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. 190–213

A Mata Geweke–Hajivassiliou–Keane multivariate normal simulator

Richard Gates
StataCorp
College Station, TX
rgates@stata.com

Abstract. An accurate and efficient numerical approximation of the multivariate normal (MVN) distribution function is necessary for obtaining maximum likelihood estimates for models involving the MVN distribution. Numerical integration through simulation (Monte Carlo) or number-theoretic (quasi–Monte Carlo) techniques is one way to accomplish this task. One popular simulation technique is the Geweke–Hajivassiliou–Keane MVN simulator. This paper reviews this technique and introduces a Mata function that implements it. It also computes analytical first-order derivatives of the simulated probability with respect to the variables and the variance–covariance parameters.

Keywords: st0102, GHK, maximum simulated likelihood, Monte Carlo, quasi–Monte Carlo, importance sampling, number-theoretic statistics

1 Introduction

Estimation of parameters for probit models, such as the multinomial probit ([R] **asmprobit**) and the multivariate probit (`mvprobit`, Cappellari and Jenkins [2003, 2005, 2006]), requires numerical integration to approximate the multivariate normal (MVN) distribution. There are several techniques to carry out this task, but this paper concentrates on the Geweke–Hajivassiliou–Keane (GHK) MVN simulator (Geweke [1989], Hajivassiliou and McFadden [1998], and Keane [1994]), an importance-sampling technique that samples recursively from truncated normals after a Cholesky transformation. I review this technique and introduce a Mata function, `ghk()`, available in Stata 9.1, that implements it. This function also computes the first-order derivatives of the simulated probability with respect to the variables and the variance–covariance parameters as described by Bolduc (1999).

The following sections will discuss the GHK algorithm as a series of transformations as presented by Genz (1992), a technique that has a more number-theoretic flavor. Following is a discussion of the algorithm as an importance-sampling technique to numerical integration of the MVN density function (Geweke 1989). Section 3 gives an overview of the first-order analytic derivatives of the simulated probability (Bolduc 1999). I present Mata code that implements the simulator followed by documentation of the Mata function `ghk()`, introduced in release 9.1 of Stata. Finally, I give an example using the function `ghk()` to estimate the parameters of a multinomial probit model.

2 GHK MVN simulator

Let $\mathbf{X} = (X_1, X_2, \dots, X_m)$ be distributed multivariate normal, $\mathbf{X} \sim \text{MVN}_m(\mathbf{0}, \mathbf{\Sigma})$, with density $\phi_m(\mathbf{x}|\mathbf{\Sigma})$ and distribution function

$$\begin{aligned}\Phi_m(\mathbf{x}|\mathbf{\Sigma}) &= \int_{-\infty}^{x_1} \dots \int_{-\infty}^{x_m} \phi_m(\mathbf{y}|\mathbf{\Sigma})\ d\mathbf{y} \\ &= \frac{1}{(2\pi)^m |\mathbf{\Sigma}|^{\frac{1}{2}}} \int_{-\infty}^{x_1} \cdots \int_{-\infty}^{x_m} \exp\left(\tfrac{-1}{2}\mathbf{y}'\mathbf{\Sigma}^{-1}\mathbf{y}\right) d\mathbf{y}\end{aligned}$$

One approach to (quasi) Monte Carlo integration of the MVN density function is to transform the domain of integration to the unit interval of dimension m, $C^m = [0,1]^m$,

$$\Phi_m(\mathbf{x}|\mathbf{\Sigma}) = \int_0^1 \dots \int_0^1 f(\mathbf{u})\ d\mathbf{u} \tag{1}$$

I will postpone defining the function f until the next section, but assuming f can be obtained, then for a set of n vectors $\{\tilde{\mathbf{u}}_i \in [0,1)^m, i = 1, \dots, n\}$ with elements that are either independently distributed uniform on $[0,1)$, $\text{U}(0,1)$, or a deterministic set of points that have a uniform spread on $[0,1)$, the approximation to $\Phi_m(\mathbf{x}|\mathbf{\Sigma}) = \int_{C^m} f(\mathbf{u})\ d\mathbf{u}$ is $1/n \sum_{i=1}^n f(\tilde{\mathbf{u}}_i)$. I next discuss these transformations as presented by Genz (1992).

2.1 Transformations

The discussion presented here for transforming the domain of integration to the unit interval is taken from Genz (1992). First, define $\phi_m(\cdot) = \phi_m(\cdot\ |\mathbf{I}_m)$ and $\Phi_m(\cdot) = \Phi_m(\cdot\ |\mathbf{I}_m)$, where $\mathbf{I}_m$ is the $m \times m$ identity matrix. $\Phi_1(\cdot)$ is the univariate standard normal distribution. I start by first taking the Cholesky factorization of the variance–covariance $\mathbf{\Sigma} = \mathbf{T}\mathbf{T}'$ and making the change of variables $\mathbf{y} = \mathbf{T}\mathbf{z}$ so that $d\mathbf{y} = |\mathbf{T}|d\mathbf{z} = |\mathbf{\Sigma}|^{1/2}d\mathbf{z}$. The bounds of integration are $-\infty < \mathbf{T}\mathbf{z} \leq \mathbf{x}$ and can be rewritten as $-\infty < z_1 \leq x_1/t_{11} = b_1$ and $-\infty < z_i \leq (x_i - \sum_{j=1}^{i-1} t_{ij}z_j)\Big/t_{ii} = b_i(z_1, \dots, z_{i-1})$, for $i = 2, \dots, m$. Now I have

$$\begin{aligned}\Phi_m(\mathbf{x}|\mathbf{\Sigma}) = \Phi_m(\mathbf{b}) &= (2\pi)^{-m} \int_{-\infty}^{b_1} \int_{-\infty}^{b_2(z_1)} \cdots \int_{-\infty}^{b_m(z_1,\dots,z_{m-1})} \exp\left(\tfrac{-1}{2}\mathbf{z}'\mathbf{z}\right)\ d\mathbf{z} \\ &= \int_{-\infty}^{b_1} \int_{-\infty}^{b_2(z_1)} \cdots \int_{-\infty}^{b_m(z_1,\dots,z_{m-1})} \phi_m(\mathbf{z})\ d\mathbf{z}\end{aligned} \tag{2}$$

Using the transformations $z_i = \Phi_1^{-1}(v_i)$, $i = 1, \dots, m$, and noting that $d\mathbf{z} = d\mathbf{v}/\phi_m(\mathbf{z})$, the integral simplifies to

$$\Phi_m(\mathbf{x}|\boldsymbol{\Sigma}) = \int_0^{a_1} \int_0^{a_2(v_1)} \cdots \int_0^{a_m(v_1,\ldots,v_{m-1})} d\mathbf{v}$$

where $a_1 = \Phi_1(x_1/t_{11})$ and $a_i(v_1, \ldots, v_{i-1}) = \Phi_1 \left[\left\{ x_i - \sum_{j=1}^{i-1} t_{ij} \Phi_1^{-1}(v_j) \right\} \Big/ t_{ii} \right]$, for $i = 2, \ldots, m$. Finally, substituting $v_i = a_i u_i$, $u_i \in [0, 1)$, $i = 1, \ldots, m-1$, yields

$$\Phi_m(\mathbf{x}|\boldsymbol{\Sigma}) = a_1 \int_0^1 a_2 \int_0^1 \cdots \; a_m \int_0^1 d\mathbf{u}$$

The approximation of (1) at a single point $\mathbf{u} = (u_1, \ldots, u_{m-1})'$ can be implemented by the following algorithm:

1. $a_1 = f_1 = \Phi_1(x_1/t_{11})$
2. for $i = 2, \ldots, m$, compute
   a. $z_{i-1} = \Phi^{-1}(u_{i-1}\; a_{i-1})$
   b. $a_i = \Phi_1 \left\{ \left(x_i - \sum_{j=1}^{i-1} t_{ij} z_j \right) \Big/ t_{ii} \right\}$
   c. $f_i = f_{i-1}\; a_i$

Upon completion $f(\mathbf{u}) = f_m$, where the $u_i \in [0, 1)$ are either pseudorandom uniform variates or from a deterministic sequence that has uniform coverage on $[0, 1)$.

In the next section, I review the GHK MVN simulator as an importance-sampling technique (Geweke 1989).

2.2 GHK as an importance-sampling technique

In importance sampling, we use a distribution, F, with density $f(\cdot)$ and support $(-\infty, b)$, that is similar to Φ_1 and easy to sample from. Then for the univariate case

$$\int_{-\infty}^{b} \phi_1(z)\; dz = \int_{-\infty}^{b} \frac{\phi_1(z)}{f(z)} f(z)\; dz = E_F \left\{ \frac{\phi_1(Z)}{f(Z)} \right\} \approx \frac{1}{n} \sum_{i=1}^{n} \frac{\phi_1(z_i^*)}{f(z_i^*)}$$

where $Z \sim F$ and the z_i^*, $i = 1, \ldots, n$ are n draws from distribution F. Our application of importance sampling uses the (singly) truncated normal distribution for F, with density $f(z) = \phi_1(z)/\Phi_1(b)$ (Johnson, Kotz, and Balakrishnan 1994).

Using (2), I can write

$$
\begin{aligned}
\Phi_m(\mathbf{b}) = \int_{-\infty}^{b_1} \phi_1(z_1) \int_{-\infty}^{b_2(z_1)} \phi_1(z_2) \cdots \int_{-\infty}^{b_m(z_1,\ldots,z_{m-1})} \phi_1(z_m)\ d\mathbf{z} \\
= \Phi_1(b_1) \int_{-\infty}^{b_1} f(z_1)\ \Phi_1\left\{b_2(z_1)\right\} \int_{-\infty}^{b_2(z_1)} f(z_2)\ \Phi_1\left\{b_3(z_1,z_2)\right\}\ \cdots \\
\cdots\ \Phi_1\left\{b_m(z_1,\ldots,z_{m-1})\right\} \int_{-\infty}^{b_m(z_1,\ldots,z_{m-1})} f(z_m)\ d\mathbf{z} \\
\approx \frac{1}{n}\sum_{i=1}^{n} \Phi_1\left(b_1\right) \cdot \prod_{j=2}^{m} \Phi_1\left\{b_j(z^*_{i1},\ldots,z^*_{i,j-1})\right\}
\end{aligned}
$$

where in the last equation the z^*_{ij}, $j = 1,\ldots,m-1$, are draws from the (singly) truncated normal distribution. Truncated normal variates in this case are obtained by

$$z^*_{i1} = \Phi^{-1}\left\{u_{i1}\Phi\left(b_1\right)\right\}$$

and

$$z^*_{ij} = \Phi^{-1}\left[u_{ij}\Phi\left\{b_j(z^*_{i1},\ldots,z^*_{i,j-1})\right\}\right]$$

for $j = 2,\ldots,m-1$, where the u_{ij} are draws from the $\mathrm{U}(0,1)$ distribution.

Next I outline an algorithm in Mata to carry out the Monte Carlo integration.

2.3 Mata implementation of the GHK algorithm

I will use the results from section 2.1 to create Mata code to implement the GHK simulator since the series of transformations presented by Genz (1992) is conceptually programmatic.

In Mata, I would like to avoid looping and take advantage of Mata's vector operator, `:` (see [M-2] **op_colon**). Because of the recursive nature of the simulator, I will need to loop over the dimensions, but I can process all n simulated values in a vector algorithm and gain some efficiency at the expense of memory consumption. In the code snippet below, assume that `V` $= \boldsymbol{\Sigma}$ and that `x` is a vector of length m containing the upper bounds of integration.

```
z = J(n,m-1,0)
a = J(n,1,x[1])
p = J(n,1,1)
T = cholesky(V)'
for (j=1; j<=m; j++) {
   if (j > 1) a = J(n,1,x[j]) - z[,1::(j-1)]*T[1::(j-1),j]

   a = normal(a:/T[j,j])
   p = p:*a
   if (j < m) z[.,j] = invnormal(uniform(n,1):*a)
}
pr = sum(p)/n
```

Upon completion `pr`, a scalar, contains the simulated probability.

3 First-order derivatives

Computational speed of maximum simulated-likelihood estimates using the GHK MVN simulator is greatly enhanced with the ability to compute analytical first-order derivatives of the simulated probability with respect to the variables and the variance–covariance parameters. This enhancement can be surpassed only by analytical second-order derivatives, but to my knowledge these have not been derived to date. The clever derivation of the first-order derivatives presented here is taken from Bolduc (1999).

Here I deal only with the univariate standard normal distribution functions, so $\Phi(\cdot) \equiv \Phi_1(\cdot)$ and $\phi(\cdot) \equiv \phi_1(\cdot)$. I will concentrate on one simulated probability since the derivatives for the n simulated values will simply be the mean of the individual derivatives. Also for ease of notation, I will denote a vector of the first i elements of $\mathbf{z}^*$ as $\mathbf{z}^*_{(i)} = (z^*_1, \ldots, z^*_i)'$ and $\mathbf{t}_{(i)} = (t_{11}, t_{21}, \ldots, t_{i1}, t_{22}, \ldots, t_{ii})' = \text{vech}(\mathbf{T}_{(i)})$, where vech$(\cdot)$ is the half-vectorization operator (Lütkepohl 1996) and $\mathbf{T}_{(i)}$ is the submatrix of $\mathbf{T}$ that includes the first i rows and columns. Finally, let $\boldsymbol{\delta} = (\mathbf{x}',\text{vech}(\mathbf{T})')'$.

I denote $p(\cdot)$ as the simulated probability and then recall from section 2 that

$$p(\boldsymbol{\delta}) = \Phi\left\{b_1(x_1, t_{11})\right\} \cdot \prod_{i=2}^{m} \Phi\left\{b_i(\mathbf{x}_{(i)}, \mathbf{z}^*_{(i-1)}, \mathbf{t}_{(i)})\right\}$$

where the z^*_is are recursive functions of $\mathbf{x} = (x_1, \ldots, x_m)'$ and $\mathbf{T}$:

$$z^*_1(x_1, t_{11}) = \Phi^{-1}\left\{u_1 \cdot \Phi(x_1/t_{11})\right\} \text{ and}$$

$$z^*_j(\mathbf{x}_{(j)}, \mathbf{z}^*_{(j-1)}, \mathbf{t}_{(j)}) = \Phi^{-1}\left[u_j \cdot \Phi\left\{b_j(\mathbf{x}_j, \mathbf{z}^*_{(j-1)}, \mathbf{t}_{(j)})\right\}\right]$$

Here I have added the dependency of the functions $b(\cdot)$ on the x's and the t_{ij}'s since $b_1(x_1, t_{11}) = x_1/t_{11}$ and $b_i(\mathbf{x}_{(i)}, \mathbf{z}^*_{(i-1)}, \mathbf{t}_{(i)}) = \left\{x_i - \sum_{j=1}^{i-1} t_{ij} z^*_j(\mathbf{x}_{(j)}, \mathbf{z}^*_{(j-1)}, \mathbf{t}_{(j)})\right\}/t_{ii}$ for $i = 2, \ldots, m$.

First, the simulated probability can be expressed as $p = \exp\left[\log\left\{\prod_{j=1}^{m} \Phi(b_j)\right\}\right]$ so

$$\begin{aligned}\frac{\partial p}{\partial \delta_l} &= p \sum_{j=1}^{m} \frac{\partial \log\left\{\Phi(b_j)\right\}}{\partial \delta_l} \\ &= p \sum_{j=1}^{m} \frac{\phi(b_j)}{\Phi(b_j)} \frac{\partial b_j}{\partial \delta_l} \qquad (3)\end{aligned}$$

I continue using the chain rule for $x_i = \delta_l$

$$\frac{\partial b_j}{\partial x_i} = \begin{cases} \dfrac{1}{t_{jj}} & \text{if } i = j \\ -\dfrac{1}{t_{jj}} \displaystyle\sum_{k=1}^{j-1} t_{jk} \frac{\partial z^*_k}{\partial x_i} & \text{if } i < j \\ 0 & \text{otherwise} \end{cases} \qquad (4)$$

I end this chain, recursing back to the last equations

$$\begin{aligned}\frac{\partial z_k^*}{\partial x_i} &= \frac{u_k \cdot \phi(b_k)}{\phi\left[\Phi^{-1}\left\{u_k \cdot \Phi(b_k)\right\}\right]}\frac{\partial b_k}{\partial x_i} \\ &= \frac{u_k \cdot \phi(b_k)}{\phi\left(z_k^*\right)}\frac{\partial b_k}{\partial x_i} \end{aligned} \tag{5}$$

Now I continue the chain rule in (3) with $t_{ik} = \delta_l$, $i \geq k$:

$$\frac{\partial b_j}{\partial t_{ik}} = \begin{cases} -\dfrac{x_1}{t_{11}^2} & \text{if } i = j = k = 1 \\ \dfrac{-x_j + \sum_{h=1}^{j-1} t_{jh} \cdot z_h^*}{t_{jj}^2} & \text{if } i = j = k > 1 \\ -\dfrac{z_k^*}{t_{jj}} & \text{if } i = j > k \\ -\dfrac{1}{t_{jj}}\displaystyle\sum_{h=1}^{j-1} t_{jh}\frac{\partial z_h^*}{\partial t_{ik}} & \text{if } j > i \geq k \end{cases}$$

or equivalently

$$\frac{\partial b_j}{\partial t_{ik}} = \begin{cases} -\dfrac{b_j}{t_{jj}} & \text{if } i = j = k \\ -\dfrac{z_k^*}{t_{jj}} & \text{if } i = j > k \\ -\dfrac{1}{t_{jj}}\displaystyle\sum_{h=1}^{j-1} t_{jh}\frac{\partial z_h^*}{\partial t_{ik}} & \text{if } j > i \geq k \end{cases} \tag{6}$$

As before, this chain ends by recursing back to the last equations

$$\frac{\partial z_h^*}{\partial t_{ik}} = \frac{u_h \cdot \phi(b_h)}{\phi\left(z_h^*\right)}\frac{\partial b_h}{\partial t_{ik}} \tag{7}$$

3.1 Mata implementation of the derivatives

To implement the derivative computations, I must return to the GHK simulator algorithm and save some intermediate computations.

(*Continued on next page*)

```
z = J(n,m-1,0)
a = J(n,1,x[1])
b = J(n,m,0)
dp = J(n,m,0)
dz = J(n,m-1,0)
p = J(n,1,1)
T = cholesky(V)'
for (j=1; j<=m; j++) {
   if (j > 1) a = J(n,1,x[j]) - z[,1::(j-1)]*T[1::(j-1),j]
   a = a:/T[j,j]
   b[.,j] = a
   f = normalden(a)
   a = normal(a)
   dp[.,j] = f:/a
   p = p:*a
   if (j < m) {
      u = uniform(n,1)
      a = invnormal(u:*a)
      dz[.,j] = u:*f:/normalden(a)
      z[.,j] = a
   }
}
pr = sum(p)/n
```

Upon completion, the $n \times m$ matrix `dp` contains the ratios $\phi(b_j)/\Phi(b_j)$, $j = 1, \ldots, m$, and the $n \times m - 1$ matrix `dz` contains $u_j \cdot \phi(b_j)/\phi\left(z_j^*\right)$, $j = 1, \ldots, m - 1$.

I continue with the Mata code to compute $\partial p/\partial x_j$, $j = 1, \ldots, m$. Here I use a recursive Mata function, `dbdx`, and perform vector computations to all n simulation points.

(*Continued on next page*)

```
real colvector dbdx(real scalar j, real scalar i, real matrix dz, real matrix T)
{
   real scalar k, n
   real colvector dxk, dx

   n = rows(dz)
                                    /* equations (4) and (5) */
   dx = J(n,1,1/T[j,j])
   if (j > i) {
      dxk = J(n,1,0)
      for (k=i; k<j; k++) dxk = dxk - J(n,1,T[j,k]):*dz[,k]:*dbdx(k,i,dz,T)
      dx = dx:*dxk
   }
   return(dx)
}
T = T'
dx = J(1,m,0)
for (l=1; l<=m; l++) {
   dxl = J(n,1,0)
                                    /* equation (3) */
   for (j=l; j<=m; j++) dxl = dxl + dp[,j]:*dbdx(j,l,dz,T)
   dx[l] = sum(p:*dxl)/n
}
```

Recall that I transposed T to upper triangular, so I first return it to lower triangular. Upon completion, the vector `dx` of length m contains $\partial p/\partial x_j$, $j = 1, \dots, m$.

I next present Mata code to compute $\partial p/\partial \text{vech}(T)$. Again I use a recursive function, `dbdt`, to carry out the vector computations.

```
real colvector dbdt(real scalar j, real scalar i, real scalar k, real matrix dz,
                    real matrix z, real matrix b, real matrix T)
{
   real scalar k, n
   real colvector dt

   n = rows(dz)
                                    /* equations (6) and (7) */
   if (i==k && j==k) dt = -b[,j]
   else if (i==j) dt = -z[,k]
   else {
      dt = J(n,1,0)
      for (h=i; h<j; h++) dt = dt - J(n,1,T[j,h]):*dz[,h]:*dbdt(h,i,k,dz,z,b,T)
   }
   return(dt:/J(n,1,T[j,j]))
}

l = 0
dt = J(1,m*(m+1)/2,0)
for (k=1; k<=m; k++) {
   for (i=k; i<=m; i++) {
      dtl = J(n,1,0)
                                    /* equation (3) */
      for (j=i; j<=m; j++) dtl = dtl + dp[,j]:*dbdt(j,i,k,dz,z,b,T)
      dt[++l] = sum(p:*dtl)/n
   }
}
```

Upon completion, the vector `dt` contains $\partial p/\partial \text{vech}(T)$.

I promised that I would produce the first-order derivatives of the simulated probability with respect to $\text{vech}(\boldsymbol{\Sigma})$. Again Bolduc (1999) provides the matrix differential calculus to carry out this task, which is easily implemented using Mata. First, I review the calculus.

I can express $\text{vech}(\boldsymbol{\Sigma})$ as

$$\begin{aligned}\text{vech}(\boldsymbol{\Sigma}) = \text{vech}(\mathbf{T}\mathbf{T}') &= \mathbf{L}_m' \left(\mathbf{T} \otimes \mathbf{I}_m\right) \mathbf{L}_m \ \text{vech}(\mathbf{T}) \\ &= \mathbf{L}_m' \left(\mathbf{I}_m \otimes \mathbf{T}\right) \mathbf{K}_m \ \text{vech}(\mathbf{T})\end{aligned}$$

where $\otimes$ is the Kronecker product (Magnus and Neudecker 1988) and the $m \cdot m \times m(m+1)/2$ matrices $\mathbf{L}_m$ and $\mathbf{K}_m$ are such that $\text{vec}(\mathbf{T}) = \mathbf{L}_m \ \text{vech}(\mathbf{T})$ and $\text{vec}(\mathbf{T}') = \mathbf{K}_m \ \text{vech}(\mathbf{T})$. Here $\text{vec}(\mathbf{T})$ is the vector operator that returns a vector of length $m \cdot m$ containing the matrix columns stacked on top of one another (Magnus and Neudecker [1988] or Lütkepohl [1996]). The Jacobian of the Cholesky transformation is

$$\frac{\partial \text{vech}(\boldsymbol{\Sigma})}{\partial \text{vech}(\mathbf{T})'} = \mathbf{L}_m' \left\{ \left(\mathbf{T} \otimes \mathbf{I}_m\right) \mathbf{L}_m + \left(\mathbf{I}_m \otimes \mathbf{T}\right) \mathbf{K}_m \right\} = \mathbf{A} \tag{8}$$

so the derivatives I seek are

$$\begin{aligned}\frac{\partial p}{\partial \text{vech}(\boldsymbol{\Sigma})'} &= \frac{\partial p}{\partial \text{vech}(\mathbf{T})'} \frac{\partial \text{vech}(\mathbf{T})}{\partial \text{vech}(\boldsymbol{\Sigma})'} \\ &= \frac{\partial p}{\partial \text{vech}(\mathbf{T})'} \mathbf{A}^{+}\end{aligned}$$

where $\mathbf{A}^{+}$ is the Moore–Penrose (MP) inverse of $\mathbf{A}$ (Magnus and Neudecker 1988).

I carry out these last computations by first introducing the Mata function `duplower()` that computes $\mathbf{L}_m$ and $\mathbf{K}_m$, but in vector form. The matrices $\mathbf{L}_m$ and $\mathbf{K}_m$ are matrices of 0s with a single 1 in each row and column (or none at all, since there are only $m(m-1)/2$ of them). Just as with permutation matrices, I can gain efficiency by using vectors that address the column (row) indices of the matrix instead of using matrix multiplication. (See also [M-1] **permutation** for more on permutation matrices.) For example, the matrix multiplication $\mathbf{A}\mathbf{L}_3$, where $\mathbf{A}$ is 9×9 and

$$\mathbf{L}_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

can be implemented in Mata as `A[,vL]` where `vL` is

```
      1   2   3   4   5   6
    +-------------------------+
  1 |   1   2   3   5   6   9 |
    +-------------------------+
```

```
void duplower(real scalar m, real vector vL, real vector vK)
{
   real scalar i, j, k, l

   vK = vL = J(m*(m+1)/2,1,.)
   k = l = 0
   for (j=1; j<=m; j++) {
      for (i=j; i<=m; i++) {
         vL[++l] = ++k
         vK[l] = (i-1)*m+j
      }
      k = k + j
   }
}
vK = vL = J(0,1,.)
duplower(m, vL, vK)
dv = dt*qrinv(((T#I(m))[,vL]+(I(m)#T)[,vK])[vL,])
```

Here I carry out the matrix computation for $\partial p/\partial \mathrm{vech}(\boldsymbol{\Sigma})$ in one line of Mata code making use of Mata's Kronecker operator `#` (see [M-2] **op_kronecker**). I could have used the Mata function `pinv()` ([M-5] **pinv()**), which uses singular value decomposition to carry out the MP inverse with more accuracy, but I chose the QR-based function `qrinv()` ([M-5] **qrinv()**), reasoning that it may be a bit faster.

I next introduce the Mata function `ghk()`, which carries out the computations discussed thus far.

4 The Mata function ghk()

4.1 Syntax

real scalar `ghk(`*real vector x*, *real matrix V*, *real vector opt*, *real scalar rank*`)`

real scalar `ghk(`*real vector x*, *real matrix V*, *real vector opt*, *real scalar rank*,
real rowvector dfdx, *real rowvector dfdv*`)`

(*Continued on next page*)

4.2 Description

`ghk(`x`,` V`,` *opt*`,` *rank*`)` returns a real scalar containing the simulated value of the MVN distribution with variance–covariance V at the point x. *opt* is a vector of length 4 containing the following simulator options:

$opt[1] = 1$	Halton sequence
2	Hammersley sequence
3	uniform pseudorandom sequence
$opt[2] > 0$	number of points to use in the simulation
$opt[3] > 0$	index of the first draw (optional for Halton or Hammersley sets)
$opt[4] \neq 0$	use antithetic draws (optional)

On return, *rank* contains the rank of V.

`ghk(`x`,` V`,` *opt*`,` *rank*`,` *dfdx*`,` *dfdv*`)` does the same thing but also returns the first-order derivatives of the simulated probability with respect to x in *dfdx* and the simulated probability derivatives with respect to `vech(`V`)` in *dfdv*, where `vech()` is the half-vectorization operator (see [M-5] **vec()**).

4.3 Remarks

Halton and Hammersley point sets are composed of deterministic sequences on [0,1) and, for sets of dimension less than 10, generally provide better coverage than the uniform pseudorandom sequences.

Antithetic draws effectively double the number of points and reduce the variability of the simulated probability. For draw u, the antithetic draw is $1 - u$.

If you are using `ghk()` in a likelihood evaluator for `ml`, be sure to use the same sequence with each call to the likelihood evaluator. For a uniform pseudorandom sequence ($opt[1] = 3$), you must set the uniform random-number generator seed, `uniformseed()`, to the same value with each call to the likelihood evaluator. If you are using the Halton or Hammersley sets, you will want to keep the sequences going with each call to `ghk()` within one likelihood evaluation. This task is done by first initializing $opt[3] = 1$ on entering the `ml` likelihood evaluator and computing the increment $opt[3] = opt[3] + opt[2]$ after each call to `ghk()` to compute the likelihood of each observation.

4.4 Conformability

`ghk(`x`,` V`,` *opt*`,` *rank*`)`:

input:

x:	$1 \times m$ or $m \times 1$
V:	$m \times m$ (symmetric, positive definite)
opt:	1×4 or 4×1

output:

result:	1×1
rank:	1×1

`ghk(`x`,` V`,` *opt*`,` *rank*`,` *dfdx*`,` *dfdv*`)`:

input:

x:	$1 \times m$ or $m \times 1$
V:	$m \times m$ (symmetric, positive definite)
opt:	1×4 or 4×1

output:

result:	1×1
rank:	1×1
dfdx:	$1 \times m$
dfdv:	$1 \times m(m+1)/2$

4.5 Diagnostics

The maximum dimension, m, is 20.

The V must be symmetric and preferably positive definite. `ghk()` will not terminate if V is not positive definite, where the returned value of *rank* will be less than `rows(`V`)`. The function uses a Cholesky routine that pivots out the rows and columns of V (as well as elements of x) that make V indefinite. The corresponding elements of *dfdx* and *dfdv* are zero. Although the MVN distribution is not defined for indefinite variance–covariance matrices, an indefinite V can occur early in an `ml` optimization, and this flexible behavior may allow the optimization process to continue. If this is not desirable behavior in your program, add the line

```
if (rank < rows(V)) exit(3353)
```

just after the call to `ghk()`.

5 A multinomial probit example

The Mata function `ghk()` can be used in any Stata `ml` likelihood evaluator that involves the MVN distribution. I will demonstrate its use in estimating the regression and variance–covariance parameters of the multinomial probit model. This implementation of the model will be somewhat simpler than that of the Stata program `asmprobit`.

I first give a quick review of the key features of the multinomial probit model that I will have to address to interface with the `ghk()` function. See *Methods and Formulas* in [R] **asmprobit** for more details of the model or, better yet, Train (2003), chapter 5. I then develop another Mata function to implement the multinomial probit–specific computations. Finally, I produce the ado-likelihood evaluator that is `ml` callable. For simplicity, the ado-code will be specific to my example, which will estimate the multinomial probit parameters for the travel data demonstrated in [R] **asmprobit**.

5.1 A multinomial probit model synopsis

The multinomial probit model applied to a discrete choice problem with m alternatives requires the evaluation of the $m-1$ dimension MVN distribution function. One alternative is chosen as the base alternative to normalize location, giving us $m-1$ latent variable equations. For simplicity of discussion, I will assume that the base alternative is the one associated with index m. I can express the equations for one case (individual) as

$$\begin{aligned} y_j &= (\mathbf{x}_j - \mathbf{x}_m)'\boldsymbol{\beta} + \mathbf{z}'\boldsymbol{\alpha}_j + \boldsymbol{\epsilon}_j \\ &= \boldsymbol{\eta}_j + \boldsymbol{\epsilon}_j \end{aligned} \tag{9}$$

for $j = 1, \ldots, m-1$, and where the $\mathbf{x}_j$ are alternative-specific variables that vary with each alternative, $\boldsymbol{\beta}$ are their associated coefficients, and the $\mathbf{z}$ are the case-specific variables that are constant for each alternative (but not for each case), with the $\boldsymbol{\alpha}_j$ their associated coefficients (one set for each alternative, less the base). The vector $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_{m-1})'$ are distributed $\text{MVN}_{m-1}(\mathbf{0}, \boldsymbol{\Sigma})$. The alternative chosen is the one associated with index j such that y_j is the maximum and $y_j > 0$, $j = 1, \ldots, m-1$. If all $y_j < 0$, then alternative m is chosen.

For this model, I estimate the p parameters of $\boldsymbol{\beta}$ and the $q(m-1)$ parameters of the $\boldsymbol{\alpha}_j = (\alpha_{j1}, \ldots, \alpha_{jq})'$, $j = 1, \ldots m-1$. The covariance matrix $\boldsymbol{\Sigma}$ has only $m(m-1)/2-1$ identifiable parameters, or one less than the unique values of $\boldsymbol{\Sigma}$, or $\text{vech}(\boldsymbol{\Sigma}) = (\sigma_{ij}), i \geq j$. I will fix $\sigma_{11} = 1$, thereby normalizing the scale.

5.2 A Mata multinomial probit function

Here I describe a Mata function, `mnp()`, that manipulates latent variables computed by my `ml` likelihood evaluator to create the input matrices for the Mata `ghk()` function. It will also take the first-order derivatives computed by the `ghk()` function and make the necessary Jacobian transformations to create the score variables needed by the likelihood evaluator.

The equations defined in my call to `ml` will provide the `mnp()` function with m variables, one for each alternative, such that $v_j = \mathbf{x}_j'\widehat{\boldsymbol{\beta}} + \mathbf{z}'\widehat{\boldsymbol{\alpha}}_j$, for $j = 1, \ldots, m-1$ and $v_m = \mathbf{x}_m'\widehat{\boldsymbol{\beta}}$. I need to transform these variables into vectors that have the form of (9). Assume that I have an $m \times 1$ vector $\mathbf{v} = (v_1, \ldots, v_m)'$ for an individual case. I can generate an $m-1 \times m$ matrix $\mathbf{N}_4$ such that

$$\mathbf{N}_4 = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{pmatrix}$$

so that $\boldsymbol{\eta} = \mathbf{N}_4\mathbf{v}$. Further assume that the individual chose the second alternative, so I need to compute the probability that this choice is made given the current estimates $\widehat{\boldsymbol{\beta}}$ and $\widehat{\boldsymbol{\alpha}}_j$, $j = 1, \ldots, m-1$. That is, I need to compute the probability that y_2 is the largest and that it is not less than zero. This probability can be expressed as $\Pr\{y_1 - y_2 \leq 0, y_3 - y_2 \leq 0, -y_2 \leq 0\}$. I transform the variables by using the 3×3 matrix $\mathbf{M}_2$

$$\mathbf{M}_2 = \begin{pmatrix} 1 & -1 & 0 \\ 0 & -1 & 1 \\ 0 & -1 & 0 \end{pmatrix}$$

so I have $\mathbf{w} = \mathbf{M}_2\mathbf{N}_4\mathbf{v} = \mathbf{N}_2\mathbf{v}$, where the matrix $\mathbf{N}_j$ is an $m-1 \times m$ matrix constructed from the $m \times m$ identity matrix with the jth column replaced with a vector of -1s and the jth row removed. The matrix $\mathbf{M}_j$ is then the matrix $\mathbf{N}_j$ with the fourth column removed. The error terms for the transformed variables $\mathbf{w}$, $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_{m-1})'$, say, are $\textsc{mvn}_{m-1}(\mathbf{0}, \mathbf{M_2 \Sigma M_2'})$, and the probability statement is now $\Pr\{\gamma_1 \leq -w_1, \ldots, \gamma_{m-1} \leq -w_{m-1}\}$. I can estimate this probability with the Mata function `ghk()`.

I need to ensure that the estimated variance–covariance matrix remains positive definite throughout the optimization. One safe way is to optimize on the lower triangular elements of the $m-1 \times m-1$ Cholesky-factored covariance matrix, $\mathbf{T}$, $\boldsymbol{\Sigma} = \mathbf{TT}'$. Moreover, I have transformed the variance–covariance, $\boldsymbol{\Omega} = \mathbf{M}_c\boldsymbol{\Sigma}\mathbf{M}_c'$, where c is the index of the choice made, $c = 1, \ldots, m$. The Mata function `ghk()` will be computing $\partial p / \partial \mathrm{vech}(\boldsymbol{\Omega})'$, and I will use these derivatives to compute $\partial p / \partial \mathrm{vech}(\mathbf{T})'$. Again I use the results from Bolduc (1999).

$$\frac{\partial p}{\partial \mathrm{vech}(\mathbf{T})'} = \frac{\partial p}{\partial \mathrm{vech}(\boldsymbol{\Omega})'} \mathbf{L}_{m-1}' \left\{ \left(\mathbf{M}_c\mathbf{T} \otimes \mathbf{M}_c\right) \mathbf{L}_{m-1} + \left(\mathbf{M}_c \otimes \mathbf{M}_c\mathbf{T}\right) \mathbf{K}_{m-1} \right\} \tag{10}$$

Finally, I need to compute the first-order derivatives of p with respect to $\mathbf{v}$ from those returned from `ghk()` (i.e., $\partial p / \partial \mathbf{w}'$) as

$$\frac{\partial p}{\partial \mathbf{v}'} = -\frac{\partial p}{\partial \mathbf{w}'} \mathbf{N}_c \tag{11}$$

(*Continued on next page*)

Below I define the Mata function `mnp()`.

```
void function mnp(string scalar sX, string scalar schoice, string scalar sT,
                  real scalar meth, real scalar draws, string scalar spr,|
                  string scalar sXscr, string scalar sTscr)
{
   real scalar i, m, m1, n, c, todo, rank
   real rowvector opt, g, s
   real colvector x, choice, pr
   real vector vL, vK
   real matrix V, R, M, X, T, MT, Xscr, Tscr
   pointer(real matrix) vector N

   pragma unset X
   st_view(X,.,tokens(sX))
   pragma unset pr
   st_view(pr,.,spr)
   n = rows(X)
   m = cols(X)
   T = st_matrix(sT)
   m1 = rows(T)
   V = T*T'

   pragma unset rank
   pragma unset g
   pragma unset s
   pragma unset choice
   st_view(choice,.,schoice)

   if (todo=(args()==9)) {
      pragma unset Xscr
      pragma unset Tscr
      st_view(Xscr,.,tokens(sXscr))
      st_view(Tscr,.,tokens(sTscr))
      pragma unset vL
      pragma unset vK
      duplower(m1, vL, vK)
   }
   N = J(m,1,NULL)
   for (c=1; c<=m; c++) N[c] = &mnp_N(m,c)
   opt = (meth, draws, 1, 0)
   for (i=1; i<=n; i++) {
      c = choice[i]
      x = -(*N[c])*X[i,.]'
      M = (*N[c])[,1::m1]
      R = M*V*M'
      if (todo > 0) {
         pr[i] = ghk(x,R,opt,rank,g,s)
         MT = M*T
                                        /* equation (10) */
         Tscr[i,.] = s*((MT#M)[,vL]+(M#MT)[,vK])[vL,]
                                        /* equation (11) */
         Xscr[i,.] = -g*(*N[c])
      }
      else pr[i] = ghk(x,R,opt,rank)

      if (meth < 3) opt[3] = opt[3] + draws
   }
}
```

In mnp() the function mnp_N(), with scalar arguments m and c, computes the matrix $\mathbf{N}_c$ for choice c of m alternatives. This function is simple enough that I do not display the code.

Next I create an ado-likelihood evaluator that is callable from ml and uses my Mata function mnp().

5.3 Multinomial probit maximum simulated likelihood using ml

I will write an ado-likelihood evaluator that is specific to this problem, using the same travel data that demonstrate [R] **asmprobit**. These data contain information on 210 individuals' choices of travel mode between Sydney and Melbourne. The four choices are air, train, bus, and car, with indices 1, 2, 3, and 4, respectively, and are stored in the variable mode. I will use two alternative-specific variables: travelcost, a measure of generalized cost of travel that is equal to the sum of in-vehicle cost and a wage-like measure times the amount of time spent traveling; and termtime, the terminal time, which is zero for car transportation. Household income, income, is a case-specific variable. Finally, the variable id is the integer variable identifying each case, and the variable choice is a byte (0/1) variable indicating which choice is made.

To understand the logic of the likelihood evaluator, I first introduce the data reshaping, initial estimate computations, and the ml call. I reshape the dataset so that it is in wide format, but I use clogit to compute initial estimates for my call to ml first. I chose the wide format for this demonstration since the data are balanced, there are four alternatives for each case, and the ml likelihood evaluator tools mleval and mlvecsum are more suited for the wide data format. The result is that less data manipulation and ado-code are required in the likelihood evaluator so that I do not distract from the use of the ghk() function itself. The program asmprobit expects the data in long format since it must handle unbalanced data, where the number of alternatives varies with each case, so this example also provides a different perspective on estimating the parameters of a multinomial probit model in Stata.

The program clogit finds the regression estimates that maximize the conditional logit likelihood. Here I condition the number of choices made by each case identified in the variable id, the group variable in clogit jargon. Since there is one choice for each group, the conditional logit likelihood is easily evaluated and although it has different distributional assumptions, it produces good initial estimates for the multinomial probit model. The initial estimates are further improved by scaling by the variance of the extreme value distribution, $\pi^2/6$. Below is the ado-code.

```
use http://www.stata-press.com/data/r9/travel, clear

/* alternative 4 (car) is base alternative */
forvalues i=1/3 {
   gen int inc`i' = cond(mode==`i',income,0)
   gen byte cons`i' = (mode==`i')
   local model `model' inc`i' cons`i'
}
qui clogit choice termtime travelcost `model', group(id)
```

```
/* scale by extreme value variance */
mat b0 = e(b)*sqrt(6)/c(pi)
drop `model'
```

The case-specific variable `income` does not vary within `id`, so I need `clogit` to compute an income regression coefficient for each level of `mode` except for car transportation, the base alternative. To do so I need to do some legwork and generate new variables that are the product of `income` with indicators for the first three modes of travel. Moreover, the generated indicator variables will be included in the `clogit` *varlist* to obtain alternative-specific intercepts. The Stata program `asmprobit` takes care of this detail for you.

Next I reshape the dataset to wide format. Once done, there will be one record for each case. In the *varlist* for `reshape`, I include the alternative-specific and the dependent variables that are used in the model. Four new variables for each variable in *varlist* will be generated, each prefixed with the variable name followed by the indices `1`, `2`, `3`, and `4`. I must drop the remaining two alternative-specific variables that exist in the dataset, `invehiclecost` and `traveltime`, before the call to `reshape`. Finally, I generate a new integer variable, `choice`, containing the index of the choice made by each individual.

```
drop invehiclecost traveltime
reshape wide choice termtime travelcost, i(id) j(mode)

gen int choice = 1 if choice1 == 1
replace choice = 2 if choice2 == 1
replace choice = 3 if choice3 == 1
replace choice = 4 if choice4 == 1

drop choice1 choice2 choice3 choice4
```

The model specification in the call to `ml` will contain nine equations: the first four are for each mode of travel and the last five are for the Cholesky matrix parameters. The equations for modes air, train, and bus transportation include `termtime`, `travelcost`, and `income`, and by default `ml` will include a constant term for each. The equation for mode car transportation includes only `termtime` and `travelcost`, and I add the option `noconstant` since it is the base alternative. Below is the ado-code.

```
mat b0 = (b0[1,1..2],b0[1,3..4],b0[1,1..2],b0[1,5..6],b0[1,1..2], ///
          b0[1,7..8],b0[1,1..2],J(1,5,0))

/* alternative-specific variables  */
constraint 1 [air]termtime1 = [train]termtime2
constraint 2 [train]termtime2 = [bus]termtime3
constraint 3 [bus]termtime3 = [car]termtime4

constraint 4 [air]travelcost1 = [train]travelcost2
constraint 5 [train]travelcost2 = [bus]travelcost3
constraint 6 [bus]travelcost3 = [car]travelcost4
```

```
ml model d1 travel_lf                               ///
   (air: choice=termtime1 travelcost1 income)       ///
   (train: choice=termtime2 travelcost2 income)     ///
   (bus: choice=termtime3 travelcost3 income)       ///
   (car: choice=termtime4 travelcost4, nocons)      ///
   /t21 /t31 /t22 /t32 /t33, init(b0,copy) max      ///
   constraints(1-6) search(off) tech(nr) collinear ///
   shownrtol
```

The additional equation specifications `/t21 /t31 /t22 /t32 /t33` are for the parameters of the Cholesky-factored variance–covariance. There are 5, not $3 \cdot 4/2 = 6$, since there are only $m(m-1)/2-1$ identifiable variance–covariance parameters. I fix $\sigma_{11} = 1$ to scale the estimates, and this restriction means that $t_{11} = 1$. I also use the log transform for the diagonal elements `t22` and `t33`, so their initial estimates are $0 = \log(1)$.

Finally, I constrain the alternative-specific parameter estimates to be equal for each variable since there is only one parameter for each alternative-specific variable. I must also use the option `collinear` to prevent `ml` from dropping `termtime4` (it is all zeros), and I know that the constraints will do the rank reduction that is necessary. This last set of legwork is a side effect of having the dataset in wide format.

In the call to `ml`, the likelihood evaluator is identified as `travel_lf`. I will present this ado-code next with a discussion.

5.4 The ml likelihood evaluator

The `ml` likelihood evaluator, `travel_lf`, acts as an interface between ado and Mata. The Mata functions discussed thus far, `ghk()` and `mnp()`, are doing most of the computations. The ado-program `travel_lf` will compose the factored variance–covariance matrix from the transformed parameter estimates and compute the latent variables by using the `ml` helper program `mleval`. It then identifies the latent variables, score variables, choice variable, and the factored variance–covariance matrix to `mnp()` by passing their names as strings. Below is the ado-code.

```
program define travel_lf
   args todo b lnf g negH sair strain sbus scar st21 st31 st22 st32 st33

   if ‘todo’ > 0 {
      tempvar st11
      local xscrs ‘sair’ ‘strain’ ‘sbus’ ‘scar’
      local tscrs ‘st21’ ‘st31’ ‘st22’ ‘st32’ ‘st33’

      foreach scr of varlist ‘xscrs’ ‘tscrs’ {
         qui replace ‘scr’ = 0
      }
      qui gen ‘st11’ = 0
   }
   tempname t21 t31 t22 t32 t33
   tempvar lf air train bus car
   local lvars ‘air’ ‘train’ ‘bus’ ‘car’
   local tpars ‘t21’ ‘t31’ ‘t22’ ‘t32’ ‘t33’
```

```
        local k = 0
        foreach l in `lvars' {
           mleval `l' = `b', eq(`++k')
        }
        foreach t in `tpars' {
           mleval `t' = `b', scalar eq(`++k')
        }
        scalar `t22' = exp(`t22')
        scalar `t33' = exp(`t33')

        tempname T
        mat `T' = (1, 0, 0 \ `t21', `t22', 0 \ `t31',`t32', `t33')
        qui gen double `lf' = .
        if `todo' > 0 {
           mata: mnp("`lvars'", "$ML_y1", "`T'", 2, 200, "`lf'", ///
                     "`xscrs'", "`st11' `tscrs'")

           foreach scr of varlist `xscrs' `tscrs' {
              qui replace `scr' = `scr'/`lf'
           }
           qui replace `st22' = `st22'*`t22'
           qui replace `st33' = `st33'*`t33'
        }
        else mata: mnp("`lvars'", "$ML_y1", "`T'", 2, 200, "`lf'")

        qui replace `lf' = ln(`lf')
        mlsum `lnf' = `lf'

        if `todo' > 0 {
           tempname g1
           local k = 0
           cap mat drop `g'
           foreach scr of varlist `xscrs' `tscrs' {
              mlvecsum `lnf' `g1' = `scr', eq(`++k')
              matrix `g' = (nullmat(`g'),`g1')
           }
        }
     end
```

Upon return from the function `mnp()`, some final computations are required. First, the function `mnp()` caches the simulated probabilities in the temporary variable identified as `` `lf' ``. I must log `` `lf' `` before using the `ml` utility `mlsum`, summing the elements of `` `lf' `` to give the log simulated likelihood for the current parameter estimates. Because I take the log of the simulated probability, more computations are required for all the score variables. Also I am using the log transform for the two diagonal elements of the Cholesky matrix, identified in the temporary names `` `t22' `` and `` `t33' ``, to ensure that the diagonal elements of the Cholesky matrix are positive. Therefore, more computations for the score variables `` `st22' `` and `` `st33' `` are required before using the `ml` utility `mlvecsum` to produce the gradient vector `` `g' ``.

5.5 The run

Below are the segments of the log generated from my do-code driver. The GHK simulator here used 200 points from the Hammersley set (Fang and Wang 1994).

```
. use http://www.stata-press.com/data/r9/travel, clear
  (output omitted)
. ml model d1 travel_lf (air: choice=termtime1 travelcost1 income)
>                 (train: choice=termtime2 travelcost2 income)
>                 (bus: choice=termtime3 travelcost3 income)
>                 (car: choice=termtime4 travelcost4, nocons)
>                 /t21 /t31 /t22 /t32 /t33, init(b0,copy) max
>                 constraints(1-6) search(off) tech(nr) collinear
>                 shownrtol
Iteration 0:   log likelihood = -211.89315
Iteration 1:   log likelihood = -200.02487  (not concave)
Iteration 2:   log likelihood = -193.38753
  (output omitted)
Iteration 9:   log likelihood = -190.09418
               g inv(H) g'    =  5.545e-20
. ml display, neq(4) noheader
 ( 1)  [air]termtime1 - [train]termtime2 = 0
 ( 2)  [train]termtime2 - [bus]termtime3 = 0
 ( 3)  [bus]termtime3 - [car]termtime4 = 0
 ( 4)  [air]travelcost1 - [train]travelcost2 = 0
 ( 5)  [train]travelcost2 - [bus]travelcost3 = 0
 ( 6)  [bus]travelcost3 - [car]travelcost4 = 0
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
air						
termtime1	-.0306287	.0084317	-3.63	0.000	-.0471546	-.0141028
travelcost1	-.0079355	.0019771	-4.01	0.000	-.0118107	-.0040604
income	.0039867	.0064421	0.62	0.536	-.0086396	.0166131
_cons	1.489141	.6563072	2.27	0.023	.202803	2.77548
train						
termtime2	-.0306287	.0084317	-3.63	0.000	-.0471546	-.0141028
travelcost2	-.0079355	.0019771	-4.01	0.000	-.0118107	-.0040604
income	-.0197278	.0081871	-2.41	0.016	-.0357742	-.0036814
_cons	1.945315	.4947447	3.93	0.000	.9756328	2.914996
bus						
termtime3	-.0306287	.0084317	-3.63	0.000	-.0471546	-.0141028
travelcost3	-.0079355	.0019771	-4.01	0.000	-.0118107	-.0040604
income	-.0063689	.004835	-1.32	0.188	-.0158453	.0031075
_cons	1.44271	.3779816	3.82	0.000	.7018798	2.18354
car						
termtime4	-.0306287	.0084317	-3.63	0.000	-.0471546	-.0141028
travelcost4	-.0079355	.0019771	-4.01	0.000	-.0118107	-.0040604

```
. mat b = e(b)
. mat R = (1,0,0\b[1,15],b[1,16],0\exp(b[1,17]),b[1,18],exp(b[1,19]))
. mat li R
R[3,3]
            c1          c2          c3
r1           1           0           0
r2   .09313086   .07669325           0
r3   .70495283   .30962656   .34010134
. mat V = R*R'
```

```
. mat li V

symmetric V[3,3]
           r1          r2          r3
r1          1
r2  .09313086  .01455521
r3  .70495283  .08939913  .70849602

. mat D  = syminv(cholesky(diag(vecdiag(V))))

. mat R = D*V*D

. mat li R

symmetric R[3,3]
           r1          r2          r3
r1          1
r2  .77194141           1
r3   .8375126   .8803499           1
```

6 Discussion

This paper presented some of the mathematical background to the GHK MVN simulator and the first-order derivatives of the simulated probability, along with Mata code to numerically implement them. The Mata function `ghk()` released in version 9.1 of Stata is really a Mata wrapper to C code that implements generating the sequences and computing the simulated probability and the derivatives. By doing so, the function `ghk()` in Stata/MP will use multiple processors (see the performance graphs for `asmprobit` in the white paper found at http://stata.com/statamp/report.pdf). Furthermore, the C code also pivots the wider bounds of integration to the inside, thereby moving the larger values of x to the end of the vector and pivoting the corresponding rows and columns of V. This pivot is a recommendation made in Genz (1992). The pivoted GHK algorithm reduces the variability of the simulated probability dramatically, which probably reflects improved accuracy. With the implementation of pivoting, we have a pivot matrix, $\mathbf{P}$, and we simply modify (8) as

$$\frac{\partial \text{vech}(\mathbf{\Sigma})}{\partial \text{vech}(\mathbf{T})'} = \mathbf{L}_m' \left\{ \left(\mathbf{PT} \otimes \mathbf{P} \right) \mathbf{L}_m + \left(\mathbf{P} \otimes \mathbf{PT} \right) \mathbf{K}_m \right\} \tag{12}$$

To demonstrate the advantage of the GHK simulator with pivoting, I ran the same example with a version of the GHK simulator that does not use pivoting and compared the maximum simulated log likelihood between the two techniques. These results are presented in table 1.

Table 1: Comparison of simulated log likelihood when using pivoting

No. of points	Pivoting Yes	Pivoting No
200	−190.094	−190.038
400	−190.096	−190.067
600	−190.093	−190.078
800	−190.096	−190.082
1,000	−190.092	−190.085
1,200	−190.094	−190.086
2,000	−190.093	−190.089
2,200	−190.093	−190.088

Here we see that the log simulated-likelihood from the GHK algorithm with pivoting is stable in the second decimal place even at 200 points, whereas without pivoting it does not stabilize at the second decimal place until after about 800 points. Moreover, the no-pivoting algorithm is slowly approaching a −190.09 asymptote, the value that the GHK algorithm with pivoting achieved with only 200 points.

The function `ghk()` also uses a Cholesky factoring function that will use pivoting in case the $m \times m$ matrix V is not numerically positive definite. For an indefinite $m \times m$ symmetric V with rank $r < m$ it will reduce V to an $r \times r$ matrix and compute the Cholesky factor of the reduced matrix as well as reduce x to a vector of length r. `ghk()` then computes the simulated probability of the reduced vector. The Cholesky-factored matrix T in (12) will be $m \times m$ with only the first r rows and columns nonzero. The derivative vectors *dfdx* and *dfdv* are of length m and $m(m+1)/2$, respectively, with the first r and $r(r+1)/2$ elements nonzero.

I have found that in the first iterations of the optimization it is common to have an (numerically) indefinite V. This finding occurs after `ml` has computed the direction of the next step and is searching for a step length (as indicated when the `todo` macro is 0 following a call to the likelihood evaluator with `todo` equal to 1). The pivoting by `ghk()` to deal with the indefinite variance–covariance is one way to handle the problem, but I could also `capture` an error thrown by Mata's `cholesky()` function and force `ml` to step-halve by returning a missing value for the log likelihood.

For example, if I use a Mata GHK function that does not use pivoting when running the multinomial probit example from section 5, the estimated variance–covariance computed by `R = M*V*M'` in the `mnp()` function is not positive definite when trying to make the first step in the optimization. I must `capture` the Mata error code 3352 (singular matrix) from the `cholesky()` function and force `ml` to step-halve. Below is the snippet of code implementing the `capture`.

```
cap mata: mnp("`lvars'", "$ML_y1", "`T'", 2, $NPTS, "`lf'")
   if _rc == 3352 {
      if ($PDMSG) di in gr "Covariance is not positive definite. Step halving"
      scalar `lnf' = .
      exit 0
   }
   else if (_rc) exit _rc
```

Eventually the variance–covariance estimates produce a positive-definite R.

Here I chose to use the Hammersley set, a variant of the Halton set, in the GHK algorithm. The uniform coverage of the Hammersley set on C^{m-1} is superior to the pseudorandom sequences and a bit better than the Halton set for low dimensional problems (Fang and Wang [1994] or Niederreiter [1992]).

Instead of using `ml display` to show the estimates of the (transformed) Cholesky parameters, I chose to compute the estimate of the 3×3 covariance matrix $\mathbf{\Sigma}$ and the correlation matrix. If I had expressed the latent variable equations in (9) as m equations, one for each alternative, instead of $m-1$ equations, the matrix $\mathbf{\Sigma}$ could be interpreted as the variance–covariance matrix of the latent errors of the first $m-1$ alternatives differenced with that of the mth (the base alternative). This procedure is the differenced parameterization described in the Stata online help for `asmprobit`. These variance–covariances and correlations are difficult to interpret, but we know that if the m equation model errors were independent and homoskedastic (the independence of irrelevant alternatives property) then the variance–covariance (and correlation) matrix would have the form

$$\mathbf{\Sigma} = \begin{pmatrix} 1 & .5 & .5 \\ .5 & 1 & .5 \\ .5 & .5 & 1 \end{pmatrix} = \mathbf{N}_4 \begin{pmatrix} .5 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 \\ 0 & 0 & .5 & 0 \\ 0 & 0 & 0 & .5 \end{pmatrix} \mathbf{N}_4'$$

The correlation estimates all exceed 0.5 and the variances vary dramatically from the scale value of 1. Although I did not compute estimate standard errors, this result does indicate a violation of the independence of irrelevant alternatives property.

7 References

Bolduc, D. 1999. A practical technique to estimate multinomial probit models in transportation. *Transportation Research, Part B* 33: 63–79.

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

———. 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.

Fang, K.-T., and Y. Wang. 1994. *Number-theoretic Methods in Statistics*. London: Chapman & Hall.

Genz, A. 1992. Numerical computation of multivariate normal probabilities. *Journal of Computational and Graphical Statistics* 1: 141–149.

Geweke, J. 1989. Bayesian inference in econometric models using Monte Carlo integration. *Econometrica* 57: 1317–1339.

Hajivassiliou, V., and D. McFadden. 1998. The method of simulated scores for the estimation of LDV models. *Econometrica* 66: 863–896.

Johnson, N. L., S. Kotz, and N. Balakrishnan. 1994. *Continuous Univariate Distributions*, vol. 1. 2nd ed. New York: Wiley.

Keane, M. P. 1994. A computationally practical simulation estimator for panel data. *Econometrica* 62: 95–116.

Lütkepohl, H. 1996. *Handbook of Matrices*. Chichester, UK: Wiley.

Magnus, J. R., and H. Neudecker. 1988. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Chichester, UK: Wiley.

Niederreiter, H. 1992. *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia: SIAM.

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

About the author

Richard Gates is a senior statistical software engineer at StataCorp.