*, Boca Raton, FL: Chapman & Hall/CRC

Joseph M. Hilbe is a Solar System Ambassador, Jet Propulsion Laboratory, CalTech, an emeritus professor, University of Hawaii, and an adjunct professor of statistics, Arizona State University. Prof. Hilbe also teaches five courses on statistical modeling with Statistics.com.