**Documentation for ECON 899 Code**

Author: Garrett Anstreicher

This document is intended to provide some light documentation for my code for Econ 899. This code is all written in the Julia programming language but is intended to be concise and well-commented enough to be a helpful resource for students writing in whatever language they prefer. On advantage of Julia is that optimizing performance need not sacrifice readability (as it frequently does in Matlab, for instance), so the logic of the provided code will hopefully be clear and intuitive.

Those interested in coding in Julia may download the language here: <https://julialang.org/>. A splendid resource for those wanting to learn about Julia can be found here: <https://lectures.quantecon.org/jl/>

I am also fond of the Juno IDE for Julia, which can be found here: <http://junolab.org/>.

**Basic Structure**

All code for problem sets are broken up into two pieces:

* **[#]\_[name]\_model.jl:** Contains all the main functions that compute whatever model the problem set has you working on.
* **[#]\_[name]\_compute.jl:** Runs said functions and creates whatever plots or reports of model results are called for in the problem set.

All computationally intensive procedures are placed inside functions for speed purposes (see performance section). Operations not inside functions are ones that generally execute instantaneously, like plotting (conditional on the plots package having been precompiled, which takes a little bit.)

I recommend reading the code for 02\_optimal\_growth first to get a sense for how I typically structure my code before going on to read the more serious programs.

**Performance**

Thematically, there are three things I do to maximize the speed of my code:

* Putting everything in functions, as mentioned before
* Extensive type declarations. Julia appreciates it when it knows the type (e.g. integer, float, array, etc) of a variable it is working with.
* Minimizing taking the powers of floating-point numbers. Taking powers of floats is slow in Julia, so I often construct a grid of possible utilities before value function iteration so that I don’t have to compute new utilities (which often entails power functions) with every iteration of the Bellman equation.

Parallelization is easy in Julia but seldom useful in this class, as the procedure of distributing tasks to multiple cores and re-collecting the results is often slower than just having one core do all the work if the individual tasks can be completed quickly. Resources for parallel computing in Julia can be found here: <https://docs.julialang.org/en/v1/manual/parallel-computing/index.html>.

Good luck!

Garrett