



AgEcon SEARCH
RESEARCH IN AGRICULTURAL & APPLIED ECONOMICS

The World's Largest Open Access Agricultural & Applied Economics Digital Library

This document is discoverable and free to researchers across the globe due to the work of AgEcon Search.

Help ensure our sustainability.

Give to AgEcon Search

AgEcon Search
<http://ageconsearch.umn.edu>
aesearch@umn.edu

*Papers downloaded from **AgEcon Search** may be used for non-commercial purposes and personal study only. No other use, including posting to another Internet site, is permitted without permission from the copyright owner (not AgEcon Search), or as allowed under the provisions of Fair Use, U.S. Copyright Act, Title 17 U.S.C.*

Creating synthetic discrete-response regression models

Joseph M. Hilbe
Arizona State University
and Jet Propulsion Laboratory, CalTech
Hilbe@asu.edu

Abstract. The development and use of synthetic regression models has proven to assist statisticians in better understanding bias in data, as well as how to best interpret various statistics associated with a modeling situation. In this article, I present code that can be easily amended for the creation of synthetic binomial, count, and categorical response models. Parameters may be assigned to any number of predictors (which are shown as continuous, binary, or categorical), negative binomial heterogeneity parameters may be assigned, and the number of levels or cut points and values may be specified for ordered and unordered categorical response models. I also demonstrate how to introduce an offset into synthetic data and how to test synthetic models using Monte Carlo simulation. Finally, I introduce code for constructing a synthetic NB2-logit hurdle model.

Keywords: st0186, synthetic, pseudorandom, Monte Carlo, simulation, logistic, probit, Poisson, NB1, NB2, NB-C, hurdle, offset, ordered, multinomial

1 Introduction

Statisticians use synthetic datasets to evaluate the appropriateness of fit statistics and to determine the effect of modeling after making specific alterations to the data. Models based on synthetically created datasets 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 datasets 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 datasets for given types of binomial, Poisson, negative binomial, proportional odds, multinomial, and hurdle models using Stata's pseudorandom-number generators. I demonstrate standard models, models with an offset, and models having user-defined binary, factor, or nonrandom continuous predictors. Typically, synthetic models have predictors with values distributed as pseudorandom uniform or pseudorandom normal. This will be our paradigm case, but synthetic datasets do not have to be established in such a manner—as I demonstrate.

In 1995, Walter Linde-Zwirble and I developed several pseudorandom-number generators using Stata's programming language (Hilbe and Linde-Zwirble 1995, 1998), 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 datasets. A synthetic binomial dataset could be created, for example, having randomly generated predictors with corresponding user-specified parameters and denominators. One could also specify whether the data was to be logit, probit, or any other appropriate binomial link function.

Stata's pseudorandom-number generators are not only based on a different method from those used in the earlier `rnd*` suite of generators but also, in general, use different parameters. The examples 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 discuss creation of synthetic count response models—specifically, Poisson, log-linked negative binomial (NB2), linear negative binomial (NB1), and canonical negative binomial (NB-C) models. Second, I develop code for binomial models, which include both Bernoulli or binary models and binomial or grouped logit and probit models. Because the logic of creating and extending such models was developed in the preceding section on count models, I do not spend much time explaining how these models work. The 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 two-part models having binary and count components. 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.

2 Synthetic count models

I first create a simple Poisson model because 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 dataset) commands. Uniform random variates work as well as and at times superior to random normal variates for the creation of continuous predictors, which are used to create many of the models below. The mean of the resultant fitted value will be lower using the uniform distribution, but the model results are nevertheless identical.

(Continued on next page)

```

* SYNTHETIC POISSON DATA
* [With predictors x1 and x2, having respective parameters of 0.75 and -1.25
* and an intercept of 2]
* poi_rng.do 22Jan2009
clear
set obs 50000
set seed 4744
generate x1 = invnormal(runiform()) // normally distributed: values between
// ~ -4.5 - 4.5
generate x2 = invnormal(runiform()) // normally distributed: values between
// ~ -4.5 - 4.5
generate xb = 2 + 0.75*x1 - 1.25*x2 // linear predictor; define parameters
generate exb = exp(xb) // inverse link; define Poisson mean
generate py = rpoisson(exb) // generate random Poisson variate with mean=exb
glm py x1 x2, nolog family(poi) // model resultant data

```

The model output is given as

```

. glm py x1 x2, nolog family(poi)

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]
Log likelihood    = -119589.3262 AIC              = 4.783693
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 r-class ado-file, 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
* 9Feb2009
program poi_sim, rclass
    version 11
    drop _all
    set obs 50000
    generate x1 = invnormal(runiform())
    generate x2 = invnormal(runiform())
    generate xb = 2 + 0.75*x1 - 1.25*x2
    generate exb = exp(xb)
    generate py = rpoisson(exb)
    glm py x1 x2, nolog family(poi)
    return scalar sx1 = _b[x1]
    return scalar sx2 = _b[x2]
    return scalar sc = _b[_cons]
end

```

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

```

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

```

Variable	Obs	Mean	Std. Dev.	Min	Max
mx1	100	.7499039	.000987	.7473155	.7524396
mx2	100	-1.250145	.0009411	-1.25298	-1.248092
mcon	100	1.9999	.0015481	1.995079	2.003942

Using a greater number of repetitions will result in mean values closer to the user-specified values of 0.75, -1.25 , and 2. 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, α , may also be simulated for negative binomial models. In fact, any statistic that 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 extra dispersion; 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 good overview of overdispersion may also be found in Hardin and Hilbe (2007).

Most synthetic models use either pseudorandom uniform or normal variates for predictors. However, it is possible to create both random and fixed-level categorical predictors as well. Next I create a three-level predictor and a binary predictor to build the synthetic model. I create the categorical variables by using the `irecode()` function, with specified percentages indicated. `x1` is partitioned into three levels: `x1_1` consists

of the first 50% of the data (or approximately 25,000 observations). `x1_2` has another 30% of the data (approximately 15,000 observations), and `x1_3` has the final 10% of the data (approximately 10,000 observations). `x1_1` is the referent. `x2` is binary with approximately 30,000 zeros and 20,000 ones. The user-defined parameters are `x1_2 = 2`, `x1_3 = 3`, and `x2 = -2.5`. The intercept is specified as 1.

```
* SYNTHETIC POISSON DATA
* poif_rng.do 6Feb2009
* x1_2=2, x1_3=3, x2=-2.5, _cons=1
clear
set obs 50000
set seed 4744
generate x1 = irecode(runiform(), 0, 0.5, 0.8, 1)
generate x2 = irecode(runiform(), 0.6, 1)
tabulate x1, gen(x1_)
generate xb = 1 + 2*x1_2 + 3*x1_3 - 2.5*x2
generate exb = exp(xb)
generate py = rpoisson(exb)
glm py x1_2 x1_3 x2, nolog family(poi)
```

The model output is given as

```
. glm py x1_2 x1_3 x2, nolog family(poi)
Generalized linear models               No. of obs      =       50000
Optimization      : ML                 Residual df      =       49996
                                                Scale parameter =           1
Deviance          = 50391.75682         (1/df) Deviance =  1.007916
Pearson           = 50115.71287         (1/df) Pearson  =  1.002394
Variance function: V(u) = u            [Poisson]
Link function     : g(u) = ln(u)        [Log]
                                                AIC              =  3.959801
Log likelihood    = -98991.02229        BIC              = -490553.9
```

	OIM					
py	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
x1_2	1.995445	.0053683	371.71	0.000	1.984923	2.005966
x1_3	2.996465	.0051336	583.70	0.000	2.986404	3.006527
x2	-2.490218	.0059027	-421.88	0.000	-2.501787	-2.478649
_cons	1.00166	.0048605	206.08	0.000	.9921333	1.011186

We can obtain exact numbers of observations for each level by using the `inrange()` function. Using the same framework as above, we can amend `x1` to have exactly 25,000, 15,000, and 10,000 observations in the factored levels by using the following example code:

```
generate x1 = _n
replace x1 = inrange(_n, 1, 25000)*1 + inrange(_n, 25001, 40000)*2 + //
           inrange(_n, 40001, 50000)*3
tabulate x1, gen(x1_)
```

The tabulation output is given as

```
. tabulate x1, gen(x1_)
```

x1	Freq.	Percent	Cum.
1	25,000	50.00	50.00
2	15,000	30.00	80.00
3	10,000	20.00	100.00
Total	50,000	100.00	

Poisson models are commonly parameterized as rate models. As such, they use an offset, which reflects the area or time over which the count response is generated. Because 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 dataset. The shortcut code used to create this variable is given below. 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. The commented code better shows what is being done and can be used by those who are uncomfortable using the shortcut.

```
* SYNTHETIC RATE POISSON DATA
* poio_rng.do 22Jan2009
clear
set obs 50000
set seed 4744
generate off = 100 + 100*int((_n-1)/10000)    // creation of offset

* generate 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

generate loff = ln(off)                // log offset prior to entry into model
generate x1 = invnormal(runiform())
generate x2 = invnormal(runiform())
generate xb = 2 + 0.75*x1 - 1.25*x2 + loff    // offset added to linear predictor
generate exb = exp(xb)
generate py = rpoisson(exb)
glm py x1 x2, nolog family(poi) offset(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 used, with the exception that the estimated intercept is highly inflated.

(Continued on next page)

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

<code>. glm py x1 x2, nolog family(poi) offset(loff)</code>						
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	

py	Coef.	OIM Std. Err.	z	P> z	[95% Conf. Interval]	
x1	.7500656	.0000562	1.3e+04	0.000	.7499555	.7501758
x2	-1.250067	.0000576	-2.2e+04	0.000	-1.25018	-1.249954
_cons	1.999832	.0001009	2.0e+04	0.000	1.999635	2.00003
loff	(offset)					

I mentioned earlier that a Poisson model having a Pearson dispersion greater than 1.0 indicates possible overdispersion. The 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 NB-C model (see Hilbe [2007]). `rnbinomial()` may be used to create a synthetic 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
* nb2_rng.do 22Jan2009
clear
set obs 50000
set seed 8444
generate x1 = invnormal(runiform())
generate x2 = invnormal(runiform())
generate xb = 2 + 0.75*x1 - 1.25*x2 // same linear predictor as Poisson above
generate a = .5 // value of alpha, the NB2 heterogeneity
// parameter
generate ia = 1/a // inverse alpha
generate exb = exp(xb) // NB2 mean
generate xg = rgamma(ia, a) // generate random gamma variate given alpha
generate xbg = exb*xg // gamma variate parameterized by linear
// predictor
generate nby = rpoisson(xbg) // generate mixture of gamma and Poisson
glm nby x1 x2, family(nb ml) nolog // model as negative binomial (NB2)
```

The model output is given as

<code>. glm nby x1 x2, family(nb ml) nolog</code>					
Generalized linear models			No. of obs	=	50000
Optimization	:	ML	Residual df	=	49997
			Scale parameter	=	1
Deviance	=	54131.21274	(1/df) Deviance	=	1.082689
Pearson	=	49994.6481	(1/df) Pearson	=	.999953
Variance function: $V(u) = u + (.5011)u^2$			[Neg. Binomial]		
Link function : $g(u) = \ln(u)$			[Log]		
			AIC	=	6.148235
Log likelihood = -153702.8674			BIC	=	-486825.2

nby	Coef.	OIM Std. Err.	z	P> z	[95% Conf. Interval]	
x1	.7570565	.0038712	195.56	0.000	.749469	.764644
x2	-1.252193	.0040666	-307.92	0.000	-1.260164	-1.244223
_cons	1.993917	.0039504	504.74	0.000	1.986175	2.00166

Note: Negative binomial parameter estimated via ML and treated as fixed once

The values of the parameters and of α closely approximate the values specified in the algorithm. These values may of course be altered by the user. Note also the values of the dispersion statistics. The Pearson dispersion approximates 1.0, indicating an approximate “perfect” fit. The deviance dispersion is 8% greater, demonstrating that it is not to be used as an assessment of overdispersion. In the same manner in which a Poisson model may be Poisson overdispersed, an NB2 model may be overdispersed as well. It may, in fact, overadjust for Poisson overdispersion. Scaling standard errors or applying a robust variance estimate can be used to adjust standard errors in the case of NB2 overdispersion. See Hilbe (2007) for a discussion of NB2 overdispersion and how it compares with Poisson overdispersion.

If you desire to more critically test the negative binomial dispersion statistic, then you should use a Monte Carlo simulation routine. The NB2 negative binomial heterogeneity parameter, α , is stored in `e(a)` but must be referred to using single quotes, `'e(a)'`. Observe how the remaining statistics we wish to use in the Monte Carlo simulation program are stored.

```
* SIMULATION OF SYNTHETIC NB2 DATA
* x1=.75, x2=-1.25, _cons=2, alpha=0.5
program nb2_sim, rclass
version 11
clear
set obs 50000
generate x1 = invnormal(runiform())           // define predictors
generate x2 = invnormal(runiform())
generate xb = 2 + 0.75*x1 - 1.25*x2           // define parameter values
generate a = .5
generate ia = 1/a
generate exb = exp(xb)
generate xg = rgamma(ia, a)
generate xbg = exb*xg
generate nby = rpoisson(xbg)
```

```

glm nby x1 x2, nolog family(nb ml)
return scalar sx1 = _b[x1]           // x1
return scalar sx2 = _b[x2]           // x2
return scalar sxc = _b[_cons]         // intercept (_cons)
return scalar pd = e(dispers_p)       // Pearson dispersion
return scalar dd = e(dispers_s)       // deviance dispersion
return scalar _a = `e(a)´             // alpha
end

```

To obtain the Monte Carlo averaged statistics we desire, use the following options with the `simulate` command:

```

. simulate mx1=r(sx1) mx2=r(sx2) mxc=r(sxc) pdis=r(pd) ddis=r(dd) alpha=r(_a),
> reps(100): nb2_sim
(output omitted)
. summarize

```

Variable	Obs	Mean	Std. Dev.	Min	Max
mx1	100	.750169	.0036599	.7407614	.758591
mx2	100	-1.250081	.0037403	-1.258952	-1.240567
mx	100	2.000052	.0040703	1.987038	2.010417
pdis	100	1.000241	.0050856	.9881558	1.01285
ddis	100	1.084059	.0015233	1.079897	1.087076
alpha	100	.5001092	.0042068	.4873724	.509136

Note the range of parameter and dispersion values. The code for constructing synthetic datasets produces quite good values; i.e., the mean of the parameter estimates is very close to their respective target values, and the standard errors are tight. This is exactly what we want from an algorithm that creates synthetic data.

We may use an offset into the NB2 algorithm in the same manner as we did for the Poisson. Because the mean of the Poisson and NB2 are both $\exp(\mathbf{x}\mathbf{b})$, we may use the same method. The synthetic NB2 data and model with offset is in the `nb2o_rng.do` file.

The NB1 model is also based on a Poisson-gamma mixture distribution. The NB1 heterogeneity or ancillary parameter is typically referred to as δ , not α as with NB2. Converting the NB2 algorithm to NB1 entails defining `idelta` as the inverse of the value of delta, the desired value of the model ancillary parameter, multiplying the result by the fitted value, `exb`. The terms `idelta` and `1/idelta` are given to the `rgamma()` function. All else is the same as in the NB2 algorithm. The resultant synthetic data are modeled using Stata's `nbreg` command with the `disp(constant)` option.

```

* SYNTHETIC LINEAR NEGATIVE BINOMIAL (NB1) DATA
* nb1_rng.do 3Apr2006
* Synthetic NB1 data and model
* x1= 1.1; x2= -.8; x3= .2; _c= .7
* delta = .3 (1/.3 = 3.3333333)
quietly {
    clear
    set obs 50000
    set seed 13579
    generate x1 = invnormal(runiform())
    generate x2 = invnormal(runiform())
    generate x3 = invnormal(runiform())
    generate xb = .7 + 1.1*x1 - .8*x2 + .2*x3
    generate exb = exp(xb)
    generate idelta = 3.3333333*exb
    generate xg = rgamma(idelta, 1/idelta)
    generate xbg = exb*xg
    generate nb1y = rpoisson(xbg)
}
nbreg nb1y x1 x2 x3, nolog disp(constant)

```

The model output is given as

```

. nbreg nb1y x1 x2 x3, nolog disp(constant)
Negative binomial regression
Dispersion      = constant
Log likelihood = -89323.313
Number of obs   =      49910
LR chi2(3)      =    82361.44
Prob > chi2     =      0.0000
Pseudo R2      =      0.3156

```

nb1y	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
x1	1.098772	.0022539	487.49	0.000	1.094354	1.103189
x2	-.8001773	.0022635	-353.51	0.000	-.8046137	-.7957409
x3	.1993391	.0022535	88.46	0.000	.1949223	.2037559
_cons	.7049061	.0038147	184.79	0.000	.6974294	.7123827
/lndelta	-1.193799	.029905			-1.252411	-1.135186
delta	.3030678	.0090632			.2858147	.3213623

```

Likelihood-ratio test of delta=0:  chibar2(01) = 1763.21 Prob>=chibar2 = 0.000

```

The parameter values and value of delta closely approximate the specified values.

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

```

* SYNTHETIC CANONICAL NEGATIVE BINOMIAL (NB-C) DATA
* nbc_rng.do 30dec2005
clear
set obs 50000
set seed 7787
generate x1 = runiform()
generate x2 = runiform()

```

```

generate xb = 1.25*x1 + .1*x2 - 1.5
generate a = 1.15
generate mu = 1/((exp(-xb)-1)*a)          // inverse link function
generate p = 1/(1+a*mu)                    // probability
generate r = 1/a
generate y = rnbinomial(r, p)
cnbreg y x1 x2, nolog

```

I wrote a maximum likelihood NB-C command, **cnbreg**, in 2005, which was posted to the Statistical Software Components (SSC) site, and I posted an amendment in late February 2009. The statistical results are the same in the original and the 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

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

Also posted to the site is a heterogeneous NB-C regression command that allows parameterization of the heterogeneity parameter, α . 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 that is the same model as is the generalized negative binomial 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. I have specified values of **x1** and **x2** as 1.25 and 0.1, respectively, and an intercept value of -1.5 . α is given as 1.15. The model closely reflects the user-specified parameters.

```

. cnbreg y x1 x2, nolog
initial:      log likelihood =      -<inf>   (could not be evaluated)
feasible:     log likelihood = -85868.162
rescale:      log likelihood = -78725.374
rescale eq:   log likelihood = -71860.156

Canonical Negative Binomial Regression          Number of obs   =       50000
                                                Wald chi2(2)     =       6386.70
                                                Prob > chi2      =        0.0000

Log likelihood = -62715.384

```

y	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
x1	1.252675	.015776	79.40	0.000	1.221754	1.283595
x2	.1009038	.0091313	11.05	0.000	.0830068	.1188008
_cons	-1.504659	.0177159	-84.93	0.000	-1.539382	-1.469937
/lnalpha	.133643	.0153947	8.68	0.000	.1034699	.1638161
alpha	1.142985	.0175959			1.109012	1.177998

AIC Statistic = 2.509

3 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 pseudorandom 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, 0/1. y is binary and is considered here to be the response variable that is explained by the values of $x1$ and $x2$. Data such as this is typically modeled using a logistic regression. A probit or complementary log-log 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 1s, 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` command. I recommend using the `glm` command because `glm` is accompanied with a wide variety of test statistics and is based directly on the binomial probability distribution. Moreover, alternative linked binomial models may easily be applied.

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 models by generating data using the same parameters. First, the Bernoulli-logit model, or logistic regression:

```

* SYNTHETIC BERNOULLI-LOGIT DATA
* berl_rng.do 5Feb2009
* x1=.75, x2=-1.25, _cons=2
clear
set obs 50000
set seed 13579
generate x1 = invnormal(runiform())
generate x2 = invnormal(runiform())
generate xb = 2 + 0.75*x1 - 1.25*x2
generate exb = 1/(1+exp(-xb))           // inverse logit link
generate by = rbinomial(1, exb)         // specify m=1 in function
logit by x1 x2, nolog

```

The output is displayed as

```

. logit by x1 x2, nolog
Logistic regression               Number of obs   =       50000
                                LR chi2(2)        =    10861.44
                                Prob > chi2        =       0.0000
                                Pseudo R2         =       0.2266
Log likelihood =   -18533.1

```

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

Second, the code for constructing a synthetic binomial, or grouped, model:

```

* SYNTHETIC BINOMIAL-LOGIT DATA
* binl_rng.do 5feb2009
* x1=.75, x2=-1.25, _cons=2
clear
set obs 50000
set seed 13579
generate x1 = invnormal(runiform())
generate x2 = invnormal(runiform())
* =====
* Select either User Specified or Random denominator.
* generate d = 100 + 100*int((_n-1)/10000) // specified denominator values
generate d = ceil(10*runiform())           // integers 1-10, mean of ~5.5
* =====
generate xb = 2 + 0.75*x1 - 1.25*x2
generate exb = 1/(1+exp(-xb))
generate by = rbinomial(d, exb)
glm by x1 x2, nolog family(bin d)

```

The final line calculates and displays the output below:

. glm by x1 x2, nolog family(bin d)						
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
<hr/>						
by	Coef.	OIM 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. Displayed is code for generating a random denominator and code for specifying the same values as were earlier used for the Poisson and negative binomial offsets.

See Cameron and Trivedi (2009) for 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, written after the do-files that are presented here were developed.

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 log-log data. I show the probit because it is frequently used in econometrics.

```
* SYNTHETIC BINOMIAL-PROBIT DATA
* binp_rng.do 5feb2009
* x1=.75, x2=-1.25, _cons=2
clear
set obs 50000
set seed 4744
generate x1 = runiform()           // use runiform() with probit data
generate x2 = runiform()
* =====
* Select User Specified or Random Denominator. Select Only One
* generate d = 100+100*int((_n-1)/10000) // specified denominator values
generate d = ceil(10*runiform())      // pseudorandom-denominator values
* =====
generate xb = 2 + 0.75*x1 - 1.25*x2
generate double exb = normal(xb)
generate double by = rbinomial(d, exb)
glm by x1 x2, nolog family(bin d) link(probit)
```

The model output is given as

<code>. glm by x1 x2, nolog family(bin d) link(probit)</code>						
Generalized linear models			No. of obs	=	50000	
Optimization : ML			Residual df	=	49997	
			Scale parameter	=	1	
Deviance	=	35161.17862	(1/df) Deviance	=	.7032658	
Pearson	=	50277.67366	(1/df) Pearson	=	1.005614	
Variance function: V(u) = u*(1-u/d)			[Binomial]			
Link function : g(u) = invnorm(u/d)			[Probit]			
			AIC	=	1.132792	
Log likelihood = -28316.80908			BIC	=	-505795.3	

by	Coef.	OIM Std. Err.	z	P> z	[95% Conf. Interval]	
x1	.7467577	.0148369	50.33	0.000	.717678	.7758374
x2	-1.247248	.0157429	-79.23	0.000	-1.278103	-1.216392
_cons	2.003984	.0122115	164.11	0.000	1.98005	2.027918

The `normal()` function is the inverse probit link and replaces the inverse logit link.

4 Synthetic categorical response models

I have previously discussed the creation of synthetic ordered logit, or proportional odds, data in Hilbe (2009), and I refer you to that source for a more thorough examination of the subject. I also examine multinomial logit data in the same source. Because of the complexity of the model, the generated data are 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
* oprobit_rng.do 19Feb 2008
display in ye "b1 = .75; b2 = 1.25"
display in ye "Cut1=2; Cut2=3,; Cut3=4"
quietly {
    drop _all
    set obs 50000
    set seed 12345
    generate double x1 = 3*runiform() + 1
    generate double x2 = 2*runiform() - 1
    generate double y = .75*x1 + 1.25*x2 + invnormal(runiform())
    generate 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 (oppr1 oppr2 oppr3 oppr4), pr
```

The modeled data appears as

```
. oprobit ys x1 x2, nolog
Ordered probit regression
```

Number of obs	=	50000
LR chi2(2)	=	24276.71
Prob > chi2	=	0.0000
Pseudo R2	=	0.2127

```
Log likelihood = -44938.779
```

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 are 0.75 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 hundredths place.

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

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

Finally, synthetic ordered slope models may easily be expanded to having more predictors as well as additional levels by using the same logic as shown in the above algorithm. Given three predictors with values assigned as $x_1 = 0.5$, $x_2 = 1.76$, and $x_3 = 1.25$, and given five levels with cuts at 0.8, 1.6, 2.4, and 3.2, the amended part of the code is as follows:

```
generate double x3 = runiform()
generate y = .5*x1 + 1.75*x2 - 1.25*x3 + invnormal(uniform())
generate int ys = 1 if y<=.8
replace ys=2 if y<=1.6 & y>.8
replace ys=3 if y<=2.4 & y>1.6
replace ys=4 if y<=3.2 & y>2.4
replace ys=5 if y>3.2
oprobit ys x1 x2 x3, nolog
```

(Continued on next page)

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

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

I have amended the `uniform()` function in the original code to `runiform()`, which is Stata's newest version of the pseudorandom-uniform generator. Given the nature of the multinomial probability function, the above code is rather self-explanatory. The code may easily be expanded to have more than three levels. New coefficients need to be defined and the probability levels expanded. See Hilbe (2009) for advice on expanding the code. The output of the above `mlogit_rng.do` is displayed as

```
. mlogit y x1 x2, baseoutcome(1) nolog
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]	
1	(base outcome)					
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	.9389566	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

By amending the `mlogit_rng.do` code to an r-class ado-file, with the following lines added to the end, the following Monte Carlo simulation may be run, verifying the parameters displayed from the do-file:

```
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 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
(output omitted)
. summarize
```

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

The user-specified values are reproduced by the synthetic multinomial program.

5 Synthetic hurdle models

Finally, I show an example of how to 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 several hurdle models, which are currently on the SSC web site. This example is shown 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 and Hilbe (2003) for generalized estimating equations models.

Hurdle models are discussed in Long and Freese (2006), Hilbe (2007), Winkelmann (2008), and Cameron and Trivedi (2009). The traditional method of parameterizing hurdle models is to have both binary and count components be of equal length, which makes theoretical sense. However, they may be of unequal lengths, as are zero-inflated models. Moreover, hurdle models can be used to estimate both over- and underdispersed count data, unlike zero-inflated models.

The binary component of a hurdle model is typically a logit, probit, or cloglog binary response model. However, the binary component may take the form of a right-censored Poisson model or a censored negative binomial model. In fact, the earliest applications

of hurdle models consisted of Poisson–Poisson and Poisson-geometric models. However, it was discovered that the censored geometric component has an identical log likelihood to that of the logit, which has been preferred in most recent applications. I published censored Poisson and negative binomial models to the SSC web site in 2005, and truncated and econometric censored Poisson models in 2009. They may be used for constructing this type of hurdle model.

The synthetic hurdle model below is perhaps the most commonly used version—a NB2-logit hurdle model. It is a combination of a 0/1 binary logit model and a zero-truncated NB2 model. For the logit portion, all counts greater than 0 are given the value of 1. There is no estimation overlap in response values, as is the case for zero-inflated models.

The parameters specified in the example synthetic hurdle model below are

```
* SYNTHETIC NB2-LOGIT HURDLE DATA
* nb2logit_hurdle.do  J Hilbe  26Sep2005; Mod 4Feb2009.
* LOGIT: x1=-.9, x2=-.1, _c=-.2
* NB2   : x1=.75, n2=-1.25, _c=2, alpha=.5
clear
set obs 50000
set seed 1000
generate x1 = invnormal(runiform())
generate x2 = invnormal(runiform())
* NEGATIVE BINOMIAL- NB2
generate xb = 2 + 0.75*x1 - 1.25*x2
generate a = .5
generate ia = 1/a
generate exb = exp(xb)
generate xg = rgamma(ia, a)
generate xbg = exb*xg
generate nby = rpoisson(xbg)
* BERNOULLI
drop if nby==0
generate pi = 1/(1+exp(-(.9*x1 + .1*x2 + .2)))
generate 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

<code>. hnblogit y x1 x2, nolog</code>						
Negative Binomial-Logit Hurdle Regression			Number of obs	=	43443	
Log likelihood = -84654.938			Wald chi2(2)	=	5374.14	
			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
AIC Statistic = 3.897						

The results approximate the specified values. A Monte Carlo simulation was performed, demonstrating that the algorithm does what it is aimed to do.

6 Summary remarks

Synthetic data can be used with substantial efficacy for the evaluation of statistical models. In this article, I have presented algorithmic code that can be used to create several 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. I have used these models, or ones based on earlier random-number generators, in Hardin and Hilbe (2007) and in both of my single authored texts (Hilbe 2007, 2009) for assessing model assumptions. With computers gaining in memory and speed, it will soon be possible to construct far more complex synthetic data than we have here. I hope that the rather elementary examples discussed in this article will encourage further use and construction of artificial data.

7 References

- Cameron, A. C., and P. K. Trivedi. 2009. *Microeconometrics Using Stata*. College Station, TX: Stata Press.
- Hardin, J. W., and J. M. Hilbe. 2003. *Generalized Estimating Equations*. Boca Raton, FL: Chapman & Hall/CRC.
- . 2007. *Generalized Linear Models and Extensions*. 2nd ed. College Station, TX: Stata Press.

- Hilbe, J. M. 2007. *Negative Binomial Regression*. New York: Cambridge University Press.
- . 2009. *Logistic Regression Models*. Boca Raton, FL: Chapman & Hall/CRC.
- Hilbe, J. M., and W. Linde-Zwirble. 1995. sg44: Random number generators. *Stata Technical Bulletin* 28: 20–21. Reprinted in *Stata Technical Bulletin Reprints*, vol. 5, pp. 118–121. College Station, TX: Stata Press.
- . 1998. sg44.1: Correction to random number generators. *Stata Technical Bulletin* 41: 23. Reprinted in *Stata Technical Bulletin Reprints*, vol. 7, p. 166. College Station, TX: Stata Press.
- Long, J. S., and J. Freese. 2006. *Regression Models for Categorical Dependent Variables Using Stata*. 2nd ed. College Station, TX: Stata Press.
- Winkelmann, R. 2008. *Econometric Analysis of Count Data*. 5th ed. Berlin: Springer.

About the author

Joseph M. Hilbe is an emeritus professor (University of Hawaii), an adjunct professor of statistics at Arizona State University, and Solar System Ambassador with the NASA/Jet Propulsion Laboratory, CalTech. He has authored several texts on statistical modeling, two of which are *Logistic Regression Models* (Chapman & Hall/CRC) and *Negative Binomial Regression* (Cambridge University Press). Hilbe is also chair of the ISI Astrostatistics Committee and Network and was the first editor of the *Stata Technical Bulletin* (1991–1993).