

# Julia Syntax and Algorithm

Hui-Jun Chen

The Ohio State University

September 29, 2022

# THE MOST important coding rules

Leggi le leggi

(“Read the manuals” in Italian)

# Resources on Syntax

- Julia Official Tutorial: <https://julialang.org/learning/tutorials/>
- Wikibook on Introducing Julia: [https://en.wikibooks.org/wiki/Introducing\\_Julia](https://en.wikibooks.org/wiki/Introducing_Julia)
- QuantEcon w/ Julia: <https://julia.quantecon.org/intro.html>
- Julia in 100 Seconds: [https://www.youtube.com/watch?v=JYs\\_94znYy0](https://www.youtube.com/watch?v=JYs_94znYy0)

# The REPL

REPL stands for **R**ead, **E**valuate, **P**rint, and **L**oops.

Julia's REPL is the best I have ever seen, includes

- Unicode transformation: type `\alpha` and `tab` leads to  $\alpha$
- Package management: type `]` to enter `Pkg` mode to `add` packages
- Manual query: type `?` to enter `help` mode and find function manual
- Tab completion: type `\a1` and `tab` gives you possible commands
- `↑` / `↓`: up/down arrow key cycle through executed command history

# The Language: Good and Bad

Julia language is designed with scientific computing in mind, and thus

- Unicode variable: directly use  $\alpha$  as variable, not `alpha`.
- Multiple dispatch: multiple “methods” in one function for input types
- Type system: use `struct` to build custom types ( $\approx$  but  $\neq$  OOP)

But also have some weird behavior that I am not used to:

- **Weird scope**: variables defined inside loops (`while`, `for`) are **local**.
- Speed needs discipline: **well-written** code v.s. **sloppy-written** code
- Memory usage: might directly crash the Julia session ( $\therefore$  LLVM?)

Best practice?

Still Searching...

## Syntax: generating a grid

- Usually the Macro coding starts with the grids of choice variables.
- A grid is a finite sample of continuous choice variable.
- Key to construct a grid is the `collect` and `range` function.
- `range` syntax requires `start` pt, `stop` pt and `length` of this grid
- `collect` then “collect” this `range` object into an array.

```
cnum = 100  
lnum = 100  
cgrid = collect( range( 0.01, 10.0, length = cnum ) )  
lgrid = collect( range( 0.01, 1.0, length = lnum ) )
```

## Syntax: Array manipulation

To get one element of a grid, we use `[]` syntax.

```
cval = cgrid[1] # get the first element of cgrid  
lval = lgrid[5] # get the fifth element of lgrid
```

To create an array, you can use manual or automatic way.

```
# manually type all the elements  
a = [1.0, 2.0, 3.0, 4.0, 5.0 ]  
  
# automatically generate an "empty" array  
#           type      dim empty row column  
utility = Array{Float64, 2}(undef, cnum, lnum)  
utility = zeros(cnum, lnum) # zero array  
utility = ones(cnum, lnum) # one array
```

## Syntax: `for` loop

To calculate the utility value at each  $(C, l)$  bundle, use `for` loop

```
utility = Array{Float64, 2}(undef, cnum, lnum)
for indl in 1:1:lnum
    # get the each value in leisure grid
    lval = lgrid[indl]
    for indc in 1:1:cnum
        # get the each value in consumption grid
        cval = cgrid[indc]
        # log utility in both c and l
        utility[indc, indl] = log(cval) + log(lval)
    end
end
```

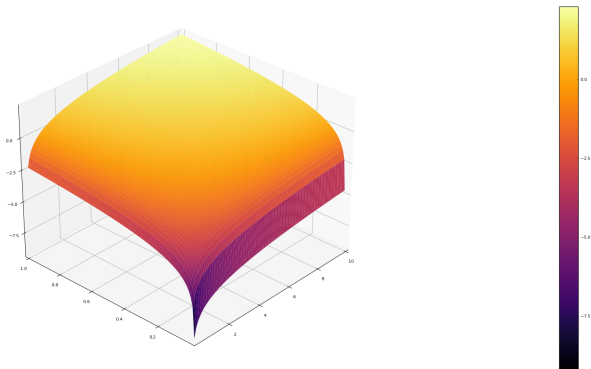


## Syntax: 3-D plotting

Install `Plots` and `PyPlot` by typing `] add Plots PyPlot`

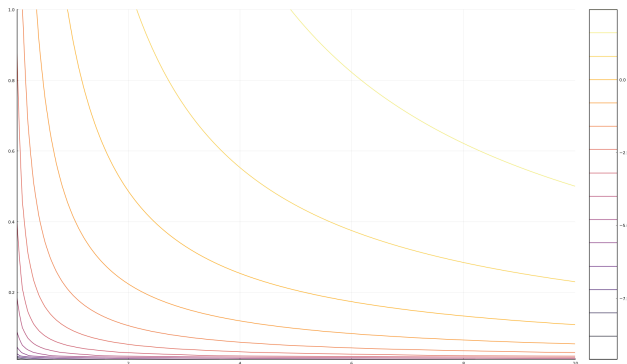
Plot the `utility` array by

```
using Plots; pyplot();  
surface(cgrid, lgrid, utility) # 3-D figure
```



# Syntax: contour plotting

```
using Plots; pyplot();  
contour(cgrid, lgrid, utility)  # contour figure
```



## Syntax: `println` print something out

To show some info inside the `for` loop, `println` is a convenient tool.  
If you want to know what  $(C, l)$  bundle leads to  $U(C, l) = 0.0$ ,

```
for indl in 1:1:lnum
    for indc in 1:1:cnum
        # the abs of u is close enough to 0.0
        if abs(utility[indc, indl]) < 1e-2
            # '$': string interpolation (IMO inefficient)
            println("U ~ 0 at (C, l) = ($indc, $indl)")
        end
    end
end
```

`println` v.s. `print`: `println` add additional `\n`

## Syntax: `while` loop

`while` loop mostly used when iteration only halt in some conditions. In my experience it is mostly used if something needs convergence. The following code is **NOT** an efficient way to find minimum location. (should use `argmin` for `minimum` and `argmax` for `maximum`)

```
dist = 1.0; iter = 0;
while (dist > 1e-2)
    iter += 1 # same as "iter = iter + 1"
    indc = rand(1:1:cnum); indl = rand(1:1:lnum)
    dist = utility[indc, indl] - minimum(utility)
    if (dist < 1e-2)
        println("Find minimum at ($indc, $indl)")
        println("Iterates $iter times")
    end
end
```

# Syntax: Rounding

Mostly for exam / standardization purpose.

```
round(pi)                # 3.0
round(pi, digits = 1)    # 3.1
round(pi, digits = 2)    # 3.14
round(pi, digits = 3)    # 3.142
round(pi, digits = 4)    # 3.1416
round(pi, digits = 5)    # 3.14159
```

## Grid search

Just calculate value **on** the grid points! Like `for loop` slide

Recall the formula with gov spending:

$$\max_l \frac{(z(1-l)^{1-\alpha} - G)^{1-b}}{1-b} + \frac{l^{1-d}}{1-d}. \quad (1)$$

We want to solve  $l(z, G)$ , but how to choose the `Ggrid`?

From the FOC we know

$$G = F(l) = z(1-l)^{1-\alpha} - \left[ \frac{l^{-d}}{(1-\alpha)z(1-l)^{-\alpha}} \right]^{-\frac{1}{b}}. \quad (2)$$

Our first step starts with generating a TFP grid:

```
znum = 100
zgrid = collect( range( 0.8, 1.2, length = znum ) )
```

# Grid search: preperation

We want to find the upper/lower bound of `Ggrid`:

```
a = 1/2; b = 2; d = 3/2;
GovFOC(z, l) = z*(1-l)^(1-a) - # line continuation!
    ( (1^(-d)) / ( (1-a)*z*(1-l)^(-a) ) )^(-1/b)
# upper & lower bound of Ggrid
Gbound = Array{Float64, 2}(undef, znum, 2)
for indz = 1:1:znum
    zval = zgrid[indz]
    Gbound[indz, 1] = GovFOC(zval, 0.99) # lower bound
    Gbound[indz, 2] = GovFOC(zval, 0.01) # upper bound
end
```

## Grid search: preperation (cont.)

```
# lower bound should higher than 0.01
Glow = max(0.0, minimum(Gbound))
Ghigh = maximum(Gbound)
# build Ggrid
Gnum = 100
Ggrid = collect( range( Glow, Ghigh, length = Gnum ) )
# build lgrid
lnum = 100
lgrid = collect( range( 0.01, 1.0, length = lnum ) )
```

and then find the optimal leisure using the value on this grid:



# Grid search: structure

```
a = 1/2; b = 2; d = 3/2;  
# define implicit utility function  
utility(l, z, G) = ( ( z*(1-l)^(1-a) - G )^(1-b) ) /  
                  ( 1-b ) +  
                  ( 1^(1-d) ) / (1-d)  
  
# Array for storage  
## for temporary storage  
uvec = Array{Float64, 1}(undef, lnum)  
## for optimal utility value  
ustar = Array{Float64, 2}(undef, znum, Gnum)  
## for optimal leisure given z, G  
lstar = Array{Float64, 2}(undef, znum, Gnum)
```

## Grid search: structure (cont.)

```
for indG = 1:1:Gnum
    Gval = Ggrid[indG]
    for indz = 1:1:znum
        zval = zgrid[indz]
        for indl = 1:1:lnum
            lval = lgrid[indl]
            cval = zval*(1-lval)^(1-a) - Gval
            uvec[indl] = ( cval < 0.0 ? -Inf :
                           utility(lval, zval, Gval) )
        end
        ustar[indz, indG] = maximum(uvec)
    end
end
```

## Grid search: analysis

Notice that in previous slide, I check whether `cval < 0.0` and also we find the highest utility and the corresponding  $(z, G)$  value by

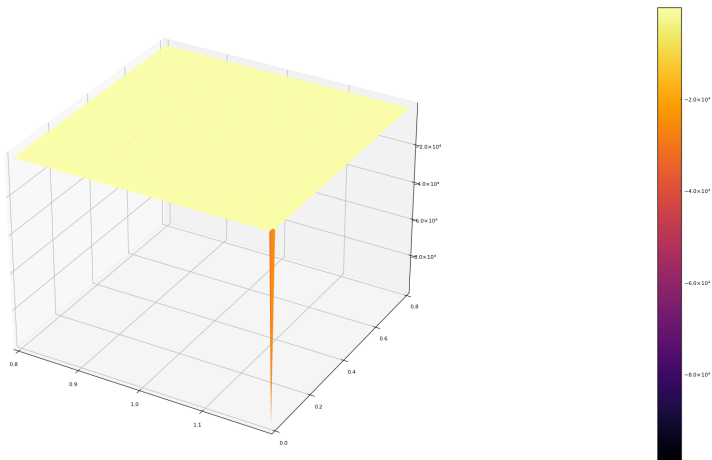
```
umax = maximum(ustar)
zloc = argmax(ustar)[1]
Gloc = argmax(ustar)[2]
zmax = zgrid[zloc]
Gmax = Ggrid[Gloc]
```

But if you plot you will see that the plot is slightly “off”:

```
using Plots; pyplot()
surface(zgrid, Ggrid, ustar)
```

$\therefore$  large negative point that drag down the scale of every point.

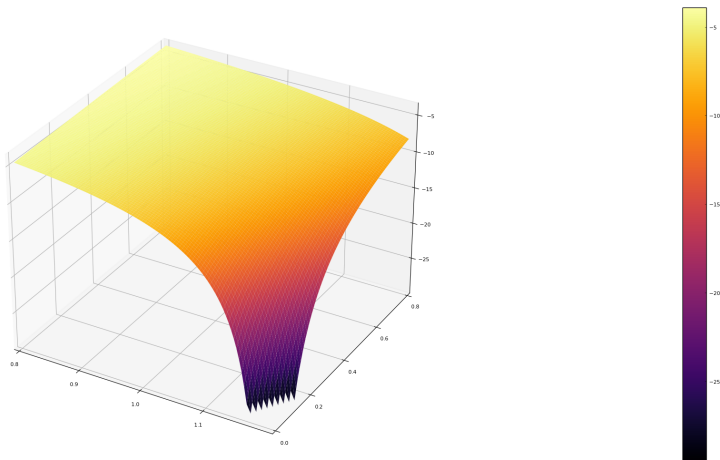
# grid search: misleading figure



Grid search: revising (erase `uval < -30.0`)

```
for indG = 1:1:Gnum
    Gval = Ggrid[indG]
    for indz = 1:1:znum
        zval = zgrid[indz]
        for indl = 1:1:lnum
            lval = lgrid[indl]
            cval = zval*(1-lval)^(1-a) - Gval
            uval = utility(lval, zval, Gval)
            uvec[indl] = ( (cval < 0.0 || uval < -30.0)
                          ? -Inf : uval )
        end
        ustar[indz, indG] = maximum(uvec)
    end
end
```

# grid search: correct figure



## Grid search method: additional details

Calculate on the grid point  $\Rightarrow$  result are correct but speed is slow.

Notice that when you choose the grid points, better to avoid some value:

### Example

When I create `cgrid` and `lgrid`, I avoid the start point of 0.0, but 0.01, since  $\log(0.0) = \infty$ .

In general, if theoretical range, say leisure, is  $[0, 1]$ , then it is safe to build a grid from  $[0.01, 0.99]$ .