ftools: a faster Stata for large datasets

Sergio Correia, Board of Governors of the Federal Reserve
2017 Stata Conference, Baltimore

sergio.correia@gmail.com
http://scorreia.com
https://github.com/sergiocorreia/ftools
1. Motivation: `bysort` is slow with large datasets
2. Solution: replace it with hash tables
3. Implementation: new Mata object
4. Implementation: new Stata commands
5. Going forward: faster internals and more commands
1. Motivation
Motivation (1/3)

- Stata is fast for small and medium datasets, but gets increasingly slower as we add more observations.
- Writing and debugging do-files is very hard if `collapse`, `merge`, etc. take hours to run.
- Example:

```stata
set obs `N'
gen int id = ceil(runiform() * 100)
gen double x = runiform()
collapse (sum) x, by(id) fast
```
Figure 1: Speed of collapse per observation, by number of obs.
• **collapse** gets slower because underneath it lies a **sort** command such as:

```
bysort id: replace x = sum(x)
by id: keep if _n == _N
```

• Sorting in Stata is **probably** implemented through quicksort, which is an \( O(n \log n) \) algorithm.

• Thus, **collapse** is also \( O(n \log n) \)

• This goes beyond **collapse**, as many Stata commands rely on **bysort** (**egen**, **merge**, **reshape**, **isid**, **contract**, etc.)

• See “**Speaking Stata: How to move step by: step**” (Cox, SJ 2002)
2. Solution
Solution

• When appropriate, replace **bysort** with a hash table
  • Already implemented by **Pandas, Julia, Apache Spark, R, etc.**
  • Also, internally by some Stata **users**

• A hash function is “any function that can be used to map data of arbitrary size to data of fixed size”

• Implemented in Stata:
  
  . mata: hash1(”John”, 100)
  52

• How does this work? Let’s implement **collapse** with a hash table!
Solution: collapse with a hash table

// Alternative to: collapse (sum) price, by(turn)
sysuse auto
mata:
   id = st_data(.,, "turn")
   val = st_data(.,, "price")
   index = J(1000, 2, 0) // Create hash table of size 1000
   for (i=1; i<=rows(id); i++) {
      h = hash1(id[i], 1000) // Compute hash
      index[h, 1] = id[i] // Store value of turn
      index[h, 2] = index[h, 2] + val[i] // Construct sum
   }
   index = select(index, index[.,1]) // Select nonempty rows
   sort(index, 1) // View results
end
• Sometimes two different values can return the same hash:

  . mata: hash1("William", 100)
      43

  . mata: hash1("Ava", 100)
      43

• To solve this, Mata’s **asarray()** stores lists of all colliding values

• Instead, **ftools** uses **linear probing**
3. Implementation
ftools is two things:

1. A Mata class that deals with factors or categories (ftools = factor tools)
2. Several Stata commands based on this class (fcollapse, fmerge, fegen, etc.)

To install:

- ssc install ftools
- ssc install moremata (used in “collapse (median) ...”)
- ssc install bootest (for Stata 11 and 12)
- ftools, compile (if we want to use the Mata functions directly)
Implementation: Factor class

```
sysuse auto
mata: F = factor("turn \_\_foreign") // New object
mata: F.num\_levels // Number of distinct values
mata: F.keys, F.counts // View values and counts
```

- `help ftools` describes in detail the methods and properties of this class
- These will remain stable, so you can implement your own commands based on it
- Please do so!
unique (from SSC) counts the number of unique values but is very slow on large datasets:

```
. sysuse auto
(1978 Automobile Data)
. unique turn
Number of unique values of turn is 18
Number of records is 74
```

Alternative:

```
mata: F = factor("turn")
mata: F.num_levels, F.num_obs
```

10x faster with 10mm obs.
Creating new commands: example 2 - xmiss

- **xmiss** (from SSC) counts missing values per variable

```stata
dsuse nlsw88  
(NLSW, 1988 extract)
.xmiss race union
```

<table>
<thead>
<tr>
<th>race</th>
<th>Missing</th>
<th>Total</th>
<th>% missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>white</td>
<td>284</td>
<td>1637</td>
<td>17.3</td>
</tr>
<tr>
<td>black</td>
<td>82</td>
<td>583</td>
<td>14.1</td>
</tr>
<tr>
<td>other</td>
<td>2</td>
<td>26</td>
<td>7.7</td>
</tr>
<tr>
<td><strong>All</strong></td>
<td>358</td>
<td>2246</td>
<td>16.4</td>
</tr>
</tbody>
</table>

- *Alternative* (12x faster with 10mm obs.)

```stata
mata: F = factor("race")
mata: F.panelsetup()
mata: mask = rowmissing(st_data(., "union"))
mata: missings = panelsum(F.sort(mask), F.info)
mata: missings, F.counts
```
4. Stata commands included with ftools
Commands included with ftools

- `fcollapse` (replaces `collapse`, `contract`, and most of `egen`)
- `fegen group`
- `fisid`
- `fmerge and join`
- `flevelsof`
- Also see: `reghdfe`
To use it: add `f` before your existing `collapse` calls

- Supports all standard functions (mean, median, count, etc.), all weights, etc.
- Can be extended through Mata functions (see `help fcollapse` for an example)
- `fcollapse ... , merge` merges the collapsed data back into the original dataset, making it equivalent to `egen`.
- `fcollapse ... , freq` is the equivalent to `contract`
- `fcollapse ... , smart` checks if the data is already sorted, in which case it just calls `collapse`
Figure 2: Speed of collapse per observation, by number of obs.
Figure 3: Speed of collapse and fcollapse by number of observations
Performance

Figure 4: Elapsed time of `collapse` and `fcollapse` by num. obs.
4. Going forward
Going forward

- The principles behind **ftools** allow Stata to work efficiently with large datasets (1mm obs. and higher)
- Still, there is large room for improvement
- **ftools** could be significantly speed up through improvements in Mata (better hash functions, more built-in functions, integer types, etc.)
- **gtools**, a very new package by Mauricio Caceres, implements some commands as a C plugin (**gcollapse, gegen**):
Going forward: gtools

Figure 5: Speed of collapse, fcollapse, and gcollapse
Going forward: 28s --> 10s --> 2s

Figure 6: Elapsed time of collapse, fcollapse and gcollapse
Conclusion

• With ftools, working with large datasets is no longer painful
• Still, we can
  • Speed it up (built-in functions, gtools)
  • Extend it to more commands (reshape, table, distinct, egenmore, binscatter, etc.)
The End
Additional Slides
• Caceres, M. (2017). gtools
• Cox, NJ. (2002). *Speaking Stata: How to move step by: step*. Stata Journal 2(1)
• Gomez, M. (2017). *Stata-R benchmark*
• Guimaraes, P. (2015). *Big Data in Stata*
• Maurer, A. (2015). *Big Data in Stata*
• McKinney, W. (2012). *A look inside pandas design and development*
• Stepner, M. (2014). *fastxtile*
Tricks learned while writing ftools (advanced)

- If you want to write fast Mata code, see these tips
- If you want to distribute Mata code as libraries, but don’t want to deal with the hassle of compiling the code, see this repo
- If you usually declare your Mata variables, consider including this file at the beginning of your .mata file
Mata Wishlist

Any of the following would significantly speed up $ftools$:

- **Integer types** so we can loop faster
- A **rowhash1()** function that computes hashes in parallel for every row
- A faster alternative of **hash1()**, such as SpookyHash, from the same author
- An optimized version of $x[i] = x[i] + 1$
- Radix sort function for integer variables (recall that counting sort is $O(n)$)