The Stata Journal

The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go "beyond the Stata manual" in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

For more information on the *Stata Journal*, including information for authors, see the webpage

http://www.stata-journal.com

The *Stata Journal* is indexed and abstracted in the following:

- CompuMath Citation Index®
- Current Contents/Social and Behavioral Sciences®
- RePEc: Research Papers in Economics
- Science Citation Index Expanded (also known as SciSearch®)
- Scopus™
- Social Sciences Citation Index®

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

The Stata Journal (2012)
12, Number 1, pp. 29–44

Scrambled Halton sequences in Mata

Stanislav Kolenikov
University of Missouri
Columbia, MO
kolenikovs@missouri.edu

Abstract. In this article, I discuss the need for Halton sequences and discuss the Mata implementation of scrambling of Halton sequences, providing several examples of scrambling procedures.

Keywords: st0244, Halton sequence, quasi–Monte Carlo, scrambled Halton sequence

1 Halton sequences

Many areas of scientific simulation rely on methods for generating random or pseudorandom numbers from the interval $[0, 1)$ or the unit cube $[0, 1)^s$ in s dimensions. A common application is multivariate integration. Suppose an integral $I(f) = \int_{[0,1)^s} f(\mathbf{u})\, \mathrm{d}\mathbf{u}$ needs to be evaluated. Integration over different domains can be handled by appropriate transformations as is done in the Geweke–Hajivassiliou–Keane estimator (see [M-5] **ghk( )**). In the context of random variables, integration may be handled by the transformation from the range of the random variables into quantiles that are marginally distributed as $U[0, 1]$. A sequence $\mathbf{u}_i$ $(i = 1, \ldots, N;\ 0 \leq u_{il} < 1;\ l = 1, \ldots, s)$ can be used to approximate the integral $I(f)$ as

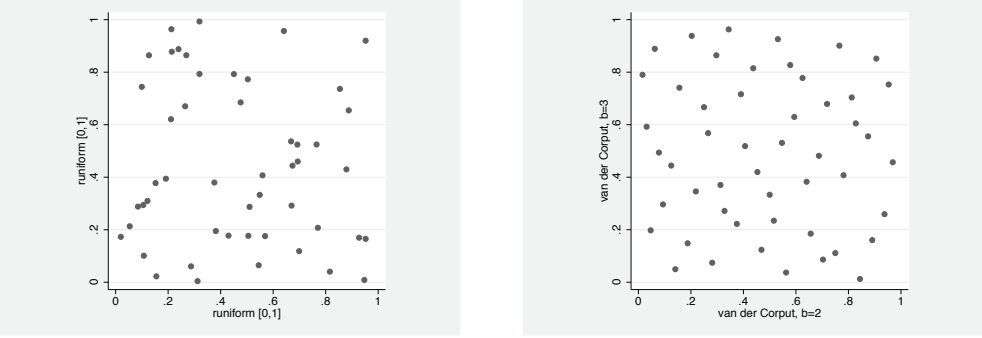
$$\widehat{I} = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{u}_i) \tag{1}$$

provided that the numbers $\mathbf{u}_1, \mathbf{u}_2, \ldots$ behave in a way similar to uniform numbers. In fact, multivariate integration does not require the draws to be independent, and a regular distribution of the finite sequence over $[0, 1)^s$ plays a more important role. Although the usual pseudorandom numbers, such as those generated by Stata's `runiform()` random-number generator, will serve the purpose, somewhat more accurate results can be achieved with quasi–Monte Carlo methods (Niederreiter 1992; Lemieux 2009).

Let us compare the two panels in figure 1. The plot on the left uses the pseudorandom numbers generated by `runiform()`. As is natural for random data, some points are clumped together and some areas are left relatively unpopulated. In contrast, the plot on the right, generated using the Halton sequence (described in the next paragraph), does not show such artifacts and provides a highly uniform coverage of the space. From the point of view of numeric integration, if the points are clumped together, they will likely have close values of the evaluated functions, adding little additional information to (1). Computing time would be better spent spreading points farther apart. At the same time, in the areas that are left sparsely populated, some important features of

the integrand may be missed. Thus the Halton sequence will likely work better in applications that require a uniform spread of points but not their independence.

```
set obs 50
set seed 10203
generate rx = runiform()
generate ry = runiform()
scatter ry rx, aspect(1)
graph export Halton1a.eps, replace
generate qmcx = .
generate qmcy = .
mata: st_store(., ("qmcx qmcy"), halton(50, 2))
scatter qmcy qmcx, aspect(1)
graph export Halton1b.eps, replace
```



(a) (b)

Figure 1. Random and quasi–Monte Carlo points in $[0,1)^2$

To define the Halton sequence, we need to make a short excursion into number theory and the presentation of numbers in different bases. For an integer $b \geq 2$, positive integers can be represented in base b using the relation

$$\begin{aligned} n &= a_0(n) + a_1(n)b + a_2(n)b^2 + \cdots + a_k(n)b^k \\ &= \sum_{j=0}^{k} a_j(n)b^j, \quad 0 \leq a_j(n) < b \quad \text{for all} \quad j = 0, \ldots, k \end{aligned}$$

which can be represented graphically as $(\overline{a_k a_{k-1} \ldots a_1 a_0})_b$. The upper bar indicates that this is a sequence of digits in base b rather than a product of numbers (its dependence on n suppressed to reduce clutter). A *radical inverse function in base b* is a reflection of this representation with respect to the decimal point: $(\overline{0.a_0 a_1 \ldots a_{k-1} a_k})_b$, that is,

$$\phi_b(n) = \sum_{j=0}^{k} a_j(n) b^{-j-1}$$

Note that $\phi_b(n) \in [0,1)$. For an integer $b \geq 2$, the *van der Corput sequence in base b* is the sequence $\{\phi_b(n)\}_{n=0}^{\infty}$. A generalization of van der Corput sequences into multiple

dimensions is provided by the *Halton sequence* (Halton 1960). For a dimension s, let $b_1, \ldots, b_s$ be integers greater than 1. Then the *Halton sequence in the bases* $b_1, \ldots, b_s$ is the sequence

$$x_n = \{\phi_{b_1}(n), \ldots, \phi_{b_s}(n)\}$$

The quantitative description of the regularity properties of sequences such as pseudorandom numbers or Halton sequences is provided by various forms of discrepancies. They measure the difference between the frequency with which the sequence passes through a certain set and the Lebesgue measure (multidimensional volume) $\lambda(\cdot)$ of this set. Let the number of hits of set B by sequence P be denoted by

$$A(B, P) = \sum_{n=1}^{\mathcal{N}} \mathbb{1}_B(x_n)$$

where $\mathbb{1}_B(x)$ is an indicator function of a set B, $\mathbb{1}_B(x) = 1$ if $x \in B$, and $\mathbb{1}_B(x) = 0$ otherwise. For a collection of sets $\mathcal{B}$, define the discrepancy as the greatest difference between the Lebesgue measure and the proportion of hits over all sets in $\mathcal{B}$:

$$D_{\mathcal{N}}(\mathcal{B}; P) = \sup_{B \in \mathcal{B}} \left| \frac{A(B, P)}{\mathcal{N}} - \lambda(B) \right|$$

The family of s-dimensional boxes $\mathcal{I}^* = \{[0, v_1) \times \cdots \times [0, v_s) : v_1, \ldots, v_s < 1\}$ starting from the origin and the family $\mathcal{I} = \{[u_1, v_1) \times \cdots \times [u_s, v_s) : 0 \le u_l < v_l < 1 \text{ for all } l = 1, \ldots, s\}$ of arbitrary s-dimensional boxes in $[0, 1)^s$ are two leading examples:

$$D^*_{\mathcal{N}}(P) = D_{\mathcal{N}}(\mathcal{I}^*; P), \quad \mathcal{I}^* = \left\{ \prod_{l=1}^{s} [0, u_l) : 0 \le u_l \le 1 \right\}$$

$$D_{\mathcal{N}}(P) = D_{\mathcal{N}}(\mathcal{I}; P), \quad \mathcal{I} = \left\{ \prod_{l=1}^{s} [u_l, v_l) : 0 \le u_l \le v_l \le 1 \right\}$$

$D^*_{\mathcal{N}}(P)$ is usually referred to as the star discrepancy and $D_{\mathcal{N}}(P)$ as the extreme discrepancy. The relation between them is $D^*_{\mathcal{N}}(P) \le D_{\mathcal{N}}(P) \le 2^s D^*_{\mathcal{N}}(P)$.

It has been established (Niederreiter 1992) that the asymptotic discrepancy of the van der Corput sequence in base b is

$$\overline{\lim_{\mathcal{N}\to\infty}} \frac{\mathcal{N} D^*_{\mathcal{N}}(S_b)}{\ln \mathcal{N}} = \overline{\lim_{\mathcal{N}\to\infty}} \frac{\mathcal{N} D_{\mathcal{N}}(S_b)}{\ln \mathcal{N}} = \begin{cases} \dfrac{b^2}{4(b+1)\ln b}, & b \in 2\mathbb{N} \\ \dfrac{b-1}{4\ln b}, & b \in 2\mathbb{N}+1 \end{cases}$$

Assuming that S is the Halton sequence in pairwise relatively prime bases $b_1, \ldots, b_s$, then (Atanassov 2004)

$$
\begin{aligned}
D^*_{\mathcal{N}}(S) &< \frac{s}{\mathcal{N}} + \frac{1}{\mathcal{N}} \prod_{l=1}^{s} \left(\frac{b_l - 1}{2l \ln b_l} \ln \mathcal{N} + \frac{b_l + 1}{2} \right) \\
&= A(b_1, \ldots, b_s) \mathcal{N}^{-1} \ln^s \mathcal{N} + O\left(\mathcal{N}^{-1} \ln^{s-1} \mathcal{N} \right) \\
A(b_1, \ldots, b_s) &= \frac{1}{s! 2^s} \prod_{l=1}^{s} \frac{b_l - 1}{\ln b_l} \to 0 \quad \text{as} \quad s \to \infty
\end{aligned} \tag{2}
$$

This discrepancy features a better asymptotic rate $O(\mathcal{N}^{-1} \ln^s \mathcal{N})$ than the random Monte Carlo rate $O(\mathcal{N}^{-1/2})$ that arises from the central limit theorem.

Implementation of the Halton sequence is available in Mata (Drukker and Gates 2006) with the `halton()` suite of functions. These functions allow for the generation of van der Corput sequences with an arbitrary base, as well as the generation of Halton sequences skipping the initial entries. See [M-5] **halton( )**.

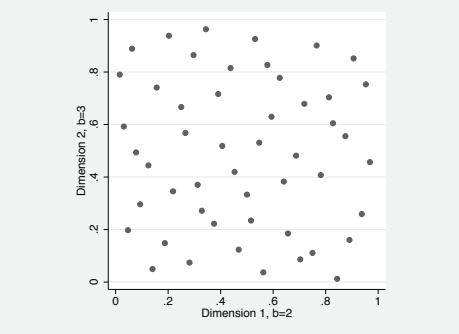
The Halton sequence is one of the examples of low-discrepancy sequences studied in the area of quasi–Monte Carlo sampling. Other sequences that provide the same asymptotic discrepancy rate are known. Among these, the Halton sequence is the only one in which an extra point $\mathbf{u}_{N+1}$ can be added quite naturally. With other sequences, the points need to be added in large blocks to improve the actual finite-length discrepancy.

2 Scrambled Halton sequences

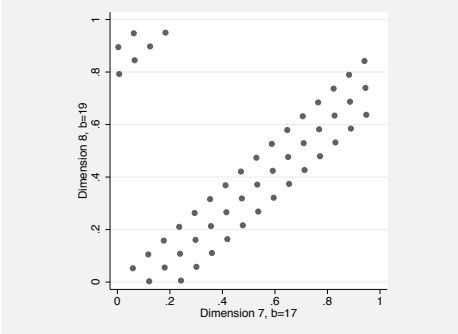
Each of the margins of the Halton sequence, that is, the van der Corput sequence in base b_l, is organized in blocks of consecutive points at the distance $1/b_l$ from each other, somewhere in the corresponding intervals $[0, 1/b_l)$, $[1/b_l, 2/b_l)$, ... This behavior generates an undesirable artifact observed in short sequences with larger dimensions.

Consider figure 2. The left panel reproduces the scatterplot of the first 50 elements of Halton sequences in the first two dimensions corresponding to $b_1 = 2$ and $b_2 = 3$. It shows the regularity that we are seeking. The right panel gives the scatterplot of the first 50 elements in dimensions 7 and 8 corresponding to the primes $b_7 = 17$ and $b_8 = 19$. Contrary to the plausible expectations of regularity, the points form a very strong pattern on parallel lines with slope $17/19 = 0.895$. Is it possible to avoid this problem while retaining the desirable regularity properties of the Halton sequence?

```
clear
set obs 50
foreach p of numlist 2 3 5 7 11 13 17 19 {
generate h`p´ = .
local hlist `hlist´ h`p´
}
mata: st_store(., tokens("`hlist´"), halton(50, 8))
scatter h3 h2, aspect(1)
graph export Halton2a.eps, replace
scatter h19 h17, aspect(1)
graph export Halton2b.eps, replace
```



(a) Dimensions 1 and 2, $b_1 = 2$ and $b_2 = 3$



(b) Dimensions 7 and 8, $b_7 = 17$ and $b_8 = 19$

Figure 2. Initial elements of the Halton sequences in $[0, 1)^2$

The answer is yes. The asymptotic discrepancy properties of Halton sequences are retained when an extra *scrambling* operation is performed when the radical inverse is taken. Namely, let $\sigma_l : \{0, 1, \ldots, b_l - 1\} \mapsto \{0, 1, \ldots, b_l - 1\}$ be a permutation of numbers $0, 1, \ldots, b_l - 1$. If the radical inverse is taken of $[\sigma_l\{a_0(n)\}, \sigma_l\{a_1(n)\}, \ldots, \sigma_l\{a_k(n)\}]_b$ instead of $(\overline{a_0 a_1 \ldots a_k})_b$, it produces a number

$$\phi_{b,\sigma_l}(n) = \sum_{j=0}^{\infty} \sigma_l\{a_j(n)\} b^{-j-1}$$

The *scrambled Halton sequence*, with the given set of permutations $\sigma_1(\cdot), \ldots, \sigma_s(\cdot)$, is then obtained by collecting the scrambled sequences over margins.

What is a "good" permutation, and how can one be found? Certain permutations were shown to improve the multiplicative constant A in front of the leading term in asymptotic discrepancy (2). They are, however, very cumbersome to derive and code. Let us instead motivate several simple options.

One of the reasons that prime numbers are chosen as the bases for Halton sequences is that for any prime p, the residues $0, 1, \ldots, p - 1$ form an Abelian group with respect to the operation $(x + y) \mod p$ (Dummit and Foote 2004). Any nonzero element can act as the generator of the group, meaning that if x is a nonzero element, then the set of

numbers $\{0, x \mod p, 2x \mod p, \ldots, (p-1)x \mod p\}$ coincides with $\{0, 1, \ldots, p-1\}$. Hence, a simple permutation rule is provided by

$$\sigma(a) = xa \mod p, \quad x \neq 0, \quad a \in \{0, 1, \ldots, p-1\} \tag{3}$$

Let us introduce the necessary Mata tools, and then give some examples.

3 Syntax

The Mata library `lscrhalton.mlib` distributed with this article contains the following functions:

- Mata functions to generate prime numbers:
  - `PrimesUpTo(`n`)` returns all prime numbers less than or equal to n.
  - `FirstPrimes(`n`)` returns the first n primes.
- Mata functions to convert bases:
  - `tolongbase(`b`, `n`)` returns a real vector containing the b-digits of representation of n in base b.
  - `fromlongbase(`b`, `v`)` returns a real value corresponding to the vector v.
  - The functions `tolongbase()` and `fromlongbase()` are mutually reciprocal.
- Mata functions to generate scrambled Halton sequences:
  - `ScrHalton(`n`, `d`, &f(), `*optional arguments*`)` generates a scrambled Halton sequence of length n in d dimensions. Scrambling is performed by the user-written function `f()`, whose syntax must conform to specifications given below.
  - `_ScrHalton(`k`, `b`, &f(), `*optional arguments*`)` returns a scalar between 0 and 1 obtained by expanding the integer k in base b, applying the scrambling procedure defined in the user-written function `f()`, and computing the radical inverse of the result. Up to five additional parameters are supported and passed through to `f()`.

The user-written function `f()` must satisfy the following requirements. Its syntax is `f(`a_j`, `b`, `j`, `*optional arguments*`)`, where a_j is the integer number between 0 and $b-1$ to be scrambled, b is the base of the transformation, j is the position of the digit in the representation of an underlying number in base b, and *optional arguments* can be passed through to `f()`. It will return the scrambled number between 0 and $b-1$, and it will leave the value 0 intact.

4 Examples

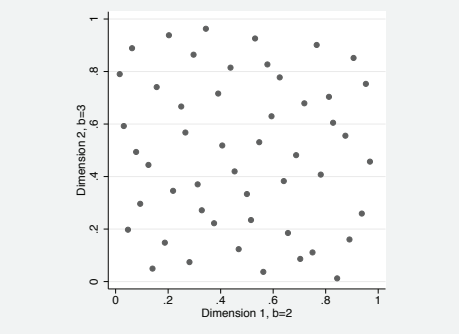
Examples given in this section operate by replacing the call to Stata's standard `halton()` function in the code snippet from page 33. Thus we shall give the body of the scrambler function and an example of the call. The complete code is provided in the do-file `scrambled-halton.do` distributed with the article. The data preparation commands and the plotting commands are the same as in the aforementioned code. When reproducing these examples, make sure that all the dimensions of the resulting scrambled Halton sequences match those of the data. In this example, there are 50 observations on 8 variables.

▷ Example 1: Square-root scrambler

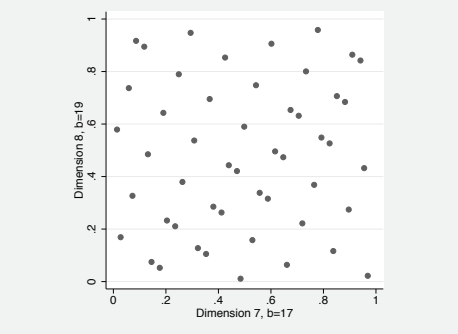
As the first example, let us use $\lfloor\sqrt{b_j}\rfloor$ as the generator of the set $\{1,\dots,b_j-1\}$, that is, as the multiplier x in (3), with $p=b_j$. Here $\lfloor\cdot\rfloor$ is Stata's `floor()` function, which computes the nearest smaller integer of the argument.

```
real scalar SqrtScrambler(real scalar k, real scalar b,
                          real scalar j) {
   return(mod(k*floor(sqrt(b)), b))
}
st_store(., tokens("h2 h3 h5 h7 h9 h11 h13 h17 h19"),
   ScrHalton(50,8,&SqrtScrambler())
```

The resulting plot of the selected pairs of dimensions is given in figure 3. Both graphs show reasonably regular behavior, although the graph of the higher dimensions with greater bases still demonstrates the points on parallel lines.



(a) Dimensions 1 and 2, $b_1=2$ and $b_2=3$



(b) Dimensions 7 and 8, $b_7=17$ and $b_8=19$

Figure 3. Initial elements of the Halton sequences in $[0,1)^2$ scrambled with the square-root multiplier

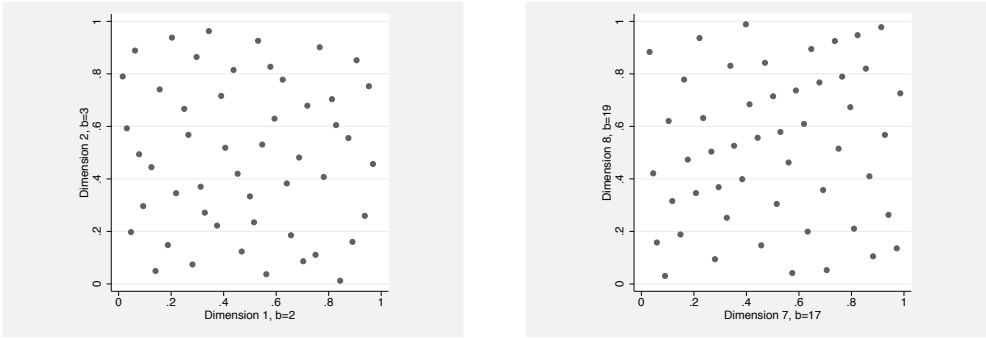
◁

▷ Example 2: Negative square-root scrambler

An equally reasonable option is to use $\{-\lfloor\sqrt{b_j}\rfloor\} \bmod b_j = b_j - \lfloor\sqrt{b_j}\rfloor$ as the generator. As a slight modification, the nearest integer to the square root rather than the nearest smaller integer is used in the code below:

```
real scalar MinusSqrtScrambler(real scalar k, real scalar b,
                               real scalar j) {
   return(mod(k*(b-round(sqrt(b),1)), b))
}
st_store(., tokens("h2 h3 h5 h7 h9 h11 h13 h17 h19"),
   ScrHalton(50,8,&MinusSqrtScrambler())
```

The resulting plot of the selected pairs of dimensions is given in figure 4, with results similar to those of figure 3. Arguably, the performance of the scrambled sequence for the higher two primes is improved, although some regular patterns remain.



(a) Dimensions 1 and 2, $b_1 = 2$ and $b_2 = 3$

(b) Dimensions 7 and 8, $b_7 = 17$ and $b_8 = 19$

Figure 4. Initial elements of the Halton sequences in $[0, 1)^2$ scrambled with the negative square-root multiplier

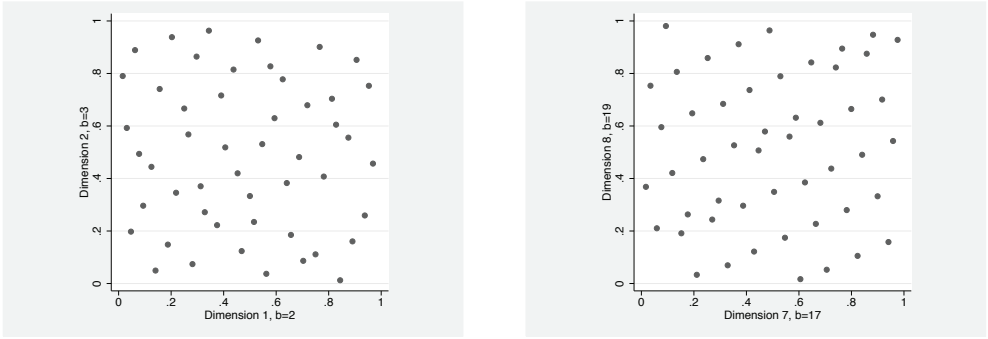
◁

▷ Example 3: Random multiplier scrambler

To demonstrate the flexibility of the code in supplying additional options, let us consider the following weak version of randomization: draw the multiplier x in (3) uniformly from $\{1, 2, \ldots, b_j - 1\}$. To ensure that the same permutation is applied consistently to all elements of the scrambled Halton sequence, the random seed is reset each time the procedure is called. This seed acts as a parameter of the procedure and is supplied to `ScrHalton()`.

```
real scalar RandomMultipleScrambler (real scalar k, real scalar b,
                                     real scalar j, real scalar seed) {
   real scalar mult;

   rseed(seed)
   mult = ceil(runiform(1,1)*(b-1))
   return(mod(k*mult, b))

}
st_store(., tokens("h2 h3 h5 h7 h9 h11 h13 h17 h19"),
   ScrHalton(50,8,&RandomMultipleScrambler(),10203)
```

The resulting plot of the selected pairs of dimensions is given in figure 5. The reader is encouraged to try different values of the seed and observe the resulting differences.



(a) Dimensions 1 and 2, $b_1 = 2$ and $b_2 = 3$

(b) Dimensions 7 and 8, $b_7 = 17$ and $b_8 = 19$

Figure 5. Initial elements of the Halton sequences in $[0, 1)^2$ scrambled with the random multiplier

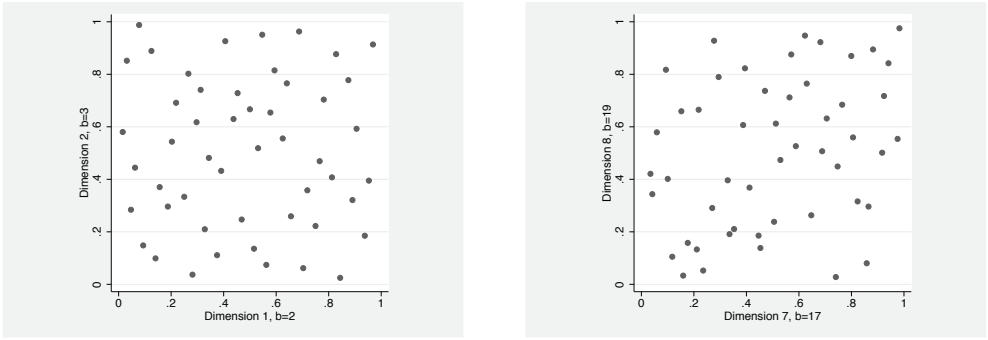
◁

▷ Example 4: Random permutation scrambler

To make more extensive use of the randomization possibilities, we can generate the whole permutation $\sigma_l(\cdot)$ randomly. To ensure that the same permutation is applied consistently to all elements of the scrambled Halton sequence, the random seed is reset each time the procedure is called. This seed acts as a parameter of the procedure and is supplied to `ScrHalton()`.

```
real scalar RandomPermuteScrambler(real scalar k, real scalar b,
                                   real scalar j, real scalar seed) {
   real colvector permut

   rseed(seed)
   permut = (0, jumble(1::(b-1)))
   return(permut[k+1])
}
st_store(., tokens("h2 h3 h5 h7 h9 h11 h13 h17 h19"),
   ScrHalton(50,8,&RandomPermuteScrambler(),1234576)
```

The resulting plot of the selected pairs of dimensions is given in figure 6. Arguably, the scatterplots are more characteristic of the pseudorandom number sequences like those in figure 1(a), with clumping of points and empty regions of the plot. Although the useful regularity properties of the Halton sequence will be preserved asymptotically, this property is not guaranteed for short series.



(a) Dimensions 1 and 2, $b_1 = 2$ and $b_2 = 3$ (b) Dimensions 7 and 8, $b_7 = 17$ and $b_8 = 19$

Figure 6. Initial elements of the Halton sequences in $[0, 1)^2$ scrambled with the random multiplier

◁

▷ Example 5: Atanassov's modified Halton sequence

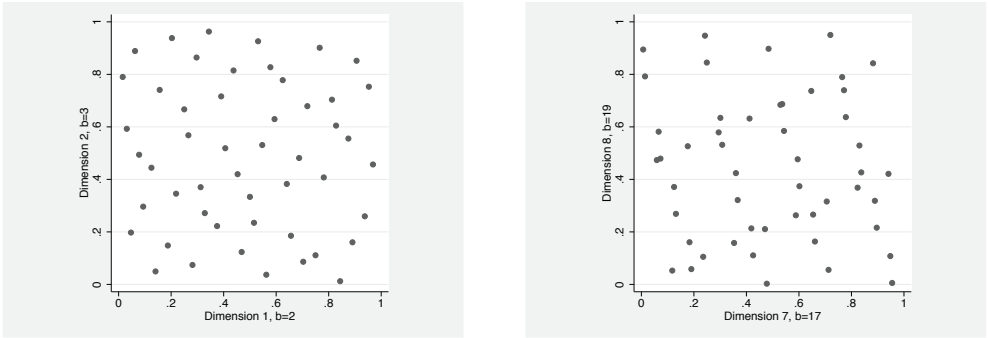
An advanced example of scrambling procedures based on the abstract algebra concepts comes from Atanassov (2004). He proposed a permutation that depends on the position of the digit after the decimal point as

$$\sigma_{lj}(a) = ak_l^j \mod b_j \tag{4}$$

for specially chosen numbers k_l related to the primitive roots in the field Z_{b_j}. The set of "good" numbers is hard-coded in the routine, and is transferred back and forth between the scrambler and the `_ScrHalton()` routine as a Mata matrix.

```
real scalar AtanassovGHaltonScrambler(real scalar k, real scalar b,
                                      real scalar j, real matrix K) {

   real colvector sel, kj, prod, i;

   if (K==.) {
      K = (2, 1 \ 3, 1 \ 5, 4 \ 7, 2 \ 11, 9 \ 13, 9 \ 17, 2 \ 19, 1 \ 23, 13)
      K = (K \ 29, 6 \ 31, 22 \ 37, 7 \ 41, 37 \ 43, 36 \ 47, 36 \ 53, 39 \ 59, 4)
      K = (K \ 61, 26 \ 67, 13 \ 71, 12 \ 73, 35 \ 79, 66 \ 83, 60 \ 89, 68)
      K = (K \ 97, 63 \ 101, 47 \ 103, 15 \ 107, 104 \ 109, 4 \ 113, 64)
   }

   sel = !(K[.,1] :- b)
   kj  = select(K, sel)[1,2]

   prod = 1
   for(i=1; i<=j; i++) {
      prod = mod(kj*prod, b)
   }

   return(mod(k*prod,b))
}
st_store(., tokens("h2 h3 h5 h7 h9 h11 h13 h17 h19"),
   ScrHalton(50,8,&AtanassovGHaltonScrambler(),K=.)
```

The resulting plot of the selected pairs of dimensions is given in figure 7. Although the strong lined patterns of figure 2(b) disappeared, other patterns are present in the higher dimensions.



(a) Dimensions 1 and 2, $b_1 = 2$ and $b_2 = 3$

(b) Dimensions 7 and 8, $b_7 = 17$ and $b_8 = 19$

Figure 7. Initial elements of the Halton sequences in $[0, 1)^2$ scrambled with the Atanassov multiplier

◁

▷ Example 6: Braaten and Weller's permutations

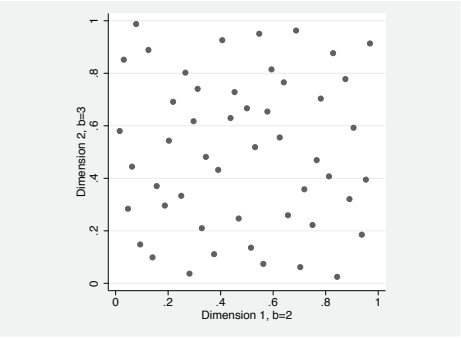
The first impetus to work with scrambled Halton sequences came from an article by Braaten and Weller (1979). To find good permutations, they used a greedy search algorithm that minimizes another version of discrepancy, the L_2 discrepancy $T^*_{\mathcal{N}}$. The latter can be computed explicitly.

This scrambler consists of three functions. The main function called by `_ScrHalton()` is `BraatenWellerScrambler()`; it uses an optional argument, Mata matrix **M**, to store the "good" permutations. These permutations are found by `BWPermutation()`, which in turn calls `_Tstar()` to compute the discrepancy of a sequence. An optional starting value of the permutation can be provided to `BWPermutation()`, which must be a number from $[0, 1)$. The default value of 0 reproduces the results of Vandewoestyne and Cools (2006), who reproduced most of the sequences from Braaten and Weller (1979) or produced other sequences with identical discrepancies.

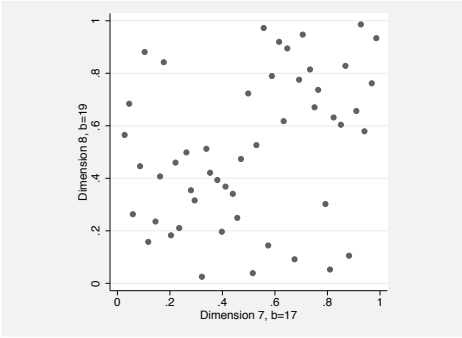
```
real scalar BraatenWellerScrambler(real scalar k, real scalar b,
                    real scalar j, real matrix M, | real scalar start) {

   // M has permutation stored by rows;
   //   the first column is the associated prime
   real rowvector perm;
   real scalar whereb;

   // starting point
   if (start == .) start = 0;
   else if (start<0 | start>=1) return(.);

   if (max(!(M[.,1]:-b))==0) {
     // initialize the permutation for the prime b
     perm = BWPermutation(b,start)
     if (M==.) M = (b, perm)
     else M = (M, J(rows(M),cols(perm)-cols(M)+1,.)  (b, perm))
  }
   else {
     // find where the permutation is stored
     whereb = sum((M[.,1]:<=b))
     perm   = M[whereb,2..b+1]
  }
   return(perm[k+1])
}
```

```
real rowvector BWPermutation(real scalar b, | real scalar start) {

   real rowvector perm;
   real scalar i, j, argmin, minT, currentT;

   // starting point
   if (start == .) start = 0;
   else if (start<0 | start>=1) return(.);

   perm = floor(start*b)
   // ensure that 0 maps to 0
   if (perm != 0) perm = (0, perm)

   for(i=1;i<b;i++) {
      // add a point to minimize the overall discrepancy
      // of the sequence accumulated so far
      minT = .
      for(j=0;j<b;j++) {
         // is j in the perm vector already?
         if (sum(!(perm:-j))) continue;
         else {
            // compute discrepancy if the point j is added
            currentT = _Tstar((perm:/b,j:/b))
            if (currentT < minT) {
               argmin = j
               minT   = currentT
            }
         }
      }
      perm = (perm, argmin)
  }
   return((0, perm))
}
// compute the T-star discrepancy of a series
real scalar _Tstar (real rowvector x) {

   real scalar i, j, sum1, sum2, N;

   N = cols(x)
   sum1 = sum2 = 0
   for(i=1;i<=N;i++) {
      if(x[i]<0 | x[i]>=1) return(.)
      for(j=1;j<=N;j++) {
        sum1 = sum1 + 1-max((x[i], x[j]))
      }
      sum2 = sum2 + (1-x[i]*x[i])
  }
   return(sum1/(N*N) - sum2/N + 1/3)
}

st_store(., tokens("h2 h3 h5 h7 h9 h11 h13 h17 h19"),
   ScrHalton(50,8,&BraatenWellerScrambler(),M=.)

st_store(., tokens("h2 h3 h5 h7 h9 h11 h13 h17 h19"),
   ScrHalton(50,8,&BraatenWellerScrambler(),M=.,0.5)
```

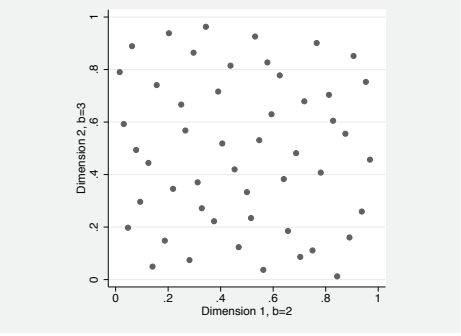
As a by-product of this procedure, the Mata matrix **M** contains the permutations found by Braaten–Weller's algorithm. The resulting plot of the selected pairs of dimensions is given in figure 8. The top row corresponds to the default starting value of 0, and the bottom row corresponds to the custom starting value of 0.5. The plots in higher dimensions are not ideal because some areas are sparsely populated.



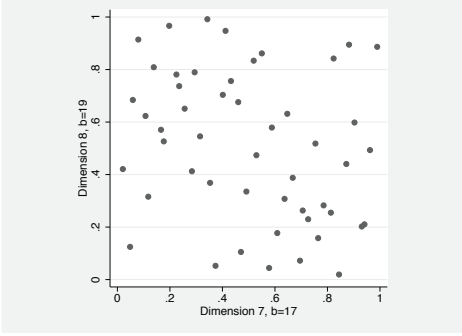
(a) Dimensions 1 and 2, $b_1 = 2$ and $b_2 = 3$; starting point is 0



(b) Dimensions 7 and 8, $b_7 = 17$ and $b_8 = 19$; starting point is 0



(c) Dimensions 1 and 2, $b_1 = 2$ and $b_2 = 3$; starting point is $\lfloor b/2 \rfloor$



(d) Dimensions 7 and 8, $b_7 = 17$ and $b_8 = 19$; starting point is $\lfloor b/2 \rfloor$

Figure 8. Initial elements of the Halton sequences in $[0, 1)^2$ scrambled with the Braaten–Weller procedure

◁

5 Discussion

Although scrambling Halton sequences, as described above, helps overcome "autocorrelations" of the original Halton sequences with higher values of the prime bases, scrambling is not a panacea: some artifacts may remain or new artifacts may be introduced. For reviews of different scrambling procedures, see Vandewoestyne and Cools (2006), Schlier (2008), and Faure and Lemieux (2009).

In some applications where Halton sequences or their generalizations are used to approximate multivariate integrals, a measure of accuracy of the approximation is required. It can be obtained by randomizing the sequence so that each point is distributed uniformly on $[0,1)^s$, and taking several sequences with independent random starting points. If the sequence $\mathbf{u}_i^{(k)}$, $i = 1, \ldots, N$, is produced from a random starting point $k = 1, \ldots, K$ and the integral $I(f) = \int_{[0,1)^s} f(\mathbf{u})\, \mathrm{d}\mathbf{u}$ is approximated by $\widehat{I}_k = 1/N \sum_{i=1}^{N} f(\mathbf{u}_i^{(k)})$, then the overall approximation to the integral in question is $\overline{\widehat{I}} = 1/K \sum_k \widehat{I}_k$, and its variance is estimated by $s_I^2 = 1/(K-1) \sum_k (\widehat{I}_k - \overline{\widehat{I}})^2$.

A simple randomization rule is to add a uniform random number (mod 1) to all elements of the (scrambled) Halton sequence. More advanced procedures will "continue" the Halton sequence from a random starting point by using an alternative representation of the iterations between the consecutive elements of the Halton sequence using additions with carry-over in base b. Of course, either idea can be combined with scrambling.

6 Acknowledgment

I thank David Roodman, the author of the `cmp` package (Roodman 2011), for helpful discussions and bug reports.

7 References

Atanassov, E. I. 2004. On the discrepancy of the Halton sequences. *Mathematica Balkanica* 18: 15–32.

Braaten, E., and G. Weller. 1979. An improved low-discrepancy sequence for multidimensional quasi-Monte Carlo integration. *Journal of Computational Physics* 33: 249–258.

Drukker, D. M., and R. Gates. 2006. Generating Halton sequences using Mata. *Stata Journal* 6: 214–228.

Dummit, D. S., and R. M. Foote. 2004. *Abstract Algebra*. 3rd ed. Hoboken, NJ: Wiley.

Faure, H., and C. Lemieux. 2009. Generalized Halton sequences in 2008: A comparative study. *ACM Transactions on Modeling and Computer Simulations* 19: 1–31.

Halton, J. H. 1960. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik* 2: 84–90.

Lemieux, C. 2009. *Monte Carlo and Quasi-Monte Carlo Sampling*. New York: Springer.

Niederreiter, H. 1992. *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia: Society for Industrial and Applied Mathematics.

Roodman, D. 2011. Fitting fully observed recursive mixed-process models with cmp. *Stata Journal* 11: 159–206.

Schlier, C. 2008. On scrambled Halton sequences. *Applied Numerical Mathematics* 58: 1467–1478.

Vandewoestyne, B., and R. Cools. 2006. Good permutations for deterministic scrambled Halton sequences in terms of L_2-discrepancy. *Journal of Computational and Applied Mathematics* 189: 341–361.

About the author

Stanislav (Stas) Kolenikov is a senior survey statistician at Abt SRBI and an adjunct assistant professor in the Department of Statistics at the University of Missouri in Columbia, MO. His research interests include statistical methods in social sciences, with a focus on structural equation models, microeconometrics, and analysis of complex survey data. He began using Stata in 1998 with version 5.