Online Appendix for: A Probabilistic Theory of Economic Equilibria (Section 4.1)

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

Abstract This document present the code solving the models in A Probabilistic Theory of Economic Equilibria.

Table of contents

1 Ramsey-Koopmans-Cass (RKC) Model

Consider a growth model where capital owners are faced with the problem of maximizing a lifetime payoff function $\bar{U} = \mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t U(c_t)\right]$. As usual, c_t represents consumption, and $U(c) = (1-\sigma)^{-1}\left[c^{1-\sigma} - 1\right]$ is a concave, strictly increasing, and twice differentiable instantaneous utility function. The exogenous state is governed by a stochastic process $z' = \rho_z z + \epsilon$, with $\epsilon \sim N(0, \sigma_z^2)$, and the capital stock evolves in time according to $k' = (1-\delta)k + f(k,z) - c$, where $y = f(k,z) = e^z k^\alpha$ is an aggregate production function, and $\delta \in [0,1]$ is the depreciation rate.

1.1 Computation

The following code describes the model parameters and the packages used for the solutions.

```
using Plots, Distributions, LaTeXStrings, Optim, QuantEcon, LinearAlgebra,
IterTools,CSV, DataFrames, KernelDensity, StatsBase
plot_font = "Computer Modern"
default(fontfamily=plot_font)
include("s_approx.jl") # Obtained from Quantecon
"Model Parameters"
n=400 # Discrete points
function create_savings_model(; β=0.9, δ=0.06, σ=0.9, α=0.4,
    k_min=0.05, k_max=10.0, k_size=n, ρ=0.9, ν=0.01, z_size=20)
 k_grid = LinRange(k_min, k_max, k_size)
 mc = tauchen(z_size, ρ, ν)
 z_grid, Q = exp.(mc.state_values), mc.p
```

```
return (; β, σ, δ, α, k_grid, z_grid, Q)
end
```

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

As noted in the main text, the lifetime payoff function for this optimization problem is given by $B(k', s) = U(k', s) + \beta \int_Z v(s')p(z' \mid z)\mathrm{d}z'$, with $s = (k, z)$, $c \geq 0$, $k \geq 0$, and $\Gamma(s) = [0, (1 - \delta)k + f(s)]$. The following figure presents the code defining $B(k', s)$.

```
function B(i, j, h, v, model)
    (; β, σ, δ, α, k_grid, z_grid, Q) = model
    k, z, k′ = k_grid[i], z_grid[j], k_grid[h]
    u(c) = c^(1-σ) / (1-σ)
    f(k,z) = z*k^(α)
    c =(1-δ)*k+f(k,z)-k′
    @views value = c >= 0 ? u(c) + β * dot(v[h, :], Q[j, :]) : -Inf
    return value
end
```

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

The following code borrows from Sargent & Stachurski (2023) and estimates the deterministic solution of the RKC model.

```
"The Bellman operator."
function T(v, model)
    k_idx, z_idx = (eachindex(g) for g in (model.k_grid, model.z_grid))
    v_new = similar(v)
    for (i, j) in product(k_idx, z_idx)
        v_new[i, j] = maximum(B(i, j, h, v, model) for h ∈ k_idx)
    end
    return v_new
end
"Compute a v-greedy policy."
function get_greedy(v, model)
       k_idx, z_idx = (eachindex(g) for g in (model.k_grid, model.z_grid))
    σ = Matrix{Int32}(undef, length(k_idx), length(z_idx))
    for (i, j) in product(k_idx, z_idx)
        _, σ[i, j] = findmax(B(i, j, h, v, model) for h ∈ k_idx)
    end
    return σ
end
"Value function iteration routine."
function value_iteration(model, tol=1e-5)
    vz = zeros(length(model.k_grid), length(model.z_grid))
```

```
    v_star = successive_approx(v -> T(v, model), vz, tolerance=tol)
    return get_greedy(v_star, model)
end
function vfi(model, tol=1e-5)
    vz = zeros(length(model.k_grid), length(model.z_grid))
    v_star = successive_approx(v -> T(v, model), vz, tolerance=tol)
    return v_star
end
```

```
vfi (generic function with 2 methods)
```

The dynamic logit function

$$\hat{p}(k' \mid s) = \frac{e^{\frac{B(k',s)}{\lambda}}}{\int_{\Gamma} e^{\frac{B(k',s)}{\lambda}} \cdot \mathrm{d}k'}.$$

is estimated in the following code

```
"Dynamic logit function"
function QR(λ,i,j,v,model)
    (; β, σ, δ, α, k_grid, z_grid, Q) = model
 N = length(k_grid)
 P=zeros(N)
  for h=1:N
   bb =B(i, j, h, v, model)
   P[h] = exp(bb/λ)
  end
 return P./sum(P)
end
```

```
Main.Notebook.QR
```

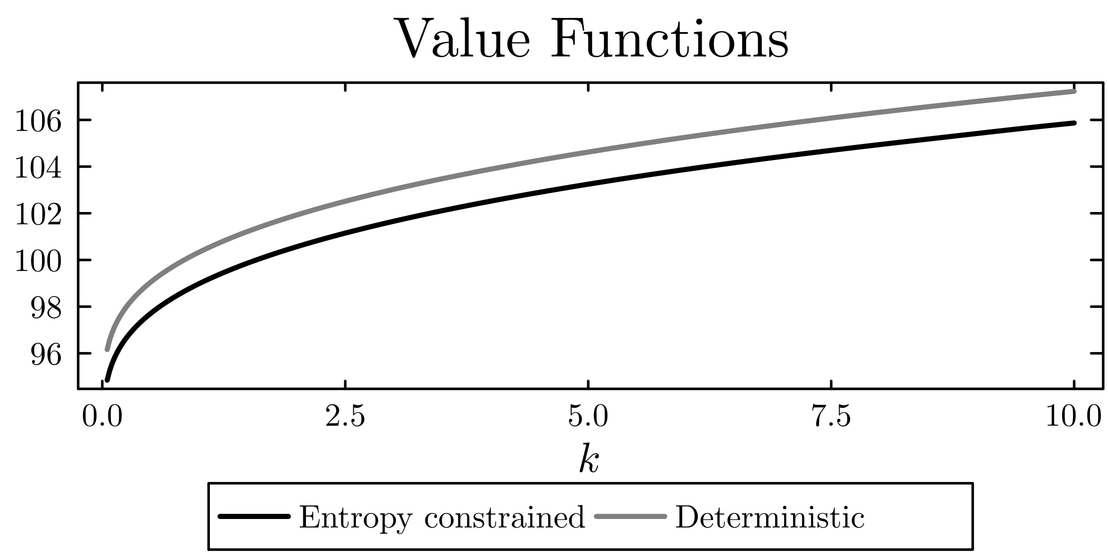
The following code solves Theorem 1 in the main text for the RKC model.

```
#Value function
function T_QR(λ,v,model)
    (; β, σ, δ, α, k_grid, z_grid, Q) = model
     k_idx, z_idx = (eachindex(g) for g in (model.k_grid, model.z_grid))
     v_new = similar(v)
     N=length(k_grid)
     for (i, j) in product(k_idx, z_idx)
            P = QR(λ,i,j,v,model)
            bb= zeros(N)
      for h=1:N
          bb[h] = (B(i,j,h,v,model) > -Inf)
```

```
       end
         NN = sum(Int32,bb) # This avoids getting NaN
         v_new[i, j] = sum(P[h]*B(i,j,h,v,model) for h=1:NN)
      end
       return v_new
end
"Contraction Mapping Following Theorem 1"
function solve_ramsey_modelQR(λ, v_init, model)
    (; β, σ, δ, α, k_grid, z_grid, Q) = model
    v_star = successive_approx(v -> T_QR(λ, v, model), v_init)
    return v_star
end
"Model solution"
λ=0.3
model=create_savings_model()
v_init =  zeros(length(model.k_grid), length(model.z_grid))
v_E = solve_ramsey_modelQR(λ,v_init,model)
σ_star =  value_iteration(model)
v_star = vfi(model);
```

The following graph contrasts the value function with probabilistic and deterministic solutions.

```
plot(model.k_grid,v_E[:,5],color=:black,      label="Entropy      constrained",
xlabel=L"k",w=2)
plot!(model.k_grid,v_star[:,5],color=:gray,label="Deterministic",  title="Value
Functions",box=:on, grid=:false,w=2)
plot!(legend=:outerbottom, legendcolumns=2)
plot!(size=(450,250))
```



Given the solution of $v(s)$, I estimate the dynamic logit function as follows:

```
"Compute Probability density"
function quantal(λ,model,v)
    (; β, σ, δ, α, k_grid, z_grid, Q) = model
    k_idx, z_idx = (eachindex(g) for g in (model.k_grid, model.z_grid))
    Ps = zeros(length(model.k_grid),length(model.z_grid), length(model.k_grid))
 for (i, j) in product(k_idx, z_idx)
        Ps[i, j,:]= QR(λ,i,j,v,model)
    end
    return Ps
end
Ps=quantal(λ,model,v_E) # MaxEnt pdf
"Deterministic Solution" # see Figure 2 in the main text
    function k_0entropy(model)
        (; β, σ, δ, α, k_grid, z_grid, Q) = model
         K_0star=zeros(length(model.z_grid))
                for i=1:length(model.z_grid)
                    K_0star[i]=(((1/β)-1+δ)/(α*z_grid[i]))^(1/(α-1))
                end
            return K_0star
     end
k0=k_0entropy(model);
```

Generate data to use the law of large numbers.

```
#Simulate capital dynamics (indices rather than grid values)
    m=150000
    mc = MarkovChain(model.Q)
        # Compute corresponding capital 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] = 5  # initial condition
    k_idx_series2[1] = 5  # 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
end
k_qr_low=k_dynamics(m,mc,4,Ps) # 4 represents a specific level of the technology
shock
# Compute values of consumption
c_qr=                       (k_qr_low[1:end-1].^model.α).-k_qr_low[2:end].+(1-
model.δ).*k_qr_low[1:end-1]
# Deterministic equilibrium of consumption
c0_star=k0[4]^(model.α)-model.δ*k0[4]
```

```
1.4785451811842374
```

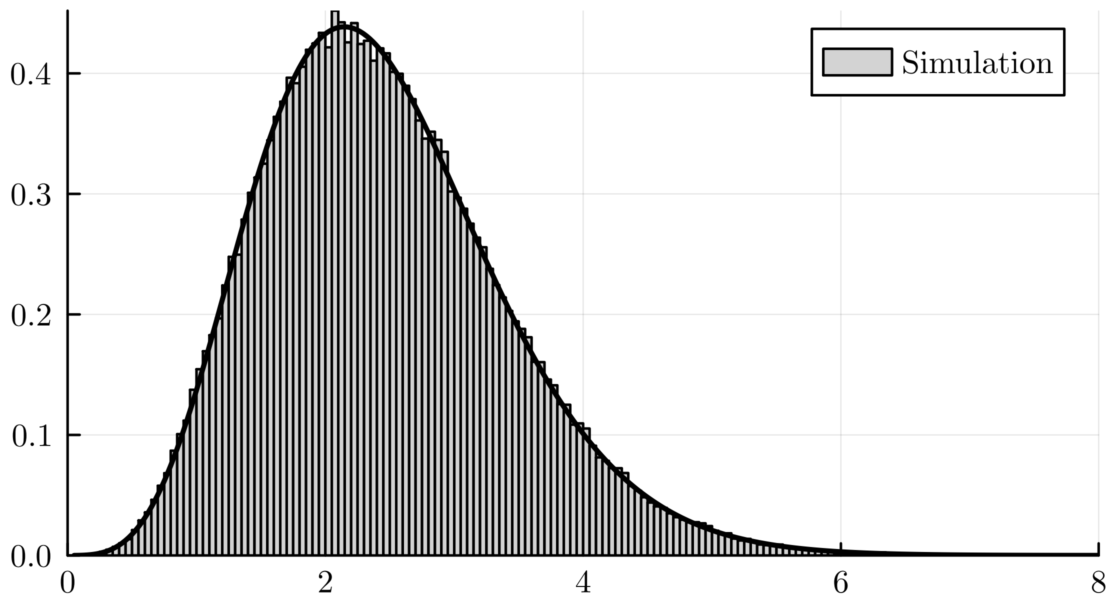
The following code generates the Chapman-Kolmogorov equation described in Theorem 2 of the main text.

```
"Compute Stationary density"
function stationary_dist(Ps,initial,iterations, shock)
     N =size(Ps,1)
  Tp=zeros(N,iterations)
        Tp[:,1] = Ps[initial,shock,:]
  for h=2:iterations
            Tp[:,h] = sum(Ps[i,shock,:].*Tp[i,h-1] for i=1:N)
        end
        return Tp
    end
```

```
Tps1=stationary_dist(Ps,5,100,4) # Initial value k_0
Tps2=stationary_dist(Ps,155,100,4); # Initial value k_{1}
```

Note that theoretical and simulated stationary density coincide

```
histogram(k_qr_low,      normalize=:pdf,      bins=305,      label="Simulation",
color=:lightgray)
plot!(model.k_grid,40*Tps2[:,end],      color=:black,      w=2,      label=:false,
xlims=(0,8))
plot!(size=(450,250))
```



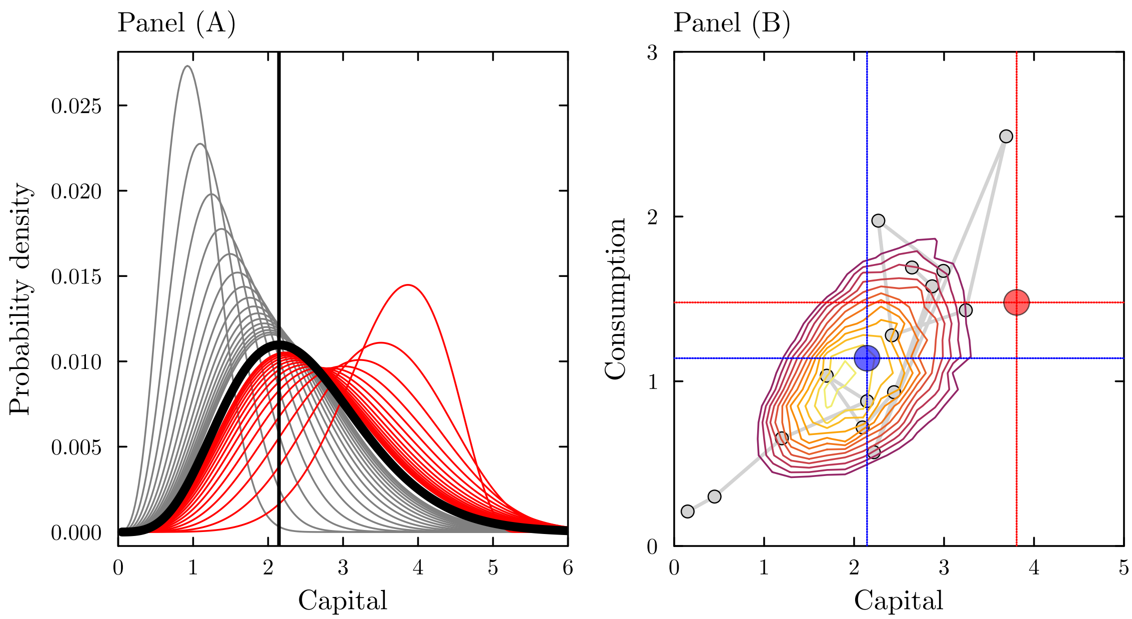
The following graph is the exact solution of Figure 3 in the main text.

```
h_kc=fit(Histogram, (k_qr_low[1000:end-1],c_qr[1000:end]), closed = :left, nbins
= (50, 50))
normalize(h_kc, mode=:pdf)
                    gap = maximum(h_kc.weights)/100
                    levels_ck = 40*gap:5*gap:200*gap
                     pp2=plot(k_qr_low[1:2:30],c_qr[1:2:30], color=:lightgray,
w=2, title="Panel (B)", titlelocation = :left,
                    ylabel="Consumption", xlabel="Capital", label=:false)
                     scatter!(k_qr_low[1:2:30],c_qr[1:2:30], color=:lightgray,
label=:false, titlefont=10, xguidefontsize=10, yguidefontsize=10)
                    contour!(midpoints(h_kc.edges[1]),
                                                         midpoints(h_kc.edges[2]),
                                                                    h_kc.weights';
                                                                            levels
```

```
= levels_ck, label=:false, colorbar_entry=false)
                vline!(k0[4]*ones(1), color=:red, linestyle=:dot, label=:false)
                        hline!(c0_star*ones(1), xlims=(0,5), ylims=(0,3), w=1,
color=:red, linestyle=:dot, label=:false)
                         scatter!([k0[4]],[c0_star],color="red", label=:false,
markersize=8, alpha=0.6)
                            vline!(2.144*ones(1), color=:blue, linestyle=:dot,
label=:false)
                    hline!(1.14*ones(1), xlims=(0,5), ylims=(0,3), color=:blue,
linestyle=:dot, label=:false)
               scatter!([2.144],[1.14],color="blue", label=:false,markersize=8,
alpha=0.6)

"Equivalent to Figure 2"
pp1=plot(model.k_grid,Tps1[:,4:1:19], color=:gray, label=:false)
plot!(model.k_grid,Tps2[:,1:1:15],  color=:red,  label=:false,  xlims=(0,6),
ylabel="Probability density", xlabel="Capital")
vline!(model.k_grid[argmax(Tps2[:,end])]*ones(1),label="",color=:black,w=2)
plot!(model.k_grid,Tps2[:,end], color=:black, w=5, label=:false, xlims=(0,6),
title="Panel  (A)",  titlelocation  =  :left,  titlefont=10,  xguidefontsize=10,
yguidefontsize=10)

plot(pp1,pp2,   layout=(1,2),   framestyle=:box,left_margin   =   3Plots.mm,
grid=:false,bottom_margin = 3Plots.mm)
plot!(size=(650,350))
```



Bibliography

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