Statistical Programming in Julia

Thomas Wiemann
University of Chicago

TA Discussion # 1

October 6, 2021

Outline

1. Logistics
2. General Advice for the Problem Sets
3. Statistical Programming in Julia
   - Motivation
   - Types and Multiple Dispatch
   - Getting started with Julia Modules

Logistics

TA office hours on Fridays, 9-10am, in the grad lounge (SHFE 201).

Problem set solutions will not be discussed during the discussion sessions.

1. Answer keys will be posted shortly after the submission deadlines.

Instead, the discussion sessions will typically:

- Explore applications of a method studied in class, or
- Discuss tools you may need for subsequent problem sets, or
- Review a strand of related literature.

I'll post my material on canvas a few hours before class. (May still contain typos and could be updated after class.)

General Advice for the Problem Sets

Problem sets for this class are labor intensive.

- Start working on them as early as possible.

Cooperate with peers, but submit your own work only.

- Don't copy answers or code from your peers (it is not worth it).

The four Cs of ~~diamonds~~ problem sets:

- **C**omprehensive – answer all (sub) parts of the questions. All necessary derivations should be included. (Do not cite outside work.)
- **C**oncise – a few lines of math speak a thousand words.
- **C**lear – define all variables and parameters clearly. Highlight when you use non-trivial results.
- **C**orrect – avoid errors. (Typically the most difficult.)

General Advice for the Problem Sets (Contd.)

For the coding exercises:

- Carefully (but not excessively) comment your code.

```
# Good
beta <- solve(crossprod(X)) %*% crossprod(X, y) # OLS coefficient

# Bad
beta[1] < 0 # checks whether beta[1] is less than zero
```

- Do not exceed 80 characters per line of code! Most IDEs have an option to display a vertical line at 80 characters. (Use it.)

- Don't reinvent the wheel: make use of style guides for Julia (e.g., Blue), R (e.g., Hadley Wickham), or Python (e.g., PEP 8).

- Code smart: think about design patterns. We'll dive into this in the second part of today.

General Advice for the Problem Sets (Contd.)

Only submit LaTeXtables. (Never screenshots from software output. Maybe this helps: excel2latex) Your tables should be well structured and contain all necessary annotations. For example:

Table 1: Simulation Results

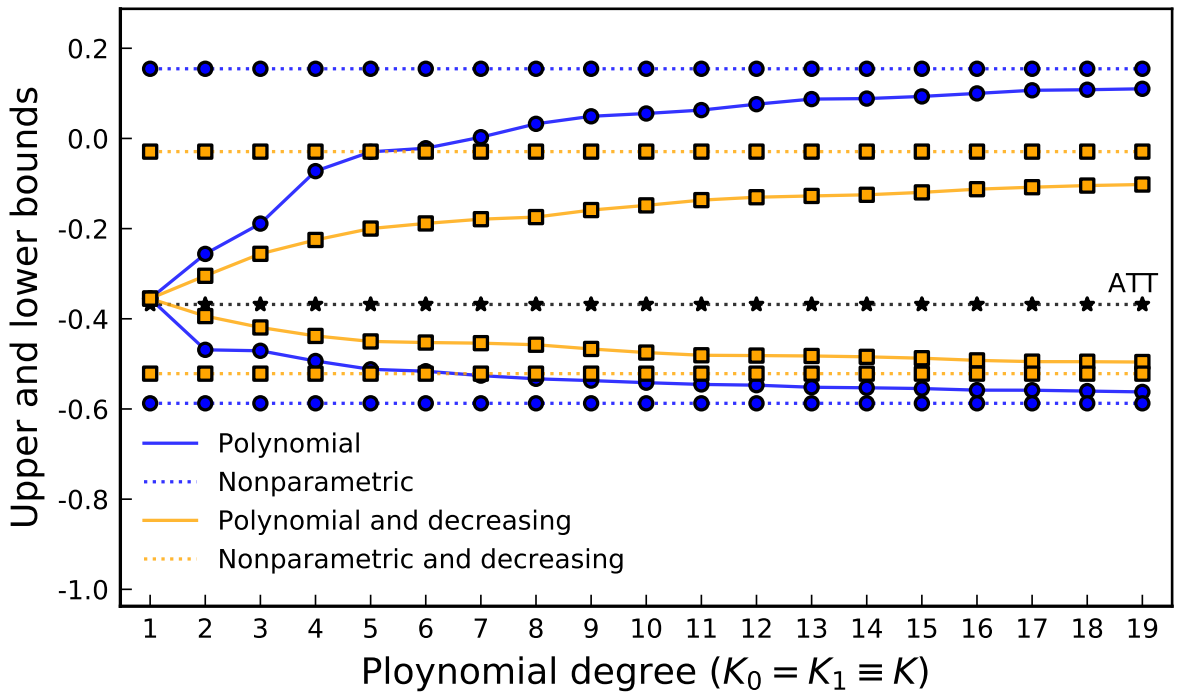
	N=100 (1)	N=200 (2)	N=400 (3)	N=800 (4)
OLS				
Bias	0.5878	0.5879	0.5879	0.5882
Median Abs. Error	0.5883	0.5872	0.5875	0.5882
Standard Error	0.0630	0.0449	0.0315	0.0222
Cov. Rate (95%)	0.0000	0.0000	0.0000	0.0000
TSLS				
Bias	0.2773	0.1639	0.0910	0.0470
Median Abs. Error	0.2826	0.1682	0.0942	0.0499
Standard Error	0.1112	0.0907	0.0724	0.0541
Cov. Rate (95%)	0.3736	0.5760	0.7314	0.8436
JIVE				
Bias	0.4445	-0.0482	-0.0189	-0.0096
Median Abs. Error	-0.0398	-0.0232	-0.0117	-0.0058
Standard Error	42.5482	0.1770	0.1007	0.0635
Cov. Rate (95%)	0.7504	0.8602	0.9066	0.9364

Notes. Statistics are based on 5,000 simulations. N denotes the sample size of the simulated data.

General Advice for the Problem Sets (Contd.)

Submit "publication ready," self-contained plots. When possible, include vectorized figures (e.g., pdf rather than png). For example:

Figure 1: ATT Bounds



Notes. Reproduction of Figure 6 in Mogstad and Torgovitsky (2018). Bernstein polynomials were used for the computation of the parametric polynomial bounds. Constant splines where used for the computation of the nonparametric bounds as outlined in Mogstad et al. (2018).

Questions

Any questions?

Next up: Statistical Programming in Julia

Statistical Programming in Julia: Motivation

In this course, you are going to code *a lot*.

A *selection* of estimators I implemented last year for this course: GLS, LPR (with various kernels), sieve estimators (with various bases, including Bernstein and standard polynomial basis, linear and cubic splines), KNN, matching, probit, logit, TSLS, JIVE, LARF, kernel density estimators, the Anderson Rubin test, and – of course – the bootstrap.

Most of the estimators accompanied by complementary functions (e.g., for inference or bandwidth choice). You'll also test your implementations, both through Monte Carlo simulations and through replication exercises.

Suggestion: Don't let the code go to waste! Use this as an opportunity to develop your own applied econometrics Julia/Python/R package.

Table 2: Pros and Cons List

Pros	Cons
code for future research	~~fixed cost of learning design patterns~~
code for future courses	
learn how to develop good code	

Notes. If you're in this TA session, the fixed costs will be incurred anyway and thus don't enter *your* decision problem.

Motivation (Contd.)

Today we will discuss how to structure and develop an applied econometrics package in Julia.

Why use Julia?

- Julia is Econ student-friendly (high level syntax, dynamically typed), but at CS student-speed (very high performance, nearly C-levels).
- Solves the two language problem. Most *fast* Python or R functions are just high-level interfaces to low-level implementations (e.g., in C). With Julia, no other programming language is needed to write high-performance code.
- Extended standard library and numerous third-party packages (e.g., `Gurobi.jl`, `Flux.jl`, `Turning.jl`)
- There's a growing (research) community using Julia.

But you don't have to use Julia. Python and R are good alternatives.

- R: Wickham (2015, 2019) (both freely available online)
- Python: Ceder (2018)

Statistical Programming in Julia: Types and Multiple Dispatch

A naive implementation of the estimators for this class may look something like this:

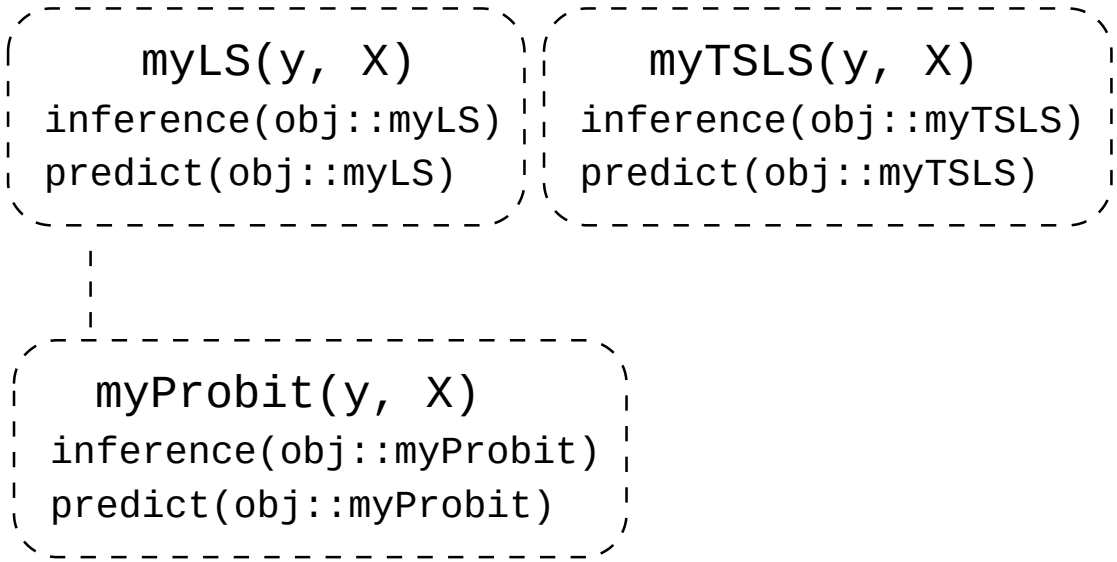
```
myLS(fit, y, X)                    myTSLS(y, D, Z, X)
inference_myLS(β_hat, y, X)        inference_myTSLS(β_hat, y, D, Z, X)
predict_myLS(β_hat, X)             predict_myTSLS(β_hat, X)
  ¦
  ¦
  ¦
myProbit(y, X)
inference_myProbit(β_hat, y, X)
predict_myProbit(β_hat, X)
```

This bad practice. Heavy use of subscripts makes your code cluttered, decreasing readability. Arguments are not recycled across complementary functions, which makes code more error prone and possibly causes issues with reproduction.

Types and Multiple Dispatch (Contd.)

A better approach borrows ideas from object-oriented programming:

- Encapsulation. Here: Bundle arguments into (immutable) objects.
- Polymorphism. Here: Leverage Julia's multiple dispatch.



Whenever you run a command in Julia, a process – called multiple dispatch – is launched, which determines which function to execute *using the types of the function arguments*. This implies that multiple functions can carry the same name as long as their argument types differ.

Types and Multiple Dispatch (Contd.)

Definition of a Custom Composite Type

```
struct myLS
    b::Array{Float64} # coefficient
    y::Array{Float64} # response
    X::Array{Float64} # features

    # Define constructor function
    function myLS(y::Array{Float64}, X::Array{Float64})
    # Calculate LS
    b = (X' * X) \ (X' * y)
    # Organize and return output
    new(b, y, X)
    end #MYLS
end #MYLS
```

Defining a Complementary Prediction Function

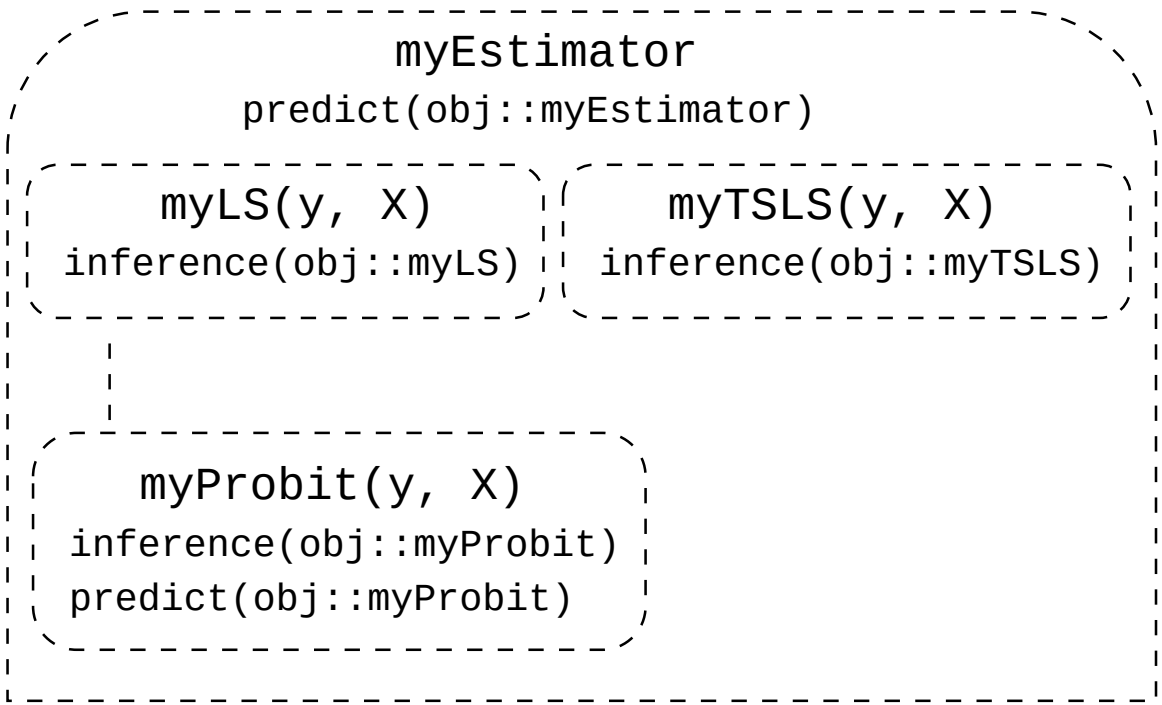
```
function predict(fit::myLS, data = nothing)
  # Check for new data, then calculate and return predictions
  isnothing(data) ? fitted = fit.X * fit.b : fitted = data * fit.b
  return(fitted)
end #PREDICT.MYLS
```

See also: thomaswiemann.com/Types-and-Multiple-Dispatch-in-Julia

Types and Multiple Dispatch (Contd.)

Notice that there are two additional pillars of object-oriented programming to which we (may be able to) pay less attention to:

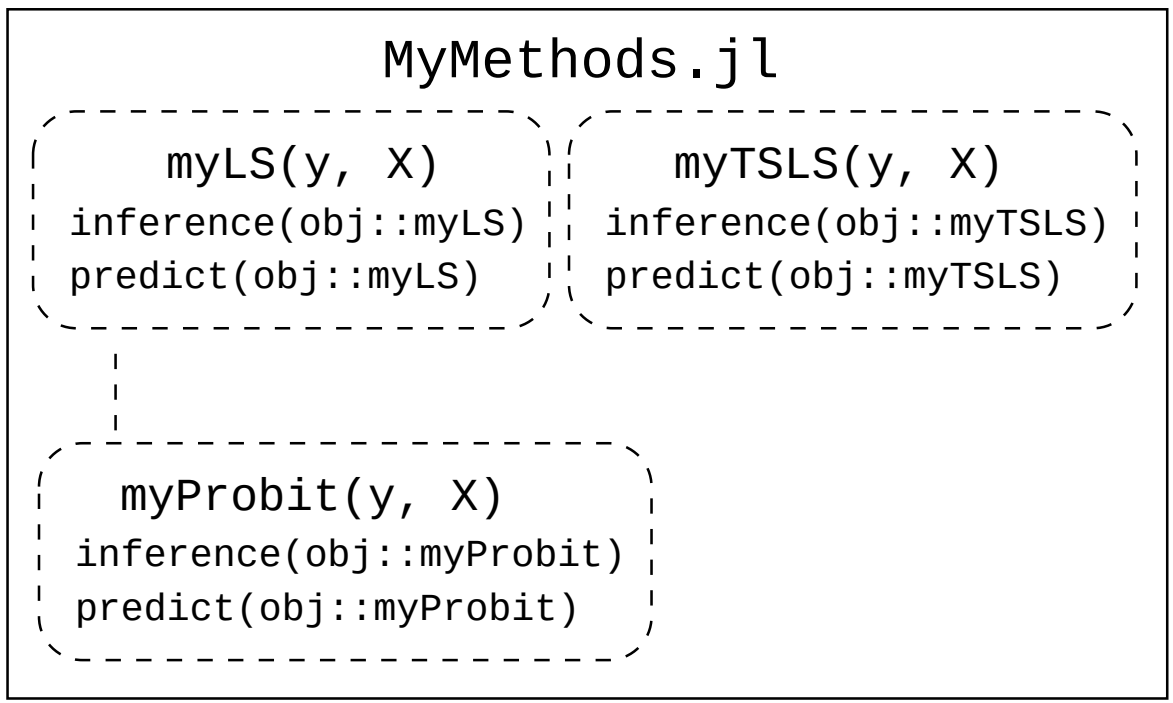
- Abstraction. Here: Provide easy-to-understand methods for, e.g., inference. (We've already done that in the naive approach.)
- Inheritance. Here: Possibly define a type hierarchy so that certain complementary functions can be shared across estimators. (This may make sense in a few cases, including the bootstrap.)



Statistical Programming in Julia: Getting Started with Modules

In Julia, you can load source code into your session using `include()`, but that can quickly become messy (latest by problem set 3).

A convenient way to load and share your newly developed estimators is through modules.



Getting Started with Julia Modules (Contd.)

Once defined, modules can be loaded into your current environment with the `using` keyword.

If you're hosting your module publicly on GitHub, you can also install it on any machine with:
`Pkg.add(url="https://github.com/[USRNAME]/[MODULENAME]")`.

For a tutorial on how to set up your applied econometrics Julia module, see: thomaswiemann.com/Getting-Started-With-Julia-Modules.

It covers:

- Using `PkgTemplates.jl` so you don't have to start from scratch.
- Setting up Git and GitHub for your Julia module.
- The development workflow:
  1. Adding (and testing) source code.
  2. Specifying dependencies.
  3. Defining public and private objects/functions.

Last Comments

If you're just getting started with Julia, you may find these resources helpful:

- Quantecon (link1, link2)
- Julia Cheat Sheet (link)
- Lauwens and Downey (2019), freely available online (link)
- Alvaro Carill's guide on setting up Atom with Julia (link)

In case you're already working with Julia and would like to develop additional skills, these two books could be interesting: Kwong (2020) and Sengupta (2019). But you should only consider purchasing these books if you *really* want to get into the weeds. (They are far exceeding what you would need for this course.)

References

Ceder, N. (2018). *The quick Python book*. Simon and Schuster.

Kwong, T. (2020). *Hands-On Design Patterns and Best Practices with Julia*. Packt Publishing Ltd.

Lauwens, B. and Downey, A. B. (2019). *Think Julia: How to think like a computer scientist*. O'Reilly Media.

Mogstad, M., Santos, A., and Torgovitsky, A. (2018). Using instrumental variables for inference about policy relevant treatment parameters. *Econometrica*, 86(5):1589–1619.

Mogstad, M. and Torgovitsky, A. (2018). Identification and extrapolation of causal effects with instrumental variables. *Annual Review of Economics*, 10:577–613.

Sengupta, A. (2019). *Julia: High Performance Programming*. Packt Publishing Ltd.

Wickham, H. (2015). *R packages*. O'Reilly Media, Inc.

Wickham, H. (2019). *Advanced R*. CRC press.