HW5 - João Lazzaro

This is just a rough draft to organize my thoughts

In this homework, we were asked to use Reiter's method to solve a model with aggregate uncertainty and hetereogenous agents and elastic labor. I first started this trying to keep the model from the previous homework in orderr to compare both methods. But I could not find a way of incorporating the joint distribution of the aggregate and idiosyncratic shocks. So I decided that it would be a good idea to make a model similar to the Aiyagari of HW3, but without taxes, with persistent idiosyncratic shocks to generate a nice dispersion of the Steady State assets distribution (in a 2 states unemployed/employed model the distribution tend to concentrate in a single point in any reasonable calibration).

Model

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:

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

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

$$logZ_{t+1} = \rho_Z logZ_t + \varepsilon \ \ , \varepsilon \sim N(0, \sigma_Z)$$

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:

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

$$a' \geq 0$$

$$e_{t+1} = 1 - \rho_e + \rho_e e_t + \nu_t \ \ , \nu \sim N(0, \sigma_Z)$$

Market clearing:

$$L = \int ned\lambda(a, e)$$

$$K' = \int ad\lambda(a, e)$$

Reiter's Method

At a broad level, the solution method includes three steps:

1 - Approximate the infinite dimensional equilibrium objects, which in our example involves the policy functions of the household and cross-sectional distribution of individual states, by some finite dimensional object. This allows us to have a finite parametrization of the model.

2 - Compute the stationary equilibrium of the discrete model. This will take into account the idiosyncratic uncertainty but not the uncertainty coming from aggregate shocks.

3 - Linearize the discrete model equations with respect to our finite parametrization around steady-state and compute the rational expectation solution of the linearized system taking into account aggregate shocks.

Here I have done steps 1 and 2 but I still could not do step 3.

The big idea in Reiter's method is to approximate the infinite dimensional variables by a finite approximation such as the assets distribution in the Steady State, similar to one of the ways of solving Ayiagari's model, thus the agents keep track of the whole distribution and not only some moments as in KS. Then, instead of simulating the economy the idea is to do a linear approximation around that.

So assuming that the economy is in the Steady State, the agent intertemporal FOC is:

$$u_{ct} = \beta E[R_{ss} u_{ct+1} | e_t]$$

Let n_a^d be the number of gridpoints for the distribution approximation and n_ϵ the number of idiosyncratic states. The law of motion for the pdf, $n_\varepsilon \times (n_a^d - 1) \times 2$ equations:

$$\lambda_{\varepsilon',t+1}(a_{k+1}) = \sum_{\varepsilon \in E} \pi(\varepsilon, \varepsilon') \frac{a_{k+1} - g_a(a_j;\varepsilon)}{a_{k+1} - a_k} \lambda_{\varepsilon,t}(a_j)$$

If $a_k \leq g_a(a_j;\varepsilon) \leq a_{k+1}$

$$\lambda_{\varepsilon',t+1}(a_k) = \sum_{\varepsilon \in E} \pi(\varepsilon, \varepsilon') \frac{g_a(a_j;\varepsilon) - a_k}{a_{k+1} - a_k} \lambda_{\varepsilon,t}(a_j)$$

If $a_{k-1} \leq g_a(a_j;\varepsilon) \leq a_k$ and 0 otherwise.

The algorithm to find the steaty state is: Guess R and w, get the aggregate variables implied by wages setting Z to its SS. Solve the agent problem (endogenous grid as in KS) find the invariant assets distribution implied by policy rules and aggregate the the individual assets using the formulas above calculate the new aggregate assets and wages and check convergence:

The code below implements these steps.

In [3]:
```
#Author: João Lazzaro
#This codes solves the agent problem for the Krussell-Smith case
using Interpolations, ProgressMeter,LinearAlgebra,SparseArrays
using IterativeSolvers, ForwardDiff
#using Distributed, SharedArrays
include("CRRA_utility.jl")

#Wages functions
R(z::Real,k::Real,l::Real;α = α::Real,δ=δ::Real,A=A::Real)=  α*A*z*k^(α-1)*l^(1
-α) + 1-δ
w(z::Real,k::Real,l::Real;α = α::Real,A=A::Real)=  (1-α)*A*z*k^α*l^(-α)

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


function weight(a1::Real,a::Real,e::Real,pol::Real,grid::AbstractArray)

    a1i::Real = findfirstn(grid,a1)
    ai::Real = findfirstn(grid,a)

    if pol <= grid[1] && a1i==1
        return 1.0

    elseif a1i<length(grid) && a1<= pol <= grid[a1i+1]
        return (grid[a1i+1] - pol)/(grid[a1i+1] - a1)
    elseif a1i>1 && grid[a1i-1]<=pol<= a1
        return (pol-grid[a1i-1])/(a1-grid[a1i-1])
    elseif pol >= grid[end] && a1i== length(grid)
        return 1.0
    else
        return 0.0
    end
end

function λ1(a1::Real,e1::Real,λ::AbstractArray,R::Real,w::Real,policy_a::Functi
on;E=E::AbstractArray,grid = Ad::AbstractArray,pdfE=pdfE::AbstractArray,E_size=
E_size::Real)
    lambda::Real = 0.0
    e1i::Real = findfirstn(E,e1)
    for ei =1:E_size, ai =1:Ad_size
       lambda += pdfE[ei,e1i] * weight(a1,grid[ai],E[ei],policy_a(grid[ai],E[ei
],R=R,w=w),grid) * λ[ai,ei]
    end
    return lambda
end



function weightmatrix(policy_a::Function,R::Real,w::Real;Ad_size=Ad_size::Real,
E_size = E_size::Real,
    E=E::AbstractArray,grid = Ad::AbstractArray,pdfE=pdfE::AbstractArray)
    weights::AbstractArray = ones(Ad_size,E_size,Ad_size,E_size)
  for ai=1:Ad_size
        for ei=1:E_size,e1i=1:E_size,a1i = 1:Ad_size
            weights[a1i,e1i,ai,ei] = pdfE[ei,e1i] * weight(grid[a1i],grid[ai],E
[ei],policy_a(grid[ai],E[ei],R=R,w=w),grid)
        end
    end
    return reshape(weights,E_size*Ad_size,E_size*Ad_size)
```

```
Out[3]: findSS (generic function with 1 method)
```

Below I define the parameters. The code defines a finite economy and them calculates the steady state.

In [4]:
```
include("functions.jl")
include("Reiter.jl")
include("AgentProblem.jl")

#Defining parameters they were taken frmom KS
const α = 0.36 #capital share in output
const β = 0.99 #Impatience
const δ = 0.025 #Depreciation rate
const η = 1.0/2.9 #labor /consumption elasticity
const μ = 1.0 #Intertemporal elasticity
const A= 1.0#(1/β-1+δ)/α
const lbar = 1.0#/0.9 #time endowment

const Ap_size = 30 #number of assets gridpoints for policy rules
const Ad_size = 150 #number of assets gridpoints for distribution
amin = 0.0
amax = 50.0

factor =7
const Ap =(range(0.0, stop=Ap_size-1, length=Ap_size)/(Ap_size-1)).^factor * am
ax #Capital grid for todayconst A1 = A#(range(0, stop=nA-1, length=nA)/(nA-1)).
^factor * amax #Capital grid for today#range(0., stop=amax, length=nA).^1 #Capi
tal grid for tomorrow
const Ad = range(0.0, stop=amax, length=Ad_size).^1.0 #Capital grid for todayco
nst A1 = A#(range(0, stop=nA-1, length=nA)/(nA-1)).^factor * amax #Capital grid
for today#range(0., stop=amax, length=nA).^1 #Capital grid for tomorrow

update = 0.2


#productivity shocks
ρ = 0.8
σ = 0.01
Z1(z;ρ=ρ,σ =σ) = exp(ρ * log(z) + σ * randn())
zss=1.0

E_size = 5 #Number of states for e
σe = 0.3 #Variance
ρe = 0.6
pdfE,E = Tauchen(ρe,σe,E_size,1-ρe)    #E comes from Tauchen method

#Initial guess for wages
w0 = 2.3739766754889238
R0=1.010012230693079

#First we fin the State States

policygridss,Kss,Lss,Rss,wss,λss = findSS(Ap,Ad,E,E_size,Ad_size,pdfE;tol = 1e-
6,update = .1,R0 = R0,w0=w0)
```

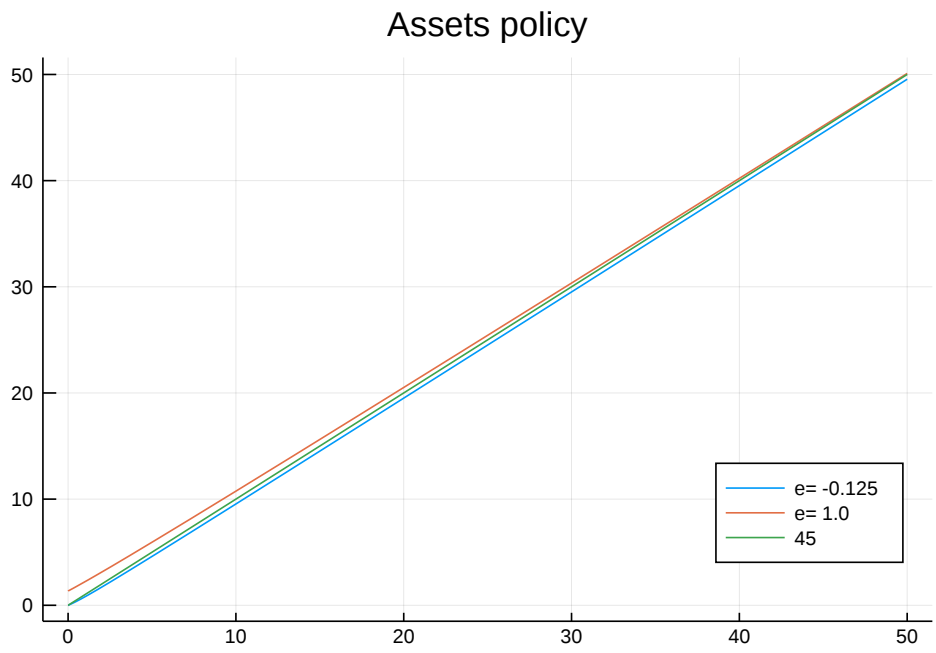
```
Looking for the SS, iterations: 1        Time: 0:00:08
  Distance:    2.0409474042849367
  AggregateK:  28.966651378988573
  AggregateL:  0.13553638176171462
  R:           1.007673000222015
Looking for the SS, iterations: 2        Time: 0:00:11
  Distance:    0.05826402040986034
  AggregateK:  9.033620294534948
  AggregateL:  0.17702172687488607
  R:           1.0073116488443605
Looking for the SS, iterations: 3        Time: 0:00:14
  Distance:    0.024426556776237174
  AggregateK:  8.389243068421699
  AggregateL:  0.17847734687447508
  R:           1.0071433860055443
Looking for the SS, iterations: 4        Time: 0:00:16
  Distance:    0.055172085999558806
  AggregateK:  8.117420333928312
  AggregateL:  0.17907026776643173
  R:           1.0070638484831216
Looking for the SS, iterations: 5        Time: 0:00:17
  Distance:    0.0660142215326287
  AggregateK:  7.9840185547191345
  AggregateL:  0.17933474990372467
  R:           1.007028680669248
Looking for the SS, iterations: 6        Time: 0:00:19
  Distance:    0.06796489135113637
  AggregateK:  7.913724921968115
  AggregateL:  0.17944913675100105
  R:           1.0070163304001538
Looking for the SS, iterations: 7        Time: 0:00:21
  Distance:    0.06556940885208462
  AggregateK:  7.876782904053631
  AggregateL:  0.17948729278509176
  R:           1.0070152191830506
Looking for the SS, iterations: 8        Time: 0:00:23
  Distance:    0.06137724485770857
  AggregateK:  7.85617554689176
  AggregateL:  0.1794895042496572
  R:           1.0070196147723498
Looking for the SS, iterations: 9        Time: 0:00:24
  Distance:    0.05650027646254019
  AggregateK:  7.844476488264016
  AggregateL:  0.17947406158644616
  R:           1.0070264532666866
Looking for the SS, iterations: 10       Time: 0:00:26
  Distance:    0.05152197472899367
  AggregateK:  7.8375913985142605
  AggregateL:  0.17945079797237073
  R:           1.0070341453309843
Looking for the SS, iterations: 11       Time: 0:00:28
  Distance:    0.0467271184686715
  AggregateK:  7.8333361929187495
  AggregateL:  0.17942481023068546
  R:           1.0070418865261772
Looking for the SS, iterations: 12       Time: 0:00:29
  Distance:    0.04224357893988495
  AggregateK:  7.8305442531528735
  AggregateL:  0.17939868182639743
  R:           1.0070492869521495
Looking for the SS, iterations: 13       Time: 0:00:31
  Distance:    0.038118089254451704
  AggregateK:  7.828589684150629
```

```
Aggregates converged with a distance of 2.2138289978990144e-7
```

Out[4]:
```
([0.0 0.228381 … 0.427203 0.47689; 1.0353e-9 0.228381 … 0.427203 0.47689; … ; 0
.863072 0.868526 … 0.882483 0.889596; 0.94855 0.952702 … 0.963724 0.96942], 7.8
169510309581005, 0.17912901784948182, 1.0071208825085896, 2.491917528290475, [0
.000504866 0.000199946 … 4.87719e-6 1.74919e-7; 0.000924772 0.000366244 … 8.933
65e-6 3.20403e-7; … ; 5.98724e-14 8.31424e-14 … 3.39015e-14 8.3996e-15; 3.2701e
-14 7.23272e-14 … 4.50633e-14 1.21595e-14])
```

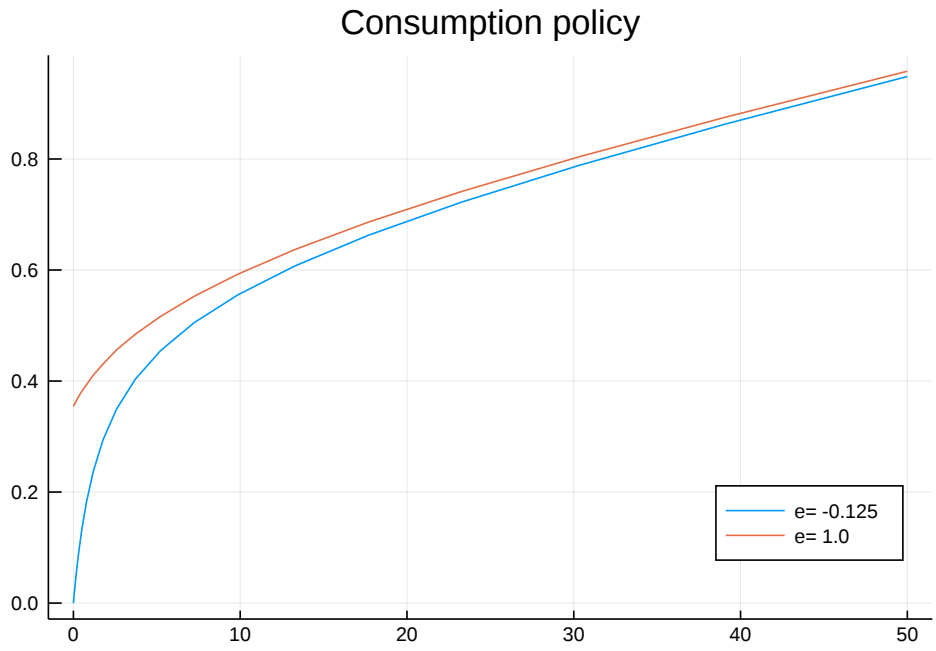
In [12]:
```
using Plots
itpc = LinearInterpolation((Ap,E),policygridss, extrapolation_bc=Line())
    policy_c(a,e) = itpc(a,e)
    policy_n(a,e;w=w0) = nstar(policy_c(a,e),e,w)
    policy_a(a,e;w=w0,R=R0) = a1star(policy_c(a,e),a,e,w,R)

plot(Ap,policy_a.(Ap,E[1]),legend=:bottomright,label = "e= $(E[1])",title="Asse
ts policy")
plot!(Ap,policy_a.(Ap,E[3]),label = "e= $(E[3])")
plot!(Ap,Ap,label = "45")
```

Out[12]:



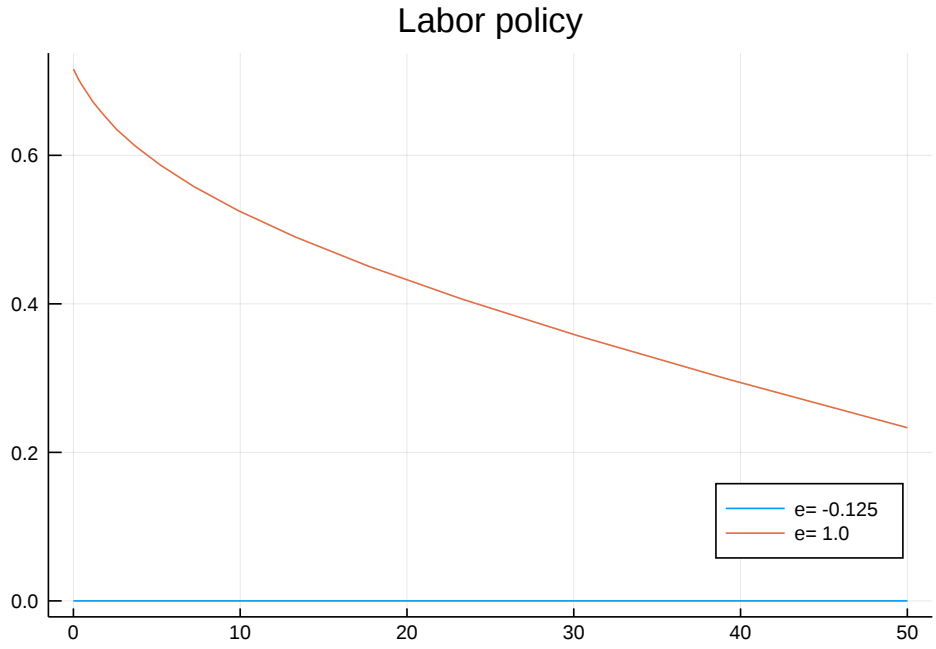
In [7]:
```
plot(Ap,policy_c.(Ap,E[1]),legend=:bottomright,label = "e= $(E[1])",title="Cons
umption policy")
plot!(Ap,policy_c.(Ap,E[3]),label = "e= $(E[3])")
```

Out[7]:



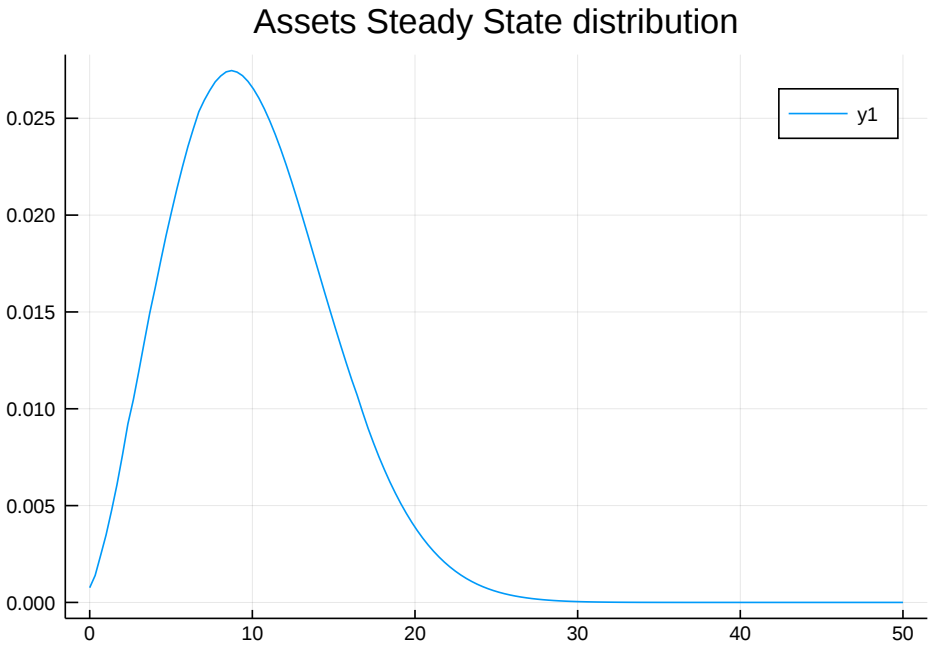
In [8]:
```
plot(Ap,policy_n.(Ap,E[1]),legend=:bottomright,label = "e= $(E[1])",title="Labo
r policy")
plot!(Ap,policy_n.(Ap,E[3]),label = "e= $(E[3])")
```

Out[8]:



In [11]:
```
plot(Ad,sum(λss,dims=2),title="Assets Steady State distribution")
```

Out[11]:



Next steps

Implement the perturbartion part (the procedure should be similiar to HW1 and 2 and also Ellen's stuff). For that I have to be able to construct the system in a way that Julia's automatic differentiation package work. I have been struggling with this for a long time. I tried to approximate derivatives taking differences but is is not working in this case. Since I have persistent idiosyncratic shocks my matrices hardly conform.