Advanced Topics Macro I - Assignment 5

Andrea Titton & Alessandro Zona Mattioli

December 2020

Question 1

As for the other problem sets, the main file for this routine is in run-five.jl and all required functions/tools are stored in the folder src/week-five.

(a)

The results for this point are generated by the first part of the function run_krusselsmith. First, the function loads the process DenHaanModel we define in week-five/denhaan.jl and then calls the Krusell-Smith algorithm to retrieve the policy function g. The rest of the routine has to do with point (c).

```
1
2  function run_krusselsmith(;stoch=true, append="", μ=0.15,ρ =0.8,
       do_plot=false, N_a=100, N_m=10)
3      model = DenHaanModel(μ)
4      g = krusellsmith(model; verbose=true, ρ=ρ, stoch=stoch)
5
6      a_grid = range(0.01, 5., length=500)
7
8      sim = stoch ? economysim_det : economysim
9      as, zs = sim(g, model; T=5_000)
10
11     if do_plot
12         plot(title="Policy function", legend=:left, xlabel="a",
               ylabel="a′(a)")
13
14         for (z_f, ε_f) in model.ζ.S
15             ys_pol = g.(a_grid, 1., z_f, collect(Float64, ε_f))
16             plot!(a_grid, ys_pol, label="a′(a | $z_f, $ε_f)")
17         end
18
19         savefig(" $plot_path/policy_krusell$append.png")
20
21         boom = zs .== 1.01
22
23         boom_as = vec(mean(as[boom, :], dims=1))
24         recession_as = vec(mean(as[.!boom, :], dims=1))
25
26         a_grid = range(0.01, 5., length=500)
27         plot(title="Steady state distribution", xaxis="a", yaxis="
               probability")
28
29         plot!(a_grid, pdf(kde(boom_as), a_grid), label="boom")
30
31         plot!(a_grid, pdf(kde(recession_as), a_grid), label="
               recession")
32
```

```
        savefig("$plot_path/stationary_distribution $append.png")
    end

    return model, g, as, zs
end
```

Turning to the krusellsmith function more in detail, this one is stored in week-five/krusell-smith/main.jl. It first loads the main elements of the model, such as utility function and the functions $R(\lambda, z)$ and $w(\lambda, z)$. Afterwords, an initial guess for B is picked and the iterations start. The following loop will call the endogenous grid method (through the function endgrid_method) to solve for the policy, then it will forward simulate the a and z with economysim and retrieve the new B' via OLS (function regress_m) in case the stochastic simulation is chosen as an option (see line 51). Otherwise the non-stochastic method is called. Finally, if the distance between B and B' is small enough, the policy function is returned, otherwise the loop continues after computing the new B with the contraction $B = B + \rho B'$.

Let us focus on the various functions called in this routine. The first is endgrid_method. The function first unpacks the necessary parameters and functions from the model and then defines the grids for the random processes ε and z and for the policy function and m, together with the conditional probabilities from the transition matrix for ε and z. At every iteration the function computes the Euler-equation solving for a (see lines 35-52). Then to get the next policy we do linear interpolation and compute the difference with the initial policy we had at the beginning of the iteration. As usual, the loop keeps on going until convergence applying the dumping parameter (which we adjust dynamically) unless the distance between the policy functions has become sufficiently low.

```

function endgrid_method(
    Ψ::Function, model::DenHaanModel, grids_sizes::NTuple{2,Int};
    ρ0=0.8,
    grid_bounds=[.01, 10.],
    max_iter=1_000, tol=1e-3,
    verbose=false)

    @unpack S_ε, S_z, ζ = model
    @unpack β, l, δ, μ = model

    u, u′, invu′ = makeutility(model)
    R, w, τ = makeproduction(model)
    cons, invcons = makeconsumption(model)

    c = positive ∘ cons
    invc = positive ∘ invcons

    ε_grid = collect(Float64, S_ε)
    z_grid = collect(S_z)
```

```
    D_z, D_ε = length(S_ε), length(S_z)
    P_cond(state) = ζ.P[findfirst(==(state), ζ.S), :]

    N_a, N_m = grids_sizes

    m_grid = range(grid_bounds..., length=N_m)
    a_grid = range(grid_bounds..., length=N_a)

    policy = repeat(a_grid, 1, N_m, D_z, D_ε)
    space = collect.(Iterators.product(a_grid, m_grid, z_grid, ε_grid
        ))

    for iter in 1:max_iter
        g = positive ∘ fromMtoFn(policy, a_grid, m_grid, z_grid, ε
            _grid)

        function next_value(state::Tuple{Float64,Int}, Ψ′, a′)
            z′, ε′ = state
            ε′ = convert(Float64, ε′)
            return R(z′, Φ′) * u′(c(a′, Φ′, z′, ε′, g))
        end

        inv_policy = similar(policy)

        @threads for (i, j, k, l) in cartesianfromsize(N_a, N_m, D_z,
             D_ε)
            # Endogenous grid method for each a′
            a′, m = a_grid[i], m_grid[j]
            z, ε = z_grid[k], ε_grid[l]

            Ψ′ = Ψ(z, m)

            rhs = β * P_cond((z, ε))′ * next_value.(ζ.S, Φ′, a′)
            c′ = invu′(rhs)
            a = invc(c′, m, z, ε, a′)

            inv_policy[i, j, k, l] = a
        end

        new_policy = similar(policy)

        @threads for (j, k, l) in cartesianfromsize(N_m, D_z, D_ε)
            origin_a = inv_policy[:, j, k, l]
            ixs = sortperm(origin_a)

            forward_policy = LinearInterpolation(origin_a[ixs], a_grid
                [ixs], extrapolation_bc=Line())
            new_policy[:, j, k, l] = forward_policy.(a_grid)
        end

        d = new_policy - policy
```

```
        err_distance = maximum(abs.(d))
        verbose && print("Iteration $iter / $max_iter: $(@sprintf("
            \%.4f", err_distance)) \r")

        if err_distance < tol
            verbose && print("Found policy in iteriterations(|x - x'| =(
                @sprintf("%.4f", err_distance)))\n")
            return g
        end

        ρ = ρ0 - iter / max_iter # Dynamic dumping parameter
        policy += ρ * d # Update with dumping parameter
    end

    throw(ConvergenceException(max_iter))
end
```

Once the loop for the convergence of the policy function is completed, we forward simulate the economy and regress via OLS to obtain the new $\Psi(\bar{m})$. The simulation first forward simulates z and then $\varepsilon|z, z'$. Once we have the sequence of both shocks, we apply the policy function $a' = g(a, m, z, \varepsilon)$ and retrieve the sequence of a, where m is the mean of a for that period.

```

function economysim(
    g::Function, model::DenHaanModel;
    drop::Int=500, N::Int=10_000, T::Int=1_500, verbose=false)

    T′ = T - drop
    zs = simulation(model.Z, T)[drop + 1:end]

    εs = conditional_simulation(model, zs, N)

    as = similar(εs)
    as[1, :] = rand(Uniform(), 1, N)

    for (t, z) in enumerate(zs[1:end - 1])
        verbose && print("Simulating economy $t / $(T′ - 1)\r")
        ε = εs[t, :]
        a = as[t, :]

        m = mean(a)

        a′ = @. g(a, m, z, ε)
        as[t + 1, :] = a′
    end

    verbose && print("\n")

    return as, zs
end
```

Finally, the regression is a simple OLS on periods of "boom" and "busts" in the economy:

```
function regress_m(logm::Vector{Float64}, boom::BitArray{1})

    afterboom = circshift(boom, 1) # All periods that follow a boom

    gX = hcat(ones(size(logm[boom])), logm[boom])
    gY = logm[afterboom]

    β_g = inv(gX' gX) * gX' gY

    bX = hcat(ones(size(logm[.!boom])), logm[.!boom])
    bY = logm[.!afterboom]

    β_b = inv(bX'bX) * bX'bY

    return β_b, β_g
end
```

With $\rho = 0.65$, the routine takes about 10 outer loops to converge to the policy function g, as reported in Figure 2. The resulting function is reported in Figure 1.

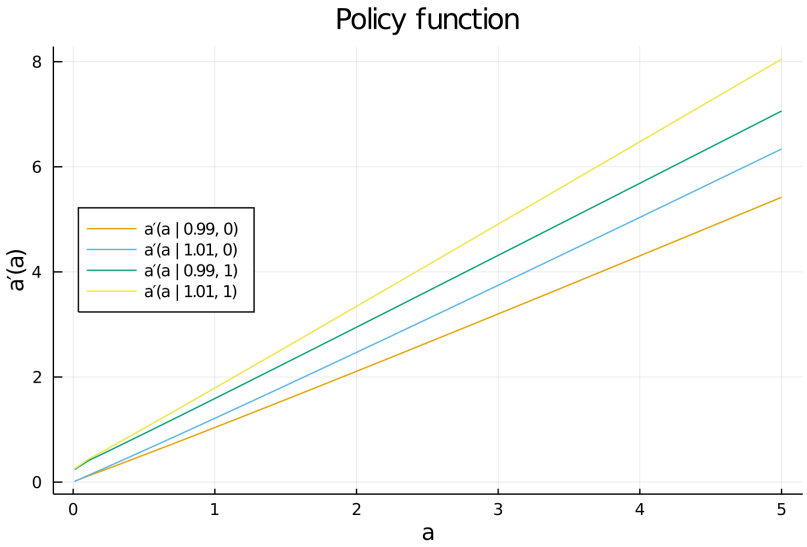

Figure 1: Policy function from the Krusell-Smith algorithm, given ε and z.

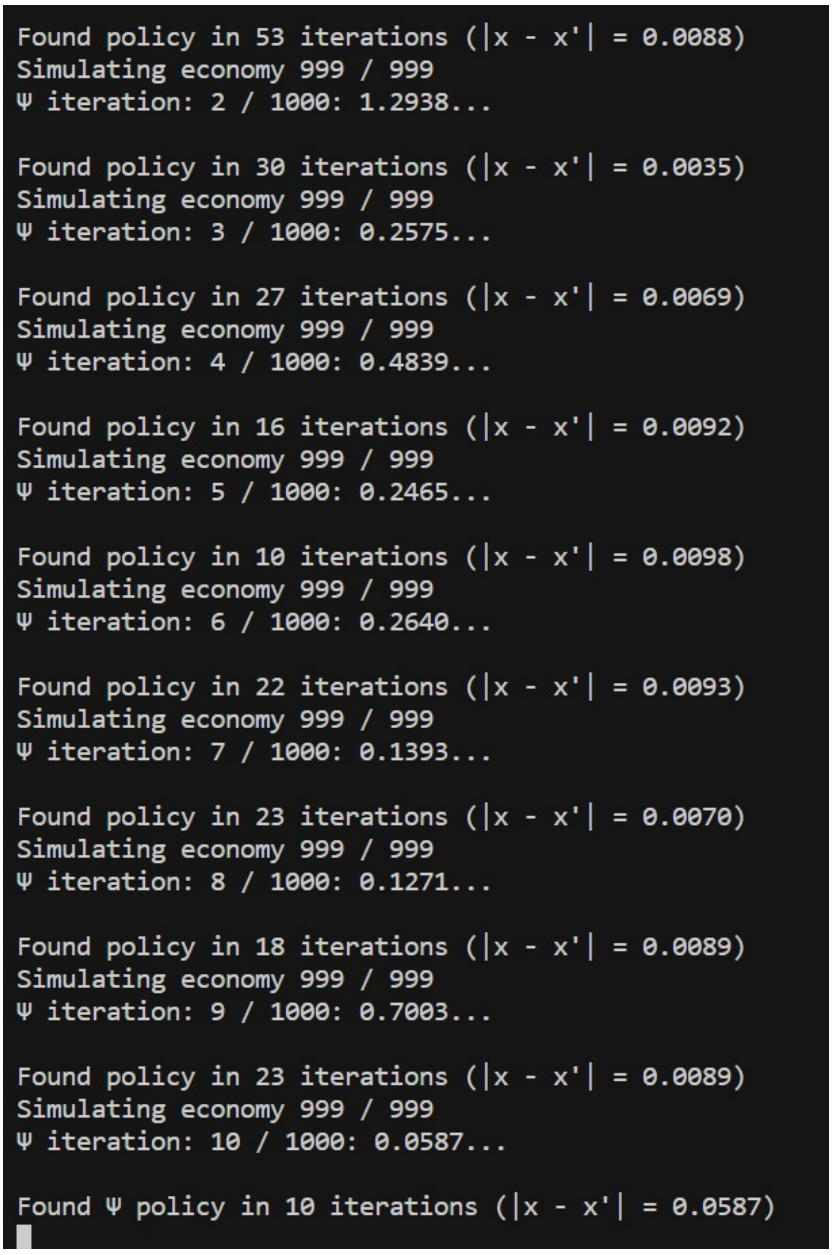

Figure 2: Convergence process of Krusell-Smith.

(b)

The non-stochastic simulation method is implemented in week-five/krusell-smith/det_simulation.jl with the economysim_det function.

We first define a fine grid for a and initialize λ on the Cartesian product of a and ϵ. Then given a simulation of the aggregate shock, every period the saving decision a' is computed with the policy function for both employed and unemployed. The next period saving decision a' is then cast linearly onto the grid based on the distance to the previous point,

$$\omega = \frac{a\prime - a_{k-1}}{a_k - a_{k-1}} \tag{1}$$

and the probability of the next period employment $\pi(\epsilon'|\cdot)$. The resulting policy function is virtually the same as the one in Figure 1.

```
1
2  function economysim_det(
3  g::Function, model::DenHaanModel;
4  T::Int=1_500, J::Int=1000, verbose=false)
5
6      S_ϵ = collect(Float64, model.S_ϵ)
7      S = length(S_ϵ)
8
9      zs = simulation(model.Z, T)
10
11     a_grid = collect(range(.01, 100., length=J))
12
13     λ0 = rand(Uniform(a_grid[1], a_grid[end]), (J, length(model.S_ϵ))
           )
14
15     λ = Array{Float64}(undef, T, J, S)
16     λ[1, :, :] = λ0
17
18     for (t, z) in enumerate(zs[1:end - 1])
19         verbose && print("Simulating economy $t / $T\r")
20
21         λt = λ[t, :, :]
22         m = mean(λt)
23         a′ = g.(λt, m, z, S_ϵ′)
24
25         λt′ = zeros(J, S)
26
27         @threads for (j, s) in cartesianfromsize(J, S)
28             ϵ = S_ϵ[s]
29             x = a′[j, s]
30
31             k = findfirst(a_k -> x < a_k, a_grid)
```

```
            dens = π_ϵ′(ϵ, zs[t + 1], z, model)

            if isnothing(k)
                λt′[end, :] += dens * x
            elseif k == 1
                λt′[1, :] += dens * x
            else
                ω = 1. - (x - a_grid[k - 1]) / (a_grid[k] - a_grid[k -
                    1])
                λt′[k - 1, :] += ω * dens * x
                λt′[k, :] += (1 - ω) * dens * x
            end
        end

        λ[t + 1,:, :] = λt′

    end

    verbose && print("\n")

    as = reshape(mean(λ, dims=3), (T, J))

    return as, zs
end
```

(c)

In the second part of run_krusselsmith we plot the ergodic savings distribution, by forward simulating the economy for 5000 periods. Then we restrict to the periods of boom and busts and take the mean. The results are reported in Figure 3 and from the figure we clearly see how in periods of boom the distribution is shifted to the right, as more sustained economic growth allows for more capital accumulation.

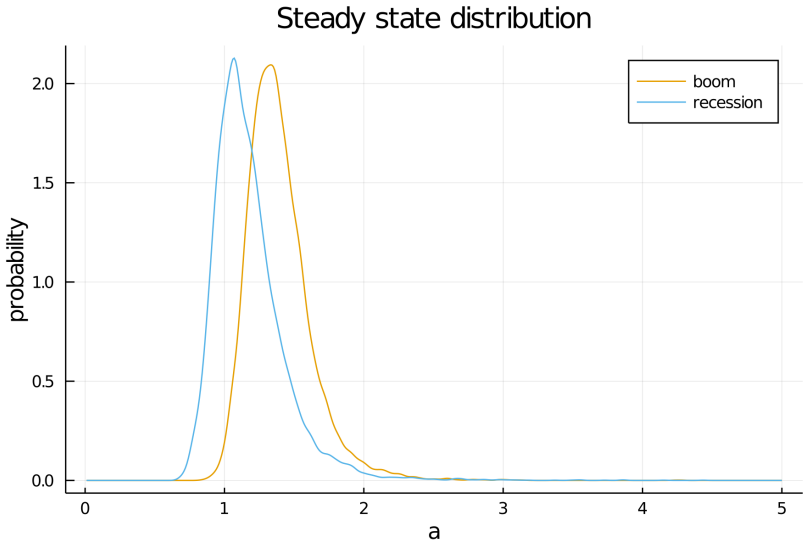

Figure 3: Convergence process of Krusell-Smith.