HW4 - João Lazzaro

In HW4 we were asked to implement Krusell & Smith (1998) model version with endogenous labor using the algorithm proposed by the authors. To check my results, I will try to replicate the figures and coefficients presented in the working paper version of the article (http://www.econ.yale.edu/smith/rcer399.pdf (http://www.econ.yale.edu/smith/rcer399.pdf)) since it contains more information than the final published paper.

Krusell & Smith Model

This section explains the model and I also comment on the strategies I took to solve it.

There is a competitive representative firm which uses capital K_t and labor H_t as inputs and faces a productivity shock Z_t. From the firm's problem we get the wages and capital return as a function of the aggregate states:

$$w(K_t, H_t, Z_t) = (1-\alpha) Z_t K_t^{\alpha} H_t^{-\alpha}$$

$$R(K_t, H_t, Z_t) = \alpha Z_t K_t^{\alpha-1} H_t^{1-\alpha} + 1 - \delta$$

Z is assumed to have 2 values: $[0.99, 1.01]$.

The economy also consists of a continuum of ex ante identical households with unit mass. Each period, agents face an idiosyncratic shock e that determines whether they are employed, $e_{n,t} = 1$,or unemployed, $e_{n,t} = 0$. An employed agent earns wage w per unit of labor. Markets are incomplete and agents can only save through capital accumulation, individual capital is denoted by a_t and R is the net rate of return. The agent must also keep track of the aggregate states of the economy. We will assume that the aggregates follow the following functional forms:

$\ln K' = b_{0,g} + b_{1,g} \ln K$ and $\ln H' = d_{0,g} + d_{1,g} \ln K$ when $Z = 1.01$

$\ln K' = b_{0,b} + b_{1,b} \ln K$ and $\ln H' = d_{0,b} + d_{1,b} \ln K$ when $Z = 0.99$

The agent recursive problem is thus the following:

$$V(a, e, K, H, Z) = max_{c,a',n} \frac{\left(c^{\eta}(1-n)^{1-\eta}\right)^{1-\mu}}{1-\mu} + \beta E\left[V(a', e', K', H', Z') | e, K, H, Z\right]$$

Subject to the budget constraint:

$$c \leq R(K,H,Z)a - a' + w(K,H,Z)en$$

$$a' \geq 0$$

And the beliefs: $\ln K' = b_{0,g} + b_{1,g} \ln K$ and $\ln H' = d_{0,g} + d_{1,g} \ln K$ when $Z = 1.01$

$\ln K' = b_{0,b} + b_{1,b} \ln K$ and $\ln H' = d_{0,b} + d_{1,b} \ln K$ when $Z = 0.99$

A recursive competitive equilibrium is an allocation that solves the agent problem and the aggregates moves consistently with the beliefs.

We parametrize the model following precisely Krusell & Smith parameters:

$\beta = 0.99,\ \delta = 0.0025,\ \mu = 1$ (which implies log utility). The stochastic process for (Z, e) is set so the unemployment rate in good states is 0.04 and 0.1 in bad times. The average duration of good and times is 8 periods and the average duration of unemployment is 1.5 in good times and 2.5 in bad times.

Solving the Consumer Problem

Due to the quantity of states, this problem is hard to solve. The usual methods I used in the previous homeworks were not able to handle this problem. Although Value Function Iteration or Euler Equation iteration worked in this case, they both were taking a ridiculous amount of time. Approximately 4 hours to solve the agent problem with a fairly small grid. The challenge was to solve the system of equations (in the case of Euler Equations) or the optimization problem (VFI) for labor and assets. It is also obvious that discrete state space that I use often for simple problems, is also not an option since it would require an enormous amount of memory.

Instead of trying to speed up the solver, I decided to try another approach. Carroll's endogenous grid method proved to be very efficient in handling this problem. With a grid 6 times larger, the method solves the agent problem in less than 5 minutes since it does not involve any optimization. Usually, the endogenous grid method does not apply directly to endogenous labor problems (in the simple growth model the method would require some tweaks (see https://www.sas.upenn.edu/~jesusfv/Endogenous_Grid.pdf (https://www.sas.upenn.edu/~jesusfv/Endogenous_Grid.pdf) for a discussion) but since wages are exogenous to consumer decisions in this case it works:

Consider the labor labor consumption FOC:

$$\frac{u_n}{u_c} = w$$

Plugging the functional forms, we can solve for n:

$n^*(c,e,K,H,Z) = 1 - \frac{1-\eta}{\eta}\frac{c}{w(K,H,Z)}$ if $e = 1$ and $n = 0$ otherwise.

From the intertemporal Euler equation we have:

$$u_c(c,n) = \beta E[R(K',H',Z')u_c(c',n')|e,K,H,Z]$$

The strategy here is to define a grid for assets in $t+1$ and to guess a functional form for c so given that gues and the labor equation found above, the right hand side of this eqution is a constant (named here RHS). Plugging the function forms for u and n^*, we get that:

$$c = \left[\frac{RHS}{\eta}\left(\frac{(1-\eta)}{\eta w(K,H,Z)}\right)^{-(1-\mu)(1-\eta)}\right]^{\frac{-1}{\mu}}$$

if $e > 0$ and $c = \left[\frac{RHS}{\eta}\right]^{\frac{-1}{\mu}}$ otherwise.

With these equations we may proceed to an usual implementation of the endogenous grid method as in Violante's slides. The code below implements the method:

In [6]:
```
using Interpolations, ProgressMeter
using Distributions, Random, DataFrames, GLM
#using Distributed, SharedArrays
include("CRRA_utility.jl")
include("functions.jl") #load some auxiliary, not interesting functions

#Wages functions
R(K::Float64,H::Float64,z::Float64;α=α::Float64,δ=δ::Float64)= z*α*K^(α-1.0)*H^
(1.0-α) + (1.0-δ)
w(K::Float64,H::Float64,z::Float64;α=α::Float64) = z*(1.0-α)*K^(α)*H^(-α)

#Law of motion functions for aggregate states
K1(K::Float64,z::Float64;b=b::Array{Float64,2},Z= Z::Array{Float64,1}) = exp(b[
findfirst(Z.==z),1]+b[findfirst(Z.==z),2]*log(K))
H0(K::Float64,z::Float64;d=d::Array{Float64,2},Z = Z::Array{Float64,1}) = exp(d
[findfirst(Z.==z),1]+d[findfirst(Z.==z),2]*log(K))

#Defining asset investment function:
#c(a,e, n ,a1,k,h,z) =  R(k,h,z)*a+e*w(k,h,z)*n-a1
a1star(c,a,e,k,h,z) = R(k,h,z)*a-c+w(k,h,z)*e*nstar(c,e,k,h,z)

#defining optimal labor choice. This comes from labor FOC:
nstar(c,e,k,h,z;η = η,lbar = lbar) = (e>0.0)*min(max((lbar - (1-η)/η * c/w(k,h,
z)),0.0),lbar)


function ENDOGENOUSGRID_KS(A::Array{Float64,1},A1::Array{Float64,1},E::Array{Fl
oat64,1},Z::Array{Float64,1},transmat::Array{Float64,2},states::NTuple,
    K::Array{Float64,1}, H::Array{Float64,1} ,b::Array{Float64,2},d::Array{Floa
t64,2};α=α::Float64,β = β::Float64, η=η::Float64, μ=μ::Float64,
     tol = 1e-6, lbar=lbar::Float64 ,  policy= zeros(nA,nE,nK,nH,nZ)::Array{Flo
at64,6},update_policy=0.5::Float64,updaterule = false)
    #This function solves the agent problem using the endogenous grid method.
    #A: Individual Asset grid in t!
    #A: Individual Asset grid in t+1
    #E: Individual productivity grid
    #Z: Aggregate shocks grid
    #transmat: transmat object with all the transition matrices
    #states: A tuple which each element is a pair of possible states
    #K: Aggregate capital grid
    #H: Aggregate labor grid
    #b: capital law of motion coefficients
    #d: labor law of motion coefficients

    #OPTIONAL ARGUMENTS
    #update_policy: Damping parameter
    #policy: Initial grid guess for the policy function
    #udpdaterule: false for regular update rule, true for extrapolation (see be
low)
    #the othere parameters are self explanatory.
    nA::Int64 = length(A)
    nZ::Int64 = length(Z)
    nE::Int64 = length(E)
    nH::Int64 = length(H)

    #RETURN the grid for the agents policy functions.

    #Guess for policy functions
    #itpn = LinearInterpolation((A,E,K,H,Z),policy[:,:,:,:,:,2],
    #extrapolation_bc=Line())
    itpc = LinearInterpolation((A,E,K,H,Z),policy, extrapolation_bc=Line())
    policy_c(a,e,k,h,z) = itpc(a,e,k,h,z)
    policy_n(a,e,k,h,z) = nstar(policy_c(a,e,k,h,z),e,k,h,z)
```

```
Out[6]: EE (generic function with 1 method)
```

Finding the Aggregate States law of motion

The algorithm proposed by Krusell & Smith is fairly simple. It is enough to simulate the economy for a large number of agents and periods and then regress the law of motion coeffients by OLS. One quick comment on the generation of shocks is that I must make sure that the distribution is consistent. For example, I generate the series of Z and for each agent the draw $e_{n,t}$ must take into acount $e_{n,t-1}, Z_{t-1}, Z_t$ since the shocks are not independent.

During the simulation I took some time to figure out that one must ensure that labor market clear every period (since capital is predetermined, this is not an issue in that market). To do so, we must each period guess an initial aggregate state for labor and then loop until the convergence (i.e. the aggregate of the indiviual labor decisions equals the aggregate labor guess). The code below implements the simulation:

In [7]:
```
function KrusselSmithENDOGENOUS(A::Array{Float64,1},A1::Array{Float64,1},
    E::Array{Float64,1},Z::Array{Float64,1},tmat::TransitionMatrix,states::NTuple{4,Array{Float64,1}},
    K::Array{Float64,1},H::Array{Float64,1},  b::Array{Float64,2},d::Array{Float64,2};
    α = α::Float64,β = β::Float64, η = η::Float64, μ=μ::Float64, tol= 1e-6::Float64,
    update_policy=0.5::Float64,updateb= 0.3::Float64,N::Int64=5000,T::Int64=11000,
    discard::Int64=1000,seed::Int64= 2803,lbar=lbar::Float64,updaterule = false)
    #This performs KS algorithm
    #A: Individual Asset grid in t!
    #A1: Individual Asset grid in t+1
    #E: Individual productivity grid
    #Z: Aggregate shocks grid
    #tmat: transmat object with all the transition matrices
    #states: A tuple which each element is a pair of possible states
    #K: Aggregate capital grid
    #H: Aggregate labor grid
    #b: capital law of motion coefficients
    #d: labor law of motion coefficients

    #OPTIONAL ARGUMENTS
    #update_policy: Damping parameter for the agent Problem
    #update_b: Damping parameter for the law of motion updates
    #N: Number of agents in the simulation
    #T: Length of the simulated time series
    #discard: number of time periods discarded for ergodicity
    #seed: set the random seed for comparable results
    #policy: Initial grid guess for the policy function
    #the othere parameters are self explanatory.


    #RETURN
    #b: Updated parameter for aggregate capital law of motions
    #d: Updated parameter for aggregate labor law of motions
    #nsim: NxT matrix with simulated labor path for each agent n
    #asim: NxT matrix with simulated assets path for each agent n
    #Ksim: T vector with simulated aggregate  capital
    #Hsim: T vector with simulated aggregate  Labor ,
    #policygrid: Grid with agents policy functions
    #K: new updated grid for aggregate capita (not used for now),
    #R2b,R2d: R-squared of b and d regressions
    #zsimd: T vector with simulated aggregate shocks
    #esim: NxT matrix with idyosincratic employment shock for each agent

    #Getting lengths
    nA::Int64 = length(A)
    nZ::Int64 = length(Z)
    nE::Int64 = length(E)
    nH::Int64 = length(H)
    nK::Int64 = length(K)

    println("Starting Krusell Smith. We are using $(nA) gridpoints for assets and")
    println("a sample of N=$(N), T=$(T). Go somewhere else, this will take a while.")

    transmat::Array{Float64,2} = tmat.P #Getting the transition matrix for the agent

    d=d::Array{Float64,2}
```

```
Out[7]: KrusselSmithENDOGENOUS (generic function with 1 method)
```

Results

I don't run the code on this notebook because it does not look nice and also the server runs it much faster than my computer. The file main.jl replicates this results. I load the directly the results of a previous run of the code.

```
In [12]: using JLD2, FileIO
         @load  "save_variables/variables_nA30.jld2"
         using Plots
```

The coefficients I find are the following:

$$lnK_{t+1} = 0.114 + 0.953 lnK_t$$

$R^2 = 0.999$ in bad times and:

$$lnK_{t+1} = 0.128 + 0.949 lnK_t$$

$R^2 = 0.999$ in good times times. The result for bad times is exactly the same as Krusell & Smith while they found for good times:

$$lnK_{t+1} = 0.123 + 0.951 lnK_t$$

For labor:

$$lnH_t = -0.560 - 0.260 lnK_t$$

$R^2 = 0.989$ in bad times and

$$lnH_t = -0.485 - 0.267 lnK_t$$

$R^2 = 0.992$ in good times times. Krusell & Smith results are:

$$lnH_t = -0.592 - 0.252 lnK_t$$

in bad times and

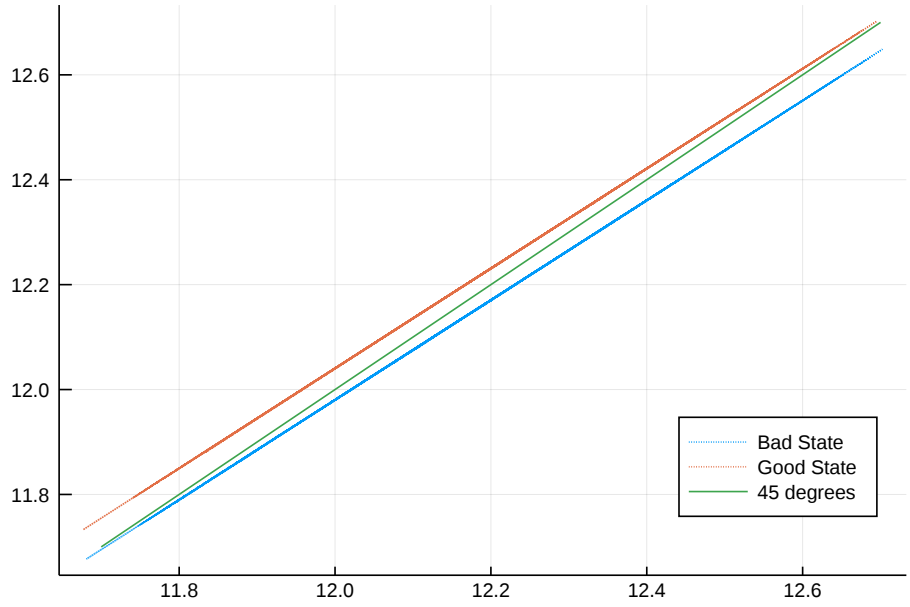
$$lnH_t = -0.544 - 0.252 lnK_t$$

in good times.

Now we plot the relationship of aggregate capital tommorow vs today which is undistinguishable from paper's figure 7:

In [13]:
```
#Aggregate Capital vs capital:
plot((Ksim[discard+1:end-1][zsim[1:end-1].==Z[1]]) ,(Ksim[discard+2:end][zsim[1
:end-1].==Z[1]]),
xlabel = "\$K_t\$",ylabel = "\$K_{t+1}\$",linestyle = :dot, label = "Bad State"
,legend = :bottomright)
plot!((Ksim[discard+1:end-1][zsim[1:end-1].==Z[2]]) ,(Ksim[discard+2:end][zsim[
1:end-1].==Z[2]]),
linestyle = :dot, label = "Good State")
plot!(11.7:0.1:12.7 ,11.7:0.1:12.7, label = "45 degrees")
```

Out[13]:

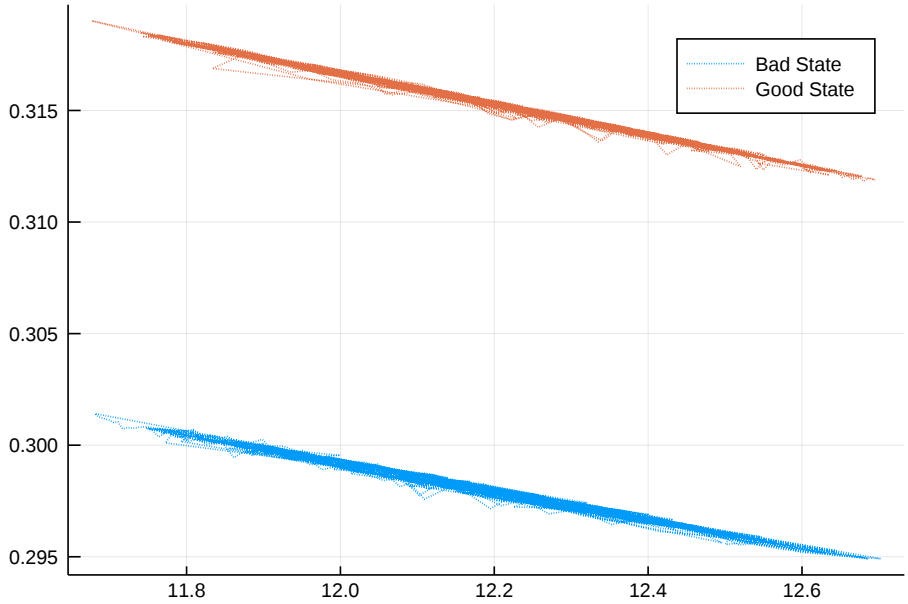



```
sh: dvipng: command not found
dvipng: PNG conversion failed
sh: dvipng: command not found
dvipng: PNG conversion failed
```

Paper figure 8 plots the relationship between aggregate labor and aggregate capital:

In [14]:
```
#Aggregate labor vs capital:
plot((Ksim[discard+1:end][zsim.==Z[1]]) ,(Hsim[discard+1:end][zsim.==Z[1]]),
xlabel = "\$K_t\$",ylabel = "\$H_t\$",linestyle = :dot, label = "Bad State")
plot!((Ksim[discard+1:end][zsim.==Z[2]]) ,(Hsim[discard+1:end][zsim.==Z[2]]),
linestyle = :dot,label = "Good State")
```

Out[14]:




```
sh: dvipng: command not found
dvipng: PNG conversion failed
sh: dvipng: command not found
dvipng: PNG conversion failed
```

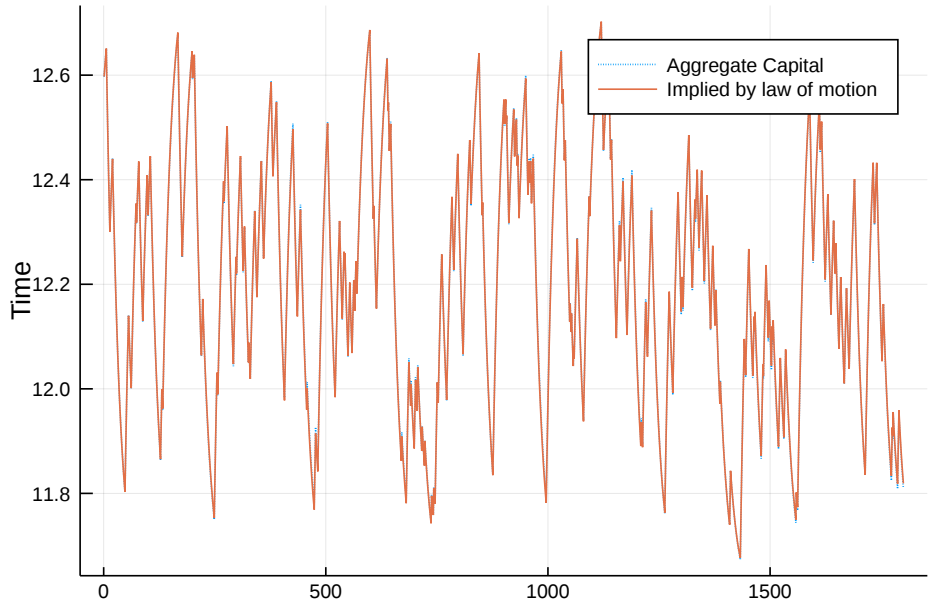
Although Krusell & Smith do not report the figures below, it has become somewhat common in the literature to plot the evolution of the actual Aggregate variables vs what the law of motion would imply:

In [16]:
```
#Law of motion functions for aggregate states
K1(K::Float64,z::Float64;b=b::Array{Float64,2},Z= Z::Array{Float64,1}) = exp(b[
findfirst(Z.==z),1]+b[findfirst(Z.==z),2]*log(K))
H0(K::Float64,z::Float64;d=d::Array{Float64,2},Z = Z::Array{Float64,1}) = exp(d
[findfirst(Z.==z),1]+d[findfirst(Z.==z),2]*log(K))

Ksimimplied = fill(Ksim[discard+1],T-discard)
Hsimimplied = fill(Hsim[discard+1],T-discard)
for t = 1:T-discard
    if t<T-discard
        Ksimimplied[t+1] = K1(Ksimimplied[t],zsim[t])
    end
    Hsimimplied[t] = H0(Ksimimplied[t],zsim[t])
end

plot(Ksimimplied,xlabel="\$K_t\$",ylabel="Time",label = "Aggregate Capital",lin
estyle = :dot)
plot!(Ksim[discard+1:end],label = "Implied by law of motion")
```

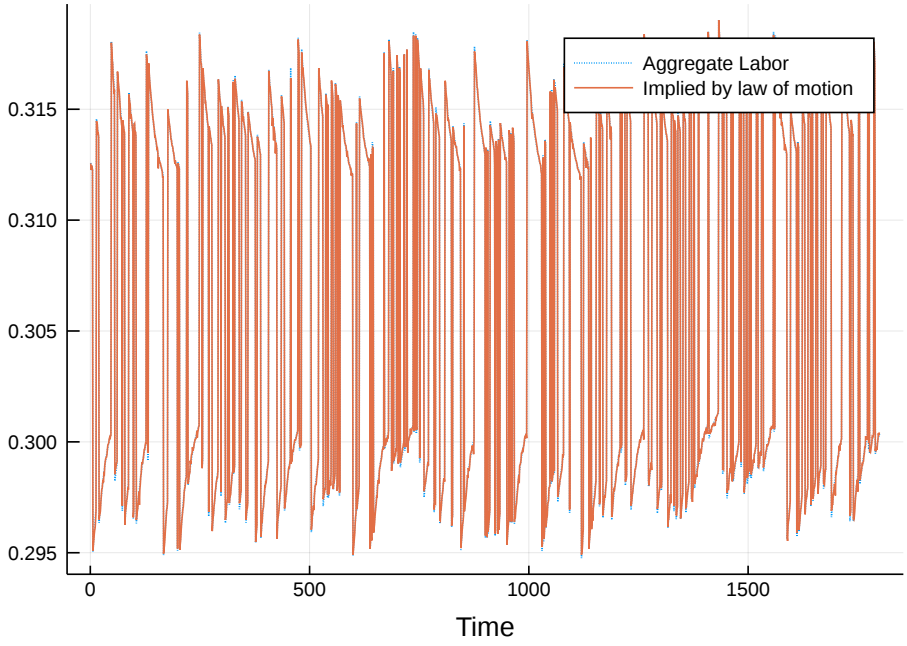
Out[16]:




```
sh: dvipng: command not found
dvipng: PNG conversion failed
```

In [17]:
```
plot(Hsimimplied,ylabel="\$H_t\$",xlabel="Time",label = "Aggregate Labor",lines
tyle = :dot)
plot!(Hsim[discard+1:end],label = "Implied by law of motion")
```

Out[17]:




```
sh: dvipng: command not found
dvipng: PNG conversion failed
```

In [ ]: