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 web page

[http://www.stata-journal.com](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®)
- Social Sciences Citation Index®

**Copyright Statement:** The *Stata Journal* and the contents of the supporting files (programs, datasets, and help files) are copyright © by StataCorp LP. The contents of the supporting files (programs, datasets, and help files) may be copied or reproduced by any means whatsoever, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the *Stata Journal*.

The articles appearing in the *Stata Journal* may be copied or reproduced as printed copies, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the *Stata Journal*.

Written permission must be obtained from StataCorp if you wish to make electronic copies of the insertions. This precludes placing electronic copies of the *Stata Journal*, in whole or in part, on publicly accessible web sites, file servers, or other locations where the copy may be accessed by anyone other than the subscriber.

Users of any of the software, ideas, data, or other materials published in the *Stata Journal* or the supporting files understand that such use is made without warranty of any kind, by either the *Stata Journal*, the author, or StataCorp. In particular, there is no warranty of fitness of purpose or merchantability, nor for special, incidental, or consequential damages such as loss of profits. The purpose of the *Stata Journal* is to promote free communication among Stata users.

The *Stata Journal* (ISSN 1536-867X) is a publication of Stata Press. Stata and Mata are registered trademarks of StataCorp LP.
Stata tip 85: Looping over nonintegers

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

A loop over integers is the most common kind of loop over numbers in Stata programming, as indeed in programming generally. *forvalues*, *foreach*, and *while* may be used for such loops. See their manual entries in the *Programming Reference Manual* for more details if desired. Cox (2002) gives a basic tutorial on *forvalues* and *foreach*. In this tip, I will focus on *forvalues*, but the main message here applies also to the other constructs.

Sometimes users want to loop over nonintegers. The help for *forvalues* contains an example:

```
forvalues x = 31.3 31.6 : 38 {
    count if var1 < `x' & var2 < `x'
    summarize myvar if var1 < `x'
}
```

It is perfectly legal to loop over such a list of numbers, because *forvalues* allows any arithmetic progression as lists, with either integer or noninteger constant difference between successive terms. However, such lists can cause problems. On grounds of precision, correctness, clarity, and ease of maintenance, the advice here is to use loops over integers whenever possible.

The precision problem is exactly that explained elsewhere (Cox 2006; Gould 2006; Linhart 2008). Stata necessarily works at machine level in binary, and so it does no calculations in decimal. Rather, it works with the best possible binary approximations of decimals and then converts to decimal digits for display. Not surprisingly, users often think in terms of decimals that they want to use in their calculations or display in their results. Commonly, the conversions required work well and are not detectable, but occasionally users can get surprising results. Here is a simple example:

```
forvalues i = 0.0(0.05)0.15 {
    display `i'
}
```

The user evidently expects display of 0, .05, .1, and .15 in turn, but the loop ends without displaying .15. Why is that? First, let us fix the loop by looping over integers and doing the noninteger arithmetic inside the loop. That is the most important trick for attacking this kind of problem.
. forvalues i = 0/3 {
    2. display `i´ * .05
    3. }

.05
.1
.15

Why did that work as desired, but not the previous loop? The default format for display is hiding from us the approximations that are being used, which are necessary because most multiples of 1/10 cannot be held as exact binary numbers. A format with many more decimal places reveals the problem:

. forvalues i = 0/3 {
    2. display %20.18f `i´ * .05
    3. }

0.000000000000000000
0.050000000000000003
0.100000000000000006
0.150000000000000022

The result 0.150000000000000022 is a smidgen too far as far as the first loop is concerned. Otherwise put, the loop terminates because Stata’s approximation to 0.05 + 0.05 + 0.05 is a smidgen more than its approximation to 0.15:

. display %20.18f 0.05 + 0.05 + 0.05
  0.150000000000000022
. display %20.18f 0.15
  0.149999999999999994

A little more cryptic, but closer to the way that Stata actually works, is a display in hexadecimal:

. display %21x 0.05 + 0.05 + 0.05
  +1.3333333333334X-003
. display %21x 0.15
  +1.333333333333X-003

The difference really is very small, but it is enough to undermine the intention behind the original loop.

Another trick that is sometimes useful to ensure desired results is the formatting of numerical values as desired. Leading zeros are often needed, and for that we just need to insist on an appropriate format:

(Continued on next page)
A subtlety to notice here is the pair of double quotes flagging to display that the macro \texttt{j} is to be treated as a string. Omitting the double quotes as in \texttt{display ’j’} would cause the formatting to be undone, because the default (numeric) display format would produce a display that started 1, 2, 3, and so forth. The syntax here for producing the local \texttt{j} is called an extended macro function and is documented in [P] macro.

If all that was desired was the display just seen, then the loop could be simplified to contain a single statement in its body, namely,

\begin{verbatim}
    . display %02.0f `i`
\end{verbatim}

However, knowing how to produce another macro with this technique has other benefits. One fairly common example is for cycling over filenames. Smart users know that it is a good idea to use a sequence of filenames such as \texttt{data01} through \texttt{data20}. Naming this way ensures that files will be listed by operating system commands in logical order; otherwise, the order would be \texttt{data1, data11}, and so forth. But those smart users then need Stata to reproduce the leading zero in any cycle over files. The loop above could easily be modified to solve that kind of problem by including a command such as

\begin{verbatim}
    . use data`j`
\end{verbatim}

Correctness, clarity, and ease of maintenance were also mentioned as advantages of looping over integers. Style preferences enter here, and programmers’ experiences vary, but on balance fewer coding errors and clearer code overall seem likely to result from the approach here. Moreover, noninteger steps, such as .05 within the very first example, are rarely handed down from high as the only possibilities. There is a marked advantage to changing just a single constant rather than a series of values from problem to problem.
References


