HW 3 - João Lazzaro

In this code, I compute a version of the HW 3 without the government. It is a simple Ayiagari with endogenous labor choice. First, Define some Parameters:

In [8]:
```
using  LinearAlgebra, Plots
include("functions.jl") #In the file functions, I defined the functions for VF
I and finding the distribution

#Defining Parameters:
β = 0.98 #Discount rate
μ = 1.5  #Elasticity of intertemporal substitution
η = 1#Utility parameter
τy = 0.0 #Income tax
ρ = 0.6 #autocorrelation
σ = 0.3 #Variance
δ = 0.075 #Depreciation rate
θ = 0.3 #Capital Share
T=0 #tax rate, to be added
Z=1 #productivity level (not used)

b = -0    #Debt limit
amax=0.1    #capital limit

nE = 5 #Number of states for e
nA = 500 #states for assets
```

Out[8]: 500

Initial Guess for interest:

In [9]:
```
r0= (1/β - 1)*rand() #initial guess for r liens in the interval (-δ, 1/β-1)
K = ((r0+δ)/(Z*θ))^(1/(θ-1)) #K for the nitial guess of r
w0=  (1-θ)*K^θ  #Initial wage given r0
```

Out[9]: 1.1822524385888202

Get the grid for E and assets. Note that in this code, I only use grids. No interpolation, therefore it is inefficient and slow, but I fully understand what is going on. I'll make it better once I understand the full algorithm.

In [10]:
```
#Defining grids
pdfE,E = Tauchen(ρ,σ,nE)    #E comes from Tauchen method
A = range(b,stop = amax, length = nA)    #Assets
```

Out[10]: 0.0:0.0002004008016032064:0.1

Get the policy functions and distribution, see the file functions.jl where I defined the ayiagari function:

In [11]:
```
@time λ,r,w, policy_a, policy_c, policy_l = ayiagary(A,E,r0,w0)
```

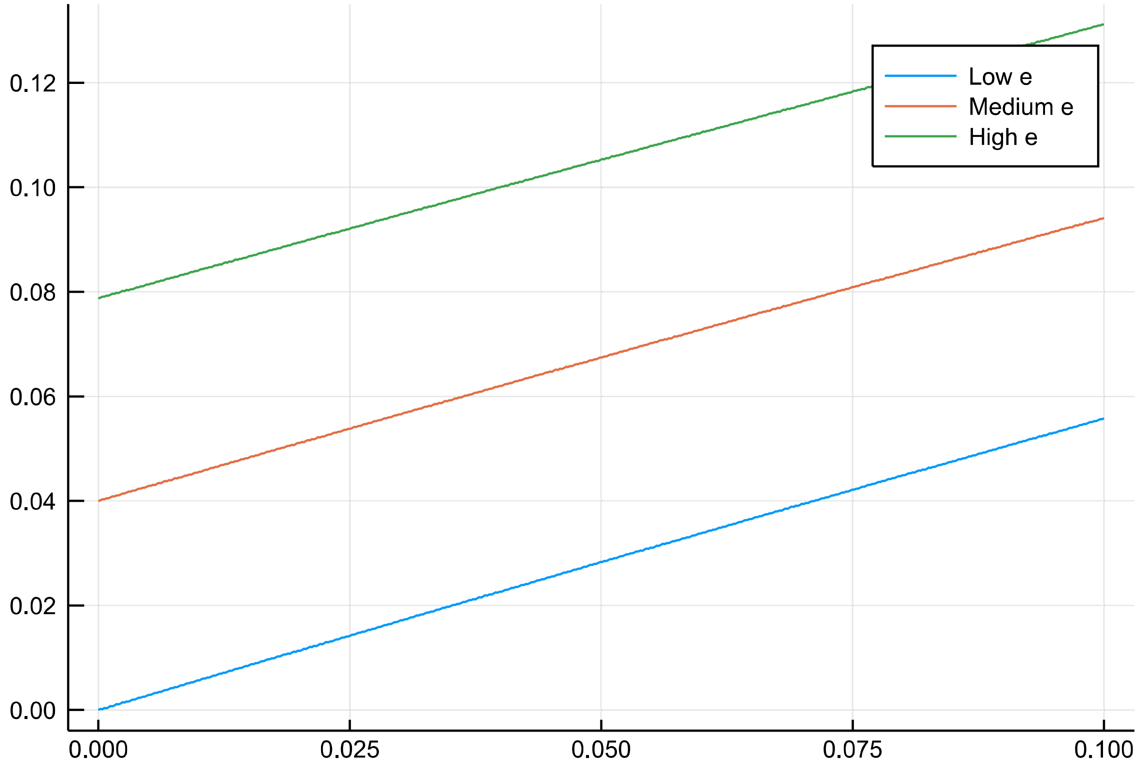
```
Iteration: 1 ,  r distance is: 0.03477533080014417
Iteration: 2 ,  r distance is: 0.017180917731642868
Iteration: 3 ,  r distance is: 0.008419506640375503
Iteration: 4 ,  r distance is: 0.004116731393197834
Iteration: 5 ,  r distance is: 0.0020061780931023634
Iteration: 6 ,  r distance is: 0.0009768144509964063
Iteration: 7 ,  r distance is: 0.00047397783748975l6
Iteration: 8 ,  r distance is: 0.00023149238822364054
Iteration: 9 ,  r distance is: 0.0001116913951787965
Iteration: 10 ,  r distance is: 5.43965823850065e-5
Iteration: 11 ,  r distance is: 2.67634567219302e-5
Iteration: 12 ,  r distance is: 1.309181308896401e-5
Iteration: 13 ,  r distance is: 6.545906544482005e-6
Iteration: 14 ,  r distance is: 3.272953272244472e-6
Iteration: 15 ,  r distance is: 1.6364766361187666e-6
Iteration: 16 ,  r distance is: 6.732729937175552e-7
r converged to -0.0550857093428804
1453.662580 seconds (60.13 G allocations: 1.001 TiB, 4.56% gc time)
```

Out[11]:
```
([6.50672e-16 2.85692e-17 … 0.0 0.0], -0.0550857093428804, 2.238361412098928
6, [1 1 … 133 271; 1 1 … 133 271; … ; 195 194 … 334 480; 196 194 … 334 481],
[-1.99883e-9 -9.99414e-10 … 0.0399438 0.0786852; 0.000189358 0.000189361 … 0.
0401332 0.0788746; … ; 0.0554243 0.0556247 … 0.0939653 0.131103; 0.0554133 0.
0558141 … 0.0941547 0.131092], [1.0 1.0 … 3.32456e-16 3.02815e-16; 1.0 1.0 …
3.32456e-16 2.87259e-16; … ; 1.0 1.0 … 4.1727e-16 3.32456e-16; 1.0 1.0 … 3.32
456e-16 3.32456e-16])
```

The consumption policy function is plotted below:

In [22]:
```
plot(A,[policy_c[:,2] policy_c[:,4] policy_c[:,5]], labels = ["Low e" "Medium
 e" "High e"])
```

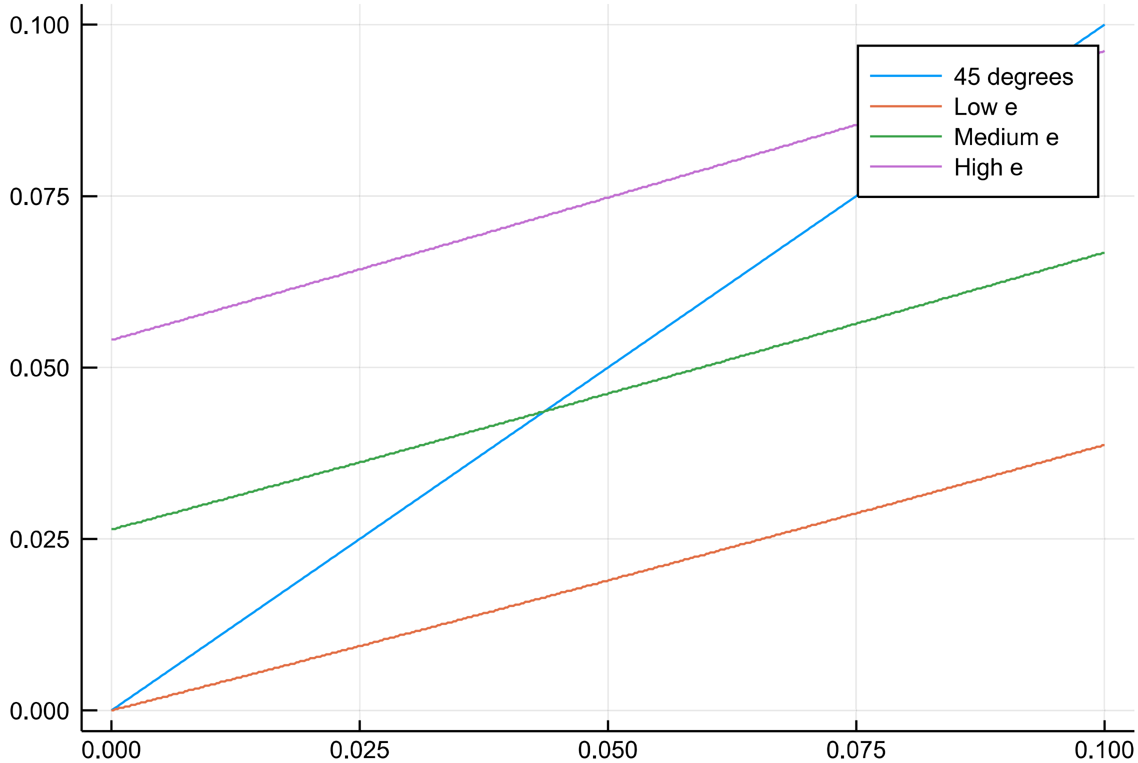
Out[22]:



The assets policy function is plotted below:

In [21]:
```
plot(A,[A,A[policy_a[:,2]],A[policy_a[:,4]],A[policy_a[:,5]]], labels = ["45 d
egrees" "Low e" "Medium e" "High e"])
```

Out[21]:



The assets distribution is below:

In [29]:
```
#reshaping λ so it gets in a format easier to plot
λ1 = ones(nA,nE)
i=0
for a=1:nA
    global i
    for e=1:nE
        i+=1
        λ1[a,e] = λ[i]
    end
end

plot(A, sum(λ1,dims=2) , label = "\\lambda")
```

Out[29]: