The Stata Journal (2003)
3, Number 3, pp. 270–277

Do-it-yourself shuffling and the number of runs under randomness

Nigel Smeeton
King's College, London, UK
nigel.smeeton@kcl.ac.uk

Nicholas J. Cox
University of Durham, UK
n.j.cox@durham.ac.uk

Abstract. A common class of problem in statistical science is estimating, as a benchmark, the probability of some event under randomness. For example, in a sequence of events in which several outcomes are possible and the length of the sequence and number of outcomes of each type known, the number of runs gives an indication of whether the outcomes are random, clustered, or alternating. This note explains and illustrates a simple method of random shuffling that is often useful. We show how the conditional probability distribution of the number of runs may be derived easily in Stata, thus yielding p-values for testing the null hypothesis that the type of outcome is random. We also compare our direct approach with that using the `simulate` command.

Keywords: st0044, alternation, categorical data, clustering, conditional distribution, forvalues, p-value, permutation, run, sequence, simulate, simulation

1 Introduction

A common class of problem in statistical science is estimating, as a benchmark, the probability of some event under randomness. Basic courses introduce methods for doing this, almost always in situations for which mathematical analysis yields (exact or approximate) p-values. Yet it is also easy to find situations, often quite simple in character, for which some kind of simulation is essential. Here, we explain and illustrate how you can "do it yourself" in Stata with a very simple and direct shuffling method.

Our working example is the number of runs under randomness. The idea of a run was, it may be guessed, one of the earliest statistical notions to emerge. Formal probabilistic interest goes back at least as far as the early 18th century, as shown by the work of Abraham De Moivre (Todhunter 1865). Barton and David (1962) reviewed the literature in a still-useful monograph, while theoretical interest in a variety of run problems is unabated (Balakrishnan and Koutras 2002).

The statistic of particular interest here is the number of runs in a sequence of several possible outcomes, which provides evidence for testing whether the observations are random. With clustering, the number of runs is relatively small, whereas alternation produces more runs than expected. The probability distribution for the number of runs follows from the total number of distinct orderings of the sequence (Wald and Wolfowitz 1940). Mood (1940) derived probability distributions for runs where three or more outcomes are involved. He distinguished between conditional distributions obtained from random arrangements of a fixed number of each outcome and unconditional distribu-

tions obtained from a binomial or multinomial population. Barton and David (1957) extended the work on conditional multiple runs distributions. Shaughnessy (1981) applied multiple runs distributions to testing for randomness in time-ordered residuals from regression analyses, usefully also tabulating various critical values.

We focus on conditional runs distributions for categorical data, especially apparent clustering, for which a one-tailed test against randomness will be applied. It should serve as an example of how easy it can be to simulate sampling distributions in Stata by simply shuffling observations within a loop, meaning precisely that observations are sorted according to a random sequence. Strictly, no programming is required (that is, you need never type `program`). We also compare our direct approach with that using the `simulate` command.

2 An example from health service research

The sequence below gives the method of delivery for 17 consecutive births in a South London hospital:

A A A A B A C C A A A A A D D A A

The codes are A for normal delivery, B for forceps, C for elective Cesarean, and D for emergency Cesarean.

Note that one category (A) accounts for most of the cases, while the other categories (B, C, D) feature only occasionally; this commonly happens with medical data and indeed more generally. The two consecutive emergency Cesareans could raise concern. They could have arisen by chance, but it is also possible that the midwife responsible for the two deliveries was more ready, compared with colleagues, to send women for an emergency Cesarean (a risky procedure), all other things being equal. The number of runs (here equal to 7) gives an indication of possible clustering. Subjectively, here one might expect a few more runs with a random pattern. However, obtaining the exact distribution analytically for the number of runs from the permutations of this sequence is not trivial. The formulas given by Mood (1940) and Barton and David (1957) are challenging, and as far as we are aware, no major statistical software has a routine that performs this analysis. Tables given by Barton and David (1957) extend only to a sample size of 12, and the group sizes in this example (12, 2, 2, 1) are far too imbalanced for Shaughnessy's tables of critical values. The approach here has been developed to simulate random permutations of sequences such as those above, leading to estimation of key probabilities associated with the runs distribution.

3 Rationale behind the code

In the sample of deliveries, there are $n(= 17)$ events and four outcomes A, B, C, and D, with $a(= 12)$, $b(= 1)$, $c(= 2)$, and $d(= 2)$ observations, respectively. First, we create a dataset containing one observation for each event. Second, we randomly shuffle the data repeatedly, calculating the number of runs after each shuffle to build up a picture of the

conditional distribution. Third, the tails of the distribution will highlight any evidence against the observations being random. In this example, clustering is of interest, so the probability in the lower tail of the distribution is pertinent.

4 Code for simulating the multiple runs distribution

We start by entering a dataset. In this example, we will use a string variable to hold categories; a numeric variable is equally possible. Naturally, in other examples, the data may already be in memory.

```
. clear
. set obs 17
. generate str1 method = "A" in 1/12
. replace m = "B" in 13/14
. replace m = "C" in 15/16
. replace m = "D" in l
```

An important consideration is the number of simulations required for reasonable accuracy of the tail probabilities in the estimated run distribution. Roughly 8,000 permutations are needed to give a 95% confidence interval of ± 0.005 for a tail probability of 0.05, as you can see directly in Stata by typing

```
. cii 8000 400
```

See [R] **ci** in the Stata manual for more on this command. Fortunately, it is easy to calculate very many more than that with even modest computer hardware. We will illustrate with 100,000.

To set up the simulation, we need first to assign places to put results. The number of runs that will be observed will certainly be an integer between 1 and 17, so we can set that up as one variable, remembering that `_n` is a built-in variable holding the observation number; see [U] **16.4 System variables (_variables)**.

```
. generate nruns = _n
```

A variable for holding random numbers must be set up, although its initial values are immaterial.

```
. generate random = .
```

It is good practice to set a random seed explicitly to allow reproducibility of results; see [R] **generate**.

```
. set seed 280352
```

Finally, we need to initialize a counter, and here it is crucial that initial values are all 0.

```
. generate frequency = 0
```

Here is the main loop:

```
. quietly forvalues i = 1/100000 {
.         replace random = uniform()
.         sort random
.         count if m != m[_n-1]
.         replace freq = freq + 1 if nruns == r(N)
. }
```

The loop as a whole is an example of `forvalues`, which is documented at [P] **forvalues** and featured in a tutorial with detailed explanations and examples (Cox 2002). Even if you have never met it before, you should be able to guess that a `forvalues` loop cycles over the range specified, here stepping through integers from 1 to 100000. Each time round the loop we get some new random numbers and `sort` the dataset on those, thus shuffling the observations. In our case, and unusually, we do not refer to the counter `i` within the loop. As the results come in random order, tagging when they arrive is presumably of no use or interest.

The number of runs is easily counted. (In passing, we commend the `count` command, which can be underrated. It is often the most direct way of getting what you want. See [R] **count**.) A new run starts whenever a value differs from the previous value: the subscript `[_n-1]` identifies the previous value, "previous" meaning, naturally, in the present order of observations. Note in particular that a condition like `if m != m[_n-1]` works properly when we look at the very first observation, for which `_n` is 1. A reference to `m[1-1]`, that is `m[0]`, will always be treated as a reference to a missing value, and any nonmissing value is evidently not equal to missing. With more complicated data than those in the current example, be aware that you may miscount if the very first value in a variable happens to be missing.

`count` leaves behind its result in `r(N)`, so we need to record the fact that one shuffle resulted in a sequence with that many runs. This requires a little care. As the data are being reshuffled every time around the loop, we need to specify that the value of `frequency` to increment (to increase by 1) is the value in the observation for which `nruns` is the same as `r(N)`. Thus if we record 7 runs, `frequency` must be incremented by 1 in (and only in) the observation for which `nruns` is 7. As programmers will note, there are other ways to do it: we could store results in a series of local macros, a series of scalars, or within a matrix, and in yet other ways. Using a variable has, at least in this case, few disadvantages and one major advantage, that users can access the results very easily without needing to learn anything particularly arcane about parts of Stata they might otherwise not know. In particular, they could proceed immediately to a table or graph.

One final detail, but one still worth flagging, is that the whole loop is controlled by `quietly`; see [P] **quietly**. This suppresses all output, except any error messages. The alternative would be a few hundred thousand lines scrolling past on your monitor (and enlarging any log file).

Now, we are on the final slope down towards home, needing only a little preparation for the final table, for which we use `tabdisp` (yet another command billed as for programmers yet often useful interactively; see [P] **tabdisp**).

```
. sort nruns
. label var nruns "# of runs"
. tabdisp nruns if freq, c(freq)
```

5 Results for the method of delivery data

The application of the above program to the birth data produced the following results:

# of runs	frequency
4	9
5	228
6	1812
7	7643
8	21072
9	34036
10	27602
11	7598

Overall, the evidence against the hypothesis of a random pattern in these data is weak. The estimated probability of seven or fewer runs is 0.0969, with an exact binomial 95% confidence interval from 0.0951 to 0.0988 (all results rounded to 4 d.p.). For comparison, note that, exceptionally, because the number of observations is only a little more than 12 and most of them are from one category (A), the exact p-value (0.0970) can be deduced by extrapolating from the table given in Barton and David (1957).

6 Comparison with use of simulate

Let us compare this method with use of the `simulate` command introduced in Stata 8. (In previous releases of Stata, `simul` was a close but not identical equivalent.) To understand this fully, you need to read the manual entry at [R] **simulate** and know a little about Stata programming, but that is not essential for our main argument.

We first set up the data as before

```
. set obs 17
. gen str1 method = "A" in 1/12
. replace m = "B" in 13/14
. replace m = "C" in 15/16
. replace m = "D" in l
```

and then initialize the random numbers

```
. gen random = .
```

We need to define a program that yields the number of runs after each shuffle

```
. program mysim, rclass
.        replace random = uniform()
.        sort random
.        qui count if m != m[_n-1]
.        return scalar N = r(N)
. end
```

and repeat it the desired number of times:

```
. simulate "mysim"  N = r(N), reps(100000)
```

`simulate` leaves in its wake a dataset with 100,000 observations, each containing a value of `N`, thus overwriting the original dataset.

```
. contract N
```

would reduce such data to a frequency distribution.

Although we refrain from making any claims about the generality of the method outlined earlier, it nevertheless has, for the problem tackled here, an appealing simplicity and directness that deserve attention.

7 Length of runs

In many run problems, the length of runs is also of interest (Balakrishnan and Koutras 2002), and so it is worth knowing how to calculate length in Stata. Continuing with our example, a run identifier is obtainable from

```
. gen runid = sum(m != m[_n-1])
```

which yields a variable with blocks of 1s, 2s, etc. To see this, note that the result is the cumulative sum of values of 1, yielded whenever a new run starts and so `m != m[_n-1]`, and of 0, yielded within a run, so that `m == m[_n-1]`. Then, the length of runs is the number of observations in each run

```
. bysort runid: gen runlength = _N
```

and that variable may be summarized as usual. In particular, this is an easy way to calculate the maximum run length, often of substantive or statistical interest. However, note that the raw mean (for example) of `runlength` will be weighted according to the number of observations in each run. The unweighted mean is obtained by using just one observation in each run:

```
. egen tag = tag(runid)
. summarize runlength if tag
```

For more background on `tag()`, see [R] **egen**.

8 A note on tsset

Readers familiar with Stata's time series functionality may have wondered why we used the subscript `[_n-1]` to indicate the previous observation, when it is possible to `tsset` the data and then use time series operators. The main reason is that on each occasion when we reshuffled the sequence, we would have to reset the time variable if we also wanted to use time series operators, an overhead easily avoided.

Two further considerations arise here. First, string outcome variables as used in our example cannot be `tsset`, but this is immaterial, as the same information could equally be held as integers with value labels. Second, if the analysis is of a subset of observations, care must be taken that references to `[_n-1]` do not refer to observations outside the exercise. In run problems, it is often simpler and safer to `drop` observations not in the analysis, having taken care to `save` the whole dataset first whenever appropriate.

9 Discussion

The code outlined in section 4 represents a straightforward technique for estimating a multiple runs distribution with reasonable accuracy. Among various practical advantages, the technique requires no special programming (although the main Stata devices are borrowed from the programmer's repertoire); it can be applied to any number of categories; and it can be extended easily both to larger data sets and to larger numbers of simulations, especially because extra memory demands are modest. Many of the details of Stata technique can also be applied to other simulation problems. In particular, almost every Stata user might want to know, sooner or later, how to shuffle randomly.

10 References

Balakrishnan, N. and M. V. Koutras. 2002. *Runs and Scans with Applications*. New York: John Wiley & Sons.

Barton, D. E. and F. N. David. 1957. Multiple runs. *Biometrika* 44: 168–178.

—. 1962. *Combinatorial Chance*. London: Griffin.

Cox, N. J. 2002. Speaking Stata: How to face lists with fortitude. *Stata Journal* 2(2): 202–222.

Mood, A. M. 1940. The distribution theory of runs. *Annals of Mathematical Statistics* 11: 367–392.

Shaughnessy, P. W. 1981. Multiple runs distributions: recurrences and critical values. *Journal of the American Statistical Association* 76: 732–736.

Todhunter, I. 1865. *A History of the Mathematical Theory of Probability from the Time of Pascal to that of Laplace*. London: Macmillan.

Wald, A. and J. Wolfowitz. 1940. On a test whether two samples are from the same population. *Annals of Mathematical Statistics* 11: 147–162.

About the Authors

Nigel Smeeton is a lecturer in medical statistics at King's College, London, UK, involved in the analysis of health service data. His interests include event clustering, capture-recapture analysis, and teaching methods in the context of medical and dental students. He is the co-author of a text on nonparametric methods.

Nicholas Cox is a statistically minded geographer at the University of Durham. He contributes talks, postings, FAQs, and programs to the Stata user community. He has also co-authored fourteen commands in official Stata. He was an author of several inserts in the *Stata Technical Bulletin* and is Executive Editor of the *Stata Journal*.