Essential numerical tools and perturbation analysis (1.a)

Introduction

Pablo Winant

African Econometric Society Workshop 14/12/23

About me

- A computational economist
- Formerly Worked in institutions (IMF, BOE)
- Now at ESCP Business School and CREST/Polytechnique
- Research on models about:
  - Inequality (heterogeneity)
  - International Finance
  - Monetary Policy
  - Artificial Intelligence
- Involved in several opensource projects (Dolo, QuantEcon, ARK)

About this course

Day 1:

- intro to Julia
- recursive sequences
- function optimization

Day 2:

- (dynamic) first order conditions
- automatic differentiation
- solution of matrix quadratic equations

- 👉 static equilibrium models
- 👉 dynamic stochastic equilibrium models

Complements the course given by Jesus Fernandez Villaverde.

About Our Work Environment Today

- We will use Julia:
  - a language, specifically designed for numerical science
- We use Pluto.jl on JuliaHub
- Two commands:
  - Shift+Enter: run a cell
  - Ctrl+Enter: run a cell and create a new one
    - for mac: Ctrl replaced by Cmd

```
12
```

```julia
3 + 9
```

About the Future

There is a variety of (free!) environments to program in Julia

- the console
- JupyterLab
- Visual Studio Code

Tip

To install Julia, instead of downloading the regular package, lookup juliaup. You get one single command to install a Julia multiplexer which can handle many Julia versions at once.

About Julia

Why Julia?

- it is opensource. There are *hundreds* of libraries
- syntax inspired by Matlab but:

    - more consistent
    - much better!
- it is Just in Time (JIT) compiled
    - one can quickly experiment with code like an interpreted language
    - code gets translated using LLVM to run at machine speed
- there is a vibrant community with tons of online resources
- you can start easy and progressively get full control

```
[4, 16, 36, 64, 100, 144, 196, 256, 324, 400, 484, 576, 676, 784, 900, 1024, 1156, 1296, 144
```

```julia
[n^2 for n in 1:100 if mod(n,2)==0 ]
```

An example of what not to do in Matlab

```
stupid_loop (generic function with 1 method)
```

```julia
function stupid_loop(I,J,K)
    t = 0.0
    for i=1:I
        for j=1:J
            for k = 1:K
                t += 1.0
            end
        end
    end
    return t
end
```

```
1.0e9
```

```julia
@time stupid_loop(1000,1000,1000)
```

```
0.892438 seconds
```

```julia
@code_llvm stupid_loop(10,10,10)
```

```
;  @ /home/pablo/Teaching/workshops/african_econometric_society/notebooks/
day_1_1_intro.jl#==#ad4ba04b-83b1-4394-a3d6-0429b4693129:1 within `stupid_loo
p`
define double @julia_stupid_loop_2689(i64 signext %0, i64 signext %1, i64 sig
next %2) #0 {
top:
;  @ /home/pablo/Teaching/workshops/african_econometric_society/notebooks/day
_1_1_intro.jl#==#ad4ba04b-83b1-4394-a3d6-0429b4693129:3 within `stupid_loop`
; ┌ @ range.jl:5 within `Colon`
; │┌ @ range.jl:397 within `UnitRange`
; ││┌ @ range.jl:404 within `unitrange_last`
     %.inv = icmp sgt i64 %0, 0
     %. = select i1 %.inv, i64 %0, i64 0
; └└└
  br i1 %.inv, label %L16.preheader, label %L85

L16.preheader:                                    ; preds = %top
  %.inv26 = icmp sgt i64 %1, 0
  %.24 = select i1 %.inv26, i64 %1, i64 0
  %.inv27 = icmp sgt i64 %2, 0
  %.25 = select i1 %.inv27, i64 %2, i64 0
;  @ /home/pablo/Teaching/workshops/african_econometric_society/notebooks/day
_1_1_intro.jl#==#ad4ba04b-83b1-4394-a3d6-0429b4693129:4 within `stupid_loop`
  %3 = select i1 %.inv26, i1 %.inv27, i1 false
  br i1 %3, label %L33.preheader.split.us.us.us, label %L85

L74.us.us:                                        ; preds = %L63.us.us.us
;  @ /home/pablo/Teaching/workshops/african_econometric_society/notebooks/day
_1_1_intro.jl#==#ad4ba04b-83b1-4394-a3d6-0429b4693129:9 within `stupid_loop`
; ┌ @ range.jl:891 within `iterate`
; │┌ @ promotion.jl:499 within `==`
    %.not20.us.us = icmp eq i64 %value_phi3.us.us, %
```

```julia
@code_native stupid_loop(10,10,10)
```

```
    .text
    .file   "stupid_loop"
    .section    .rodata.cst8,"aM",@progbits,8
    .p2align    3                               # -- Begin function julia_stu
pid_loop_2702
.LCPI0_0:
    .quad   0x3ff0000000000000              # double 1
    .text
    .globl  julia_stupid_loop_2702
    .p2align    4, 0x90
    .type   julia_stupid_loop_2702,@function
julia_stupid_loop_2702:                 # @julia_stupid_loop_2702
; ┌ @ /home/pablo/Teaching/workshops/african_econometric_society/notebooks/da
y_1_1_intro.jl#==#ad4ba04b-83b1-4394-a3d6-0429b4693129:1 within `stupid_loop`
    .cfi_startproc
# %bb.0:                                # %top
    vxorpd  %xmm0, %xmm0, %xmm0
; │ @ /home/pablo/Teaching/workshops/african_econometric_society/notebooks/da
y_1_1_intro.jl#==#ad4ba04b-83b1-4394-a3d6-0429b4693129:3 within `stupid_loop`
; │┌ @ range.jl:5 within `Colon`
; ││┌ @ range.jl:397 within `UnitRange`
; │││┌ @ range.jl:404 within `unitrange_last`
    testq   %rdi, %rdi
; │└└└
    jle .LBB0_9
# %bb.1:                                # %L16.preheader
    testq   %rsi, %rsi
; │ @ /home/pablo/Teaching/workshops/african_econometric_society/notebooks/da
y_1_1_intro.jl#==#ad4ba04b-83b1-4394-a3d6-0429b4693129:4 within `stupid_loop`
    jle .LBB0_9
# %bb.2:                                # %L16.preheader
```

Your Journey with Julia

stage 1:

- basic types (numbers, strings)
- define functions
- use macros
- enjoy syntactic sugar

stage 2:

- experiment with many libraries
- write type-stable code
- define own types
- use multiple dispatch

stage 3:

- metaprogramming
- avoid memory allocations
- ...

First discoveries

Basic Types

The following code assigns a number to a variable. Change the variable so that it contains: a float (64 bits), an integer, a float 32, a boolean (true or false)

```
using Bits
```

```
a = 2.0f0
```

```
a = Float32(2)
```

```
Float32
```

```
typeof(a)
```

```
4
```

```
sizeof(a) # the answer is given in bytes (i.e. 8 bits on most computers)
```

```
122013682599111006870123878542304692625357434280319284219241358838584537315388199760549644
```

```
factorial(BigInt(500))
```

```
Bits.BitVector1{Float32}: [false, false, false, false, false, false, false, false, false,
```

```
bits(a)
```

Operations on Numbers

No big suprise

```
5
```

```
2 + 3 # try with +, -, *, /, ^
```

```
5.0
```

```
2 + 3.0 # conversion to the most precise type
```

```
0.4
```

```
2/5 # division returns a float
```

```
1
```

```
5%2
```

```
11//15
```

```
2//5 + 1//3 # fraction
```

```
27
```

```
3^3
```

Strings (1)

There are several ways to define a string: "... " for regular strings, for multiline """ ...""".

Strings can be manipulated as objects and "printed" to the terminal using function `println()`

Warning:

In Julia quotes are used to denote a single character as in 'a'

```
txt1 =
"This is a regular string. New lines must be signalled with the \\n special charachter.\nTl
```

```
txt1 = "This is a regular string. New lines must be signalled with the \\n special
charachter.\nThe string continues.\nIf you wish to use an anteslash in the string,
you need to 'escape' it with another anteslash: \\"
```

```
# print string txt1
println(txt1)
```

```
This is a regular string. New lines must be signalled with the \n special ch
arachter.
The string continues.
If you wish to use an anteslash in the string, you need to 'escape' it with ano
ther anteslash: \
```

```
txt2 =
"In a multiline string, the first newline is ignored, \nWords flow together, in syntax the
```

```
txt2 = """
In a multiline string, the first newline is ignored,
Words flow together, in syntax they're deployed.
Yet in this silent space, a tale can be enjoyed,
As lines of code and verse are cleverly employed.
"""
```

```
# print string txt2
println(txt2)
```

```
In a multiline string, the first newline is ignored,
Words flow together, in syntax they're deployed.
Yet in this silent space, a tale can be enjoyed,
As lines of code and verse are cleverly employed.
```

Strings (2)

Strings can contain any unicode character including many alphabets and emoticons. To interpolate a value inside a string we use the dollar sign '$'.

Tip

Like in Jupyter notebooks, greek letters can be entered by typing their latex representation and hitting Tab

```
0.5
```

```julia
begin
    n = 10
    🦈 = 0.3
    🐋 = 0.5
end
```

```julia
println("After $(n) iterations, the populations of sharks and whales are respectively equal to 🦈=$🦈 et 🐋=$🐋")
```

```
After 10 iterations, the populations of sharks and whales are respectively e
qual to 🦈=0.3 et 🐋=0.5
```

```julia
# Change the code above to use unicode symbols instead of alpha and beta
```

```julia
begin
    println("Baby 🦈")
    println("tdudtu")
end
```

```
Baby 🦈
tdudtu
```

Logical Operations

Logical operators (==, <, <=, !=, >, >=) always return a boolean

```
b = 3
```

```julia
b = 3
```

```
false
```

```julia
a == b
```

```
false
```

```julia
2 == 5.2
```

```
false
```

```julia
nothing == Nothing
```

```
true
```

```julia
2 != 5.2
```

```
true
```

```julia
2 < 5.2
```

They can be combined using logical operators '&' (and) and '|'

```
true
```

```julia
(2 < 3) & (1 < 2)
```

```
true
```

```julia
(2 < 3) |  (3 < 2)
```

More Julia

```julia
# functions
```

```julia
# containers
```

```julia
# arrays
```

```julia
# control flow
```

Table of Contents