The Stata Journal

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

The Stata Journal (2007)
7, Number 1, pp. 143–145

Stata tip 43: Remainders, selections, sequences, extractions: Uses of the modulus

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

The `mod(x, y)` function produces remainders from division. It yields the remainder or residue when `x` is divided by `y`. The manual or online help definition is that `mod(x, y)` yields the modulus of `x` with respect to `y`. Mathematically, this definition is an abuse of terminology, but one that Stata shares with many other computing languages. In mathematics, the modulus is the divisor; somehow a few decades back in computing the term was transferred to the remainder.

Like several other functions, `mod()` may at first seem fairly trivial, so here are examples of some of its uses. All illustrations will be for first arguments (dividends) that are zero or positive integers and second arguments (divisors) that are positive integers. Stata's definition is more general, and yet more general definitions are possible, but the illustrations will show the main idea and cover most practical applications. Texts on discrete mathematics or the mathematics behind computing give fuller treatments (Biggs 2002; Knuth 1997; Graham, Knuth, and Patashnik 1994), but we need none of that material here. Authors often discuss these ideas under the heading of congruences.

How should you play with functions like `mod()` to get to know them? First, there is `display`:

```
. display mod(1,2)
1
. display mod(2,2)
0
. display mod(3,2)
1
```

One useful device is a loop to get several results at once:

```
. forvalues i = 0/8 {
.     display "`i' " mod(`i', 3)
. }
```

Second, there is `generate`, typically followed by `list`:

```
. set obs 9
. generate mod3 = mod(_n - 1, 3)
. list mod3
```

You can use the observation numbers `_n`, which are integers 1 and up, to produce variables corresponding to successive integers.

Third, there is Mata, released in Stata 9:

```
. mata
: x = (0..8)
: mod(x, 3)
```

The first illustration, dividing 1, 2, and 3 by 2, points up a useful detail. Evidently, on division by 2, odd numbers have remainder 1 and even numbers, remainder 0. This example gives a way of characterizing odd and even in Stata. Suppose that you want to specify every other observation. Then

```
if mod(_n, 2) == 1
```

specifies odd-numbered observations and

```
if mod(_n, 2) == 0
```

specifies even-numbered observations. No new variable need be created, as Stata does the necessary calculations on the fly. We can be even more concise:

```
if mod(_n, 2)
```

selects odd observation numbers. Given `mod(_n, 2)`, Stata evaluates it as 1 whenever `_n` is odd, which is nonzero and therefore true. Further,

```
if !mod(_n, 2)
```

selects even, as `mod(_n, 2)` is 0 whenever `_n` is even, but that result is flipped to 1 by the operator `!`, giving again 1, nonzero and true.

The idea extends easily to other divisors; for example, `if mod(year, 10) == 0` or `if !mod(year, 10)` selects values of `year` divisible by 10 such as 1990 or 2000, and `if !mod(year - 5, 10)` selects years such as 1995 or 2005 (but not 1990 or 2000).

Now let us turn to sequences. For integers x from 0 up, `mod(`x`, 3)` is

0 1 2 0 1 2 0 1 2 ...

and for any positive integer y, `mod(`x`, `y`)` repeats cycles of 0 to $y - 1$. You may often want to add 1 to get, e.g.,

1 2 3 1 2 3 1 2 3 ...

Here you should use in Stata `1 + mod(`x`, 3)` and in Mata `1 :+ mod(`x`, 3)`—note the elementwise operator `:+`.

You could get such sequences in other ways. Using `cond()` (Kantor and Cox 2005), we could type for observation numbers `_n` that run 1 upwards `cond(mod(_n, 3) == 0, 3, mod(_n, 3))`, giving the same result.

Hence, you now have a basic recipe for generating repetitive sequences. You may know that this functionality is wired into `egen`'s `seq()` function, but the approach from first principles has merit, too.

Extracting digits is yet another application. In the shadow world between numbers and strings dwell numeric identifiers and run-together dates (20070328 for 28 March 2007) or times (112233 for 11:22:33). Whether such beasts are best processed as numbers or strings can be a close call. Conversion functions `real()` and `string()` are available to throw each to the other side of the divide.

Suppose that your beasts arrive as numeric. `mod(112233, 100)` extracts the last two digits. Hence, second arguments that are 10^k will extract the last k digits from integers.

Other subsequences of digits require a little more work. We could get the first two, the second two, and the third two digits like this:

```
. local first = floor(112233/10000)
. local second = floor(mod(112233, 10000) / 100)
. local third = mod(112233, 100)
. display `first'`second'`third'
112233
```

For more on `floor()` and its twin `ceil()`, see Cox (2003). You could also use `int()` here. An alternative is to work with (say) `real(substr(string(112233),1,2))`.

Naturally, if what you are given is just 112233, you do not need Stata or even a computer to extract digits. Rather, these are examples of the kind you can try for yourself to see what is necessary to convert information given in variables from one form to another.

References

Biggs, N. L. 2002. *Discrete Mathematics*. Oxford: Oxford University Press.

Cox, N. J. 2003. Stata tip 2: Building with floors and ceilings. *Stata Journal* 3: 446–447.

Graham, R. L., D. E. Knuth, and O. Patashnik. 1994. *Concrete Mathematics: A Foundation for Computer Science*. Reading, MA: Addison–Wesley.

Kantor, D., and N. J. Cox. 2005. Depending on conditions: A tutorial on the `cond()` function. *Stata Journal* 5: 413–420.

Knuth, D. E. 1997. *The Art of Computer Programming. Volume 1: Fundamental Algorithms*. Reading, MA: Addison–Wesley.