# The Mata Book

William Gould

President
StataCorp LLC

September 2018, London

## Purpose of talk

**The Mata Book: A book for serious programmers and those who want to be**
by William Gould

The book is 428 pages!

I'll try to convince you that it's worth your time.

(That's a tall order.)

## Is this book for you?

**I wrote this book for**

**people who have added substantive features to Stata**

**. . . and those who want to**

- People like you. The people in this room.
- I'm about to tell you a story the ends with . . .
  "*like a plate of spaghetti*".
- This book is for you if you have had the tangled-code experience.

## It wasn't your fault because . . .

**You chose the wrong language.**

- Stata's ado is not rich enough.
- You needed Mata.
- But if you use Mata the same way you use ado . . .
  **nothing will change**.
- And that means the book has to be about . . .
  *more than just Mata*.

The book is about

Programming . . . Programming Techniques . . . Workflow

. . . and Software Development

Mata is the programming language that is used.

**Let's start over . . .**

## The book has four parts . . .

1. Mata's language elements
2. Writing simple programs
3. Writing complex programs (small systems)
4. Writing big systems (programs that need to be designed)

## Part 1: Mata's language elements

- I call this part the boring part.
- Piece by piece, we work our way through Mata.

Here's a piece: `while`.

Some books would introduce it like this:

```
i = 1
while (i < 10) {
        ...
}
```

## How not to be boring

Here's how I show you `while` for the first time:

```
x = 1
while (abs(f(x)) > 1e-8) {
    x = x + f(x)/fprime(x)
}
```

It's Newton's Method for solving $x : f(x) = 0$.

Or if you prefer, for solving $x : g(x) = c$.

Define $f(x) = g(x) - c$ and the loop returns $x = g^{-1}(c)$.

## How not to be boring

*Do you see how neat this is?*

We can use `while` to find the square root of 2:

```
: x = 1
: while (abs(x^2-2) > 1e-8) x = x + (x^2-2)/(2*x)
: x
: 1.414213562
```

*Remember this the next time you need to invert a function.*

## How not to be boring 2

Many pages later ...

- We discuss how to write numeric literals.
- 1, 2, 3.35, 1.0e-08, and so on.

*... What could be more boring?*

*Question:* What could be more boring?

*Answer:* Yet another way Mata lets you write numbers . . .

- 1.0x-1a means . . . *oh forget it* . . .
  1.0x-1a is approximately equal to 1.490e-08
- Not only boring, but superfluous.
- Overdone. Redundant.

. . . So I kick you in the head with a table:

Problem: Calculate `d = (f(x+h)-f(x))/h`.

```
------------------------------------------------------
                    Relative error in d vs. truth
                 -------------------------------------
                 if you code             if you code
          x      h = 1.0e-8              h = 1.0x-1a
------------------------------------------------------
          0          0                          0
          8          8.27e-08                   0
         64          6.27e-07                   0
      1,024          1.06e-05                   0
     32,768          2.83e-04                   0
  1,048,576          1.17e-03                   0
  4,194,304          2.45e-02                   0
 16,777,216          0.118                      0
 33,554,432          0.255                      0
 67,108,864          0.490                      0
134,217,728          infinity                infinity
------------------------------------------------------
```

Perhaps it's worth your time to learn about 1.0x-1a?

So much for the boring part of the book.

But realize

- It runs 127 pages.
- I told you about 7 of them.
- It's not boring.
- And it's useful.

## The non-boring parts of the book

The non-boring parts of the book are about

- **Writing Mata programs**.
- Literally.
- We don't just discuss program writing.
- We write real programs from start to finish.

Along the way, the book

- Teaches Mata details.
- And advanced features.

## How to write simple programs

Part 2 is about Simple Programs.

We write a short but serious program to calculate

$$c = \frac{n!}{(n-k)!\,k!}$$

I complicate things by spending four pages discussing alternative implementations and their numerical accuracy.

We package the simple function three ways,

1. for use in a do-file
2. for use in an ado-file
3. for use anywhere, anytime (we put the program in a library)

We discuss validation and certification.

We implement our first certification script.

We spend 30 pages.

**We are done with simple programs . . .**

. . . and complex programs are next

"**One simple function!**" I hear you yell. "**That's all?**"

*Yes*, because . . .

- I show you how to transform complex programs into simple programs.
- Lots of simple programs.

## Complex (meaning multipart) functions

In the book,

- We implement linear regression.
- A full implementation. A good implementation. One suitable to be shipped by StataCorp.
- **And we write 14 functions, each 4-lines long!**

## Complex functions

*Punchline:* **We write 14 functions, each 4-lines long . . .**

- There's a way to transform a complex program into multiple, simple programs.
- I teach it to you.
- It has wide applicability in statistical programming.

## Wherein I praise second-rate formulas and algorithms . . .

- I love second-rate formulas and algorithms!
- I love them because they are so easy to write.
- I use them when I write code.
- When the code works, I evaluate whether results are good enough. Sometimes they are.

In the book . . .

- We implement $b = (X'X)^{-1}X'y$.
- and 13 other formulas
- We discover that $(X'X)^{-1}X'y$ is inadequate.
- We substitute a far more complicated calculation for it.
- If code is well written, swapping algorithms is easy.

We spend 59 pages.

We did all of the preceding, *and* . . .

- I taught you all about structures. You are an expert.
- We could not have implemented the self-theading code design without them.
- I taught you how to use pointers to conserve memory.
- We wrote a certification script.

And we are still not done with the linear-regression problem . . .

- We reimplement the entire linear-regression system using classes.
- We should have used classes from the outset.
- After 59 pages, you'll know why.
- We spend another 40 pages adding new features (robust standard errors).
- Then we spend 3 pages on numerical accuracy when dealing with symmetric matrices.

That last one? Every scientific programmer should know it, but they don't.

And that brings me to what I do in my day job: **Part 4: Systems**.

- A system is a program that has to be designed, inside and out.
- *Outside:* What the user sees.
- *Inside:* How the code is organized.

- Some programs design themselves.
- They are so short it's obvious.
- Or you adopt a standard technique and that's the design.

**Nobody has shown you when and why to design code. I do.**

*When:*

When you can't envision the code in detail.

*Why:*

It will save you time.
It improves the chances you will finish.
Code will be easier to modify.

**Nobody has taught you how to design a system. I do.**

But I warn you, it will take 107 pages.

In the book . . .

- I show you the steps of doing a design.
- Designs often go wrong. Ours does.
- We fix it while still in design stage.
- We implement code based on design.

And even so, our system does not work well enough.

It works so poorly we will be near cancelling the project.

## What happens next?

You'll have to read the book.

Oh yes, the book has appendices.

The Appendices are

A. Writing Mata code to add new commands to Stata

B. Mata's storage type for complex numbers

C. How Mata differs from C and C++

D. Three dimensional arrays (advanced use of pointers)

No cheating and skipping directly to them.

They are written assuming you have digested all but the last part of the text.

Except for D, which assumes the last part too.

**Thank you**