```julia
# ------------------------------------------------------------------------------
# Author: Philip Coyle
# Date Created: 05/26/2020
# ps1_Q7_optimize.jl
# ------------------------------------------------------------------------------

using Parameters
using Plots
## Housekeeping
# Directory
dir = "/Users/philipcoyle/Documents/School/University_of_Wisconsin/SecondYear/Summer_2020/CodingBootcamp/ProblemSets/PS1/"

## Structures
@with_kw struct Params
    β::Float64 = 0.99
    δ::Float64 = 0.025
    θ::Float64 = 0.36

    tol::Float64 = 1e-10
    maxit::Int64 = 10000
end

@with_kw struct Shocks
    Zg::Float64 = 1.25
    Zb::Float64 = 0.2

    # Transition Probabilities
    # Pyx = Pr(Z' = Zy | Z = Zx) x∈{g,b} & y∈{g,b}
    Pgg::Float64 = 0.977
    Pbg::Float64 = 1 - Pgg
    Pbb::Float64 = 0.926
    Pgb::Float64 = 1 - Pbb
    Tmat::Array{Float64,2} = [
        Pgg Pbg
        Pgb Pbb
    ]
end

@with_kw struct Grids
    Zg::Float64 = 1.25
    Zb::Float64 = 0.2

    k_lb::Float64 = 1
    k_ub::Float64 = 75
    n_k::Int64 = 100
    k_grid::Array{Float64,1} = range(k_lb, stop = k_ub, length = n_k)

    n_Z::Int64 = 2
    Z_grid::Array{Float64,1} = [Zg, Zb]
end

mutable struct PolFuncs
    pf_c::Array{Float64,2}
    pf_k::Array{Float64,2}
    pf_v::Array{Float64,2}
end


## Functions
function solve_model()
    P = Params()
    S = Shocks()
    G = Grids()

    @unpack n_k, n_Z = G
    # Initial Guess
    pf_c = zeros(n_k, n_Z)
    pf_k = zeros(n_k, n_Z)
    pf_v = zeros(n_k, n_Z)
    PFs = PolFuncs(pf_c, pf_k, pf_v)

    converged = 0
    it = 1
    while converged == 0 && it < P.maxit
        @unpack pf_v, pf_k, pf_c = PFs

        pf_c_up, pf_k_up, pf_v_up = Bellman(P, S, G, PFs)

        diff_v = sum(abs.(pf_v_up - pf_v))
        diff_k = sum(abs.(pf_k_up - pf_k))
        diff_c = sum(abs.(pf_c_up - pf_c))
```

```
82
83          max_diff = diff_v + diff_k + diff_c
84
85          if mod(it, 250) == 0 || max_diff < P.tol
86              println(" ")
87              println("**************************************************")
88              println("AT ITERATION = ", it)
89              println("MAX DIFFERENCE = ", max_diff)
90              println("**************************************************")
91
92              if max_diff < P.tol
93                  converged = 1
94              end
95          end
96          # Update the policy functions
97          PFs = PolFuncs(pf_c_up, pf_k_up, pf_v_up)
98          it = it + 1
99      end
100
101     return G, PFs
102 end
103
104 function Bellman(P::Params, S::Shocks, G::Grids, PFs::PolFuncs)
105     @unpack β, δ, θ = P
106     @unpack Zg, Zb, Pgg, Pbg, Pbb, Pgb, Tmat = S
107     @unpack n_k, k_grid, n_Z, Z_grid = G
108     @unpack pf_c, pf_k, pf_v = PFs
109
110     # To make updating work
111     pf_k_up = zeros(n_k, n_Z)
112     pf_c_up = zeros(n_k, n_Z)
113     pf_v_up = zeros(n_k, n_Z)
114
115     for (i_Z, Z) in enumerate(Z_grid)
116         Pr = Tmat[i_Z, :]
117         for (i_k, k_today) in enumerate(k_grid)
118             # Must be defined outside loop.
119             v_today = log(0)
120             c_today = log(0)
121             k_tomorrow = log(0)
122
123             # Find optimal investment/consumption given capital level today
124             for (i_kpr, k_temp) in enumerate(k_grid)
125                 y_today = Z * k_today^θ
126                 c_temp = y_today + (1 - δ) * k_today - k_temp
127                 v_tomorrow = Pr[1] * pf_v[i_kpr, 1] + Pr[2] * pf_v[i_kpr, 2]
128                 if c_temp < 0
129                     v_temp = log(0) + β * v_tomorrow
130                 else
131                     v_temp = log(c_temp) + β * v_tomorrow
132                 end
133
134                 if v_temp > v_today
135                     v_today = v_temp
136                     c_today = c_temp
137                     k_tomorrow = k_temp
138                 end
139             end
140
141             # Update PFs
142             pf_k_up[i_k, i_Z] = k_tomorrow
143             pf_c_up[i_k, i_Z] = c_today
144             pf_v_up[i_k, i_Z] = v_today
145         end
146     end
147
148     return pf_c_up, pf_k_up, pf_v_up
149 end
150
151 function plot_pfs(dir::String, G::Grids, PFs::PolFuncs)
152     @unpack k_grid = G
153     @unpack pf_c, pf_k, pf_v = PFs
154
155     pf1 = plot(k_grid,pf_v[:,1],legend = false,color=:black, label = "Good State",lw = 2);
156     pf_1 = plot!(pf1,k_grid,pf_v[:,2],title="Value Function",legend = true,color=:blue, label = "Bad State",lw = 2);
157
158     pf2 = plot(k_grid,pf_k[:,1],legend = false,color=:black, lw = 2);
159     pf_2 = plot!(pf2,k_grid,pf_k[:,2],title="Capital Investment",legend = false,color=:blue, lw = 2);
160
161     pf3 = plot(k_grid,pf_c[:,1],legend = false,color=:black, lw = 2);
162     pf_3 = plot!(pf3,k_grid,pf_c[:,2],title="Consumption",legend = false,color=:blue, lw = 2);
163
```

```
    pf = plot(pf_1,pf_2,pf_3,layout=(1,3),size = (600,400)) #Size can be adjusted so don't need to mess around with 'blank space'
    xlabel!("Initial Capital Stock")
    # savefig(dir * "Q7_PFs.pdf")
end

## Main Code
@time G, PFs = solve_model()
plot_pfs(dir, G, PFs)
```