ECON 899 – Problem Set 3

Danny, Mitchell, Ryan, Yobin, and Hiroaki

December 15, 2021

The optimal labor supply

Houshold's problem is

$$\max_{c,l} \frac{\left[c^{\gamma}(1-l)^{1-\gamma}\right]^{1-\sigma}}{1-\sigma} \text{ s.t. } c = (1-\theta)we(z,\eta_j)l + (1+r)a - a' \tag{1}$$

The first order condition yields

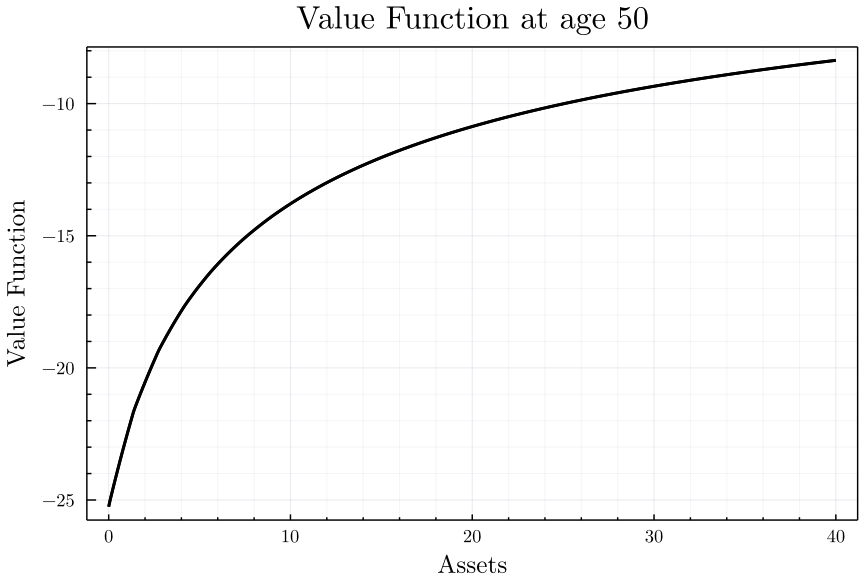
$$c = \frac{\gamma}{1-\gamma}(1-\theta)we(z,\eta_j)(1-l) \tag{2}$$

Using this and the budget constraint, we get the optimal l in the PS.

Exercise 1

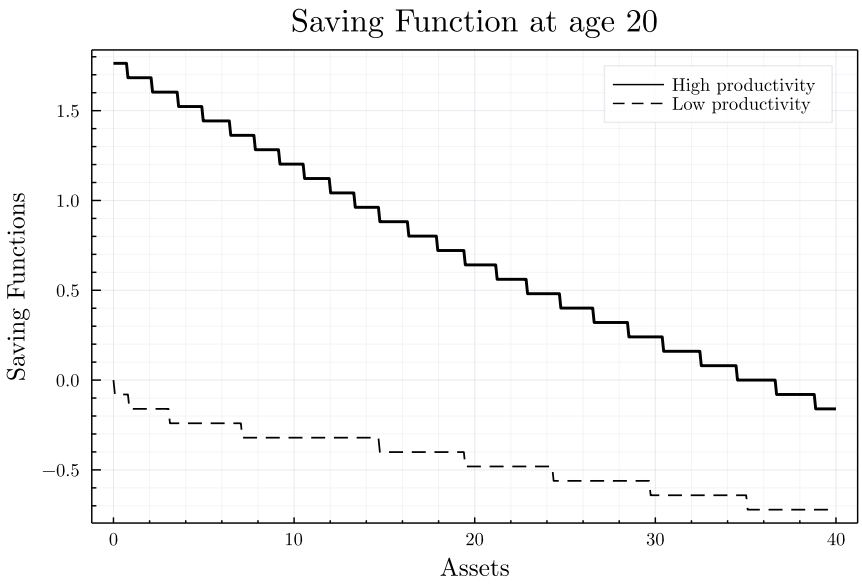
Solve the dynamic programming problem of retirees and workers. Plot the value function over a for a retired agent at the model-age 50. Is it increasing and concave? Plot the savings function for a worker at the model-age 20, $a'_{20}(z,a)$. Is saving increasing in a? Is it increasing in z?

Answer: The value function over a for a retired agent at the model-age 50 is given below:



It is increasing and concave.

The savings function for a worker at the model-age 20, $a'_{20}(z,a) = a' - a$ is the following.



The saving is decreasing in a and increasing in z. ∎

Exercise 2

After solving for agent's dynamic programming problem, solve for the steady-state distribution of agents over age, productivity and asset holdings, $F_j(z, a)$. Find first the relative sizes of each cohort of age j (denoted by μ_j) using the expression below:

$$\mu_{i+1} = \frac{\mu_i}{1+N}, \text{ for } i = 1, \ldots, N-1$$

with any $\mu_1 = \tilde{\mu}_1 > 0$. Then normalize μ, so that it sums up to 1 across all age groups. Finally, start with the newborn generation with zero wealth: given its distribution, $F_1(z^H, 0) = \mu_1 0.2037$ and $F_1(z^L, 0) = \mu_1 0.7963$, compute the distribution of agents over asset holdings at subsequent ages by applying the optimal decision rules.

Answer: The code used to solve this portion is included in the appendix, and the results shown in Exercise 3 use this code from Exercise 2. ∎

Exercise 3

1. First, solve for the benchmark model with social security. Is this economy dynamically efficient (compare the interest rate with the implicit return from social security, which is equal to the population growth rate)? Now eliminate social security by setting $\theta = 0$. Observe how aggregate capital accumulation and labor supply change as a result of the tax reform. Provide intuition in terms of insurance and output efficiency. How does aggregate welfare change? Who benefits and who loses due to this reform? How does the reform affect cross-sectional wealth inequality? You can use table 1 to support your answers.

   Answer: Table 1 summarises the results. This economy is not efficient because it is not equal to population growth. The total welfare decreases. Figure 1 shows the effect of this policy change by age. Only a few young people are benefited. ∎

2. In the second experiment, there is no idiosyncratic risk. Assume that at each age j, $z^L = z^H = 0.5$. First, compute the aggregate variables for the case with social security. How does the aggregate capital stock change relative to the benchmark model? Provide intuition in terms of capital as a buffer stock. Then, eliminate social security. How does the aggregate welfare change? What can you conclude about social security as an insurance device against idiosyncratic risk? Comment on the extent, to which these welfare comparisons across steady states are meaningful or misleading.

   Answer: The welfare decreases. Social security decreases comparing with the benchmark model. The aggregate capital decreases because people don't have saving demand for the shock. If social security is eliminated, the ag-

Table 1: Results of policy experiments

	Benchmark Model		No risk, $z^L = z^H = 0.5$		Exogenous labor, $\gamma = 1$	
capital, K	3.362	4.59	1.073	1.347	6.952	9.12
labor, L	0.343	0.365	0.162	0.174	0.754	0.754
wage, w	1.455	1.592	1.263	1.338	1.424	1.57
interest, r	0.024	0.011	0.048	0.037	0.027	0.013
pension benefit, b	0.225	0.0	0.092	0.0	0.484	0.0
total welfare, W	-35.797	-37.116	-44.906	-44.825	-23.26	-25.438
cv(wealth)	—	-0.195	—	-0.087	—	-0.194

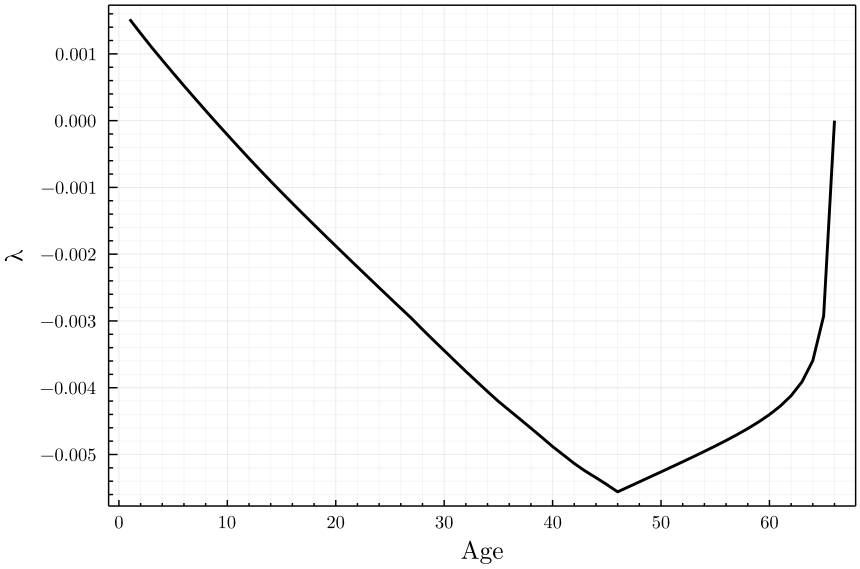


Figure 1: Effects of Policy Change by Age

gregate welfare does not change well. The result suggests that social security works as insurance because welfare decreases in the benchmark when social security is removed. ∎

3. Consider the case, when labor supply is exogenous ($\gamma = 1$). Compare the distortionary effect of social security on the aggregate labor supply. How does the support for social security change with exogenous labor supply?

   Answer: Social security does not change the aggregate labor supply even without social security. Without social security, total welfare decreases. So, fewer people support the social security change. ∎

Appendix

The first codefile runs the code:

```
@time begin
using Distributed, SharedArrays, NaNMath, JLD

#add processes
workers()
addprocs(2)


#@Distributed.everywhere include("./PS3/JuliaCode/conesa_kueger.jl");
@Distributed.everywhere include("./conesa_kueger.jl");

#prim, res = Initialize(); #=
#@time V_ret(prim, res);
#@time V_workers(prim, res);
#@time V_Fortran(res.r, res.w, res.b);
#@time SteadyStateDist(prim, res); =#

#agridf, consumption = V_Fortran(res.r, res.w, res.b);
#=
agridf
prim.a_grid
hcat(prim.a_grid, agridf, prim.a_grid - agridf)
=#
@time out_prim, out_res = MarketClearing(use_Fortran=false, tol = 1e-3);

using Plots, LaTeXStrings

theme(:vibrant)
default(fontfamily = "Computer Modern", framestyle=:box) #LaTex Style
plot(out_res.val_fun[:,:, end])
plot!(out_res.val_fun[:,:, end-1])
plot!(out_res.val_fun[:,:, end-2])

# Exercise 1
plot(out_prim.a_grid, out_res.val_fun[:,:, 50],
     title = "Value Function at age 50",
     xlabel = "Assets",
     ylabel = "Value Function",
     color=:black,
     legend = false,
     lw = 2)
savefig("../Figures/value_function50.pdf")


plot(out_prim.a_grid, out_res.val_fun[:, 1, end])
plot!(out_prim.a_grid, out_res.val_fun[:, 2, end])
plot(out_prim.a_grid, out_res.val_fun[:, 1, 20])
plot!(out_prim.a_grid, out_res.val_fun[:, 2,20])

plot(out_prim.a_grid, out_res.pol_fun[:,1,20])
plot!(out_prim.a_grid,out_res.pol_fun[:,2,20])

plot(out_prim.a_grid, out_res.val_fun[:, 1, 34])
plot!(out_prim.a_grid, out_res.val_fun[:, 2, 34])

# Calculate savings
savings = out_res.pol_fun[:,:,:] .- out_prim.a_grid;

plot(out_prim.a_grid, savings[:,1,20],
     title = "Saving Function at age 20",
     xlabel = "Assets",
     ylabel = "Saving Functions",
     label = "High productivity",
     color=:black,
     lw = 2,
     legend=:topright)
plot!(out_prim.a_grid, savings[:,2,20],
      label = "Low productivity",
      color=:black,
      line=:dash)
savefig("../Figures/savings_20.pdf")


#Plotting Policy Functions
    #Find a_hat (The asset point beyond which everyone dissaves)
```

```
    function Find_a_hat()
        a_hat=zeros(out_prim.nZ, out_prim.N_final)
        for zi=1:out_prim.nZ, ni=1:out_prim.N_final
            for ai=1:out_prim.nA
                if out_res.pol_fun[ai,zi,ni]<=out_prim.a_grid[ai]
                    a_hat[zi,ni]=out_prim.a_grid[ai]
                    break
                end
            end
        end
        return a_hat
    end
    a_hat=Find_a_hat()
    plot(out_prim.a_grid, out_res.pol_fun[:,1,20])
    plot!(out_prim.a_grid, out_res.pol_fun[:,2,20])
    vline!([a_hat[1,20]], label=L"\hat{a}")

    plot(1:out_prim.N_final,a_hat[1,:],
        xlabel="Age", ylabel=L"\hat{a}",label="High Shock")
    plot!(1:out_prim.N_final,a_hat[2,:], title="Maximum Assets at which an Agent Saves
over the Life Cycle",
        xlabel="Age", ylabel=L"\hat{a}",label="Low Shock")

plot!(out_prim.a_grid, out_prim.a_grid)

plot(savings[500,1,:])
plot!(savings[500,2,:])
plot!

# graph distributions

# accross assets for workers and retirees
a_dist = sum(out_res.F, dims = 2:3)
plot(out_prim.a_grid, a_dist[:, 1, 1])

plot(out_prim.a_grid,out_res.F[:,1,50])

# conduct policy experiments
@time prim_noSS, res_noSS               = MarketClearing(use_Fortran=false, tol =
1e-3, ss = false);

    ## for PS4
    save("../../PS4/Data/Initial_Conditions.jld",
         "Γ_0", out_res.F,
         "V_0", out_res.val_fun,
         "K_θ", out_res.K,
         "K", res_noSS.K,
         "L_θ", out_res.L,
         "L", res_noSS.L,
         "V", res_noSS.val_fun,
         "a", res_noSS.pol_fun,
         "l", res_noSS.l_fun
         )

@time prim_noRisk, res_noRisk           = MarketClearing(use_Fortran=false, tol =
1e-2, i_risk = false);
@time prim_noRisk_noSS, res_noRisk_noSS = MarketClearing(use_Fortran=false, tol =
1e-2, ss = false, i_risk = false);
@time prim_exLab, res_exLab             = MarketClearing(use_Fortran=false, tol =
1e-3, exog_l = true);
@time prim_exLab_noSS, res_exLab_noSS   = MarketClearing(use_Fortran=false, tol =
1e-3, ss = false, exog_l = true);

# write results to table 1
open("../Tables/table1.tex", "w") do io
    write(io, string(
        "\\begin{table}",
        "\n \\centering",
        "\n \\caption{\\label{tab1} Results of policy experiments}",
        "\\begin{tabular}{lcccccc}",
        "\n \\hline",
        "\n \\hline",
        "\n &\\multicolumn{2}{c}{Benchmark Model} &\\multicolumn{2}{c}{No risk, \$z^L=z^H=0.5\$}&\\multicolumn{2}
        "\n \\hline",
        "\n capital, \$K\$ & ", round(out_res.K, digits = 3), " & ", round(res_noSS.K, digits =
3), " & ", round(res_noRisk.K, digits = 3), " & ", round(res_noRisk_noSS.K, digits =
3), " & ", round(res_exLab.K, digits = 3), " & ", round(res_exLab_noSS.K, digits =
3), " \\\\",
        "\n labor, \$L\$ & ", round(out_res.L, digits = 3), " & ", round(res_noSS.L, digits =
3), " & ", round(res_noRisk.L, digits = 3), " & ", round(res_noRisk_noSS.L, digits =
3), " & ", round(res_exLab.L, digits = 3), " & ", round(res_exLab_noSS.L, digits =
```

```
3), " \\\\",
        "\n wage, \$w\$ & ", round(out_res.w, digits = 3), " & ", round(res_noSS.w, digits =
3), " & ", round(res_noRisk.w, digits = 3), " & ", round(res_noRisk_noSS.w, digits =
3), " & ", round(res_exLab.w, digits = 3), " & ", round(res_exLab_noSS.w, digits =
3), " \\\\",
        "\n interest, \$r\$ & ", round(out_res.r, digits = 3), " & ", round(res_noSS.r, digits =
3), " & ", round(res_noRisk.r, digits = 3), " & ", round(res_noRisk_noSS.r, digits =
3), " & ", round(res_exLab.r, digits = 3), " & ", round(res_exLab_noSS.r, digits =
3), " \\\\",
        "\n pension benefit, \$b\$ & ", round(out_res.b, digits = 3), " & ", round(res_noSS.b, digits =
3), " & ", round(res_noRisk.b, digits = 3), " & ", round(res_noRisk_noSS.b, digits =
3), " & ", round(res_exLab.b, digits = 3), " & ", round(res_exLab_noSS.b, digits =
3), " \\\\",
        "\n total welfare, \$W\$ & ", round(NaNMath.sum(out_res.val_fun.*out_res.F), digits =
3), " & ", round(NaNMath.sum(res_noSS.val_fun.*res_noSS.F), digits = 3), " & ", round(NaNMath.sum(res_noRisk.val_
3), " & ", round(NaNMath.sum(res_noRisk_noSS.val_fun.*res_noRisk_noSS.F), digits =
3), " & ",round(NaNMath.sum(res_exLab.val_fun.*res_exLab.F), digits = 3), " & ", round(NaNMath.sum(res_exLab_noSS
3), " \\\\",
        "\n cv(wealth) & \\textemdash & ", round(Lambda(prim_noSS, res_noSS, out_res.val_fun), digits =
3), " & \\textemdash & ", round(Lambda(prim_noRisk_noSS, res_noRisk_noSS, res_noRisk.val_fun), digits =
3), " & \\textemdash & ", round(Lambda(prim_exLab_noSS, res_exLab_noSS, res_exLab.val_fun), digits =
3), " \\\\",
        "\n \\hline",
        "\n \\end{tabular}",
        "\n \\end{table}"))
end;


# Exercise 3
c = Lambda2(prim_noSS, res_noSS, out_res.val_fun);

plot(c[1], c[2],
     title = "",
     xlabel = "Age",
     ylabel = "λ",
     color=:black,
     legend = false,
     lw = 2)
savefig("../Figures/lambda.pdf")
end
```

The second contains the relevant functions:

```
@everywhere using Parameters, DelimitedFiles, ProgressBars, SharedArrays, LinearAlgebra, NaNMath

# Define the primitives of the model
@everywhere @with_kw mutable struct  Primitives
    N_final ::Int64             = 66         # Lifespan of the agents
    J_R     ::Int64             = 46         # Retirement age
    n       ::Float64           = 0.011      # Population growth rate
    a_1     ::Float64           = 0          # Initial assets holding for newborns
    θ       ::Float64                        # Labor income tax rate
    γ       ::Float64                        # Utillity weight on consumption
    σ       ::Float64           = 2.0        # Coefficient of relative risk aversion
    α       ::Float64           = 0.36       # Capital share in production
    δ       ::Float64           = 0.06       # Capital depreciation rate
    β       ::Float64           = 0.97       # Discount factor

    # Parameters regarding stochastic processes
    z_H     ::Float64           = 3.0        # Idiosyncratic productivity High
    z_L     ::Float64           = 0.5        # Idiosyncratic productivity Low
    z_Vals  ::Array{Float64}    = [z_H, z_L] # Vector of idiosyncratic productivity values
    nZ      ::Int64             = 2          # Number of idiosynctatic productivity levels
    p_H     ::Float64           = 0.2037     # Probability of z_H at birth
    p_L     ::Float64           = 0.7963     # Probability of z_L at birth

    # Markov transition matrix for z
    Π       ::Array{Float64,2}  = [0.9261 1-0.9261; 1- 0.9811  0.9811]

    # Functions

    util  ::Function          = (c, l) -> ( c > 0 ) ? ((c^γ * (1 - l)^(1-γ))^(1-σ))/(1-σ) :
-Inf

    # Utility of a retiree
    # Todo: Remove the next 3 lines if everything is working
```

```
    # * Note im only using the utility of a worker and setign l = 0 to obtain the utility of a retiree
    # util_R  ::Function          = (c) -> c^(γ*(1-σ))/(1-σ)

    # Optimal labor supply note that last argument is x = (1+r)*a-a_next
    l_opt   ::Function          = (e, w, r, a, a_next) ->  (γ *(1-θ)*e*w-(1-γ)*( (1+r)*a - a_next ) ) /( (1 - θ)*w*e)

    # Production technology
    w_mkt   ::Function          = (K, L) -> (1-α)*(K^α)*(L^(-α))        # Labor first order condition
    r_mkt   ::Function          = (K, L) -> α*(K^(α-1))*(L^(1-α)) - δ   # Capital first order condition

    # Government budget constraint
    b_mkt   ::Function          = (L, w, m) -> θ*w*L/m   # m is mass of retirees

    # Grids
    # Age efficiency profile
    η       ::Matrix{Float64}   = readdlm("../Data/ef.txt")
    nA      ::Int64             = 500      # Size of the asset grid
    a_min   ::Float64           = 0.0       # lower bound of the asset grid
    a_max   ::Float64           = 40.0      # upper bound of the asset grid
    a_grid  ::Array{Float64}    = collect(range(a_min, length = nA, stop = a_max)) # asset grid

end # Primitives

# Structure mutable parameters and results of the model
@everywhere mutable struct Results
    w       ::Float64                       # Wage
    r       ::Float64                       # Interest rate
    b       ::Float64                       # Benefits
    K       ::Float64                       # aggregate capital
    L       ::Float64                       # aggregate labor
    μ       ::Array{Float64, 1}             # Distibution of age cohorts
    val_fun ::SharedArray{Float64, 3}             # Value function
    pol_fun ::SharedArray{Float64, 3}             # Policy function
    l_fun   ::SharedArray{Float64, 3}             # (effective) Labor policy function

    F       ::Array{Float64,3}              # Distribution of agents over asset holdings
end # Results

# Function that initializes the model
function Initialize(; θ = 0.11, γ = 0.42)
    prim = Primitives(θ = θ, γ = γ)                 # Initialize the primitives
    w = 1.05                                        # Wage guess
    r = 0.05                                        # Interest rate guess
    b = 0.2                                         # Benefits guess
    K = 4                                           # inital capital guess
    L = 0.9                                         # initial labor guess
    val_fun = SharedArray{Float64}(prim.nA, prim.nZ, prim.N_final)    # Initialize the value function
    pol_fun = SharedArray{Float64}(prim.nA, prim.nZ, prim.N_final)    # Initialize the policy function
    l_fun = SharedArray{Float64}(prim.nA, prim.nZ, prim.N_final)

    # Before the model starts, we can set the initial value function at the end stage
    # We set the last age group to have a value function consuming all the assets and
    # with a labor supply 0 (i.e. no labor) and recieving a benefit of b

    # Calculate population distribution across age cohorts
    μ = [1.0]
    for i in 2:prim.N_final
        push!(μ, μ[i-1]/(1.0 + prim.n))
    end
    μ = μ/sum(μ)

    # Finally we initialize the distribution of the agents
    F = zeros(prim.nA, prim.nZ, prim.N_final)
    F[1, 1, 1] = μ[1] * prim.p_H
    F[1, 2, 1] = μ[1] * prim.p_L

    # Initialize the results
    res = Results(w, r, b, K, L, μ, val_fun, pol_fun, l_fun, F)

    return (prim, res)                              # Return the primitives and results
end
#=
# Value funtion for the retirees
function V_ret(prim::Primitives, res::Results)

    # unpack the primitives and the results
    @unpack nA, a_grid, N_final, J_R, util = prim
    @unpack b, r = res
```

```
    # We obtain for every age group and asset holdings level the value function using backward induction
    for j in N_final-1:-1:J_R
        @sync @distributed for a_index in 1:nA
            a = a_grid[a_index]
            vals = util.(((1+r)*a + b).- a_grid, 0) .+ res.val_fun[:, 1, j+1]
            pol_ind = argmax(vals)
            val_max = vals[pol_ind]
            res.pol_fun[a_index, :, j] .= a_grid[pol_ind]
            res.val_fun[a_index, :, j] .= val_max
            res.l_fun[a_index, :, j] .= 0
        end # for a_index
    end # for j

end # V_ret
=#
function V_last(prim::Primitives, res::Results)
    last_period_value = prim.util.( prim.a_grid .* (1 + res.r) .+ res.b, 0 )
    res.val_fun[: ,: , end] = hcat(last_period_value, last_period_value)
end

function V_ret(prim::Primitives, res::Results)
    @unpack nA, a_grid, N_final, J_R, util, β = prim
    @unpack b, r = res

    for j in N_final-1:-1:J_R

        choice_lower = 1

        for index_a = 1:nA

            a = a_grid[index_a]

            maxvalsofar = -Inf

            for index_ap = choice_lower:nA

                a_next = a_grid[index_ap]

                c = (1+r)*a+b - a_next

                if c > 0

                    vals = util.(c, 0) +
                        β*res.val_fun[index_ap, 1, j+1]

                    if vals > maxvalsofar
                        maxvalsofar = vals
                        res.pol_fun[index_a, :, j] .=
                            a_grid[index_ap]
                        choice_lower = index_ap
                    end

                end
            end
            res.val_fun[index_a, :, j] .= maxvalsofar
        end
    end
end



# Value function for the workers
function V_workers(prim::Primitives, res::Results)

    # Unopack the primitives
    @unpack nA, nZ, z_Vals, η, N_final, J_R, util, β, θ, a_grid, Π, l_opt = prim
    @unpack r, w, b = res

    # First  we iterate over the age groups
    # for j in ProgressBar(J_R-1:-1:1) # Progressbar for running in console

    for j in J_R-1:-1:1 # Without progressbar for running in jupyter notebook

        # Next we iterate over the productivity levels
        @sync @distributed for z_index in 1:nZ
            z = z_Vals[z_index] # Current idiosyncratic productivity level
            #println("Solving for productivity type $z")
            e = z * η[j] # Worker productivity level (only for working age)
            LowestChoiceInd=1 #Exploiting monotonicity in the policy function
            # Next we iterate over the asset grid
            for a_index in 1:nA
```

```
                a = a_grid[a_index] # Current asset level
                cand_val = -Inf     # Initialize the candidate value
                cand_pol = 0        # Initialize the candidate policy
                cand_pol_ind = 1    # Initialize the candidate policy index
                l_pol = 0           # Initialize the labor policy

                # Next we iterate over the possible choices of the next period's asset
                #l_grid = l_opt.(e, w, r, a, a_grid) # Labor supply grid
                # if j == 20 && z_index == 2
                #     print("\n a = $a a_next reached:")
                # end
                for an_index in LowestChoiceInd:nA
                    a_next = a_grid[an_index]   # Next period's asset level
                    l = l_opt(e, w, r, a, a_next) #l_grid[an_index]        # Implied labor supply in current peri
                    if l < 0                    # If the labor supply is negative, we set it to zero
                        l = 0
                    elseif l > 1                # If the labor supply is greater than one, we set it to one
                        l = 1
                    end
                    c = w * (1 - θ) * e * l + (1 + r)*a - a_next # Consumption of worker (All people in this loop
                    if c < 0 # If consumption is negative than this (and all future a' values) are unfeasible
                        break
                    end

                    # exp_v_next = val_fun[an_index, :, j+1] * Π[z_index , :] # Expected value of next period
                    # exp_v_next = val_fun[an_index, 1, j+1] * Π[z_index , 1] + val_fun[an_index, 2, j+1] * Π[z_i

                    # calculate expected value of next period
                    exp_v_next = 0
                    for zi = 1:nZ
                        exp_v_next += res.val_fun[an_index, zi, j+1] * Π[z_index ,
zi]
                    end # zi

                    v_next = util(c, l) + β * exp_v_next # next candidate to value function

                    if v_next > cand_val
                        cand_val = v_next       # Update the candidate value
                        cand_pol = a_next       # Candidate to policy function
                        cand_pol_ind = an_index # Candidate to policy function index
                        l_pol = e*l             # Candidate to labor policy function
                    end # if v_next > cand_val

                end # Next period asset choice loop

                res.val_fun[a_index, z_index, j] = cand_val         # Update the value function
                res.pol_fun[a_index, z_index, j] = cand_pol         # Update the policy function
                res.l_fun[a_index, z_index, j] = l_pol              # Update the labor policy function
                LowestChoiceInd=copy(cand_pol_ind)
            end # Current asset holdings loop
        end # Productivity loop
    end  # Age loop
end # V_workers

# Function to obtain the steady state distribution
function SteadyStateDist(prim::Primitives, res::Results)
    # Initialize the steady state distribution
    res.F[:,:,2:end] .= zeros(prim.nA, prim.nZ)
    # Unpack the primitives
    @unpack N_final, n, p_L, p_H, nZ, nA, Π, a_grid = prim

    # Finding relative size of each age cohort

    # Finding the steady state distribution
    for j in 2:N_final
        for z_ind in 1:nZ
            for a_ind in 1:nA
                a_next_ind = argmin(abs.(res.pol_fun[a_ind, z_ind, j-1].-a_grid))
                #= This should not ever happen
                if a_next_ind == 0 # Level not reached
                    continue
                end
                =#

                for zi = 1:nZ
                    res.F[a_next_ind, zi, j] += res.F[a_ind, z_ind, j-1] * Π[z_ind, zi] *
(res.μ[j]/res.μ[j-1])
                end # zi

            end
        end # z_ind
```

```
    end # j loop

end # SteadyStateDist

# Function to solve for market prices
function MarketClearing(; ss::Bool=true, i_risk::Bool=true, exog_l::Bool=false,
    use_Fortran::Bool=false, λ::Float64=0.7, tol::Float64=1e-2, err::Float64=100.0)

    # initialize struct according to policies
    if ~ss & exog_l
        prim, res = Initialize(θ = 0, γ = 1)
        prim.l_opt = (e, w, r, a, a_next) ->  1
        res.L = sum(res.μ[1:(prim.J_R-1)])
    elseif ~ss
        prim, res = Initialize(θ = 0)
    elseif exog_l
        prim, res = Initialize(γ = 1)
        prim.l_opt = (e, w, r, a, a_next) ->  1
        res.L = sum(res.μ[1:(prim.J_R-1)])
    else
        prim, res = Initialize()
    end

    if ~i_risk
        prim.z_H, prim.z_L, prim.z_Vals = 0.5, 0.5, [0.5, 0.5]
    end

    # unpack relevant variables and functions
    @unpack w_mkt, r_mkt, b_mkt, J_R, a_grid = prim

    n = 0 # loop counter

    # iteratively solve the model until excess savings converge to zero
    while err > tol

        # calculate prices and payments at current K, L, and F
        res.r = r_mkt(res.K, res.L)
        res.w = w_mkt(res.K, res.L)
        res.b = b_mkt(res.L, res.w, sum(res.μ[J_R:end]))

        # solve model with current model and payments
        if use_Fortran
            K, L = V_Fortran(prim,res)
        else
            V_last(prim, res);
            V_ret(prim, res);
            V_workers(prim, res);
            SteadyStateDist(prim, res);

            # calculate aggregate capital and labor
            K = sum(res.F[:, :, :] .* a_grid)
            L = sum(res.F[:, :, :] .* res.l_fun) # Labor supply grid
        end

        # calculate error
        err = maximum(abs.([res.K, res.L] - [K, L]))

        if (err > tol*10)
            # Leave λ at the default
        elseif (err > tol*5) & (λ <= 0.85)
            λ = 0.85
        elseif (err > tol*1.7) & (λ <= 0.90)
            λ = 0.90
        elseif λ <= 0.975
            λ = 0.975
        end

        # update guess
        res.K = (1-λ)*K + λ*res.K
        res.L = (1-λ)*L + λ*res.L

        n+=1

        if  n % 5 == 0
            println("$n iterations; err = $err, K = ", round(res.K, digits = 4), ", L = ",
            round(res.L, digits = 4), ", λ = $λ")
        end

    end # while err > tol
    return prim, res
end # MarketClearing
```

```
# Function to calculate compensating variation
function Lambda(prim::Primitives, res::Results, W::SharedArray{Float64, 3})

    # unpack necessary variables
    @unpack F, val_fun = res
    @unpack α, β, γ, σ = prim

    # calculate and return compensating variation
    # λ = (W ./ val_fun).^(1/(γ*(1-σ))) .- 1

    λ = (val_fun ./ W).^(1/(γ*(1-σ))) .- 1
    # Is W denominator?

    return NaNMath.sum(F.*λ)

end

# To anwer "who benefits" in Exercise 3.
function Lambda2(prim::Primitives, res::Results, W::SharedArray{Float64, 3})

    # unpack necessary variables
    @unpack F, val_fun = res
    @unpack α, β, γ, σ = prim

    # calculate and return compensating variation
    λ = (val_fun ./ W).^(1/(γ*(1-σ))) .- 1
    a = [1:1:66;]
    b = dropdims(sum(sum(F.*λ, dims = 1), dims = 2), dims =1)'
    c = [a, b]
    return c
end
```