1. Derive the following asymptotic moments associated with $m_3(x)$: mean, variance, first order auto-correlation. Furthermore, compute $\nabla_b g(b_0)$. Which moments are informative for estimating b?

Answer: We obtain the asymptotic moments as follows:

$$\begin{aligned}
\mathbb{E}[x_t] &= \mathbb{E}[\rho_0 x_{t-1} + \epsilon_t] = \rho_0\,\mathbb{E}[x_{t-1}] + \mathbb{E}[\epsilon_t] = \rho_0\,\mathbb{E}[\rho_0 x_{t-2} + \epsilon_{t-1}] = \rho_0^t\,\mathbb{E}[x_0] = 0. \\
\mathbb{E}[(x_t - \mathbb{E}[x_t])^2] &= \mathbb{E}[x_t^2] = \mathbb{E}[(\rho_0 x_{t-1} + \epsilon_t)^2] \\
&= \rho_0^2\,\mathbb{E}[x_{t-1}^2] + 2\rho_0\,\mathbb{E}[x_{t-1}\epsilon_t] + \mathbb{E}[\epsilon_t^2] \\
&= \rho_0^2\,\mathbb{E}[(\rho_0 x_{t-2} + \epsilon_{t-1})^2] + \sigma_0^2 \\
&= \rho_0^{2T}\,\mathbb{E}[x_0^2] + \sigma_0^2 \sum_{i=0}^{T-1} (\rho_0^2)^i \to \frac{\sigma_0^2}{1-\rho_0^2} = \frac{4}{3}. \\
\mathbb{E}[(x_t - \mathbb{E}[x_t])(x_{t-1} - \mathbb{E}[x_{t-1}])] &= \mathbb{E}[x_t x_{t-1}] \\
&= \rho_0\,\mathbb{E}[x_{t-1}^2] + \mathbb{E}[\epsilon_t x_{t-1}] \\
&= \rho_0\,\mathbb{E}[(\rho_0 x_{t-2} + \epsilon_{t-1})^2] \\
&= \rho_0^3\,\mathbb{E}[x_{t-2}^2] + \rho_0\,\mathbb{E}[\epsilon_{t-1}^2] \\
&= \rho_0^{2T-1}\,\mathbb{E}[x_0^2] + \rho_0\sigma_0^2 \sum_{i=0}^{T-2} (\rho_0^2)^i \to \frac{\rho_0\sigma_0^2}{1-\rho_0^2} = \frac{2}{3}
\end{aligned}$$

Moreover, we have that $\nabla_b = \begin{pmatrix} \frac{\partial}{\partial \rho} \\ \frac{\partial}{\partial \sigma^2} \end{pmatrix}$. Evaulating the derivative of $g(\cdot)$ at $b = b_0$ gives us

$$\nabla_b g(b_0) = \begin{bmatrix} 0 & 0 \\ \frac{2\rho_0\sigma_0^2}{(1-\rho_0)^2} & \frac{1}{1-\rho_0^2} \\ \frac{\sigma_0^2(1+2\rho_0^2)}{(1-\rho_0^2)^2} & \frac{\rho_0}{1-\rho_0^2} \end{bmatrix} \overset{\text{true values}}{=\!=\!=\!=} \begin{bmatrix} 0 & 0 \\ 4 & \frac{4}{3} \\ \frac{8}{3} & \frac{2}{3} \end{bmatrix}$$

Both the variance and the first order correlation are informative for estimating b. This is because we can estimate the true parameters given the two moments as follows

$$\rho_0 = \frac{\mathbb{E}[(x_t - \mathbb{E}[x_t])(x_{t-1} - \mathbb{E}[x_{t-1}])]}{\mathbb{E}[(x_t - \mathbb{E}[x_t])^2]} \qquad \sigma^2 = \mathbb{E}[(x_t - \mathbb{E}[x_t])^2] - \frac{\mathbb{E}[(x_t - \mathbb{E}[x_t])(x_{t-1} - \mathbb{E}[x_{t-1}])]}{\mathbb{E}[(x_t - \mathbb{E}[x_t])^2]}$$

■

2. Simulate a series of "true" data of length $T = 200$ using (1). We will use this to compute $M_T(x)$.

3. Set H = 10 and simulate H vectors of length T = 200 random variables e_t from $N(0,1)$. We will use this to compute $M_{TH}(y(b))$. Store these vectors. You will use the same vector of random variables throughout the entire exercise. Since this exercise requires you to estimate σ^2, you want to change the variance of e_t during the estimation. You can simply use σ_{e_t} when the variance is σ^2.

4. We will start by estimating the$l = 2$ vector b for the just identified case where m_2 uses mean and variance. Given what you found in part (1), do you think there will be a problem? Of course, in general we would not know whether this case would be a problem, so hopefully the standard error of the estimate of b as well as the J test will tell us something. Let's see.

Answer: The code for completing this question, as well as the other questions for this problem set, is included in the appendix. From the results of part (1), we know that the mean will be uninformative about the true parameter values, and so in this case only the variance will be useful in identifying the parameter values. One problem that we've notice here and elsewhere is that occasionally the random draws selected by the code will produce issues which make it so that a solution cannot be found. In this problem and for the remainder of the problem set, then, we set a specific random seed value that avoid this issue and produce nice results. This value is 9193, as shown in the code.

(a) $\hat{b}^1_{TH} = [0.7178, 0.855]$. Figure 1 displays a graph of the objective function.

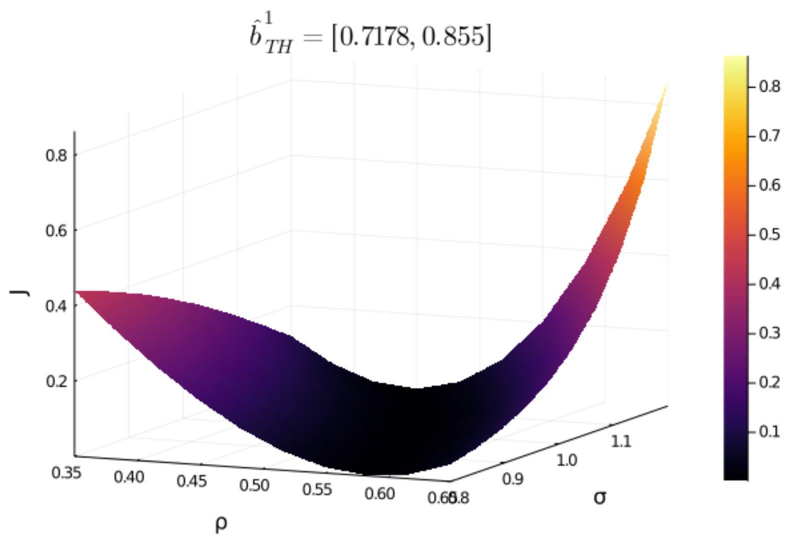

Figure 1: Objective Function for Exercise 4

(b) $\hat{b}^2_{TH} = [0.7179, 0.8549]$ after the Newey-West correction. Figure 2 displays a graph of the objective function.

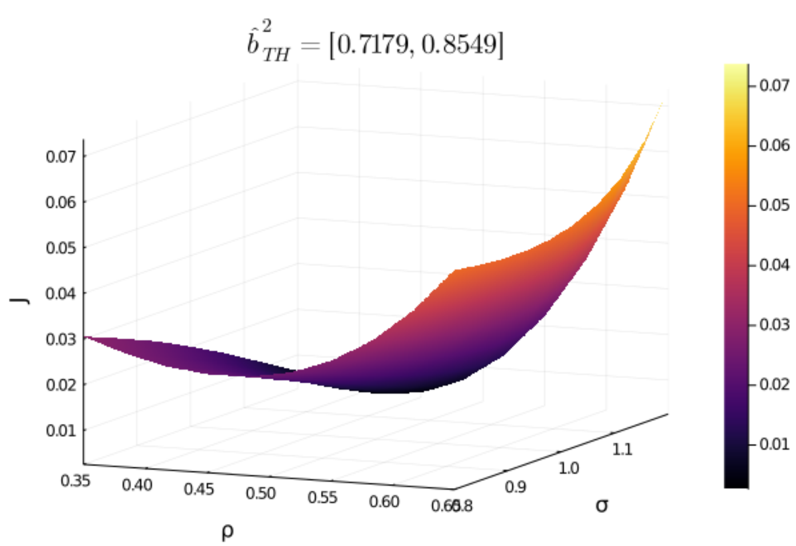

Figure 2: Objective Function for Exercise 4 with the Newey-West Correction

(c)

$$\nabla_b g_T(\hat{b}^2_{TH}) = \begin{bmatrix} 0.22 & 0.06 \\ 3.66 & 3.11 \end{bmatrix}$$

The variance-covariance matrix of $\hat{b}^2_{TH}$ is given by

$$\begin{bmatrix} 0.01 & -4.07e-3 \\ -4.07e-3 & 0.01 \end{bmatrix}$$

The standard errors are given by

$$\begin{bmatrix} 0.08 \\ 0.11 \end{bmatrix}$$

For local identification, it is useful to have the elements of $\nabla_b g_T(\hat{b}^2_{TH})$ to be relatively large. This is because if the distance between the modeled moments and the data moments is changing rapidly around the optimal parameter choices, we can be more confident in the precision with which we've identified the local optimal parameter values.

(d) The J-test value is $1.17e-6$.

∎

5. Next we estimating the $l = 2$ vector b for the just identified case where m_2 uses the variance and autocorrelation. Given what you found in part (1), do you now think there will be a problem? If not, hopefully the standard error of the estimate of b as well as the J test will tell us something. Let's see. For this case, perform steps (a)-(d) above.

Answer: Given our results from part (1), we know that the variance and first order correlation are both informative for estimating b. Therefore, we do not anticipate a major problem here.

(a) $\hat{b}^1_{TH} = [0.4408, 1.0751]$. Figure 3 displays a graph of the objective function.

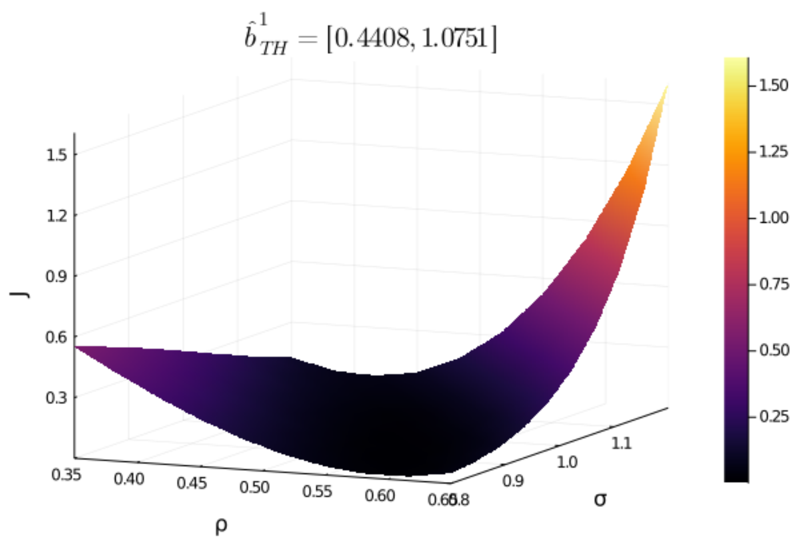

Figure 3: Objective Function for Exercise 5

(b) $\hat{b}^2_{TH} = [0.4409, 1.075]$ after the Newey-West correction. Figure 4 displays a graph of the objective function.

(c)
$$\nabla_b g_T(\hat{b}^2_{TH}) = \begin{bmatrix} 1.44 & 2.78 \\ 1.83 & 1.22 \end{bmatrix}$$

The variance-covariance matrix of $\hat{b}^2_{TH}$ is given by

$$\begin{bmatrix} 7.43e-5 & -1.02e-3 \\ -1.02e-3 & 0.01 \end{bmatrix}$$

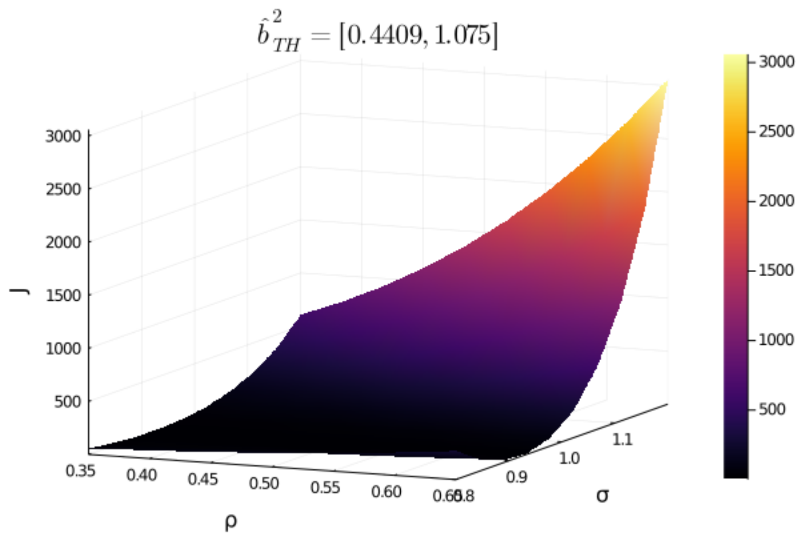

Figure 4: Objective Function for Exercise 5 with the Newey-West Correction

The standard errors are given by

$$\begin{bmatrix} 0.01 \\ 0.12 \end{bmatrix}$$

For local identification, it is useful to have the elements of $\nabla_b g_T(\hat{b}^2_{TH})$ to be relatively large. This is because if the distance between the modeled moments and the data moments is changing rapidly around the optimal parameter choices, we can be more confident in the precision with which we've identified the local optimal parameter values.

(d) The J-test value is $2.29e - 7$.

■

6. Next, we will consider the overidentified case where m3 uses the mean, variance and autocorrelation. Let's see. For this case, perform steps (a)-(d) above. Furthermore, bootstrap the the finite sample distribution of the estimators using the following algorithm:

   i. Draw ϵ_t and e^h_t from $N(0,1)$ for $t = 1, 2, \ldots, T$ and $h = 1, 2, \ldots, H$. Compute $(\hat{b}^1_{TH}), \hat{b}^2_{TH}$ as described.

   ii. Repeat (e) using another seed.

Answer: .

(a) $\hat{b}^1_{TH} = [0.4414, 1.0748]$. Figure 5 displays a graph of the objective function.

(b) $\hat{b}^2_{TH} = [0.3894, 0.4101]$ after the Newey-West correction. Figure 6 displays a graph of the objective function.

(c)
$$\nabla_b g_T(\hat{b}^2_{TH}) = \begin{bmatrix} 0.02 & 0.03 \\ 0.14 & 0.92 \\ 0.24 & 0.34 \end{bmatrix}$$

The variance-covariance matrix of $\hat{b}^2_{TH}$ is given by

$$\begin{bmatrix} 3.58e - 3 & 0.02 \\ 0.02 & 0.07 \end{bmatrix}$$

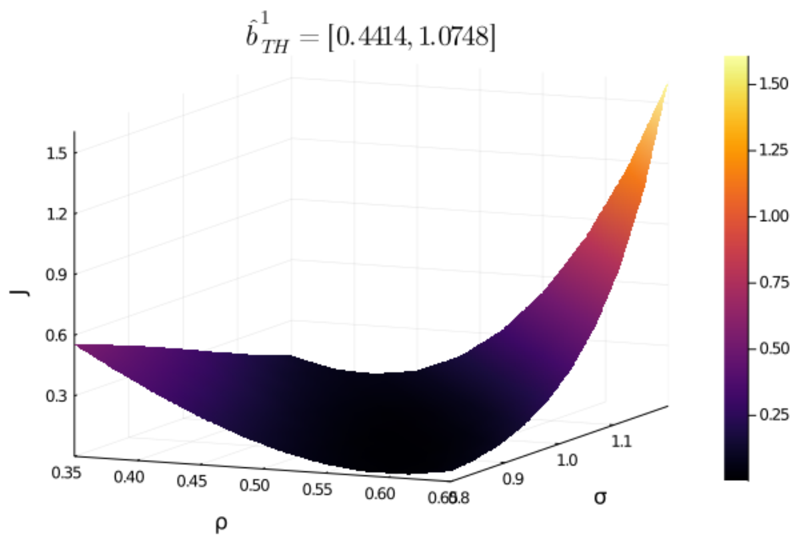

Figure 5: Objective Function for Exercise 6

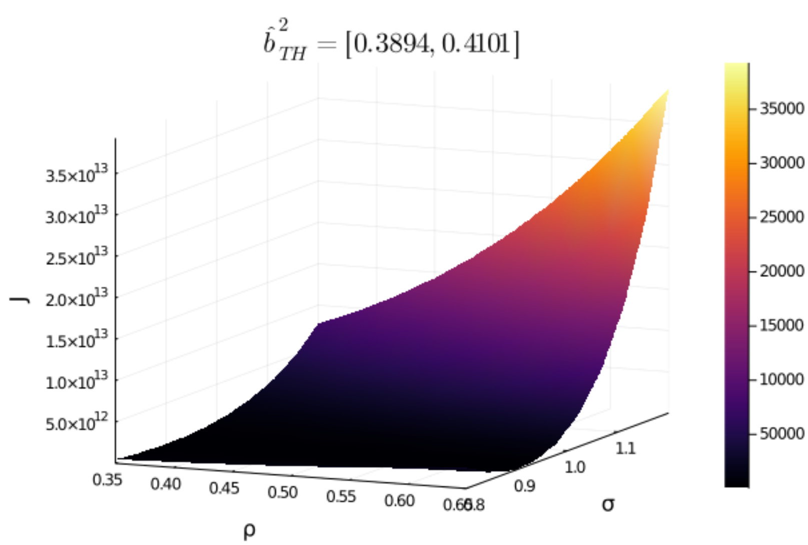

Figure 6: Objective Function for Exercise 6 with the Newey-West Correction

The standard errors are given by

$$\begin{bmatrix} 0.06 \\ 0.27 \end{bmatrix}$$

For local identification, it is useful to have the elements of $\nabla_b g_T(\hat{b}^2_{TH})$ to be relatively large. This is because if the distance between the modeled moments and the data moments is changing rapidly around the optimal parameter choices, we can be more confident in the precision with which we've identified the local optimal parameter values.

(d) The J-test value is 27.19.

Bootstrapping: Figures 7 and 8 display the bootstrapped densities of parameter estimates for $\hat{b}^1_{TH}$ and $\hat{b}^2_{TH}$, respectively. The density of $\hat{b}^2_{TH}$ is calculated using only those random seeds which allowed for a solution to be found, as shown in the code. ∎

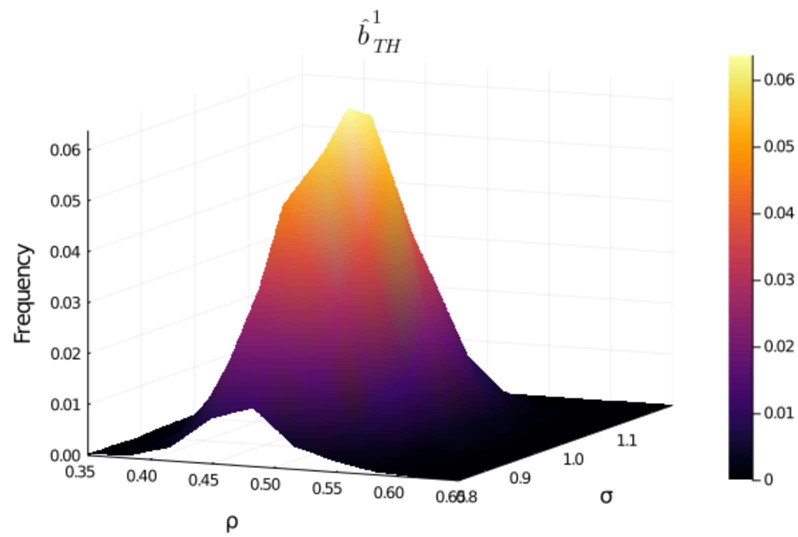

Figure 7: Bootstrapped Density of Estimates for $\hat{b}^1_{TH}$

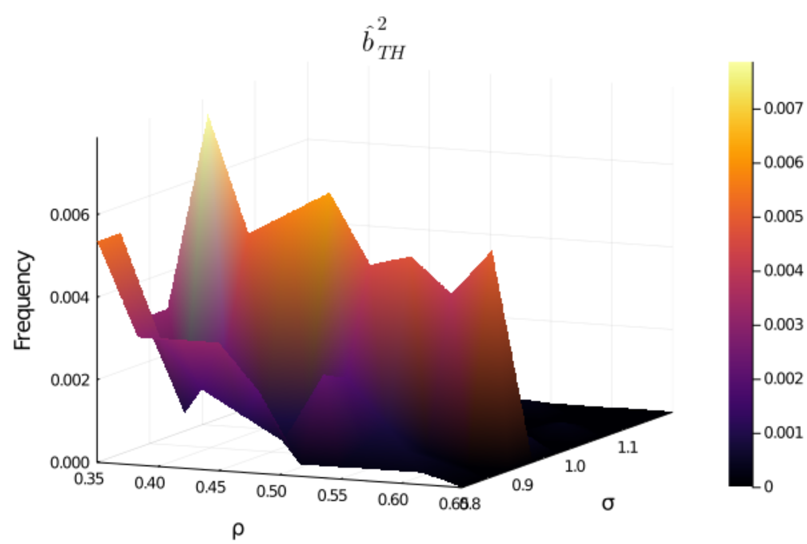

Figure 8: Bootstrapped Density of Estimates for $\hat{b}^2_{TH}$

Appendix

The first codefile named "RunCode.jl" runs the code.

```
# theme(:juno)
# default(fontfamily="Computer Modern", framestyle=:box)
# Load the Code
    include("./Code.jl");

#Run The Code
    #All exercises are done at once like this since they are supposed to all use
        #the same e draw
    #Note that the results are highly dependent upon the epsilon draw,
    #estpecially for exercise 6. Oftentimes standard errors cannot be
    #calculated. Setting UseRandomSeed to false gives a consistent draw where this
    #is not the case. Refresh Julia after you do this to reset the seed somewhere
    #more truly random
    StepsAThroughD(UseRandomSeed=false)
        #StepsAThroughD(UseRandomSeed=true)
```

The second codefile named "Code.jl" contains the relevant functions.

```
using Parameters, Optim, Distributions, LinearAlgebra, Plots, LaTeXStrings,
    Random

@with_kw mutable struct  Primitives
    T       ::Int64      #Time Series Length
    H       ::Int64      #Number of Simulations
    ρ0      ::Float64           = .5
    σ0      ::Float64           = 1
    x0      ::Float64           = 0
    gpoints ::Int64             = 10
    ρgrid   ::Array{Float64}    = collect(range(0.35, length = gpoints, stop = 0.65))
    σgrid   ::Array{Float64}    = collect(range(0.8, length = gpoints, stop = 1.2))
    iT      ::Int64             = 4
end # Primitives
@with_kw mutable struct  Results
    bHat1TH       ::Array{Float64} = [0,0]
    bHat2TH       ::Array{Float64} = [0,0]
    b_dist        ::Array{Float64} = zeros(5000,2,2) #Iteration, bhat1 or 2, then each parameter
    e             ::Array{Float64}
    JTest         ::Float64        =100
    td            ::Array{Float64} = [0, 0] #The true data
end # Primitives

#=
function GetMoments(ρ, σ, T, H)

    Random.seed!(100)
    ϵ = zeros(T,H)
    for hi=1:H
        ϵ[:,hi]=rand(Normal(0,σ),T)
    end

    xₜ = zeros(T+1, H)
    for t=2:(T+1)
        xₜ[t,:] = ρ₀ .* xₜ[t-1,:] .+ ϵ[t-1,:]
    end
    x_t_1 = xₜ[1:end-1,:]
    xₜ = xₜ[2:end,:]
    barx = mean(xₜ, dims=1)

    m1 = mean(mean(xₜ, dims=1), dims = 2)[1]
    m2 = mean(mean((xₜ .- mean(xₜ, dims =1)).^2, dims =1), dims = 2)[1]
    m3 = mean((xₜ.- barx) .* (x_t_1 .- barx))[1]

    m = [m1, m2, m3]

    return m

end
=#
```

```
function TrueData(prim)
    @unpack T, σ0, x0, ρ0 = prim
    ϵ = rand(Normal(0,sqrt(σ0)), T)
    xt=zeros(T+1)
    xt[1]=x0
    for t=2:(T+1)
        xt[t]=ρ0*xt[t-1]+ϵ[t-1]
    end
    xt=xt[2:end]
    return xt
end
function eDrawsForModel(prim;URS=false)
    if ~URS
        Random.seed!(9193) #543
    end
    @unpack H,T=prim
    e=zeros(T,H)
    for hi=1:H
        e[:,hi]=rand(Normal(0,1),T)
    end
    return e
end
function ModelData(prim,e,b)
    @unpack T, σ0, x0, ρ0, H = prim
    #if b[2]<0
    #    #Here J is set to return a very high value so this will not be picked
    #    return zeros(T,H)
    #else
        yt=zeros(T+1,H)
        for hi=1:H
            for t=2:(T+1)
                # yt[t,hi]=b[1]*yt[t-1,hi]+sqrt(b[2])*e[t-1,hi]
                yt[t,hi]=b[1]*yt[t-1,hi]+ b[2]*e[t-1,hi] # Removed sqrt if sigma is being used
            end
        end
        yt=yt[2:end,:]
        return yt
    #end
end
function FindM2_MeanVar(data)
    M=zeros(2)
    if size(data,2)>1 #We are dealing with the Model Data
        Hmeans=sum(data,dims=1)./size(data,1)
        M[1]=sum(Hmeans)/size(data,2)
        #data.-Hmeans does the subtraction for each row
        Hvars=sum((data.-Hmeans).^2,dims=1)./size(data,1)
        M[2]=sum(Hvars)/size(data,2)
    else
        M[1]=sum(data)/length(data)
        M[2]=sum((data.-M[1]).^2)/length(data)
    end
    return M
end
function m_MeanVar(x,ind, mean,prim)
    return [x[ind]  (x[ind]-mean)^2]
end
function FindM2_VarCoVar(data)
    M=zeros(2)
    if size(data,2)>1 #We are dealing with the Model Data
        Hmeans=sum(data,dims=1)./size(data,1)
        #data.-Hmeans does the subtraction for each row
        Hvars=sum((data.-Hmeans).^2,dims=1)./size(data,1)
        M[1]=sum(Hvars)/size(data,2)
        HCovars=sum((data[2:end,:].-Hmeans).*(data[1:(end-1),:].-Hmeans),dims=1)./(size(data,1)-1)
        M[2]=sum(HCovars)/size(data,2)
    else #We are dealing with the true data
        Mean=sum(data)/length(data)
        M[1]=sum((data.-Mean).^2)/length(data)
        M[2]=sum((data[2:end].-Mean).*(data[1:(end-1)].-Mean))/(length(data)-1)
    end
    return M
end
function m_VarCovar(x,ind,mean,prim)
    @unpack x0 = prim
    if ind - 1 == 0
        return [(x[ind]-mean)^2     (x[ind]-mean)*(x0-mean)]
    else
        return [(x[ind]-mean)^2     (x[ind]-mean)*(x[ind-1]-mean)]
    end
end
function FindM3(data)
```

```
    M=zeros(3)
    if size(data,2)>1 #We are dealing with the Model Data
        Hmeans=sum(data,dims=1)./size(data,1)
        M[1]=sum(Hmeans)/size(data,2)
        #data.-Hmeans does the subtraction for each row
        Hvars=sum((data.-Hmeans).^2,dims=1)./size(data,1)
        M[2]=sum(Hvars)/size(data,2)
        HCovars=sum((data[2:end,:].-Hmeans).*(data[1:(end-1),:].-Hmeans),dims=1)./(size(data,1)-1)
        M[3]=sum(HCovars)/size(data,2)
    else #We are dealing with the true data
        Mean=sum(data)/length(data)
        M[1]=Mean;
        M[2]=sum((data.-Mean).^2)/length(data)
        M[3]=sum((data[2:end].-Mean).*(data[1:(end-1)].-Mean))/(length(data)-1)
    end
    return M
end
function m3(x,ind,mean,prim)
    @unpack x0 = prim
    if ind - 1 == 0
        return [x[ind]   (x[ind]-mean)^2      (x[ind]-mean)*(x0-mean)] #We can replace this by x0
    else
        return [x[ind]   (x[ind]-mean)^2      (x[ind]-mean)*(x[ind-1]-mean)]
    end
end
function J(g,W,b) #The objective function
    if b[2]<0 #We shouldn't allow there to be a negative variance term
        return 1e25
    else
        return transpose(g)*W*g
    end
end

function GraphAndFindbHat(W,prim,res,FindM,Exercise; NeweyWest=false,Graph=true)
    @unpack ρgrid,σgrid,gpoints=prim
    @unpack e, td = res
    Jgrid=zeros(gpoints,gpoints)
    for ρi=1:gpoints,σi=1:gpoints
        md=ModelData(prim,e, [ρgrid[ρi] σgrid[σi]])
        Jgrid[ρi,σi]=J(FindM(td).-FindM(md),W,[ρgrid[ρi] σgrid[σi]])
    end
    Solution=optimize(b->J(FindM(td).-FindM(ModelData(prim,e,b)),W,b),[.3 1.2], NelderMead())
    # Solution=optimize(b->J(FindM(td).-FindM(ModelData(prim,e,b)),W,b),
    #     [.3 1.2],[ρgrid[1] σgrid[1]],[ρgrid[end] σgrid[end]], NelderMead()) #I want to restrict the parameter s
    bHat=Solution.minimizer
    if Graph
    # println("The minimizer is ", bHat)
    if NeweyWest
        plot(ρgrid,σgrid,Jgrid, st=:surface,
            title=L"\hat{b}^{2}_{TH}=[%$(round(bHat[1],digits=4)) , %$(round(bHat[2],digits=4)) ]", xlabel =
"ρ",
            ylabel = "σ", zlabel ="J")
        # savefig("PS7\\Figures\\Exercise$(Exercise)NeweyWestCorrection.png")
        savefig("PS7/Figures/Exercise$(Exercise)NeweyWestCorrection.png")
    else
        plot(ρgrid,σgrid,Jgrid, st=:surface,
            title=L"\hat{b}^{1}_{TH}=[%$(round(bHat[1],digits=4)) , %$(round(bHat[2],digits=4)) ]", xlabel =
"ρ",
            ylabel = "σ", zlabel ="J")
        # savefig("PS7\\Figures\\Exercise$(Exercise).png")
        savefig("PS7/Figures/Exercise$(Exercise).png")
    end
    end #if Graph statement
    return bHat
end

function NeweyWest(prim::Primitives,simdata::Array{Float64},
        m::Function, MTH::Array{Float64})
    @unpack iT,H,T = prim
    #Can either use the simulation-H specific mean
        Hmeans=sum(simdata,dims=1)./size(simdata,1)
    #Or can Try using the overall mean:
        #Hmeans=ones(size(simdata,1))*(sum(simdata)/(T*H))
    #Defining Γ function
        function ΓjTH(j)
            out=zeros(length(MTH),length(MTH))
            for hi=1:H, ti=(j+1):T
                #Necessary to add one twice above because one moment includes a lag ( I removed the twice additio
                # changed the m function to consider cases)
                out+=(m(simdata[:,hi],ti,Hmeans[hi],prim).-MTH)*
                    transpose(m(simdata[:,hi],ti-j,Hmeans[hi],prim).-MTH)
```

```
            end
            return (1/(T*H))*out
        end
    #Finding STH
        SyTH=ΓjTH(0)
        for ji=1:iT
            SyTH+=(1-ji/(iT+1))*(ΓjTH(ji)+transpose(ΓjTH(ji)))
        end
        STH=(1+1/H)*SyTH
    #Returning WStar
        return inv(STH)
end

function Find∇g(res,prim, FindM; s=1e-15)
    @unpack e, bHat2TH = res
    #∂g∂ρ= -(FindM(ModelData(prim,e,bHat2TH)).-FindM(ModelData(prim,e,bHat2TH-[s 0]))))./s
    #∂g∂σ= -(FindM(ModelData(prim,e,bHat2TH)).-FindM(ModelData(prim,e,bHat2TH-[0 s]))))./s
    #Alternate Derivative calculation for (hopefully) improved accuracy from
    #https://en.wikipedia.org/wiki/Numerical_differentiation#
    ∂g∂ρ= (FindM(ModelData(prim,e,bHat2TH+[s 0])).-FindM(ModelData(prim,e,bHat2TH-[s 0])))./(2*s)
    ∂g∂σ= (FindM(ModelData(prim,e,bHat2TH+[0 s])).-FindM(ModelData(prim,e,bHat2TH-[0 s])))./(2*s)
    return [∂g∂ρ ∂g∂σ]
end

function StepsAThroughD(;T=200,H=10,UseRandomSeed=false)
    prim=Primitives(T=T,H=H)
    res=Results(e=eDrawsForModel(prim,URS=UseRandomSeed))
    res.td=TrueData(prim)
    for Exercise=4:6
        if Exercise==4
            FindM=FindM2_MeanVar
            m=m_MeanVar
        elseif Exercise==5
            FindM=FindM2_VarCoVar
            m=m_VarCovar
        elseif Exercise==6
            FindM=FindM3
            m=m3
        end

        #Function for parts a and b
        #Part a: Graph in three Dimensions
            res.bHat1TH=GraphAndFindbHat(I,prim,res,FindM,Exercise)
            print("
________________________________________________________________________________
                    Results for Exercise $(Exercise)
________________________________________________________________________________\n")
            println("The estimate of b using W = I is ", res.bHat1TH,".")

            # ∇g1=Find∇g(res,prim,FindM)
            #     print("∇g= \n\n")
            #     display(∇g1)
            # StdErrorsbHat1TH = sqrt.(diag((1/prim.T)*inv(transpose(∇g1)*I*∇g1)))
            # print("\n The Standard errors are given by \n\n")
            #     display(StdErrorsbHat1TH)
        #Part b: Use NeweyWest to update your guess of bHat
            md_bHat1TH=ModelData(prim,res.e, res.bHat1TH)
            WStar=NeweyWest(prim,ModelData(prim,res.e, md_bHat1TH),
                    m, FindM(md_bHat1TH))
            res.bHat2TH=GraphAndFindbHat(WStar,prim,res,FindM,Exercise,
                NeweyWest=true)
            println("The estimate of b using Wstar is ", res.bHat2TH,".")

        #Part c
            ∇g=Find∇g(res,prim,FindM)
                print("∇g= \n\n")
                display(∇g)
            VarCovarbHat2TH=(1/prim.T)*inv(transpose(∇g)*WStar*∇g)
                print("\n The variance-covariance matrix for bHat2TH is given by \n\n")
                display(VarCovarbHat2TH)
            try
                StdErrorsbHat2TH=sqrt.(diag(VarCovarbHat2TH))
                    print("\n The Standard errors are given by \n\n")
                    display(StdErrorsbHat2TH)
            catch
                print("The Standard Errors cannot be found for this epsilon draw")
            end
        #Part d: Computing the Value of the J test
            res.JTest=prim.T*(prim.H/(1+prim.H))*
                J(FindM(res.td).-FindM(ModelData(prim,res.e,res.bHat2TH)),
                    WStar,res.bHat2TH)
```

```
                println("\n The J-Test is $(res.JTest)")
        #Bootstrapping for Exercise 6
        if Exercise==6
            println("\n\n Beginning Bootstrapping")
            @unpack ρgrid,σgrid,gpoints=prim
            Density=zeros(2,gpoints,gpoints)
            FailedDraws=0
            for iter=1:size(res.b_dist,1)
                res=Results(e=eDrawsForModel(prim,URS=true))
                res.td=TrueData(prim)
                #bHat1TH
                    res.b_dist[iter,1,:]=GraphAndFindbHat(I,prim,res,FindM,Exercise, Graph=false)
                #bHat2TH
                    md_bHat1TH=ModelData(prim,res.e, res.b_dist[iter,1,:])
                    try
                        WStar=NeweyWest(prim,ModelData(prim,res.e, md_bHat1TH),
                                m, FindM(md_bHat1TH))
                        res.b_dist[iter,2,:]=GraphAndFindbHat(WStar,prim,res,FindM,Exercise,
                            NeweyWest=true, Graph=false)
                    catch
                        res.b_dist[iter,2,:]=[-Inf, -Inf]
                        FailedDraws+=1
                    end
                for b_type=1:2
                    if res.b_dist[iter,b_type,1]<= ρgrid[1] ||
                        res.b_dist[iter,b_type,1]>= ρgrid[gpoints] ||
                            res.b_dist[iter,b_type,2]>= σgrid[gpoints] ||
                            res.b_dist[iter,b_type,2]<= σgrid[1]
                            #Out of range, do nothing
                    else
                        for ρi=1:gpoints,σi=1:gpoints
                            if (ρgrid[ρi+1]>=res.b_dist[iter,b_type,1]>=ρgrid[ρi] &&
                                σgrid[σi+1]>=res.b_dist[iter,b_type,2]>=σgrid[σi])
                                Density[b_type,ρi,σi]+=1
                                break
                            end
                        end
                    end
                end
                if iter % 250 ==0
                    print("\n Iteration $(iter) of Bootstrapping. Cumulative failed draws=$(FailedDraws)")
                end
            end #End bootstrapping with iter
            Density[1,:,:]=Density[1,:,:]./size(res.b_dist,1) #Divide by the number of iterations
            Density[2,:,:]=Density[2,:,:]./(size(res.b_dist,1)-FailedDraws)
            for b_type=1:2
                plot(ρgrid,σgrid,Density[b_type,:,:], st=:surface,
                    title=L"\hat{b}^{%$(b_type)}_{TH}", xlabel = "ρ",
                    ylabel = "σ", zlabel ="Frequency")
                savefig("PS7/Figures/Exercise$(Exercise)Bootstrapping$(b_type).png")
            end
        end
    end #Exercise Loop
end #End Function StepsAThroughD
```