JuliaMono[Extension = .ttf, Path = ./, Scale = 0.87, Contextuals = Alternate, UprightFont = *-Light, BoldFont = *-SemiBold, ItalicFont = *-Lightitalic, BoldItalicFont = *-SemiBolditalic,]

This problem set was completed by Danny Edgel, Mitchell Valdes Bobes, Ryan Mather, and Yobin Timilsena.

I. Consider the same environment as Huggett (1993, JEDC) except assume that there are enforceable insurance markets regarding the idiosyncratic shocks to earnings and that there are no initial asset holdings. Solve for a competitive equilibrium. What are prices? What is the allocation? (Hint: think about the planner's problem and then decentralize).

Answer: Under the assumptions of enforceable insurance markets + locally non-satiated preferences, the basic first and second welfare theorems hold. Hence, we will solve the planner's problem for allocations and then decentralize by setting asset prices that support the allocations as a CE.

The planner's problem can be written as

$$\mathop{\mathbb{E}}_{0}\sum_{t=0}^{\infty}\beta^t[\pi(e)u(c_{e,t})+\pi(u)u(c_{u,t})] \quad \text{s.t.} \quad \pi(e)c_{e,t}+\pi(u)c_{u,t}\leq \pi(e)y_t(e)+\pi(u)y_t(u),\ \forall t$$

The first order conditions are

$$[c_{e,t}]:\beta^t\pi(e)u'(c_{e,t})=\lambda\pi(e) \qquad [c_{u,t}]:\beta^t\pi(u)u'(c_{u,t})=\lambda\pi(u)$$

Combined, we have that $u'(c_{e,t})=u'(c_{u,t})\Leftrightarrow c_{e,t}=c_{u,t}=\bar{c}$.

Plugging into the BC, we get $\bar{c}=y_t(u)+\pi(e)[y_t(e)-y_t(u)]$. For instance, in Part II, we are given $\pi(e)=0.94$, $y_t(e)=1$ and $y_t(u)=0.5$, which would imply $\bar{c}=0.97$.

The decentralized EE for an individual i is $\beta^t u'(c_t^i)=\lambda q_t \Leftrightarrow q_{t+1}=\beta q_t=\beta^{t+1}q_0$. ∎

II. Now compute Huggett (1993, JEDC) with incomplete markets. The following takes you through the steps of solving a simple general equilibrium model that generates an endogenous steady state wealth distribution. The basic algorithm is to: 1) taking a price of discount bonds $q\in[0,1]$ as given , solve the agent's dynamic programming problem for her decision rule $a'=g_\theta(a,s;q)$ where $a\in A$ are asset holdings, $s\in S\subset R_{++}$ is exogenous earnings, and θ is a parameter vector; 2) given the decision rule and stochastic process for earnings, solve for the invariant wealth distribution $\mu^*(A,S;q)$; 3) given μ^*, check whether the asset market clears at q (i.e. $\int_{A,S}g_\theta(a,s;q)\mu^*(da,ds;q)=0$). If it is, we are done. If not (i.e. it is not within an acceptable tolerance), then bisect $[0,1]$ in the direction that clears the market (e.g. if $\int_{A,S}a'\mu^*(da,ds;q)>0$), then choose a new price $\hat{q}=q+[1-q]/2$ and go to step 1.

4. After finding fixed points of the T and T^* operators, answer the following questions:

a. Plot the policy function $g(a,s)$ over a for each s to verify that there exist $\hat{a}$ where $g(\hat{a},s)<\hat{a}$ as in Figure 1 of Huggett. (Recall this condition establishes an upper bound on the set A necessary to obtain an invariant distribution).

Answer: The policy function is graphed below in Figure 1. As can be seen, there does exist an $\hat{a}$ beyond which modeled agents always, whether employed or unemployed, dissave on net. ∎

b. What is the equilibrium bond price? Plot the cross-sectional distribution of wealth for those employed and those unemployed on the same graph.

Answer: The equilibrium bond price is $q=0.9943074$, and the resulting cross-sectional distribution of wealth is shown in Figure 2. ∎

c. Plot a Lorenz curve. What is the gini index for your economy? Compare them to the data. For this problem set, define wealth as current earnings (think of this as direct deposited into your bank, so it is your cash holdings) plus net assets. Since market clearing implies aggregate assets equal zero, this wealth definition avoids division by zero in computing the Gini and Lorenz curve.

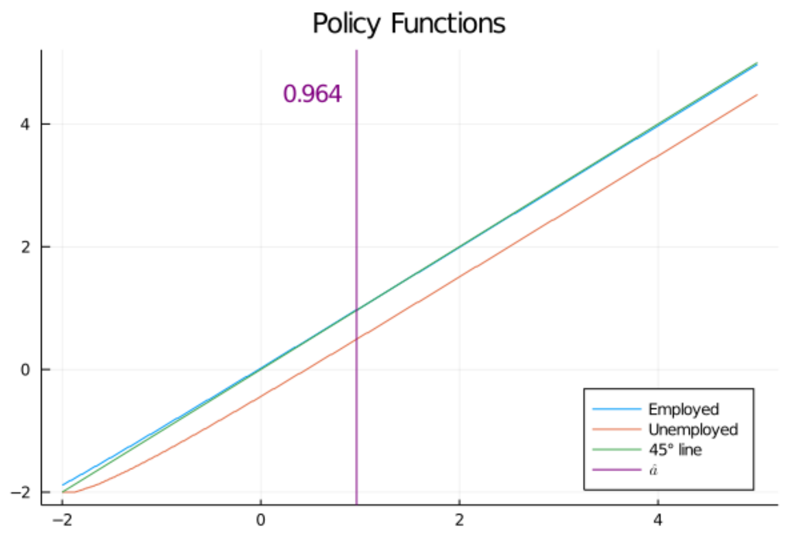

Figure 1: Problem 4(a)

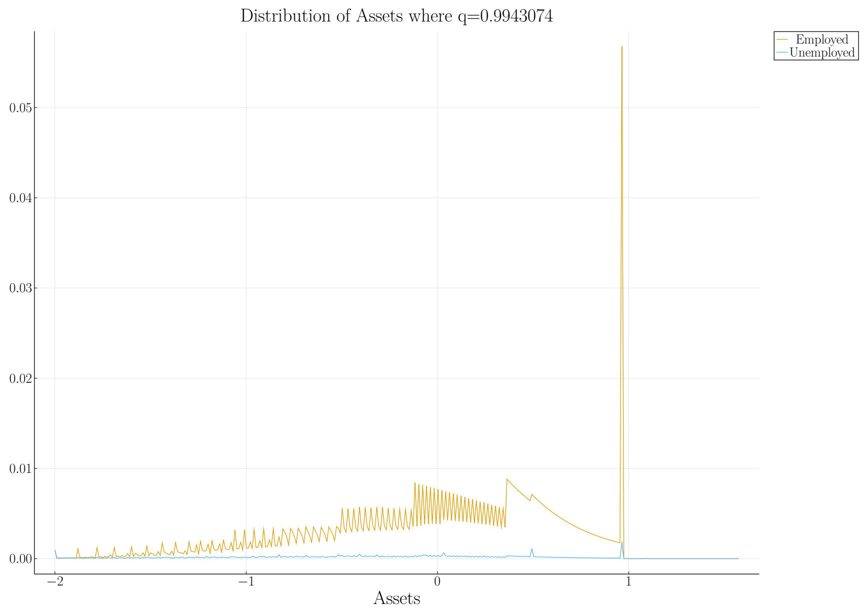

Figure 2: Problem 4(b)

Answer: The Lorenz curve is shown in Figure 3, and implies a Gini coefficient of 0.29969592. The true Income Gini coefficient for the US economy in 2020 was .458,[1] so if we take this to be the relevant statistic against which to compare our model results, our model explains about 65% of inequality as captured by the Gini Coefficient. ■

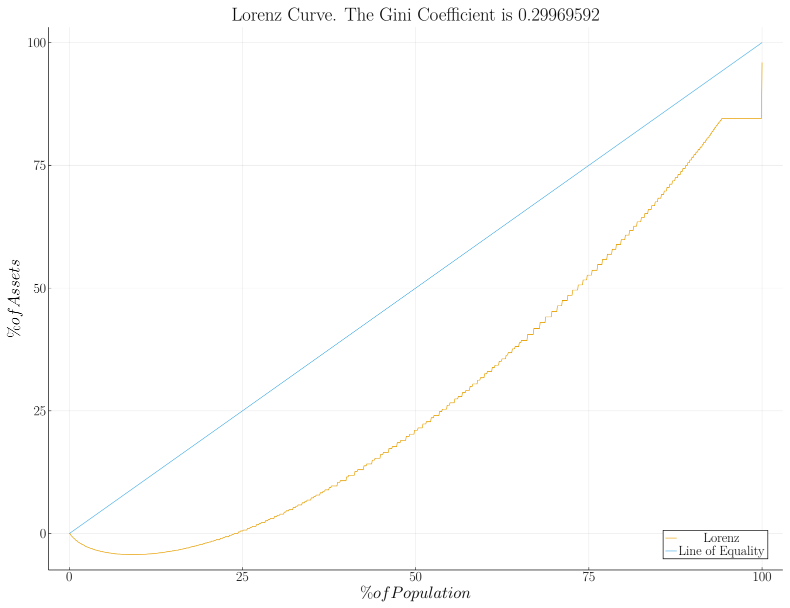

Figure 3: Problem 4(c)

III. (a) Plot $\lambda(a, s)$ across a for both $s = e$ and $s = u$ in the same graph.

Answer:

■

(b) What is W^{FB}? What is $W^{INC} = \sum_{(a,s)\in A\times S} \mu(a, s)\nu(a, s)$? What is WG?

Answer:

■

(c) What fraction of the population would favor changing to complete markets? That is $\sum_{(a,s)\in A\times S} \mathbb{1}_{\lambda(a,s)\geq 0}(a, s)\mu(a, s)$.

Answer:

■

Code Appendix

This is the "computation" code that does most of the numerical work involved for the solutions above:

```
#kywor - n  l   strutur to hol  mo l primitivs
@ v rywh r  @with_kw strut Primitivs
    β::Flo t  =  .9    #isount r t
    α::Flo t  =  .5 # pit l  sh r
    S_gri::Arry{Flo t , } = [ ,  .5] #Ernings whn  mploy   n  unmploy
    Π::Arry{Flo t , } = [.9  .0 ; .5 .5] #Trnsition Mtrix  t w n  mploymnt  n  unmploymnt
     n ::Int  =     #Num r  o   ss t  gri  points
     _gri ::Arry{Flo t , } =  oll t(r ng (- .0,lngth=n , .0))
```

[1] U.S. Census Bureau, Current Population Survey, 1968 to 2021 Annual Social and Economic Supplements (CPS ASEC), Table F-4

```
 n

#strutur  tht hols mo l rsults
@ v rywh r  mut l  strut Rsults
    v l_ un ::Arry{Flo t ,  } #v lu  untion
    pol_un ::Arry{Flo t ,  } #poliy untion
    q::Flo t
    q_Bouns::Arry{Flo t , }
 n

#untion  or initilizing mo l primitivs  n  rsults
untion Initiliz()
    prim = Primitivs() #initiliz primtiivs
    v l_ un  = zros(prim.n , ) #initil v lu  untion guss
    pol_un = zros(prim.n , ) #zros(prim.nk, ) #initil poliy untion guss
    q = (prim.β+ )/
    q_Bouns=[prim.β,  ]
    r s = Rsults(v l_ un , pol_un , q, q_Bouns) #initiliz rsults strut
    prim, r s #rturn  l i v r l s
 n

#Mking   untion  or th  innr loop
@ v rywh r  moul  IL

    untion Fin _p(S_in x, _in x,r s,prim)
        #unp k  mo l primitivs
            _gri , β, α,  n , Π, S_gri = prim. _gri , prim.β, prim.α,
               prim.n , prim.Π, prim.S_gri #unp k  mo l primitivs
        #Utility Funtion
            untion U(x)
                i  x<
                    rturn -In
                 l s
                    rturn (x^( -α)- )/( -α)
                 n
             n
        #Exploiting Monotoniity  o  V
         u g t =S_gri[S_in x] +  _gri[ _in x];
        #S r h  or v l_m x in th  oun  intrvl.
        m x_v l,m x_ p=-In ,
         or  p_in x= :n
            v l=U( u g t  - r s.q* _gri[p_in x]) +
               β*(tr nspos (Π[S_in x,:])*[ r s.v l_ un[p_in x, ] ; r s.v l_ un[p_in x, ]])
            i  v l>m x_v l
                m x_v l=v l;
                m x_ p= p_in x;
             l s i  v l<m x_v l
                 r  k #Th  V lu  untion is now  lining
             n
         n
        rturn [m x_v l, m x_ p]
     n
 n


#B l l m n  Oprtor
untion B l l m n(prim::Primitivs,r s::Rsults)
    @unp k  n ,  _gri  = prim
    v_nxt = zros(n , ) #nxt guss  o  v lu  untion to ill
     or  S_in x= :
        out=pmp( _in x -> IL.Fin _p(S_in x, _in x,r s,prim), :n )
         or  _in x= :n #Unp king th  pmp rsults
            v_nxt[ _in x,S_in x]=out[ _in x][ ]
            r s.pol_un[ _in x,S_in x]=out[ _in x][ ]
         n
     n
    v_nxt #rturn nxt guss  o  v lu  untion
 n



#V lu  untion itrtion
untion V_it r t(prim::Primitivs, r s::Rsults; tol::Flo t  =  e-4, rr::Flo t  =
   .0)
    n =   #ountr
    whil  rr>tol # gin itrtion
        v_nxt = B l l m n(prim, r s) #spit out nw  v tors
         rr =   s.(mximum(v_nxt.-r s.v l_ un))/  s(v_nxt[prim.n ,  ]) #r s t  rror  l v l
        r s.v l_ un  = .8*v_nxt+.2*r s.v l_ un  #up t   v lu  untion
        n+=
        i  mo(n,  )==
            println("V lu  Funtion itrtion $(n), Error $( rr)")
         n
     n
     println("V lu  untion  onv rg  in ", n, " itrtions.")
 n

#M rk t  l ring  or  ss ts/sts   n w q
 untion MC_ss ts(prim,r s; ist_tol::Flo t  =  e-6, ist_rr::Flo t  =    .0, ES_tol= e-2, Don= ls )
    @unp k Π, n ,  _gri = prim
    TrnsM t=zros( *n , *n ) #Th  irst n points  r  or mploy olks,  n  th  nxt n   r  or unmploy
     or  _in x= :n
        TrnsM t[Int (r s.pol_un [ _in x, ]), _in x]=Π[ , ] #Svings  hoi   or thos moving rom mp->mp
        TrnsM t[Int (r s.pol_un [ _in x, ])+n , _in x]=Π[ , ] #Svings  hoi   or thos moving rom mp->unmp
        TrnsM t[Int (r s.pol_un [ _in x, ]), _in x+n ]=Π[ , ] #Svings  hoi   or thos moving rom unmp->mp
        TrnsM t[Int (r s.pol_un [ _in x, ])+n , _in x+n ]=Π[ , ] #Svings  hoi   or thos moving rom unmp->mp
     n
    Dist=on s( *n )*( /( *n )) #
    Dist_nw= opy(Dist)
    whil  ist_rr>ist_tol
         or i= :    #It r t  until w  r  h  th  st y-st t  istriution
            Dist_nw=TrnsM t*Dist_nw
         n
         ist_rr= s .(mximum(Dist_nw.-Dist))
        Dist= opy(Dist_nw)
     n
    #Fin  Ex ss Supply  n  r st q
        Ex ssSupply=trnspos (Dist)*v t( _gri , _gri )

         i   s (Ex ssSupply)>ES_tol
            #Do v rint  o  Bis tion  M tho
            i  Ex ssSupply<
                r s.q_Bouns[ ]=r s.q
                #Wight slightly towr  ol  q to  voi  wil  lututions
                r s.q=r s.q_Bouns[ ]*.3+ r s.q_Bouns[ ]*.7
             ls
                r s.q_Bouns[ ]=r s.q
                r s.q=r s.q_Bouns[ ]*.7 +r s.q_Bouns[ ]*.3
             n
            print("Ex ss Supply: $(Ex ssSupply), q:$(r s.q)")
         ls
            Don =tru
         n
        rturn Don
 n

#solv  th  mo l
 untion Solv _mo l() #prim::Primitivs, r s::Rsults)
    prim, r s = Initiliz ()
     onv rg = ls
    Outr_loop_Itr=
    whil ~ onv rg  && Outr_loop_Itr<
        println("Bginning Asst Cl ring Loop $(Outr_loop_Itr)")
        V_it r t (prim, r s)
         onv rg =MC_ss ts(prim,r s)
        Outr_loop_Itr+=
     n
    rturn prim, r s
 n


#G t Distriution  or Plotting
 untion FinDist_ForPlot(prim,r s; ist_tol::Flo t  = e-6, ist_rr::Flo t  =
   .0,)
    @unp k Π, n ,  _gri = prim
    TrnsM t=zros( *n , *n ) #Th  irst n points  r  or mploy olks,  n  th  nxt n   r  or unmploy
     or  _in x= :n
        TrnsM t[Int (r s.pol_un [ _in x, ]), _in x]=Π[ , ] #Svings  hoi   or thos moving rom mp->mp
        TrnsM t[Int (r s.pol_un [ _in x, ])+n , _in x]=Π[ , ] #Svings  hoi   or thos moving rom mp->unmp
        TrnsM t[Int (r s.pol_un [ _in x, ]), _in x+n ]=Π[ , ] #Svings  hoi   or thos moving rom unmp->mp
        TrnsM t[Int (r s.pol_un [ _in x, ])+n , _in x+n ]=Π[ , ] #Svings  hoi   or thos moving rom unmp->mp
     n
    Dist=on s( *n )*( /( *n )) #
    Dist_nw= opy(Dist)
    whil  ist_rr>ist_tol
         or i= :    #It r t  until w  r  h  th  st y-st t  istriution
            Dist_nw=TrnsM t*Dist_nw
         n
         ist_rr= s .(mximum(Dist_nw.-Dist))
        Dist= opy(Dist_nw)
     n
    rturn Dist, Dist[ :n ].+Dist[(n + ): *n ]
 n
##############################################################################
```

This code calls the “computation” code above and then prints some figures:

```
#Gtting th Pr l l l R y
    using Distriut #, Sh r Arr ys
    #R -initilizing th workrs
        rmpros(workrs())
          pro s( )
    @ v rywh r using P r m t rs
#Sving D t ils
    in lu ("Comput _Dr t .jl")
#Solv th Mo l
    #initiliz primitiv  n  rsults struts
    @tim out_primitivs, out_rsults = Solv _mo l() #solv th mo l!
    @unp k v l _un , pol_un = out_rsults
    @unp k _gri , n , S_gri = out_primitivs

#Plotting rsults
using Plots, LTXStrings #import th li rri s w  wnt
Plots.plot( _gri , v l _un [:, ], titl="V lu Funtion", l  l="Employ ")
    plot!( _gri , v l _un [:, ], l  l="Un mploy ")
    Plots.s v ig("Vlu_Funtions.png")
    #Plotting Poliy untions
        untion PoliyPolots()
            _h t=
            or  i= :n
                i  _gri [Int .(pol_un [i, ])]<= _gri [i]
                    _h t=[ _gri [ i]];
                    r k
                n
            n
            Plots.plot( _gri , _gri [Int .(pol_un [:, ])], titl="Poliy Funtions", l  l="Employ ")
                plot!( _gri , _gri [Int .(pol_un [:, ])], l  l="Un mploy ")
                plot!( _gri , _gri , l  l="   lin", l g n =:ottomright)
                vlin!( _h t, l  l=L"\h t{ }",olor=:purpl)
                 nnot t !( _h t[ ]-.1 ,  .5, txt("$(roun ( _h t[ ],igits= ))", :purpl, :right,   ))
                Plots.s v ig("Poliy_Funtions.png")
        n
        PoliyPolots()
    #Plotting Distriution
        untion DistPlots()
            TS_Distriution, SS_WlthDistriution=FinDist_ForPlot(out_primitivs,out_rsults)
            MxNonZro=
            ForDistPlot= opy(TS_Distriution)
            or  i= :n
                i  ForDistPlot[i]==
                    ForDistPlot[i]=N N
                n
                i  ForDistPlot[n +i]==
                    ForDistPlot[n +i]=N N
                n
                i  SS_WlthDistriution[i]!=
                    MxNonZro= opy(i)
                n
            n
            Plots.plot( _gri [ :MxNonZro], ForDistPlot[ :MxNonZro], titl="Distriution o  Assts
wh r  q=$(roun(out_rsults.q,igits= ))",
                l  l="Employ ")
                plot!( _gri [ :MxNonZro], ForDistPlot[(n + ):n +MxNonZro], l  l="Un mploy ", xl  l="Assts")
                Plots.s v ig("Distriution.png")
            #Lornz Curv
            n_lornz=
            Lornz=zros(n_lornz, )
                Lornz[:, ]= oll t(r ng ( ,lngth=n_lornz, )) #First olumn is p r nt  o  popultion
                i=
                or  _in x= :n
                    i  sum(SS_WlthDistriution[ : _in x])<=Lornz[i, ]
                        Lornz[i, ]=Lornz[i, ]+TS_Distriution[ _in x]*( _gri [ _in x]+S_gri[ ]) +
                            TS_Distriution[n + _in x]*( _gri [ _in x]+S_gri[ ]) #S on  olumn is umultiv  ssts
                    l s
                        whil  sum(SS_WlthDistriution[ : _in x])>Lornz[i, ]
                            i+=
                            Lornz[i, ]=Lornz[i- , ]+ ; #opy ovr th prvious umultiv  w lth
                        n
                    n
                    Lornz[i, ]=Lornz[i, ]+TS_Distriution[ _in x]*( _gri [ _in x]+S_gri[ ]) +
                        TS_Distriution[n + _in x]*( _gri [ _in x]+S_gri[ ])
                 n
                #Clulting Gini
                Gini=sum(Lornz[:, ].-Lornz[:, ])/(sum(Lornz[:, ].-Lornz[:, ])+sum(Lornz[:, ]))
                #Lornz[:, ]=Lornz[:, ]./Lornz[n_lornz, ] #xprss umultiv ssts s   p r nt g
                #print(Lornz)
                Plots.plot(   *Lornz[:, ],   *Lornz[:, ], titl="Lornz Curv.
 Th Gini Co i i nt is $(roun(Gini,igits= ))",
                 xl  l="% o  Popultion",
                    yl  l="% o  Assts", l g n =:ottomright, l  l="Lornz")
                    plot!(   *Lornz[:, ],   *Lornz[:, ], l  l="Lin  o  Equlity")
                    Plots.s v ig("Lornz.png")
         n
        DistPlots()
#
```