The Stata Journal

The *Stata Journal*, electronic version (ISSN 1536-8734) is a publication of Stata Press, and Stata is a registered trademark of StataCorp LP.

The Stata Journal (2006)
6, Number 2, pp. 156–189

Calculation of multivariate normal probabilities by simulation, with applications to maximum simulated likelihood estimation

Lorenzo Cappellari
Catholic University of Milan
Milan, Italy
and University of Essex
Colchester, UK
lorenzo.cappellari@unicatt.it

Stephen P. Jenkins
University of Essex
Colchester, UK
stephenj@essex.ac.uk

Abstract. We discuss methods for calculating multivariate normal probabilities by simulation and two new Stata programs for this purpose: `mdraws` for deriving draws from the standard uniform density using either Halton or pseudorandom sequences, and an `egen` function, `mvnp()`, for calculating the probabilities themselves. Several illustrations show how the programs may be used for maximum simulated likelihood estimation.

Keywords: st0101, mdraws, egen function mvnp( ), simulation estimation, maximum simulated likelihood, multivariate probit, Halton sequences, pseudorandom sequences, multivariate normal, GHK simulator

1 Introduction

This article discusses the program `mdraws`, which produces pseudorandom or Halton draws, and the `egen` function `mvnp()`, which uses the Geweke–Hajivassiliou–Keane (GHK) smooth recursive conditioning simulator to calculate multivariate normal probabilities and shows how they may be used for maximum simulated likelihood (MSL) estimation. The article is a development of our research on estimation of multivariate probit models (Cappellari and Jenkins 2003, 2005, 2006). In the earlier work, we noted that estimation of these models required evaluation of multivariate normal probability distribution functions, but functions to evaluate trivariate and higher dimensional normal distributions did not exist in Stata at the time. In the `mvprobit` program accompanying our 2003 article (updated in this issue), we used the GHK simulator to do these evaluations, using pseudorandom draws. Because the GHK simulator can be used in many other contexts, we were motivated to write stand-alone programs that could be applied more generally. The programs `mdraws` and `mvnp()` for Stata 8.2 or higher can be used in many MSL estimation applications.

Because simulation estimation is computationally intensive, we sought to reduce computing time. This reduction has been done in two ways. First, `mvnp()` has been implemented as a Stata plugin, with an option to use ado-code. As we show below, the plugin leads to substantial gains in speed. Second, `mdraws` lets users create draws

variables based on Halton sequences for use by `mvnp()` or, indeed, by other programs. (The program can also create variables based on pseudorandom uniform variates, as used by `mvprobit`.) Train (2003) argued that Halton draws are more effective for MSL estimation than pseudorandom draws because they can provide the same accuracy with fewer draws, thereby saving computer time. However, most evidence to date about this conclusion has been based on estimation of mixed logit models rather than of multivariate probit models of the types that we consider in our illustrations.

Our discussion is nontechnical and didactic. For more extensive discussion of the principles underlying MSL estimation, the GHK simulator, and drawing from densities, see Greene (2003, 931–933), Gouriéroux and Monfont (1996), and especially Train (2003), on whom we rely heavily. Also see our earlier article (Cappellari and Jenkins 2003, 2005, 2006) and the other articles in this issue. Section 2 discusses `mdraws` and Halton sequences in particular. Section 3 focuses on `mvnp()` and provides examples of its use. Section 4 concludes.

2 Multiple draws from the standard uniform density

MSL works by simulating likelihoods and then averaging over these. These calculations involve expressions that contain a multivariate density—a multivariate normal density here—and so to do the simulation, one needs to take draws from this density. The GHK simulator works by taking draws from upper-truncated univariate standard normal distributions and then recursively computing a multivariate probability value using Cholesky factorization. The draws are derived by taking values from a density that is uniform over the interval $[0, 1)$, the so-called standard uniform density. The upper-truncated standard normal distribution values are generated by inversion of the normal probability function combined with an inversion formula given by, among others, Stern (1997). The key initial step, then, is taking draws from the standard uniform density.

The most common method of generating these draws has been to use a pseudorandom number generator. In Stata, this means creating variables by using the `uniform()` function. The method is straightforward and the independence of the draws facilitates derivation of the statistical properties of the MSL estimator. On the other hand, "there are ways to take draws that can provide greater accuracy for a given number of draws" (Train 2003, 217). Train (2003, chap. 9) emphasizes that coverage and covariance are the two important criteria for assessing these methods.

With pseudorandom draws, the values may clump together in particular regions of the domain of the density that we wish to integrate, which may lead to a poor approximation of the integral. And, because the draws are independent, the covariance across draws for each observation is zero. A negative covariance across draws is better, because this reduces the variance of the simulation compared to the independent draws case: a relatively high value is balanced by a low value. The negative covariance can also lead to greater coverage. Antithetic draws are the most commonly used method for this type of variance reduction, and creation of the most straightforward type is provided as an option in `mdraws`. For each vector of draws, z, from the standard uniform distribution, the antithetic draw is $1 - z$.

A second type of covariance is the covariance of the draws across observations, which is zero when independent pseudorandom draws are used but, again, a negative correlation is more desirable. If the average of the draws for one observation is more than 1/2, we want the average for another observation to be less than 1/2. As Train (2003, 218) explains, the maximand in MSL is a sum across observations of the logs of the simulated probabilities. If draws across observations are negatively correlated, then the variance of the MSL maximand is smaller than the variance with independent pseudorandom draws (the sum of the variances for each observation).

Draws derived from Halton sequences both improve coverage of the domain of integration and induce a negative correlation between the draws from different observations. Each sequence is defined uniquely by a particular prime number, P. Sequence elements are characterized by an iterative process comprising a series of successive rounds. In the first round, the unit interval (the domain of the standard uniform density) is split into P equal-width segments, and P sequence elements with values equal to the $P-1$ segment cutpoints are defined. (If $P = 3$, the values are 1/3 and 2/3.) In the second round, each segment created in the first round is split into P new segments (nine segments if $P = 3$). Then there is a systematic cycling across the segments. In each cycle, P sequence elements are picked, and the cycling continues until all the relevant segments have been exhausted. (If $P = 3$, the values are, in order, the cutpoints from segments #1, #4, and #7 and then the cutpoints from segments #2, #5, and #8.) In the next round, each segment is split P ways again, and the systematic cycling rule is used again, and the rounds and within-round cycles continue for as long as one needs sequence elements. As the number of rounds increases, the unit interval is filled in more and more by sequence elements.

If $P = 3$, the first elements of the Halton sequence are 1/3, 2/3 (from the initial round), 1/9, 4/9, 7/9, 2/9, 5/9, 8/9 (from the second round), 1/27, 10/27, 19/27, 4/27, 13/27, 22/27, 7/27, 16/27, 25/27, 2/27, 11/27, 20/27 (from the third round), and so on. The sequence elements for any other prime are defined similarly. For example, if $P = 2$, the initial elements are 1/2 (from the first round), 1/4, 3/4 (from the second round), 1/8, 5/8, 3/8, 7/8 (from the third round), and so on.

The procedure outlined above defines a long sequence of numbers from the unit interval. For MSL estimation using data on a sample of observations, we need a set of draws for each observation, with one draw variable used in each simulation. One allocates elements to observations in bunches. With five draws, for example, the first five elements of the sequence are allocated to the first observation, the second five elements are allocated to the second observation, and so on. (This allocation is often done after discarding some initial elements of the original sequence; see below.)

Where integration is over multiple dimensions, as with calculation of multivariate normal probabilities, one Halton sequence is created for each dimension by using a separate prime, and again values are allocated to observations in bunches. Using the first M primes for an M-dimensional calculation is conventional, which is what the multinomial probit program `asmprobit` does, for example. Multiple Halton sequences generally provide better multidimensional coverage than the corresponding pseudorandom sequences,

but issues remain concerning the correlation between sequences for different primes. The initial elements of any two sequences can be highly correlated, at least during the first cycle over the unit interval (before the different cycle periods for the two primes take effect). Therefore, dropping the initial elements of each Halton sequence before allocating the elements to observations is common. Little guidance about the optimal number of elements to drop is available, except that the number of elements to drop should be greater than the largest prime used to create the sequences (Train 2003, 230). The `burn()` option in the `mdraws` command lets users choose how many elements to drop.

Also several authors have pointed to problems of high correlation between the sequences constructed using relatively large primes, i.e., when the number of dimensions is relatively high, and thence poorer multidimensional coverage. Hess and Polak (2003) and Hess, Polak, and Daly (2003) discuss the problem and argue in favor of "shuffled" Halton sequences rather than the "scrambled" Halton sequences that other researchers have suggested. The `mdraws` command offers the option of shuffled Halton sequences, but their properties are not yet well known, especially when used for lower-dimensional problems or for MSL applications other than mixed logit model estimation.

`mdraws` creates $M \times D$ new variables, where each variable contains numbers drawn from the standard uniform density. M is the number of equations (integration dimensions), and D is the number of draws created per observation. The numbers are either Halton sequences (the default) or sequences of pseudorandom numbers. The names of the variables created have a common prefix specified by the option `prefix(`*string*`)`, and the variable name suffixes are m_d for each integer $m = 1, \ldots, M$ and each integer $d = 1, \ldots, D$. Users may need to `set matsize` and `set memory` to values higher than the default to allow enough space for the new variables. Also `mdraws` uses temporary files and will not work if there is not enough free hard disk space for temporary file storage. In this situation (rare in our experience), users should check the path specified by their computer's `TEMP` environment variable, and either create space on the relevant disk or change `TEMP` to a location where there is space (e.g., a different disk).

For Halton sequences, users can specify M prime numbers of their own choosing using the `primes(`*matrix_name*`)` option. If the option is not specified, `mdraws` uses the first M prime numbers in ascending order.

2.1 Syntax for mdraws

`mdraws` [*if*] [*in*]`,` `draws(`#`)` `neq(`#`)` `prefix(`*string*`)` [`primes(`*name*`)`
`antithetics` `burn(`#`)` `random` `seed(`#`)` `hrandom` `shuffle` `replace`]

2.2 Options for mdraws

`draws(`#`)` specifies the number of draws. If the `antithetics` option is not chosen, the total number of draw variables created for each integration dimension is $D = D^*$, where D^* is the number specified in `draws()`. If the `antithetics` option is chosen, $D = 2D^*$.

`neq(#)` specifies M, the number of equations (dimensions of integration).

`prefix(`*string*`)` specifies the prefix common to the names of each of the draws variables created.

`primes(`*name*`)` specifies the name of an existing $1 \times M$ or $M \times 1$ matrix containing M different prime numbers. If the option is not specified and as long as $M \leq 20$, the program uses the first M prime numbers in ascending order.

`antithetics` specifies that antithetic draws also be created. The antithetic draw for a vector of draws, z, is $1 - z$. The variables are named in a manner consistent with the system outlined above. The first D^* variables per dimension are the original draws variables; the second D^* variables are the corresponding antithetic draws.

`burn(#)` specifies the number of initial sequence elements to drop for each equation when creating Halton sequences. The default is zero, and the option is ignored if the `random` option is specified. Train (2003, 230) recommends that # should be at least as large as the largest prime number used to generate the sequences.

`random` specifies that pseudorandom number sequences are created rather than Halton sequences (the default).

`seed(#)` specifies the initial value of the pseudorandom number seed used by the `uniform()` function if `random` is specified or if the `hrandom` or `shuffle` options are requested when Halton sequences are specified. Otherwise, it is ignored. The value should be an integer (the default value is 123456789). Use this option to ensure reproducibility of results.

`hrandom` specifies that each Halton sequence should be transformed by a random perturbation. For each dimension, a draw, u, is taken from the standard uniform distribution. Each sequence element for that dimension has u added to it. If the sum is greater than 1, the element is transformed to the sum minus 1; otherwise, the element is transformed to the sum. See Train (2003, 234).

`shuffle` specifies that "shuffled" Halton draws should be derived, as proposed by Hess and Polak (2003) and Hess, Polak, and Daly (2003). The Halton sequence for each dimension is randomly shuffled before sequence elements are allocated to observations. Philippe Van Kerm's program `_gclsort`, available from SSC, must be installed for this option to work.

`replace` specifies that existing variables named using the prefix specified by `prefix()` and the suffix defined by the relevant equation and draw number be replaced.

2.3 Saved results

`mdraws` saves the following in `r()`:

Scalars

`r(n_draws)`	scalar equal to D
`r(n_dimensions)`	scalar equal to M
`r(n_burn)`	number of draws dropped per dimension, if `burn()` option used

Macros

`r(antithetics)`	local macro containing `"yes"` if `antithetics` option specified, else containing `"no"`
`r(prefix)`	local macro containing the string specified by the `prefix()` option
`r(seed)`	local macro containing `c(seed)`, if `seed()` option used
`r(type)`	local macro containing `"halton"` if Halton draws created, else contains `"random"`

Matrices

`r(primes)`	primes used in creating a Halton draw, if `primes()` option used

2.4 Examples

We begin by reproducing the Halton draws example given by Train (2003, 227). He has two observations, $D = 5$, $P = 3$, and the first nine elements of the sequence are dropped. (Train refers to dropping 10 initial elements because he uses zero as the first element when illustrating how to construct the sequence.) `z` is the draw variable name prefix, and the command is

```
. set obs 2
obs was 0, now 2
. matrix p = (3)
. mdraws, draws(5) neq(1) prefix(z) burn(9) primes(p)
Created 5 Halton draws per equation for 1 dimensions. Number of initial
draws dropped per dimension = 9 . Primes used:
  3
. list
```

	z1_1	z1_2	z1_3	z1_4	z1_5
1.	.37037037	.7037037	.14814815	.48148148	.81481481
2.	.25925926	.59259259	.92592593	.07407407	.40740741

The values shown in the five draws variables for observation 1 are 10/27, 19/27, 4/27, 13/27, and 22/27; those in the corresponding variables for observation 2 are 7/27, 16/27, 25/27, 2/27, and 11/27 (see earlier). MSL for this sample would be based, at each iteration, on simulation of the sample likelihood using each of the five `z` variables in turn and then averaging the result across simulations.

Suppose that we now have a more realistic sample size, 1,000 observations, and three integration dimensions and require 100 draws. For pseudorandom draws, the command syntax is

```
. set obs 1000
obs was 0, now 1000
. mdraws, draws(100) neq(3) prefix(a) random
Created 100 pseudorandom draws per equation for 3 equations. Seed = 123456789
```

For 50 Halton draws plus antithetic draws, using primes 7, 11, and 13, and dropping the first 20 sequence elements in each dimension, the syntax is

```
. matrix p1 = (7, 11, 13)
. mdraws, draws(50) neq(3) prefix(b) burn(20) antithetics primes(p1)
Created 50 Halton draws per equation for 3 dimensions. Number of initial draws
dropped per dimension = 20 . Primes used:
  7 11 13
Also created 50 antithetic draws per dimension for 3 dimensions.  Note: there
are now 100 draws per equation
```

Observe in the last case that $D^* = 50$ (the number specified in the `draws()` option) but, with `antithetics` also specified, $D = 100$.

```
. return list
scalars:
           r(n_draws) =  100
      r(n_dimensions) =  3
            r(n_burn) =  20
macros:
       r(antithetics) : "yes"
            r(prefix) : "b"
              r(type) : "halton"
matrices:
            r(primes) :  1 x 3
```

3 An egen function for computing multivariate normal probabilities, mvnp()

Our `egen` function `mvnp()` calculates multivariate normal probabilities with the GHK simulator and returns the results in a new variable with storage type `double`. More specifically, the function returns the multivariate normal probability

$$\Pr\left(-\infty < x_m \leq a_m, m = 1, \ldots, M\right)$$

where the M variables x_m each have mean zero and covariance matrix V. For computational reasons, users specify not V but the lower triangular matrix C that is the Cholesky factorization of V : $C =$ `cholesky(`V`)`. From the MSL estimate of C, one can recover an estimate of V since $V = CC'$. We show how to do this using `nlcom` later. (Instead, one could use `_diparm`.)

Put another way, `mvnp()` returns the joint cumulative distribution $\Phi(a_1, a_2, \ldots, a_M; V)$ of an M-variate normal distribution with covariance matrix V, where the cumulation is over $(-\infty, a_1] \times (-\infty, a_2] \times \cdots \times (-\infty, a_M]$. The upper integration points $a_1, a_2, \ldots, a_M$ are variables specified by the user and may of course vary across observations. If the

mean of any of the x_m variables is nonzero, the upper integration points should be appropriately centered first. For MSL estimation of multivariate probit–type models, this step is typically unnecessary.

The function assumes the existence of $M \times D$ variables containing draws from the standard uniform distribution. The names of the variables must have a common prefix, specified by the option `prefix(`*string*`)`, and the variable name suffixes are m_d for each integer $m = 1, \ldots, M$ and each integer $d = 1, \ldots, D$. The variables can be created with `mdraws`.

The MSL estimator is consistent, asymptotically normal and efficient, and equivalent to ML if the number of draws tends to infinity faster than the square root of the number of observations does (Train 2003, 259). When $M = 2$, and for a large number of random draws, then calculation by `mvnp()` is asymptotically equivalent to that provided by the built-in function `binormal()`. Other things being equal, the more draws, the better. In practice, relatively few draws may work well in the sense that the change in calculated probabilities as the number of draws is increased is negligible. The user must check that this is the case.

Calculation is numerically intensive and may be slow if the number of observations is large, if D is large, or especially if M is large.

Next we introduce the syntax for the `egen` function `mvnp()` and then illustrate the function. The first example shows how the function can be used for stand-alone one-off calculations. The remaining three examples illustrate how the function may be used for MSL estimation—essentially showing how to embed calls to `mvnp()` within code for likelihood function evaluation by `ml`.

3.1 Syntax for mvnp()

`egen` *newvar* `= mvnp(`*varlist*$_1$`)` [*if*] [*in*]`, prefix(`*string*`) draws(`#`)`
[`chol(`*matrix_name*`) signs(`*varlist*$_2$`) adoonly`]

where *varlist*$_1$ refers to a list of existing variables containing upper integration points. The variable names should be separated by spaces, not commas.

3.2 Options

`prefix(`*string*`)` specifies the prefix common to each of the variables representing draws from a standard uniform density.

`draws(`#`)` specifies the number of draws used when calculating the simulated probability. The default is `draws(5)`. (See the discussion above concerning the choice of D.)

`chol(`*matrix_name*`)` specifies the lower triangular matrix that is the Cholesky factorization of the covariance matrix, V, i.e., matrix *matrix_name* = `cholesky(`V`)`. At least one of the diagonal elements of matrix *matrix_name* should equal 1. The nature of any further constraints on matrix *matrix_name* depends on the application (see the examples below). The user must ensure that the appropriate constraints are imposed.

`signs(`*varlist*$_2$`)` may be used if the function is used to evaluate multivariate probit-like likelihood functions, and it helps reduce computation time. For an ordered set of binary dependent variables $i = 1, \ldots, M$, *varlist*$_2$ contains the names of a set of variables summarizing the sign of each dependent variable. Specifically, the ith variable of *varlist*$_2$ should contain 1 for an observation with the corresponding dependent variable equal to 1 and contain -1 for an observation with the corresponding dependent variable equal to 0.

`adoonly` prevents using the Stata plugin to perform the intensive numerical calculations. Specifying this option results in slower-running code but may be necessary if the plugin is not available for your platform.

3.3 Illustration 1: one-off calculations of multivariate normal probabilities

`mvnp()` may be used to calculate multivariate probabilities in any situation in which one has a set of upper integration points and a variance matrix or, rather, the Cholesky matrix derived from the variance matrix. That is, MSL is not the only application. Let us illustrate how the program can be used in a stand-alone context. Although the first example is artificial, it demonstrates the relevant principles and compares the calculated probabilities with those generated using the built-in function `binormal()`. This example creates more than 4,600 variables and thus requires either Stata/SE or Stata/MP.

Consider probabilities from a standard bivariate normal distribution with correlation $\rho = 0.5$. By assumption, the means of the two variables are zero. The correlation matrix and its Cholesky matrix are created as follows:

```
. clear
. matrix r = (1, .5 \ .5, 1)
. matrix c = cholesky(r)
```

Now suppose that we have a set of upper integration points for each of 1,000 observations held in variables `v1` and `v2` and created for illustrative purposes in the following way:

```
. set obs 1000
. gen v1 = uniform()
. gen v2 = uniform()
```

We will compare calculations based on 50 and 1,000 pseudorandom draws and 100 Halton draws both with and without antithetic draws, creating six sets of draw variables using `mdraws`.

```
. * without antithetics
. mdraws, neq(2) dr(50) prefix(p) random seed(123456789)
Created 50 pseudorandom draws per equation for 2 equations. Seed = 123456789
. mdraws, neq(2) dr(1000) prefix(q) random seed(123456789)
Created 1000 pseudorandom draws per equation for 2 equations. Seed = 123456789
. mdraws, neq(2) dr(100) prefix(h) burn(10)
Created 100 Halton draws per equation for 2 dimensions. Number of initial draws
dropped per dimension = 10 . Primes used:
  2  3
. * with antithetics
. mdraws, neq(2) dr(25) prefix(pa) random seed(123456789) antithetics
Created 25 pseudorandom draws per equation for 2 equations. Seed = 123456789
Also created 25 antithetic draws per dimension for 2 dimensions.  Note: there
are now 50 draws per equation
. mdraws, neq(2) dr(500) prefix(qa) random seed(123456789) antithetics
Created 500 pseudorandom draws per equation for 2 equations. Seed = 123456789
Also created 500 antithetic draws per dimension for 2 dimensions.  Note: there
are now 1000 draws per equation
. mdraws, neq(2) dr(50) prefix(ha) burn(10)
Created 50 Halton draws per equation for 2 dimensions. Number of initial draws
dropped per dimension = 10 . Primes used:
  2  3
```

Now compute the probabilities using `binormal()` and `mvnp()` with and without antithetic draws, and then `summarize` the probabilities to compare them.

```
. * built-in
. gen pr_b = binormal(v1,v2,.5)
. * egen function with plugin
. egen pr_s1p = mvnp(v1 v2), dr(50) chol(c) prefix(p)
. egen pr_s1q = mvnp(v1 v2), dr(1000) chol(c) prefix(q)
. egen pr_s1h = mvnp(v1 v2), dr(100) chol(c) prefix(h)
. * with antithetics
. egen pr_s1pa = mvnp(v1 v2), dr(25) chol(c) prefix(pa)
. egen pr_s1qa = mvnp(v1 v2), dr(500) chol(c) prefix(qa)
. egen pr_s1ha = mvnp(v1 v2), dr(50) chol(c) prefix(ha)
. summarize pr_b pr_s*
```

Variable	Obs	Mean	Std. Dev.	Min	Max
pr_b	1000	.5300376	.0880073	.3459212	.7432851
pr_s1p	1000	.5293781	.0889788	.3321594	.7432417
pr_s1q	1000	.5300343	.0878787	.3447233	.7420039
pr_s1h	1000	.5300534	.0879827	.3452459	.7424432
pr_s1pa	1000	.5292217	.0891549	.3220528	.7490329
pr_s1qa	1000	.529947	.087786	.3442015	.7413688
pr_s1ha	1000	.5300763	.0880055	.3450753	.7425911

The simulated probabilities have a similar distribution to those calculated using `binormal()` regardless of the number of draws and whether antithetic draws are used. Nonetheless and unsurprisingly, the mean probability based on 1,000 pseudorandom uniform draws is markedly closer than the mean probability based on 50 pseudorandom draws to the mean probability based on `binormal()`. The mean based on 100 Halton draws, with or without antithetics, gets even closer.

The second, and perhaps more useful, application of one-off calculations is generation of predicted probabilities of multiple outcome variables after estimation of multivariate probit (and related) models. Our postestimation command `mvppred` generates predicted probabilities from multivariate probit estimates derived using `mvprobit`, but only for the probability that every observed outcome variable equals one and that every observed outcome variable equals zero. (Updated versions of these commands are available with this issue.) With `mvnp()`, the predicted probability of any combination of ones and zeros can be derived.

The multivariate probit model is characterized, for each observation, by M pairs of equations, one describing each latent dependent variable and the other describing the corresponding binary observed outcome.

$y^*_m = \beta'_m X_m + \epsilon_m, m = 1, \ldots, M$

$y_m = 1$ if $y^*_m < 0$, and 0 otherwise

$\epsilon_m, m = 1, \ldots, M$, are error terms distributed as multivariate normal, each with a mean of zero, and variance–covariance matrix V, where V has values of 1 on the leading diagonal and correlations $\rho_{jk} = \rho_{kj}$ as off-diagonal elements for $j, k = 1, \ldots, M$ and $j \neq k$.

The predicted probability of the observed outcomes for any observation is $\Phi_M(\mu; \Omega)$, where $\Phi_M(.)$ is the M-variate standard normal cumulative distribution function with arguments μ_i and Ω, and $\mu = (\kappa_1 \beta'_1 X_1, \kappa_2 \beta'_2 X_2, \ldots, \kappa_M \beta'_M X_M)$. The κ_k are signs variables, being equal to 1 or -1 depending on whether the observed binary outcome equals 1 or 0: $\kappa_k = 2y_k - 1$ for each observation for $k = 1, \ldots, M$. Matrix Ω has constituent elements Ω_{jk}, where $\Omega_{jj} = 1$ for $j = 1, \ldots, M$, and $\Omega_{jk} = \Omega_{kj} = \kappa_k \kappa_j \rho_{kj}$.

Estimates of β_m and V can be derived using the `egen`-based code shown in the next illustration or with `mvprobit`. To calculate the predicted probabilities using `mvnp()`, one first generates several new variables. There are the M-signs variables appropriate to the outcome combinations of interest; these will be the arguments specified in the `signs()` option. The upper integration point variables are the M linear index variables $I_m = b'_m X_m$, which can be derived by using `matrix score` or `mvppred, xb` after `mvprobit` (b_m is the estimate of β_m). Next use `mdraws` to generate the draws variables and, finally, calculate the probabilities by using the `egen` command with the I_m variables as arguments and referring to a matrix equal to `cholesky(`V`)` in the `chol()` option. You can save much computer time if the probability calculations are not made for every observation. Instead create a few new observations that have the specific values for

$X_1, X_2, \ldots, X_M$ that are of interest, and generate the linear index variable values for them by using Stata's ability to generate out-of-sample predictions. Then calculate the probabilities only for these observations: `mvnp()` accepts `if` and `in` qualifiers.

3.4 Illustration 2: MSL estimation of multivariate probit models

Now we show how our command may be used to fit multivariate probit models. The main advantage of using `mvnp()` rather than `mvprobit` is that you can save much computational time by taking advantage of the plugin and Halton draws. These savings may be particularly valuable when the number of outcome variables is large (four or more, say) and the number of observations is large.

The illustration considers the trivariate probit model. (It is straightforward to generalize the likelihood evaluation code below to fit multivariate probit models with more equations.) To benchmark the estimates, we create a dataset with 5,000 observations from a model with known parameters, using the same methods as in our previous work, Cappellari and Jenkins (2003, 2005, 2006):

```
. set seed 123456789
. set obs 5000
obs was 0, now 5000
. matrix R = (1, .25, .5 \ .25, 1, .75 \ .5, .75, 1)
. drawnorm u1 u2 u3, corr(R)
. correlate u*
(obs=5000)
             |       u1       u2       u3
-------------+---------------------------
          u1 |   1.0000
          u2 |   0.2501   1.0000
          u3 |   0.4913   0.7575   1.0000
. gen x1 = uniform()-.5
. gen x2 = uniform() + 1/3
. gen x3 = 2*uniform() + .5
. * Equations
. gen y1s = .5 + 4*x1 + u1
. gen y2s = 3 + .5*x1 - 3*x2 + u2
. gen y3s = 1 - 2*x1 + .4*x2 -.75*x3 + u3
. gen y1 = y1s>0
. gen y2 = y2s>0
. gen y3 = y3s>0
```

The equations for `y1s`, `y2s`, and `y3s` correspond to the equations for $y^*_{im}, i = 1, \ldots, M$, given earlier, and those for `y1`, `y2`, and `y3` correspond to those for y_{im}. The correlations between the error terms (the elements of the matrix V) are shown in the output from the `correlate` command.

The log-likelihood contribution for each observation, $\log\{\Phi_3(\mu; \Omega)\}$, needs to be calculated by the user-written evaluation program that is called by `ml`. Code for

doing this using `ml` evaluation method `lf` is set out below and then the key elements are explained. (For a general introduction to ML estimation using Stata, see Gould, Pitblado, and Sribney 2006.)

```
program define myll
     args lnf xb1 xb2 xb3 c21 c31 c32
     tempvar sp k1 k2 k3
quietly {
     gen double ‘k1’ = 2*$ML_y1 - 1
     gen double ‘k2’ = 2*$ML_y2 - 1
     gen double ‘k3’ = 2*$ML_y3 - 1
     tempname cf21 cf22 cf31 cf32 cf33 C
                // Following needed since lf evaluator
     su ‘c21’, meanonly
     scalar ‘cf21’ = r(mean)
     su ‘c31’, meanonly
     scalar ‘cf31’ = r(mean)
     su ‘c32’, meanonly
     scalar ‘cf32’ = r(mean)
                // constraints on diagonal elements
     scalar ‘cf22’ = sqrt( 1 - ‘c21’^2 )
     scalar ‘cf33’ = sqrt( 1 - ‘c31’^2 - ‘c32’^2 )
     mat ‘C’ = (1, 0, 0  \ ‘cf21’, ‘cf22’, 0 \ ‘cf31’,‘cf32’, ‘cf33’)
     egen ‘sp’ = mvnp(‘xb1’ ‘xb2’ ‘xb3’) , ///
                chol(‘C’) draws($dr) prefix(z) ///
                signs(‘k1’ ‘k2’ ‘k3’)
     replace ‘lnf’= ln(‘sp’)
}
end
```

The `args` statement refers first to `lnf`, the variable that will contain the observation-specific values of $\log\{\Phi_3(\mu;\Omega)\}$. Cited next are the variables containing the observation-specific values of the linear indices for each of the three model equations ($\beta_m' X_m$), and finally there are three variables containing scalar values of the three Cholesky factors associated with correlation matrix V. The first three lines after the `quietly` statement define the observation-specific signs variables that were introduced earlier. Next six lines define three Cholesky factor scalars that are used to specify the lower triangular Cholesky matrix (`‘C’`) that will be passed to `mvnp()`. These lines are required because, although the Cholesky factors to be estimated are scalars, each of the arguments of a method `lf` evaluator is a variable with a value that is the same for each observation. The procedure shown avoids problems that may arise if there are any observations with missing values on those variables, e.g., observations excluded using an `if` qualifier in a later `ml model` statement.

The lines defining scalars `‘cf22’` and `‘cf33’` place constraints on the Cholesky matrix, `‘C’`. Recall that for a multivariate probit model, each diagonal element of the covariance matrix V equals 1 (the variance of each error is normalized to unity). The off-diagonal elements are correlations. Since $V = CC', V_{11} = (C_{11})^2, V_{22} = (C_{21})^2 + (C_{22})^2$, and $V_{33} = (C_{31})^2 + (C_{32})^2 + (C_{33})^2$. Requiring $V_{11} = V_{22} = V_{33} = 1$ leads to the constraints shown. C_{11} is not a function of the estimated parameters and simply set equal to 1: note the first element in the definition of matrix `‘C’`.

The `egen` command calculates the observation-specific values of $\Phi_3(\mu;\Omega)$. The upper integration points are the linear indices for each equation and specified using `mvnp('xb1' 'xb2' 'xb3')`. The `draws($dr)` option refers to a global that will be filled in later, and we will create the required draws variables that have a prefix `z`. The `signs()` option refers to the signs variables created earlier in the evaluation program.

Good starting values are important. An obvious strategy for the multivariate probit model is to assume that the cross-equation correlations are each equal to zero and to set the regression coefficients in each equation equal to the corresponding univariate probit estimates for that equation. (The univariate estimates are consistent but inefficient estimators of the multivariate probit ones.)

```
quietly {
     probit y1 x1
     mat b1 = e(b)
     mat coleq b1 = y1
     probit y2 x1 x2
     mat b2 = e(b)
     mat coleq b2 = y2
     probit y3 x1 x2 x3
     mat b3 = e(b)
     mat coleq b3 = y3
     mat b0 = b1, b2, b3
}
```

To fit the trivariate probit model, we need only choose and set the number of draws and to create the draws variables using `mdraws`, the `ml model` statement, specification of the vector containing starting values using `ml init`, and then, finally, the call to `ml maximize`. We will use 250 pseudorandom draws combined with antithetic draws (500 draws in total). The `ml model` statement specifies equations corresponding to the data generation process.

```
. mdraws, dr(250) neq(3) prefix(z) random seed(123456789) antithetics replace
Created 250 pseudorandom draws per equation for 3 equations. Seed = 123456789
Also created 250 antithetic draws per dimension for 3 dimensions.  Note: there
are now 500 draws per equation
. global dr = r(n_draws)
. ml model lf myll (y1: y1=x1) (y2: y2=x1 x2) (y3: y3 = x1 x2 x3)
>  /c21 /c31 /c32, title("MV Probit by MSL, $dr pseudorandom draws")
. ml init b0
```

After estimation, the estimates of the cross-equation correlations and their standard errors can be derived with `nlcom` applying the definition $V = CC'$. The estimates were as follows:

(*Continued on next page*)

```
. ml maximize
  (output omitted)
MV Probit by MSL, 500 pseudo-random draws         Number of obs   =       5000
                                                  Wald chi2(1)    =    1705.86
Log likelihood =  -6749.249                       Prob > chi2     =     0.0000
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
y1						
x1	3.939239	.0953764	41.30	0.000	3.752304	4.126173
_cons	.496217	.0232598	21.33	0.000	.4506286	.5418054
y2						
x1	.4808888	.0708867	6.78	0.000	.3419534	.6198243
x2	-2.998213	.0805319	-37.23	0.000	-3.156052	-2.840373
_cons	2.94806	.0750775	39.27	0.000	2.800911	3.09521
y3						
x1	-2.037973	.0711971	-28.62	0.000	-2.177516	-1.898429
x2	.324464	.0648008	5.01	0.000	.1974569	.4514712
x3	-.7676863	.0316202	-24.28	0.000	-.8296607	-.7057119
_cons	1.082151	.0740849	14.61	0.000	.9369469	1.227354
c21						
_cons	.2090744	.0306232	6.83	0.000	.1490541	.2690947
c31						
_cons	.4664248	.0263437	17.71	0.000	.4147921	.5180576
c32						
_cons	.6666989	.0214556	31.07	0.000	.6246466	.7087512

```
. nlcom   (r21: [c21]_b[_cons])
>         (r31: [c31]_b[_cons])
>         (r32: [c21]_b[_cons]*[c31]_b[_cons]
>                 + sqrt(1 - [c21]_b[_cons]^2)*[c32]_b[_cons])
        r21:  [c21]_b[_cons]
        r31:  [c31]_b[_cons]
        r32:  [c21]_b[_cons]*[c31]_b[_cons]
>               + sqrt(1 - [c21]_b[_cons]^2)*[c32]_b[_cons]
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
r21	.2090744	.0306232	6.83	0.000	.1490541	.2690947
r31	.4664248	.0263437	17.71	0.000	.4147921	.5180576
r32	.7494822	.0179106	41.85	0.000	.714378	.7845864

The program provides good estimates of not only the underlying model of the regression coefficients but also the correlation matrix. The largest divergences from the "true" model are for the estimates of correlations `r21` and `r31` though, even for these, the 95% confidence interval spans the "true" value.

The gain in estimation speed from using the plugin is substantial. Using a Windows XP Pentium P4/1.8-GHz computer, convergence took about 57 minutes with the plugin, whereas the `adoonly` version of the program and `mvprobit` each took more than 3.5 hours.

Estimates will vary depending on the number and type of draws used. We (2003, 2005, 2006) illustrated this issue for a bivariate probit model fitted with `mvprobit`, varying the number of pseudorandom draws and the seed. Here we focus on differences in results for pseudorandom and Halton draws. Train (2003, 231–234) cites several studies including one of his own, each demonstrating that Halton draws are more effective for simulation than pseudorandom draws. All the studies were based on mixed logit models, and so it is of interest to know whether similar conclusions also apply to multivariate probit models.

Our comparisons took the same format as Train's. First, a trivariate probit model was fitted five times using 500 pseudorandom draws plus antithetics, but with a different seed each time, where the seeds were chosen randomly. The five seeds were selected, to avoid overlaps in sequences, using the following code (derived from Stewart 2006):

```
set seed 123456789
gen long seedvar = int((uniform() + 5 - _n)*100000000) in 1/5
local seed1 = seedvar[1]
local seed2 = seedvar[2]
local seed3 = seedvar[3]
local seed4 = seedvar[4]
local seed5 = seedvar[5]
```

Estimates were compared with those derived from the same model fitted using 50 Halton draws plus antithetic draws, with 10 initial sequence elements dropped in each dimension. Primes 2, 3, and 5 were used, in different permutations, for different dimensions to produce the different estimates. The estimates of parameters and their standard errors are summarized in tables 1 (pseudorandom draws) and 2 (Halton draws).

(*Continued on next page*)

Table 1: Trivariate probit model estimates: pseudorandom draws

500 pseudorandom draws plus antithetic draws

	Row mean	Row s.d.
−logL	6,749.6990	0.3994
Time (sec)	6,878.6380	59.6338
[y1]_b[x1]	3.9391	0.0001
[y1]_se[x1]	0.0954	0.0000
[y1]_b[_cons]	0.4964	0.0001
[y1]_se[_cons]	0.0233	0.0000
[y2]_b[x1]	0.4817	0.0004
[y2]_se[x1]	0.0709	0.0000
[y2]_b[x2]	−2.9980	0.0003
[y2]_se[x2]	0.0805	0.0000
[y2]_b[_cons]	2.9480	0.0004
[y2]_se[_cons]	0.0751	0.0000
[y3]_b[x1]	−2.0350	0.0012
[y3]_se[x1]	0.0712	0.0000
[y3]_b[x2]	0.3238	0.0015
[y3]_se[x2]	0.0648	0.0000
[y3]_b[x3]	−0.7670	0.0005
[y3]_se[x3]	0.0316	0.0000
[y3]_b[_cons]	1.0822	0.0014
[y3]_se[_cons]	0.0741	0.0000
[c21]_b[_cons]	0.2084	0.0008
[c21]_se[_cons]	0.0307	0.0000
[c31]_b[_cons]	0.4673	0.0006
[c31]_se[_cons]	0.0264	0.0000
[c31]_b[_cons]	0.6671	0.0004
[c31]_se[_cons]	0.0215	0.0000
r32	0.7498	0.0002
se(r32)	0.0179	0.0000

Note: c21 = r21 and c31 = r31.

Table 1: Trivariate probit model estimates: pseudorandom draws, continued

	500 pseudorandom draws plus antithetic draws				
Seed	413698407	364322066	255780169	160479494	68417597
	(1)	(2)	(3)	(4)	(5)
−logL	6,750.34	6,749.94	6,749.56	6,749.19	6,749.47
Time (sec)	6,803.16	6,829.94	6,906.33	6,880.56	6,973.20
[y1]_b[x1]	3.9391450	3.9392590	3.9392470	3.93907	3.9389380
[y1]_se[x1]	0.0953744	0.0953770	0.0953797	0.0953833	0.0953632
[y1]_b[_cons]	0.4963101	0.4963739	0.4964035	0.4964518	0.4962911
[y1]_se[_cons]	0.0232582	0.0232589	0.0232624	0.0232609	0.0232553
[y2]_b[x1]	0.4816162	0.4819739	0.4821864	0.4815196	0.4811402
[y2]_se[x1]	0.0708839	0.0708854	0.0708736	0.0708734	0.0708819
[y2]_b[x2]	−2.9985280	−2.9981170	−2.9976140	−2.9980880	−2.9977320
[y2]_se[x2]	0.0805410	0.0805375	0.0805302	0.0805265	0.0805371
[y2]_b[_cons]	2.9485970	2.9480640	2.9474610	2.9480880	2.9475610
[y2]_se[_cons]	0.0750873	0.0750833	0.0750729	0.0750709	0.0750777
[y3]_b[x1]	−2.0333280	−2.0352470	−2.0366050	−2.0357180	−2.0339860
[y3]_se[x1]	0.0711889	0.0712167	0.0712413	0.0711961	0.0711190
[y3]_b[x2]	0.3217575	0.3230236	0.3255635	0.3254686	0.3234068
[y3]_se[x2]	0.0647607	0.0647823	0.0648078	0.0647764	0.0646952
[y3]_b[x3]	−0.7668600	−0.7673650	−0.7674080	−0.7671380	−0.7661510
[y3]_se[x3]	0.0315997	0.0316022	0.0316263	0.0316034	0.0315734
[y3]_b[_cons]	1.0841960	1.0837090	1.0812000	1.0809790	1.0810330
[y3]_se[_cons]	0.0740604	0.0740689	0.0741090	0.0740383	0.0739905
[c21]_b[_cons]	0.2086790	0.2073670	0.2082024	0.2081035	0.2096905
[c21]_se[_cons]	0.0307609	0.0307244	0.0307358	0.0307378	0.0306816
[c31]_b[_cons]	0.4676432	0.4672187	0.4662095	0.4675385	0.4681019
[c31]_se[_cons]	0.0263845	0.0263691	0.0263675	0.0263965	0.0264144
[c31]_b[_cons]	0.6665950	0.6672873	0.6674139	0.6674673	0.6665139
[c31]_se[_cons]	0.0215065	0.0214476	0.0214307	0.0214068	0.0214541
r32	0.7495067	0.7496684	0.7498540	0.7501507	0.7498524
se(r32)	0.0179468	0.0179033	0.0178797	0.0178633	0.0178969

Note: c21 = r21 and c31 = r31.

Train (2003, tables 9.1 and 9.2) reports estimates for mixed logit models (N = 4,308), five estimated using 100 Halton draws and five estimated using 1,000 pseudo-random draws. The means of the estimated model parameters were much the same in each case, suggesting that the smaller number of Halton draws provided estimates much the same on average. But the standard deviation of his Halton estimates was lower, suggesting, given much the same mean, that with 100 Halton draws a researcher can expect to be closer to the expected values of the estimates than with 1,000 pseudorandom draws. For our trivariate probit model, we get an approximately 10-fold savings in computation time, as Train did. According to the "Time" row in table 1, mean estimation time is 10.2% smaller using Halton draws. Corresponding means of estimates are also similar (see the "row mean" entries). However, the row standard deviations are not unambiguously lower for the Halton estimates. Our results underscore Train's (2003, 233) remarks that simple statements about the relative advantages of Halton draws need to be viewed with caution. More research is required before definitive conclusions can be drawn about the tradeoff between speed and accuracy.

(*Continued on next page*)

Table 2: Trivariate probit model estimates: Halton draws

50 Halton draws plus antithetic draws (10 initial draws dropped)

	Row mean	Row s.d.
−logL	6,750.3430	0.3306
Time (sec)	698.8080	7.6409
[y1]_b[x1]	3.9387	0.0001
[y1]_se[x1]	0.0954	0.0000
[y1]_b[_cons]	0.4962	0.0001
[y1]_se[_cons]	0.0233	0.0000
[y2]_b[x1]	0.4816	0.0004
[y2]_se[x1]	0.0709	0.0000
[y2]_b[x2]	−2.9984	0.0003
[y2]_se[x2]	0.0806	0.0000
[y2]_b[_cons]	2.9481	0.0002
[y2]_se[_cons]	0.0751	0.0000
[y3]_b[x1]	−2.0345	0.0018
[y3]_se[x1]	0.0712	0.0001
[y3]_b[x2]	0.3232	0.0011
[y3]_se[x2]	0.0648	0.0001
[y3]_b[x3]	−0.7665	0.0007
[y3]_se[x3]	0.0316	0.0000
[y3]_b[_cons]	1.0816	0.0008
[y3]_se[_cons]	0.0741	0.0001
[c21]_b[_cons]	0.2078	0.0010
[c21]_se[_cons]	0.0308	0.0001
[c31]_b[_cons]	0.4655	0.0009
[c31]_se[_cons]	0.0264	0.0001
[c31]_b[_cons]	0.6669	0.0005
[c31]_se[_cons]	0.0215	0.0000
r32	0.7490	0.0005
se(r32)	0.0179	0.0000

Note: c21 = r21 and c31 = r31.

Table 2: Trivariate probit model estimates: Halton draws, continued

	50 Halton draws plus antithetic draws (10 initial draws dropped)				
Primes	2, 3, 5	3, 2, 5	5, 3, 2	2, 5, 3	3, 5, 2
	(1)	(2)	(3)	(4)	(5)
−logL	6,750.92	6,750.332	6,750.364	6,749.907	6,750.192
Time (sec)	703.99	702.30	700.03	703.92	683.80
[y1]_b[x1]	3.9388990	3.9386530	3.9385420	3.9385820	3.9385950
[y1]_se[x1]	0.0953799	0.0953742	0.0953760	0.0953675	0.0953814
[y1]_b[_cons]	0.4963090	0.4962998	0.4961965	0.4961129	0.4961550
[y1]_se[_cons]	0.0232611	0.0232602	0.0232600	0.0232571	0.0232613
[y2]_b[x1]	0.4821302	0.4818128	0.4809724	0.4819018	0.4813064
[y2]_se[x1]	0.0708865	0.0708894	0.0708929	0.0708850	0.0709040
[y2]_b[x2]	−2.9982050	−2.9984710	−2.9987860	−2.9980170	−2.9984140
[y2]_se[x2]	0.0805421	0.0805402	0.0805527	0.0805527	0.0805631
[y2]_b[_cons]	2.9480800	2.9481210	2.9484980	2.9478830	2.9481380
[y2]_se[_cons]	0.0750866	0.0750821	0.0751001	0.0750986	0.0751062
[y3]_b[x1]	−2.0325070	−2.0330360	−2.0358330	−2.0337580	−2.0373120
[y3]_se[x1]	0.0711341	0.0711083	0.0711919	0.0711608	0.0712849
[y3]_b[x2]	0.3223934	0.3221294	0.3224382	0.3249884	0.3240396
[y3]_se[x2]	0.0648024	0.0647506	0.0648555	0.0648144	0.0648916
[y3]_b[x3]	−0.7656050	−0.7659450	−0.7667760	−0.7665520	−0.7675070
[y3]_se[x3]	0.0316383	0.0316004	0.0316483	0.0315646	0.0316592
[y3]_b[_cons]	1.0813130	1.0814320	1.0826200	1.0802830	1.0823640
[y3]_se[_cons]	0.0741471	0.0740966	0.0741638	0.0740542	0.0741773
[c21]_b[_cons]	0.2084468	0.2062338	0.2089133	0.2083446	0.2069897
[c21]_se[_cons]	0.0308445	0.0307262	0.0306878	0.0308136	0.0307486
[c31]_b[_cons]	0.4658724	0.4648260	0.4651382	0.4671057	0.4644295
[c31]_se[_cons]	0.0264159	0.0263092	0.0262384	0.0264215	0.0264397
[c31]_b[_cons]	0.6667344	0.6675500	0.6662109	0.6672030	0.6666491
[c31]_se[_cons]	0.0215791	0.0215287	0.0215076	0.0214571	0.0215259
r32	0.7491984	0.7490623	0.7486839	0.7498806	0.7483437
se(r32)	0.0179779	0.0179679	0.0179383	0.0179089	0.0179304

Note: c21 = r21 and c31 = r31.

3.5 Illustration 3: MSL estimation of multivariate probit models with sample selection

`mvnp()` can also be used to fit multivariate probit models with sample selection (otherwise known as *incidental truncation*). The built-in command `heckprob` is a bivariate example of this type of model: there is one equation describing the binary outcome of interest and a second equation that characterizes whether the first outcome is observed. If the cross-equation error terms are correlated, sample selection is "endogenous", in which case simply fitting a univariate probit model for the binary outcome of interest leads to inconsistent estimators of the parameters of interest. Models with multiple outcomes of interest and possibly more than one selection equation are obvious generalizations of the `heckprob` case. For example, Jenkins et al. (forthcoming) have an equation system with four binary outcomes, two of which describe sample selections. In our 2004 paper, we modeled three binary outcomes, including one sample selection equation. We consider a similar trivariate model here.

We use data on 1,098 working-age employees who responded to the Bank of Italy's Survey of Households' Income and Wealth in a base year (either 1993 or 1995) and with whom follow-up interviews were sought 2 or 3 years later (1995 or 1998, respectively). We model the determinants of whether a respondent was low paid in the base year and whether the respondent was low paid in the later ("current") year. Low pay in each year is defined as having a wage in the poorest fifth of the earnings distribution of that year. The complication is that not all respondents in the base year provided data in the current year, and so we wish to model current year low pay probabilities controlling for the potential sample selection biases that may arise from differential sample dropout. (Dropout here includes either sample attrition or sample retention but not having a job.) In this illustrative dataset, `trial.dta`, `lph20` = 1 if low paid in the base year and 0 otherwise. For the current year, `flph20` is defined similarly but is observed only if the sample retention indicator `retent1` = 1 (`retent1` = 0 for the 382 observations who dropped out). Age (`eta`), age-squared (`eta2`), and the sex of the employee (`female` = 1 if a woman) are the only predictor variables used in this simple illustration, and all three are included as regressors in each of the three equations.

The equations for this model have the following form for each observation:

$$\begin{aligned}
\text{Low pay, base year:} \quad & L^* = W'\beta + l, \text{ where } L = I(L^* > 0) \\
\text{Sample retention:} \quad & R^* = Y'\delta + r, \text{ where } R = I(R^* > 0) \\
\text{Low pay, current year:} \quad & F^* = Z'\theta + f, \text{ where } F = I(F^* > 0) \text{ if } R = 1 \\
& \qquad \text{and is missing otherwise}
\end{aligned}$$

The variables denoted by asterisks are the latent outcomes and those without them are binary indicators summarizing the observed outcomes. $I(.)$ is the indicator function equal to one if its argument is true and zero otherwise. Observe the sample selection condition in the current year low-pay equation. We assume that the error terms $(l, r, f) \sim N_3(0, V)$, where V is a symmetric matrix with typical element $\rho_{rs} = \rho_{sr}$ for $r, s \in \{l, r, f\}$ and $r \neq s$, and $\rho_{rr} = 1$, for all r. The errors in each equation are assumed to be orthogonal to the predictors (elements of the vectors W, Y, and Z, respectively).

Define a set of signs variables $\kappa_T = 2T - 1$ for $T \in \{L, R, F\}$. The likelihood contribution for an employee who is observed in both the base year and the current year, i.e., with $R = 1$, is

$$\mathcal{L}_3 = \Phi_3(\kappa_L W'\beta, \kappa_R Y'\delta, \kappa_F Z'\theta; \kappa_L\kappa_R\rho_{lr}, \kappa_L\kappa_F\rho_{lf}, \kappa_R\kappa_F\rho_{rf})$$

By contrast, the likelihood contribution for someone who responded only in the first year is

$$\mathcal{L}_2 = \Phi_2(\kappa_L W'\beta, \kappa_R Y'\delta; \kappa_L\kappa_R\rho_{lr})$$

The log-likelihood contribution to be calculated by the evaluator function for each observation is

$$(1 - R) \log \mathcal{L}_2 + R \log \mathcal{L}_3$$

The evaluator function for method `lf` estimation is coded with a similar structure to that used for the multivariate probit example earlier. Any differences reflect that the three outcome variables are observed only for employees who are retained in the sample; for dropouts, there are only two observed outcomes.

```
program define myll
     args lnf x1 x2 x3  c21 c31 c32
     tempvar sp2 sp3 k1 k2 k3
quietly {
     gen double ‘k1’ = 2*$ML_y1 - 1
     gen double ‘k2’ = 2*$ML_y2 - 1
     gen double ‘k3’ = 2*$ML_y3 - 1
     tempname cf21 cf22 cf31 cf32 cf33 C1 C2
     su ‘c21’, meanonly
     scalar ‘cf21’ = r(mean)
     su ‘c31’, meanonly
     scalar ‘cf31’ = r(mean)
     su ‘c32’, meanonly
     scalar ‘cf32’ = r(mean)
     scalar ‘cf22’ = sqrt(1 - ‘cf21’^2)
     scalar ‘cf33’ = sqrt(1 - ‘cf31’^2 - ‘cf32’^2)
     mat ‘C1’ = (1, 0 , 0 \ ‘cf21’, ‘cf22’, 0 \ ‘cf31’ , ‘cf32’ , ‘cf33’)
     mat ‘C2’ = (1, 0 \ ‘cf21’, ‘cf22’)
     egen ‘sp3’ = mvnp(‘x1’ ‘x2’ ‘x3’) if   $ML_y1==1, ///
          chol(‘C1’) dr($dr) ml prefix(z) signs(‘k1’ ‘k2’ ‘k3’)
     egen ‘sp2’ = mvnp(‘x1’ ‘x2’ ) if   $ML_y1==0, ///
          chol(‘C2’) dr($dr) ml prefix(z) signs(‘k1’ ‘k2’)
     replace ‘lnf’= cond($ML_y1, ln(‘sp3’), ln(‘sp2’), .)
}
end
```

There are two principal differences from the earlier illustration. First, there are now two Cholesky matrices defined, `‘C1’`, `‘C2’`, with the latter being a submatrix of the former. (Having two matrices ensures that the appropriate parameter constraints hold for all observations, regardless of whether they dropped out.) Second, the call to `mvnp()` differs depending on dropout status. Adding the `if` qualifier to the `egen` command is not essential but it is wise because restricting the number of observations for whom the simulation calculations is undertaken reduces computation time.

Starting values were derived from three independent univariate probit regressions (the same method as for the trivariate probit example) and again they were stored in a matrix named `b0`. `mdraws` was used to create 100 Halton draws with antithetics, and then the calls were made to `ml model` and `ml maximize`. Because observations that dropped out of the sample have missing values for the current year low-pay status variable `flph20`, we used the `missing` option on the `ml model` statement so that these cases are not dropped from the estimation sample.

```
. mdraws, dr(100) neq(3) prefix(z) burn(10) antithetics
Created 100 Halton draws per equation for 3 dimensions. Number of initial draws
dropped per dimension = 10 . Primes used:
  2  3  5
Also created 100 antithetic draws per dimension for 3 dimensions.  Note: there
are now 200 draws per equation
. global dr = r(n_draws)
. ml model lf myll (retent1: retent1 = female eta eta2)
>          (lph20:  lph20 = female eta eta2)
>          (flph20: flph20 = female eta eta2)
>          /c21  /c31 /c32
>         , missing title("3-var probit, 1 selection, MSL, $dr Halton draws")
. ml init b0
```

The resulting estimates were

```
. ml maximize
initial:       log likelihood = -1387.3445
rescale:       log likelihood = -1387.3445
rescale eq:    log likelihood = -1387.3445
Iteration 0:   log likelihood = -1387.3445  (not concave)
Iteration 1:   log likelihood = -1347.2308  (not concave)
Iteration 2:   log likelihood = -1338.9097
Iteration 3:   log likelihood = -1338.7581
Iteration 4:   log likelihood = -1338.7247
Iteration 5:   log likelihood = -1338.7247
3-var probit, 1 selection, MSL, 200 Halton draws  Number of obs   =       1098
                                                  Wald chi2(3)    =      32.88
Log likelihood = -1338.7247                       Prob > chi2     =     0.0000
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
retent1						
female	-.0282483	.0838161	-0.34	0.736	-.1925249	.1360282
eta	.1605634	.0280407	5.73	0.000	.1056047	.2155221
eta2	-.0020581	.0003605	-5.71	0.000	-.0027646	-.0013515
_cons	-2.500076	.5174225	-4.83	0.000	-3.514206	-1.485947
lph20						
female	.1314453	.0992667	1.32	0.185	-.0631139	.3260045
eta	-.2443421	.0335981	-7.27	0.000	-.3101931	-.178491
eta2	.0026242	.00044	5.96	0.000	.0017618	.0034867
_cons	4.152685	.6008131	6.91	0.000	2.975113	5.330257

```
flph20
      female |   .2920033   .1215907     2.40   0.016       .05369    .5303167
         eta |  -.2204495   .0610417    -3.61   0.000    -.3400891   -.1008099
        eta2 |   .0025759   .0008165     3.15   0.002     .0009756    .0041762
       _cons |   3.322163   1.461342     2.27   0.023     .4579846    6.186341
-------------+----------------------------------------------------------------
c21          |
       _cons |  -.1382572   .0597633    -2.31   0.021    -.2553911   -.0211234
-------------+----------------------------------------------------------------
c31          |
       _cons |  -.2608709   .6660562    -0.39   0.695    -1.566317    1.044575
-------------+----------------------------------------------------------------
c32          |
       _cons |   .6888565   .1274599     5.40   0.000     .4390397    .9386734
------------------------------------------------------------------------------

. // Derive correlations from cholesky factors
. // C is lower triangular, with 1s on the leading diagonal
. // r21 = c21, r31 = c31, r32 = c21*c31 + c22*c32

. nlcom (r21: [c21]_b[_cons])  ///
>       (r31: [c31]_b[_cons]) ///
>       (r32: [c21]_b[_cons]*[c31]_b[_cons]  ///
>               + sqrt(1 - [c21]_b[_cons]^2 )[c32]_b[_cons] ) ///
>       , post

        r21:  [c21]_b[_cons]
        r31:  [c31]_b[_cons]
        r32:  [c21]_b[_cons]*[c31]_b[_cons]
>               + sqrt(1 - [c21]_b[_cons]^2)*[c32]_b[_cons]

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         r21 |  -.1382572   .0597633    -2.31   0.021    -.2553911   -.0211234
         r31 |  -.2608709   .6660562    -0.39   0.695    -1.566317    1.044575
         r32 |   .7183083    .055739    12.89   0.000     .6090618    .8275548
------------------------------------------------------------------------------
```

The results suggest that there is a U-shaped relationship between age and the probability of being low paid, whereas the relationship between age and the probability of sample retention is inverse-U shaped. Those with unobserved characteristics favoring sample retention are less likely to have been low paid in the base year (`r21` < 0). And low-pay propensities in the current and base years are correlated (`r32` > 0). A formal test of whether sample selection is ignorable is based on a test of the null hypothesis `r21` $= 0 =$ `r31`. This test can be implemented using the `test` command after `nlcom`. Observe the use of the `post` option to `nlcom`, which saves the `nlcom` output as e-class results.

```
. // sample selection ignorable if r21 = r31 = 0
. test r21 r31
 ( 1)  r21 = 0
 ( 2)  r31 = 0
           chi2(  2) =    5.40
         Prob > chi2 =    0.0670
```

With a p-value for the test of 0.067, we cannot reject the null hypothesis of ignorability at the 5% significance level.

3.6 Illustration 4: MSL estimation of a probit model for panel data

Our final illustration provides an example of a method `d0` estimator applied to panel data. (The earlier illustrations considered method `lf` estimators and cross-sectional data.) The dataset `long4.dta` contains data on 1,334 men and women aged 50–59 years who were respondents to the UK Quarterly Labour Force Survey between 1993 and 2004. Each individual was interviewed for five consecutive quarters ("waves"), providing a balanced panel. We investigate the predictors of the probability of being employed rather than being unemployed or economically inactive.

Because we have repeated observations on the same individuals over time, we can control for unobserved differences in employment probabilities. One common way is with a random-effects probit model (`xtprobit` in Stata), but this has the disadvantage that it imposes an equicorrelation structure on the error terms for the different periods, as well as equal variances. We generalize this model to allow for an unrestricted error variance–covariance structure.

More formally, we assume that the model for the latent employment propensity and observed employment outcome for each individual is

$$y_t^* = \beta' X_t + u_t, t = 1, \ldots, T$$
$$y_t = 1 \text{ if } y_t^* > 0, \text{ and 0 otherwise}$$

where X_t is a vector of observed predictors. The error terms u_t are assumed to have a multivariate normal distribution of dimension T, with zero mean and covariance matrix V: $u_t \sim N_T(0, V)$. The covariance matrix is unconstrained, except that one variance is normalized to unity, for identification. The random-effects probit model is a special case of this specification. It assumes $u_t = \alpha + \epsilon_t$, with $\alpha \sim N(0, \sigma_\alpha^2)$, and $\epsilon_t \sim N_T(0, \sigma_\epsilon^2 I_T)$ implying that $\text{cov}(u_t, u_s) = \sigma_\alpha^2 + \sigma_\epsilon^2$ if $t = s$, and σ_α^2 if $t \neq s$.

The likelihood contribution for each individual, given $T = 5$ in each case, is

$$\Phi_5(\kappa_1 \beta' X_1, \kappa_2 \beta' X_2, \ldots, \kappa_5 \beta' X_5; KVK)$$

where $K = \text{diag}(\kappa_1 \ldots \kappa_5)$, $\Phi_5(.)$ is the five-variate standard normal distribution function, and the signs variables (κ_t) are defined as earlier.

The likelihood-evaluation function for this model assumes that the data are in long form, with one row for each person–wave observation. The personal identifier for each individual is held in the global macro `$pid` and the numbers of waves observed (5) is held in global macro `$M`. The data are assumed to be sorted by `$pid` and wave.

```
global cs " "
global csbar " "
forvalues i = 2/$M {
     forvalues j = 1/‘i’ {
          global cs "$cs  c‘i’‘j’"
          global csbar "$csbar  /c‘i’‘j’"
     }
}

program define myll
     args todo b lnf
     tempvar theta1 T fi xb1 xb2 xb3 xb4 xb5  k1 k2 k3 k4 k5
     tempname $cs
     mleval ‘theta1’ = ‘b’ , eq(1)
     local c = 1
     forvalues i = 2/$M {
          forvalues j = 1/‘i’ {
               local c = ‘c’ + 1
               mleval ‘c‘i’‘j’’ = ‘b’  , eq(‘c’) scalar
          }
     }
     quietly {
          forvalues i = 1/$M {
               by $pid: gen double ‘k‘i’’ = (2*$ML_y1[‘i’]) - 1
               by $pid: gen double ‘xb‘i’’ = ‘theta1’[‘i’]
          }
          by $pid: gen double ‘T’ =  (_n == $M)
          tempname C
          mat ‘C’ = I($M)
          forvalues i = 2/$M {
               forvalues j = 1/‘i’ {
                    local c‘i’‘j’ = ‘c‘i’‘j’’
                    mat ‘C’[‘i’,‘j’] = (‘c‘i’‘j’’)
               }
          }
          egen ‘fi’ = mvnp(‘xb1’ ‘xb2’ ‘xb3’ ‘xb4’ ‘xb5’) , ///
              chol(‘C’) dr($dr) prefix(z) ///
              signs(‘k1’ ‘k2’ ‘k3’ ‘k4’ ‘k5’)
          mlsum ‘lnf’ = ln(‘fi’) if ‘T’
     }
end
```

The first piece of code defines, for convenience, a global macro that will hold the names of all the Cholesky matrix elements (`cs`), together with another global macro containing the corresponding equation names (`csbar`), to be used on the later `ml model` statement.

```
. di "$cs"
   c21  c22  c31  c32  c33  c41  c42  c43  c44  c51  c52  c53  c54  c55

. di "$csbar"
   /c21  /c22  /c31  /c32  /c33  /c41  /c42  /c43  /c44  /c51  /c52  /c53
   /c54  /c55
```

Because we use a method `d0` estimator, the evaluation program has a different format from those used in the earlier illustrations. In particular, the Cholesky factors are not declared on the `args` statement but by using `mleval` statements instead. (Each factor is no longer a variable, but a constant term in an equation.) The first lines within the `quietly` block create the signs variables and the linear indices ($\beta' X_t$). Observe the indexing of the variables to each quarter. And, since only one equation (`'theta1'`) was declared earlier to refer to the regression coefficients, the coefficients are constrained to be the same for each quarter, as required. The next lines specify the Cholesky matrix. No constraints are placed on the elements except that $C_{11} = 1$. (Observe that matrix `'C'` is first declared as an identity matrix and subsequent lines replace lower triangular elements with Cholesky factors—with the exception of element (1,1), which therefore stays equal to 1.) The call to `mvnp()` creates the simulated probabilities of the observed employment–nonemployment sequence for each respondent. Finally, the `mlsum` statement sums, for each individual, the log of these probabilities and stores the result in the last data row.

To fit the model, we need to start by declaring starting values and creating the draws variables for a chosen number of draws. The predictor variables used are `female` (a binary indicator equal to 1 if the individual is a woman and 0 otherwise) and `age` (in years). Code for these steps, to specify the `ml model` statement, and to maximize the model, could be

```
. probit employed female age
. mat b0 = e(b)
. mdraws, dr(50) neq($M) prefix(z) antithetics
. global dr = r(n_draws)
. ml init b0
. ml model d0 myll (employed = female age) $csbar,
> title(Multiperiod Probit, $dr Halton draws)
```

In practice, we had to use several variations on these statements to fit a model that converged satisfactorily. We often experienced nonconcavities and nonconvergence and so experimented with different numbers and types of draws, as well as with different starting values. One successful strategy was to fit the model with a few draws, to use the resulting estimates as starting values for estimation using more draws, and then to repeat this process until the estimates stabilized. When doing so, we also used the `technique(dfp nr)` option on the `ml model` statement and the `difficult` option on the `ml maximize` statement.

Example output, from a run based on 250 Halton draws plus antithetics, is as follows.

```
. ml maximize, difficult
  (output omitted)
Multiperiod Probit, 500 Halton draws              Number of obs   =       6670
                                                  Wald chi2(2)    =     118.07
Log likelihood =  -1537.467                       Prob > chi2     =     0.0000
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
employed						
female	-.4531049	.0736374	-6.15	0.000	-.5974317	-.3087782
age	-.0877273	.0083794	-10.47	0.000	-.1041507	-.0713039
_cons	5.488687	.4846868	11.32	0.000	4.538718	6.438656
c21						
_cons	1.073389	.0386209	27.79	0.000	.9976932	1.149084
c22						
_cons	.1729371	.0234292	7.38	0.000	.1270167	.2188576
c31						
_cons	1.044281	.043958	23.76	0.000	.9581245	1.130437
c32						
_cons	.1773736	.0288286	6.15	0.000	.1208705	.2338767
c33						
_cons	.1462416	.0201903	7.24	0.000	.1066694	.1858138
c41						
_cons	.9886117	.0466896	21.17	0.000	.8971018	1.080122
c42						
_cons	.1765436	.0336247	5.25	0.000	.1106404	.2424468
c43						
_cons	.1682321	.0277484	6.06	0.000	.1138461	.222618
c44						
_cons	.1135522	.0190432	5.96	0.000	.0762283	.1508762
c51						
_cons	.9701103	.0477173	20.33	0.000	.8765862	1.063634
c52						
_cons	.1569621	.0383303	4.09	0.000	.081836	.2320882
c53						
_cons	.1145818	.0341573	3.35	0.001	.0476347	.1815289
c54						
_cons	.1627055	.0317603	5.12	0.000	.1004565	.2249546
c55						
_cons	.0955757	.031963	2.99	0.003	.0329293	.1582221

The results indicate that employment probabilities are lower for women than men and decline with age. The estimate of the covariance matrix of the error terms is as follows, derived by applying `nlcom` as before.

```
. nlcom (v21: [c21]_b[_cons])
>       (v22: [c21]_b[_cons]^2 + [c22]_b[_cons]^2)
>       (v31: [c31]_b[_cons])
>       (v32: [c21]_b[_cons]*[c31]_b[_cons] + [c22]_b[_cons]*[c32]_b[_cons])
>       (v33: [c31]_b[_cons]^2 + [c32]_b[_cons]^2 + [c33]_b[_cons]^2)
>       (v41: [c41]_b[_cons])
>       (v42: [c21]_b[_cons]*[c41]_b[_cons] + [c22]_b[_cons]*[c42]_b[_cons])
>       (v43: [c31]_b[_cons]*[c41]_b[_cons] + [c32]_b[_cons]*[c42]_b[_cons]
>                + [c33]_b[_cons]*[c43]_b[_cons])
>       (v44: [c41]_b[_cons]^2 + [c42]_b[_cons]^2 +
>             [c43]_b[_cons]^2 + [c44]_b[_cons]^2)
>       (v51: [c51]_b[_cons])
>       (v52: [c21]_b[_cons]*[c51]_b[_cons] + [c22]_b[_cons]*[c52]_b[_cons])
>       (v53: [c31]_b[_cons]*[c51]_b[_cons] + [c32]_b[_cons]*[c52]_b[_cons]
>                + [c33]_b[_cons]*[c53]_b[_cons] )
>       (v54: [c41]_b[_cons]*[c51]_b[_cons] + [c42]_b[_cons]*[c52]_b[_cons]
>                + [c43]_b[_cons]*[c53]_b[_cons] + [c44]_b[_cons]*[c54]_b[_cons])
>       (v55: [c51]_b[_cons]^2 + [c52]_b[_cons]^2
>                + [c53]_b[_cons]^2 + [c54]_b[_cons]^2 + [c55]_b[_cons]^2)
>         , post
```

(*output omitted*)

employed	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
v21	1.073389	.0386209	27.79	0.000	.9976932	1.149084
v22	1.18207	.0851815	13.88	0.000	1.015118	1.349023
v31	1.044281	.043958	23.76	0.000	.9581245	1.130437
v32	1.151593	.0809325	14.23	0.000	.9929687	1.310218
v33	1.14337	.0956865	11.95	0.000	.9558277	1.330912
v41	.9886117	.0466896	21.17	0.000	.8971018	1.080122
v42	1.091696	.0792027	13.78	0.000	.936461	1.24693
v43	1.088305	.0899833	12.09	0.000	.9119406	1.264669
v44	1.049717	.0977442	10.74	0.000	.8581417	1.241292
v51	.9701103	.0477173	20.33	0.000	.8765862	1.063634
v52	1.06845	.0780511	13.69	0.000	.9154726	1.221427
v53	1.057665	.0856769	12.34	0.000	.8897413	1.225588
v54	1.024525	.0918157	11.16	0.000	.8445694	1.20448
v55	1.014488	.0985217	10.30	0.000	.821389	1.207587

The estimates of the variances are close to one, which implies—given the covariance estimates—that each correlation is also close to one. A formal test of the constant variance and equicorrelation assumptions incorporated in the random-effects probit model can be implemented by using `test` after `nlcom` with the `post` option.

```
. test _b[v22] = _b[v33] = _b[v44] = _b[v55] = 1
 ( 1)  v22 - v33 = 0
 ( 2)  v22 - v44 = 0
 ( 3)  v22 - v55 = 0
 ( 4)  v22 = 1
           chi2(  4) =    6.21
         Prob > chi2 =    0.1841
```

```
. test _b[v21] = _b[v31] = _b[v32] = _b[v41] = _b[v42] = _b[v43] = _b[v51] =
> _b[v52] = _b[v53] = _b[v54], accum
 ( 1)  v22 - v33 = 0
 ( 2)  v22 - v44 = 0
 ( 3)  v22 - v55 = 0
 ( 4)  v22 = 1
 ( 5)  v21 - v31 = 0
 ( 6)  v21 - v32 = 0
 ( 7)  v21 - v41 = 0
 ( 8)  v21 - v42 = 0
 ( 9)  v21 - v43 = 0
 (10)  v21 - v51 = 0
 (11)  v21 - v52 = 0
 (12)  v21 - v53 = 0
 (13)  v21 - v54 = 0
           chi2( 13) =   29.25
         Prob > chi2 =    0.0060
```

According to the first test, we cannot reject the null hypothesis that each error variance is equal to one. However, according to the second test, one can reject the null hypothesis of unit variances combined with equal cross-wave correlations, i.e., the assumptions of the random-effects probit model (for which the estimated cross-wave correlation is around 0.96).

One might also investigate other hypotheses about the structure of the covariance matrix, for example whether the estimates were consistent with some particular autoregressive moving average error structure and implement the test for the Cholesky factors using `testnl`. Estimation of models that incorporate autoregressive moving average error structures is harder since, e.g., `ml` does not currently allow specification of nonlinear constraints (`constraint define` refers to linear constraints). It would also be desirable to incorporate some genuine dynamics into the model, e.g., to have the current quarter's employment probability depend on whether the individual worked last quarter. This relationship, in turn, raises *initial conditions* issues. That is, the state in which the respondent is first observed is endogenous and needs to be accounted for. For a program that estimates by MSL a dynamic probit model controlling for initial conditions and with first-order autocorrelated errors, see `redpace` by Stewart (2006). Allowing for unbalanced panels would be another useful extension.

4 Conclusions

Here we have extended the range of models that can be fitted in Stata. Users can fit many types of model by MSL and do so using less computation time. Although our examples have focused on fitting multivariate and multiperiod probit models, one can also fit models in which some outcomes are continuous and some are binary. Using `mdraws` and `mvnp()` requires the user to code the appropriate likelihood evaluation function, but the template code used in our illustrations aims to make that task easier.

Effective estimation is also partly a matter of experience. Complicated models require good starting values, and finding them may require some experimentation. We have referred, for example, to tricks such as starting by fitting a model with a few draws and using estimates from this model as starting values for a model with more draws. If the process takes a long time, then saving intermediate results to disk with tools such as Michael Blasnik's `estsave` program (available from SSC) may prove useful.

There remain many gaps in our knowledge about the performance of MSL estimators and the different types of draw variables. Most empirical investigations of estimator properties have focused on mixed logit models, and it is not clear yet whether the conclusions derived in that context also apply to the multivariate normal case. Our comparisons of multivariate probit model estimates based on pseudorandom and Halton draws (tables 1 and 2) emphasize this point. Nevertheless, some promising evidence is provided by Sándor and András (2004). They studied the performance of several sampling methods for estimation of multivariate normal probabilities using the GHK simulator. Draws variables based on Halton sequences are shown to perform better than those based on pseudorandom draws (with or without antithetic draws). Both are dominated by other more complicated methods such as Niederreiter sequences and those based on orthogonal arrays.

Our programs intentionally separate the tasks of creation of the draws variables from the calculation of the multivariate normal probabilities. This modular approach means that it should be easier to incorporate extensions and innovations. These improvements might also take advantage of Mata. After we finished the first draft of this paper, Mata functions for calculation of Halton sequences and multivariate normal probabilities using the GHK simulator became available: `halton()` and `ghk()`, released in the 20 January 2006 update to Stata version 9.1; see `help mf_halton` and `help mf_ghk`. These are welcome innovations, but our programs provide similar functionality for Stata users of version 8.2 and later, and our use of a plugin means that calculations are also relatively fast. Moreover, `mdraws` lets users choose the prime numbers that are used to create Halton sequences and allocates the sequence elements to observations.

Our programs do not have to be used together. For example, `mdraws` can be used separately for many MSL applications. Haan and Uhlendorff (2006) use MSL to fit a random-intercept multinomial logit model with panel data. Because the latent outcome variables do not have a multivariate normal distribution, `mvnp()` is not applicable. However, each simulation requires a set of draws variables, and these simulations use `mdraws` to derive Halton draws.

5 Acknowledgments

This research was supported by core funding to the Institute for Social and Economic Research from the UK Economic and Social Research Council and the University of Essex.

The first draft of this article was written when Jenkins visited the SOEP Group at DIW Berlin. Mark Stewart provided many helpful comments and suggestions. The code for Halton draws is a generalization of some do-file code by Arne Uhlendorff (DIW Berlin), which in turn uses a program posted on Statalist by Nick Cox (Durham University) in August 2004: see http://www.stata.com/statalist/archive/2004-08/msg00222.html. `mdraws` also uses that program, here renamed `mdraws_h` and modified. We are grateful to the anonymous referee for comments. We owe very special thanks to Roberto Gutierrez (StataCorp) for writing the plugin to accompany `mvnp()`.

6 References

Cappellari, L., and S. P. Jenkins. 2003. Multivariate probit regression using simulated maximum likelihood. *Stata Journal* 3: 278–294.

———. 2004. Modelling low income transitions. *Journal of Applied Econometrics* 19: 593–610.

———. 2005. Software update: st0045_1: Multivariate probit regression using simulated maximum likelihood. *Stata Journal* 5: 285.

———. 2006. Software update: st0045_2: Multivariate probit regression using simulated maximum likelihood. *Stata Journal* 6: 284.

Gould, W., J. Pitblado, and W. Sribney. 2006. *Maximum Likelihood Estimation with Stata*. 3rd ed. College Station, TX: Stata Press.

Gouriéroux, C., and A. Monfont. 1996. *Simulation-Based Econometric Methods*. Oxford: Oxford University Press.

Greene, W. H. 2003. *Econometric Analysis*. 5th ed. Upper Saddle River, NJ: Prentice Hall.

Haan, P., and A. Uhlendorff. 2006. Estimation of multinomial logit models with unobserved heterogeneity using maximum simulated likelihood. *Stata Journal* 6: 229–245.

Hess, S., and J. Polak. 2003. An alternative method to the scrambled Halton sequence for removing correlation between standard Halton sequences in higher dimensions. Paper presented at the 2003 European Regional Science Conference, Jyväskylä, Finland. http://www.jyu.fi/ersa2003/cdrom/papers/406.pdf.

Hess, S., J. Polak, and A. Daly. 2003. On the performance of the shuffled Halton sequence in the estimation of discrete choice models. Paper presented at the European Transport Conference, Strasbourg, France.
http://www.cts.cv.imperial.ac.uk/StaffPages/StephaneHess/papers/Hess_Polak_Daly_ETC_oct_16.pdf.

Jenkins, S. P., L. Cappellari, P. Lynn, A. Jäckle, and E. Sala. Forthcoming. Patterns of consent: evidence from a general household survey. *Journal of the Royal Statistical Society, Series A* 169. http://www.iser.essex.ac.uk/pubs/workpaps/pdf/2004-27.pdf.

Sándor, Z., and P. András. 2004. Alternative sampling methods for estimating multivariate normal probabilities. *Journal of Econometrics* 120: 207–234.

Stern, S. 1997. Simulation-based estimation. *Journal of Economic Literature* 35: 2006–2039.

Stewart, M. 2006. Maximum simulated likelihood estimation of random-effects dynamics probit models with autocorrelated errors. *Stata Journal* 6: 256–272.

Train, K. 2003. *Discrete Choice Methods with Simulation*. Cambridge: Cambridge University Press.

About the authors

Lorenzo Cappellari is an associate professor at the Universitá Cattolica (Milan, Italy) and a research associate of the Institute for Social and Economic Research (ISER) at the University of Essex, Colchester, UK. Stephen Jenkins is a professor at ISER, research professor at DIW Berlin, and an associate editor of the *Stata Journal*. Both authors are research associates of IZA (Bonn) and CHILD (Turin).