Online Appendix for: A Probabilistic Theory of Economic Equilibria (Section 4.2)

Juan Jacobo
Universidad Externado de Colombia
juan.jacobo@uexternado.edu.co

Abstract This document describes the code solving the investment model in A Probabilistic Theory of Economic Equilibria.

Table of contents

1 Investment Model with Fixed Costs

Similar to the RKC model, I now present the Julia code solving the probabilistic investment model with fixed costs. I consider the problem of a capitalist looking to maximize the expected present value of operating profits $\pi(k, z) = e^z k^\theta$ less total investment costs $\nu \cdot c(k', k)$. As in the main text, $\nu = 0$ represents the event of inaction, $c(k', k)$ is the augmented adjustment cost function, k is the capital stock, $k' = (1 - \delta)k + I$ is next period's capital, and $z' = \rho_z z + \epsilon_z$ is a random shock with $\epsilon_z \sim N(0, \sigma_z^2)$. For practical purposes, let $c(I, k) = \gamma_0 k (I/k)^2 + \gamma_1 k + p_b I$ if gross investment $I > 0$, and $c(I, k) = \gamma_0 k (I/k)^2 + \gamma_1 k + p_s I$ if $I < 0$, with $p_b \geq p_s$. As usual, $\gamma_0 \geq 0$ captures the convex side of the adjustment costs of capital, $\gamma_1 \geq 0$ measures the fixed costs of capital, and p_s and p_b represent the selling and buying price per unit of uninstalled capital.

The value function associated to this model is given by:

$$
\begin{aligned}
v(s) &= \max_{p(\nu=0 \mid s) \in \mathcal{P}_\nu(s)} \Bigg\{ p(\nu = 0 \mid s) v_I(s) + (1 - p(\nu = 0 \mid s)) \times \\
&\quad \max_{p(k' \mid \nu=1, s) \in \mathcal{P}_{k'}(s)} \left[\int_{\Gamma(s)} p(k' \mid \nu = 1, s) B_A(k', s) \mathrm{d}k' \right] \Bigg\} \\
&= \max_{p(\nu=0 \mid s) \in \mathcal{P}_\nu(s)} \Big\{ p(\nu = 0 \mid s) v_I(s) + (1 - p(\nu = 0 \mid s)) v_A(k', s) \Big\}
\end{aligned}
$$

Where $v_I(s) = \pi(s) + \beta \int_Z v\left((1 - \delta)k, z' \right) p(z' \mid z) \mathrm{d}z'$ is the value of inaction, \$ v_{A}(s)\$ is the value of active investment, $B_A(k', s) = \pi(s) - c(k', k) + \beta \int_Z v(s') p(z' \mid z) \mathrm{d}z'$ is the lifetime value of active investment for each $k' \in \Gamma(s)$, and $\mathcal{P}_{k'}(s)$ and $\mathcal{P}_\nu(s)$ represent the admissible sets associated to the optimization problems of $p(k' \mid \nu = 1, s)$ and $p(\nu = 0 \mid s)$, respectively.

Computation

The following code specifies the parameters and packages to run the investment model with fixed costs.

```
using QuantEcon,LinearAlgebra, IterTools, Plots, LaTeXStrings,
NLSolversBase,ADNLPModels, Distributions, DSP, KernelDensity,StatsBase
plot_font = "Computer Modern"
default(fontfamily=plot_font)
include("s_approx.jl")
#### MODEL ##########
"Model"
function create_investment_model(; β=0.94, γ=0.04, δ = 0.06, p=0.99, P=1, θ =
0.56,
  K_min=1, K_max=120, K_size=800,
  ρ=0.9, ν=0.01, a_size=8)
K_grid = LinRange(K_min, K_max, K_size)
mc = tauchen(a_size, ρ, ν)
a_grid, Q = exp.(mc.state_values), mc.p
return (; β, γ, δ, p, P, θ, K_grid, a_grid, Q)
end
"Cost function"
function cost_fun(Kp,K,p,P,δ,γ)
  I=Kp - (1-δ)*K
  if    I>0
  return  (γ/2) *K*(I/K)^2 +P*I + 0.02*K
  elseif I<0
    return (γ/2) *K*(I/K)^2 +p*I +0.02*K
  end
end
```

```
Main.Notebook.cost_fun
```

Figure 1: Model Packages and Parameters

Unlike the RKC model, the investment model with fixed costs requires the definition several value functions. Particularly, we need to define the value of inaction $v_I(s)$ and the value of active investment $v_A(s)$. Figure 2 presents the Julia code that estimates the value of inaction using a discrete approximation of the state space. This is important because the state of inaction $k = (1 - \delta)k'$ may not be equal to any $k[i]$ in the grid defined in Figure 1. To solve this problem, I introduce a discrete approximation in Figure 2.

```
"Value function of inaction"
function vI(i,j,v,model)
  (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
  K, a = K_grid[i], a_grid[j]
  Kp=(1-δ)*K   # Investment is zero
  ti=Int.(argmin(abs.(K_grid.-Kp))) # Closest discrete approximation
  return  a*K^θ +  β * dot(v[ti, :], Q[j, :])
```

```
end
function full_VI(v,model)
  (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
  K_idx, A_idx = (eachindex(g) for g in (model.K_grid, model.a_grid))
  v_inactive = zeros(length(K_grid), length(a_grid))
  for (i, j) in product(K_idx, A_idx)
    v_inactive[i,j]=vI(i,j,v,model)
  end
  return v_inactive
end
```

```
full_VI (generic function with 1 method)
```

Figure 2: Code of Fixed Costs and value of Inaction

?@fig-activeinv displays the code to generate the value of active investment. **BA(i,j,h,v,model)** describes the payoff of active investment. **quantalK(A,i,j,v,model)** is the dynamic logit function $\hat{p}(k' \mid k, z, \nu = 1)$ in the main text. Finally, **vA(A,v,model)** is the value of active investment.

```
 #| label: fig-activeinv
#| fig-cap: Value of Active Investment
"Life-time reward of active investment"
function BA(i,j,h,v,model)
 (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
    K, a, K′ = K_grid[i], a_grid[j], K_grid[h]
     @views value = K′ > 0 ? a*K^θ.-cost_fun(K′,K,p,P,δ,γ).+ β * dot(v[h, :],
Q[j, :]) : -Inf
    return value
end
"Quantal response of next period's capital"
function quantalK(λA,i,j,v,model)
    (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
    K, a = K_grid[i], a_grid[j]
    N = length(K_grid)
    P=zeros(N)
    for h=1:N
  bb = BA(i,j,h,v,model)
        P[h]=exp(bb/λA)
    end
    return P./sum(P)
end
"Value function of active investment"
function vA(λA,v,model)
    (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
    K_idx, A_idx = (eachindex(g) for g in (model.K_grid, model.a_grid))
    v_active = zeros(length(K_grid), length(a_grid))
    N=length(K_grid)
```

```
    for (i, j) in product(K_idx, A_idx)
        P = quantalK(λA,i,j,v,model)
     v_active[i, j] = sum(P[h]*BA(i,j,h,v,model) for h=1:N)
    end
        return v_active
end
```

```
Main.Notebook.vA
```

The following code generates the deterministic solution used in the main text to constrast the results of the probabilistic model.

```
"Deterministic value function of active investment"
function vsA(v,model)
      (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
   K_idx, A_idx = (eachindex(g) for g in (model.K_grid, model.a_grid))
    v_new = similar(v)
    for (i, j) in product(K_idx, A_idx)
        v_new[i, j] = maximum(BA(i, j, h, v, model) for h ∈ K_idx)
    end
    return v_new
end
"The Bellman operator."
function T(v, model)
    (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
    K_idx, A_idx = (eachindex(g) for g in (model.K_grid, model.a_grid))
    v_new = similar(v)
    N=length(K_grid)
    V_A=vsA(v,model)
    V_inu = full_VI(v,model)
    for (i, j) in product(K_idx, A_idx)
        V_inaction = V_inu[i,j]
        V_action = V_A[i,j]
        v_new[i,j] =  maximum([V_inaction;V_action])
    end
return v_new
end
function get_greedy1(v,model) # deterministic sol capital
    (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
    K_idx, A_idx = (eachindex(g) for g in (model.K_grid, model.a_grid))
    σ = Matrix{Int32}(undef, length(K_idx), length(A_idx))
    for (i, j) in product(K_idx, A_idx)
                    _, σ[i, j] = findmax(BA(i, j, h, v, model) for h ∈ K_idx)
    end
    return σ
end
function get_greedy2(v, model) # deterministic sol inaction
```

```
    (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
    K_idx, A_idx = (eachindex(g) for g in (model.K_grid, model.a_grid))
    σ = zeros(length(K_idx), length(A_idx))
    V_A=vsA(v,model)
    V_inu = full_VI(v,model)
    for (i, j) in product(K_idx, A_idx)
              V_inaction = V_inu[i,j]
              V_action = V_A[i,j]
                                if V_inaction>=V_action
                        σ[i, j] = 1
                                else
                                    σ[i, j]=0
                                end
    end
    return σ
end
"Find fixed point with a contraction algorithm"
function solve_investment_model(v_init, model)
    (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
    v_star = successive_approx(v -> T(v, model), v_init)
    return v_star
end
```

```
Main.Notebook.solve_investment_model
```

Given the definition of the value functions of active and inactive investment, we can now define the logit function of inaction in $\hat{p}(\nu = 0 \mid s)$ and the value function $v(s)$ in the main text. **?@fig-valuefunI** presents the Julia code with the definition of these two function and the corresponding solution using the contraction mapping algorithm in Sargent & Stachurski (2023).

```
"Quantal response of inaction"
function QR_inaction(λI,vI,vA)
  return 1/(1+exp((vA-vI)/λI))
end
"Bellman contraction mapping"
function T_QR(λI,λA,v,model)
  (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
K_idx, A_idx = (eachindex(g) for g in (model.K_grid, model.a_grid))
   v_new = similar(v)
   N=length(K_grid)
    V_A=vA(λA,v,model)
    V_inu = full_VI(v,model)
    for (i, j) in product(K_idx, A_idx)
        V_inaction = V_inu[i,j]
        V_action = V_A[i,j]
        Pv = QR_inaction(λI,V_inaction,V_action)
        v_new[i,j] =  Pv*V_inaction + (1-Pv)*V_action
    end
return v_new
end
"Find fixed point with a contraction algorithm for probabilistic model"
function solve_investment_modelQR(λI,λA,v_init, model)
  (; β, γ, δ, p, P, θ, K_grid, a_grid, Q) = model
  v_star = successive_approx(v -> T_QR(λI,λA,v, model), v_init)
  return v_star
end
λI=1 # Lagrange multiplier of inaction
λA=0.27  # Lagrange multiplier of active investment
model = create_investment_model()
v_init =zeros(length(model.K_grid), length(model.a_grid))
v_E = solve_investment_modelQR(λI,λA,v_init,model)  # value function entropy
v_star = solve_investment_model(v_init, model)   # value function deterministic
k0_indx=get_greedy1(v_star,model);    # optimal solution deterministic
inaction_indx=get_greedy2(v_star,model);    # optimal solution deterministic
```

Figure 3

```
"Estimate probability of action and inaction"
vAE=vA(λA,v_E,model)
VIE=zeros(length(model.K_grid),length(model.a_grid))
Pv0u=zeros(length(model.K_grid),length(model.a_grid))
Pks=zeros(length(model.K_grid),length(model.a_grid),length(model.K_grid))
for i=1:length(model.K_grid)
   for j=1:length(model.a_grid)
   VIE[i,j]=vI(i,j,v_E,model)         # value of inaction
   Pv0u[i,j] = QR_inaction(λI,VIE[i,j],vAE[i,j]) #p(ν=0|k,z)
   Pks[i,j,:]=quantalK(λA,i,j,v_E,model)  # p(k'|k,z)
end
```

```
end
"Smooth probability of inaction" #
Pv0=similar(Pv0u)
for j=1:length(model.a_grid)
   Pv0[:,j]=smooth(Pv0u[:,j],51)
end
Pv1=-Pv0.+1
"Smooth deterministic choice of inaction"
function smooth_det(inaction_result, shock)
index1 = findfirst(x -> x > 0, inaction_result[:,shock])
index2 = findfirst(x -> x < 1, inaction_result[index1+35:end,shock])
smooth_inaction=zeros(size(inaction_result,1))
smooth_inaction[index1:index1+35+index2-1].=1
return smooth_inaction
end
S_σ21=smooth_det(inaction_indx, 1)
S_σ22=smooth_det(inaction_indx, 8)
#This is the equivalent of Figure 4
plot(model.K_grid,Pv0[:,1],  color=:green,  linestyle=:dash,  xlabel="Capital",
ylabel="Probability inaction/action", label="Low Productivity z")
plot!(model.K_grid,Pv1[:,1], color=:green, linestyle=:dash, label=:false)
plot!(model.K_grid,Pv0[:,end], color=:black, label="High Productivity z")
plot!(model.K_grid,Pv1[:,end],  color=:black, label=:false)
hline!(0.5*ones(1), color=:lightgray, linestyle=:dot, label=:false)
plot!(model.K_grid,S_σ21, color=:green, linestyle=:dot, w=2,label=:false)
plot!(model.K_grid,S_σ22, color=:black, linestyle=:dot, w=2,label=:false)
plot!(legend=:outertop,             legendcolumns=2,bottom_margin            =
3Plots.mm,framestyle=:box, grid=:false)
plot!(size=(500,350))
```

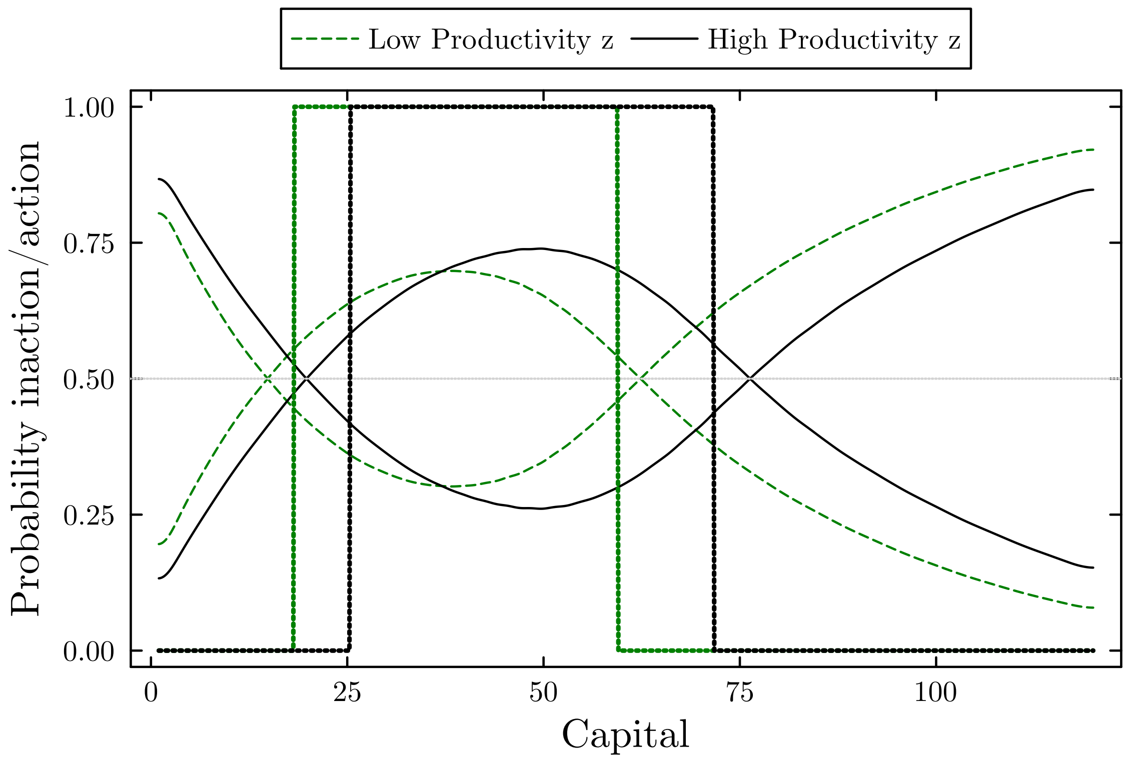


Figure 4: Probability of Action and Inaction

Using the numerical estimates of the different value functions, we can now explore the main implications of the investment model with fixed costs. To start, consider Figure 4, which is identical to Figure 4 in the main text, expect for the fact that now I also introduce the effect of technology changes on the probability of inaction. The contrast between the green and black lines reveal that a positive technology shock shifts the region of inaction to the right. This is a reasonable result since an improved technology raises the profitability of capital, which lowers the probability of inaction at low levels of the capital stock.

```
"Marginal probability density of next period's capital"
Pk_marginal=zeros(length(model.K_grid),length(model.a_grid),length(model.K_grid))
function Drac_delta(i)
  K = model.K_grid[i]
  Kp=(1-model.δ)*K # if inaction
  ti=Int.(argmin(abs.(model.K_grid.-Kp)))
  Dirac=zeros(length(model.K_grid))
  Dirac[ti]=1
  return Dirac
end
# p(k'|k,z)
for i=1:length(model.K_grid)
  for j=1:length(model.a_grid)
    for h=1:length(model.K_grid)
    Pk_marginal[i,j,h] = Pks[i,j,h]*(1-Pv0u[i,j]) + Pv0u[i,j]*Drac_delta(i)[h]
    end
  end
end


#Create contraction mapping algorithm"
function stationary_dist(Ps,initial,iterations, shock)
  N =size(Ps,1)
  Tp=zeros(N,iterations)
  Tp[:,1] = smooth(Ps[initial,shock,:],31)
  for h=2:iterations
    Tp[:,h] = smooth(sum(Ps[i,shock,:].*Tp[i,h-1] for i=1:N),31)
  end
  return Tp
end


Tps1=stationary_dist(Pk_marginal,150,100,4)
Tps2=stationary_dist(Pk_marginal,550,100,4);
```

Figure 5

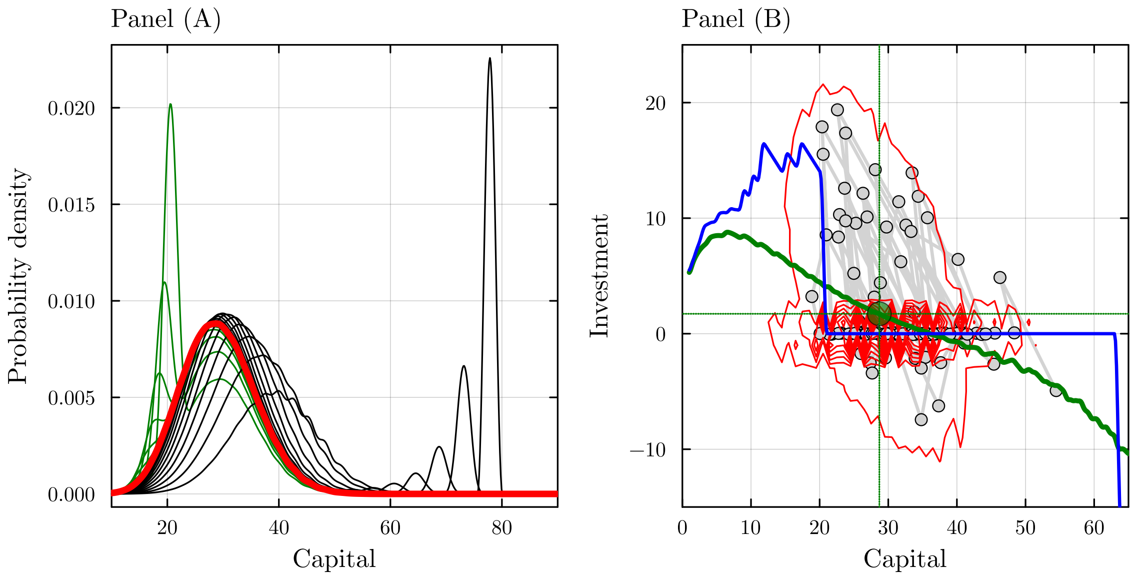
The previous code applies the Chapman-Kolmogorov equation to illustrate how the system converges to a stationary p.d.f. This replicates Figure 5 in the main text.

To finish the presentation of the code, I now simulate data of $k' \sim \hat{p}(k' \mid s)$ to show the stationary joint distribution of investment and capital depicted in Figure 6 in the main text.

```
#Deterministic decisions"
σ1=get_greedy2(v_star,model)
σk=get_greedy1(v_star,model)
S_σ25=smooth_det(inaction_indx, 4) # Deterministic solution of inaction
"Investment levels"
```

```
iks=zeros(length(model.K_grid),length(model.a_grid))
iks_QR=zeros(length(model.K_grid),length(model.a_grid))
iks_QR2=zeros(length(model.K_grid),length(model.a_grid))
for i=1:length(model.K_grid)
    for j=1:length(model.a_grid)
                    iks_QR[i,j]=   ((Pk_marginal[i,j,:]'*model.K_grid)   -   (1-
model.δ)*model.K_grid[i])
        iks[i,j] = ((-S_σ25[i].+1)*model.K_grid[σk[i,j]] - model.K_grid[i]*(1-
model.δ))*(-S_σ25[i].+1)
  end
end


# Deterministic steady state
k_det_NFC = ((((1+2*model.γ*model.δ)/model.β)-(1-model.δ)*(1+2*model.γ*model.δ)
-(model.γ)*model.δ^2)/model.θ)^(1/(model.θ-1))

#Law of large numbers"
    # Simulate capital (indices rather than grid values)
    m=150000
    mc = MarkovChain(model.Q)
        # Compute corresponding wealth time series
function k_dynamics(m,mc,shock,Ps)
    z_idx_series = simulate(mc, m) # stochastic shocks
    k_idx_series = similar(z_idx_series)
    k_idx_series2 = similar(z_idx_series)
    k_idx_series[1] = 360  # initial condition
    k_idx_series2[1] = 360  # initial condition
    k_qr = zeros(m)
    k_qr2 = zeros(m)
    k_qr[1] = model.K_grid[k_idx_series2[1]]
    for t in 1:(m-1)
                    i, j = k_idx_series[t], z_idx_series[t]
                    #k_idx_series[t+1] = σ_star[i, j]
                                                        k_idx_series2[t+1]   =
wsample(1:length(model.K_grid),Ps[k_idx_series2[t],shock,:], 1)[1]
                    k_qr[t+1] =         model.K_grid[k_idx_series2[t+1]]
    end
    return k_qr,k_idx_series2
end


k_qr_low=k_dynamics(m,mc,4,Pk_marginal)[1]
k_qr_high=k_dynamics(m,mc,8,Pk_marginal)[1]
I_qr_low = k_qr_low[2:end].-(1-model.δ)*k_qr_low[1:end-1]
k_index=k_dynamics(m,mc,4,Pk_marginal)[2]
k_index_high=k_dynamics(m,mc,8,Pk_marginal)[2]
```

```
h_ki = fit(Histogram, (k_qr_low[1:end-1], I_qr_low), closed = :left, nbins =
(50, 50))

#Plot associated to Figure 6"
gap = maximum(h_ki.weights)/15
levels_ik = 50:gap:(100*gap)
pp22=plot(k_qr_low[1:130],I_qr_low[1:130], color=:lightgray, w=2, title="Joint
probability density", ylabel="Investment", xlabel="Capital", label=:false)
scatter!(k_qr_low[1:130],I_qr_low[1:130], color=:lightgray, label=:false)
contour!(midpoints(h_ki.edges[1]),
                                                    midpoints(h_ki.edges[2]),
                                                    h_ki.weights';
                                                           levels = levels_ik,
label=:false,  color=:red,colorbar_entry=false)
                                                                         plot!
(model.K_grid,smooth(iks_QR[:,4],9),w=3, color=:green, label=false)
                                        plot!(model.K_grid,smooth(iks[:,4],9),
xlims=(0,65), w=2, ylims=(-15,20), color=:blue, label=false)
                                             vline!(model.K_grid[187]*ones(1),
label=false, linestyle=:dot, color=:green)
                                                                        hline!
(model.δ*model.K_grid[187]*ones(1), label=false, linestyle=:dot, color=:green,
ylims=(-15,25),
                                              title="Panel (B)", titlelocation
= :left, titlefont=10, xguidefontsize=10, yguidefontsize=10)
                                                   scatter!([model.K_grid[187]],
[model.δ*model.K_grid[187]],color="green",          label=:false,markersize=8,
alpha=0.6)

pp11=plot(model.K_grid,Tps1[:,2:1:19],       color=:green,       label=:false,
ylabel="Probability density", xlabel="Capital")
plot!(model.K_grid,Tps2[:,1:1:19], color=:black, label=:false, xlims=(10,90))
plot!(model.K_grid,Tps2[:,end],      color=:green,      w=2,      linestyle=:dot,
label=:false)
plot!(model.K_grid,Tps1[:,end], color=:red, label=:false,w=4,
title="Panel  (A)",  titlelocation  =  :left,  titlefont=10,  xguidefontsize=10,
yguidefontsize=10)

plot(pp11,pp22,   layout=(1,2),   framestyle=:box,left_margin   =   3Plots.mm,
bottom_margin = 3Plots.mm)
plot!(size=(700,350))
```



Bibliography

Sargent, T., & Stachurski, J. (2023). *Dynamic Programming. Finite States* (Vol. 1). Unpublished.