Introduction to Economic Modelling in Julia

IFS TECH

David Sturrock

12th July 2023

Julia

Julia is:

- …a programming language
- …increasingly used across the natural sciences and economics
- …open source and with a growing community

Julia is useful to applied economists for:

- …things requiring a lot of computation/optimisation
- …like estimating models through indirect inference (i.e. simulated method of moments)

Julia is not so useful for:

- …'standard' data analysis
- …things using advanced/particular econometrics

Julia

Julia is:

- . . . a programming language
- . . . increasingly used across the natural sciences and economics
- . . . open source and with a growing community

Julia is useful to applied economists for:

- . . . things requiring a lot of computation/optimisation
- . . . like estimating models through indirect inference (i.e. simulated method of moments)

Julia is not so useful for:

- . . . 'standard' data analysis
- . . . things using advanced/particular econometrics

Julia

Julia is:

- . . . a programming language
- . . . increasingly used across the natural sciences and economics
- . . . open source and with a growing community

Julia is useful to applied economists for:

- . . . things requiring a lot of computation/optimisation
- . . . like estimating models through indirect inference (i.e. simulated method of moments)

Julia is not so useful for:

- . . . 'standard' data analysis
- . . . things using advanced/particular econometrics

Installation and Setup

- **Downloading and installing Julia:** Download the Julia language from `https://julialang.org/downloads/` and install.

- **Setting up Julia in Visual Studio Code:**
  1. Download and install Visual Studio Code if not yet installed: `https://code.visualstudio.com/download`.
  2. Launch VS Code, go to the Extensions view by clicking on the Extensions icon in the Activity Bar on the side of the window
  3. In the Extensions view, search for ‘Julia‘ and install the extension named ‘Julia‘. Make sure this extension is enabled.
  4. Configure the Julia extension to use the Julia executable that you installed. To do this, open the command palette (‘Ctrl+Shift+P‘), type in ‘Preferences: Open User Settings‘, and hit enter. In the settings tab, search for ‘Julia: Executable Path‘ and fill in the path to the Julia.exe file.

What is special about Julia?

1. Both very fast and easy to use
   - Julia combines interpretation and ‘Just-in-Time compilation’
   - Ease of interpreted languages (Matlab) while nearly as fast as C/Fortran
     - e.g. can run interactively and debug easily

2. ‘Multiple dispatch’
   - Functions can be defined generically and specialised by use
   - Makes code more versatile while facilitating speed

3. Built-in parallel and distributed computing capabilities

4. ‘Expressive syntax’
   - Code can be made to resemble mathematical notation

5. Open source
   - It’s free! And consequently has a growing groups of users

Comparing Julia to other languages

There are a range of languages that can be used for economic modelling

- Matlab, Fortran, C++, C, Python, R(?)...

Item	**Costly/Free?**	**Slow/Fast?**	**Hard/Easy-to-use?**	**Less/more support and packages?**
Fortran	✓	✓✓	×	✓✓
C/C++	✓	✓✓	×	✓✓
Julia	✓	✓	✓	✓
Matlab	×	×	✓✓	✓
Python	✓	×	✓	✓
R	✓	× ×	✓	✓✓

- All of this depends on your previous programming, what you are doing, and subjective judgement

Basic Syntax of Julia

Variables: defined without needing a specific type, e.g.

- `x = 5` (Integer)
- `y = 3.14` (Float)
- `str = "Hello"` (String)

Arrays: Created using square brackets, indexed from 1.

- e.g., `A = [1, 2, 3]` (column vector) ; `B = [1 2 3]` (row vector)
- `A[1]` would give ‘1’.

Mathematical operations: standard operators are used: `+`, `−`, `*`, `/` etc.

- put a ‘`.`’ first to do element-wise on arrays
- i.e. `B*A` would give 14. `A.*A` would give `[1, 4, 9]`

Comments: use ‘#’; multi-line comments between ‘`#=`’ and ‘`=#`’

Functions

We can create a simple function with the following text:

```
function square_then_double( input )
# This function outputs the twice the square of the input

output = 2 * (input.^2)

return output
end
```

Rather than having in the main script, we save it as e.g. `square_then_double.jl`.

To use this function, we need to ‘include’ the file that defines it in our main script:

```
include("functions/square_then_double.jl")
E = square_then_double(A)
```

This code assigns the value `[1, 4, 9]` to `E`.

Other things to note

Scope

1. Variables are defined globally by default
2. Functions return only the final output by default; specify more with ‘return’
3. Functions will only *modify* inputs if return and re-assign

Packages

1. To use additional packages, write `` using `packageName' `` at start of script
    - useful packages include Distributed, Interpolations, QuadGK, Optim, Roots, LinearAlgebra, Random, Plots, Queryverse, XLSX, Statistics
2. To use parrallelisation, use the (built-in) “Threads” package

Example usage: estimating a lifecycle model of saving

Question

1. Are younger generations are less ‘patient’ than older generations?

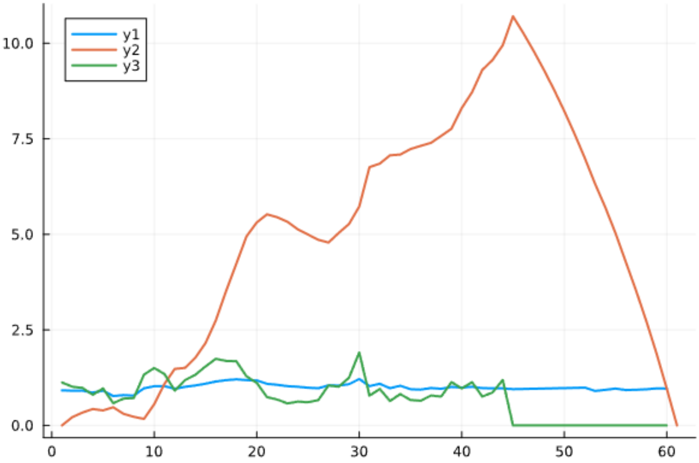
Method

1. Build model of optimal wealth accumulation given economic conditions
2. Estimate preference parameters by matching model to wealth data with simulated method of moments

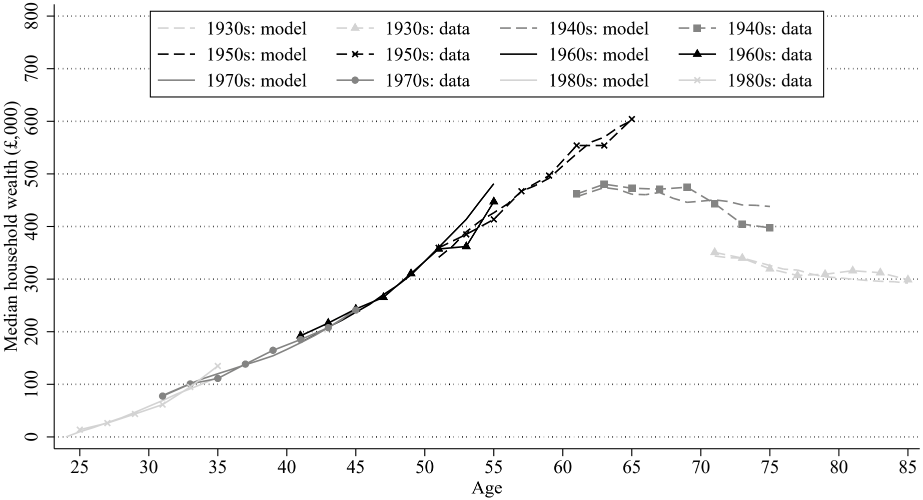
Why a quantitative model?

1. There is no closed-form solution for optimal saving in the face of uncertain future earnings, rates of return, changing household composition etc.
2. Need to solve for optimal choices using numerical methods, which involves solving hundreds of optimisation problems

Example usage: estimating a lifecycle model of saving



Example usage: estimating a lifecycle model of saving



Final thoughts

- When solving things numerically, you never know what the correct answer is
- Can never absolutely verify that there is no mistake
- Calls for disciplined coding practices:
  - Write many many many checks and warnings into your code
  - Once you think your program works - try to break it by stress-testing - select extreme parameters - may help you find bugs

Other good resources:

- `https://github.com/cpfiffer/julia-bootcamp-2022`