Structural Estimation Pset 2

Timothy Schwieg

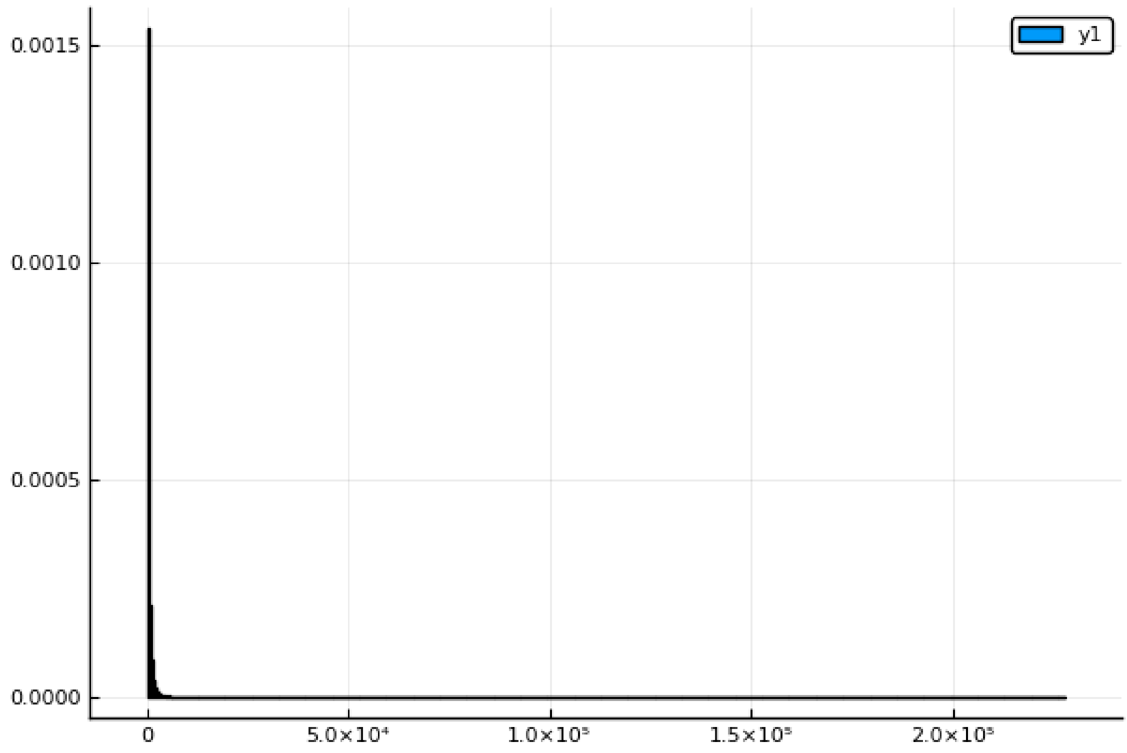
1 Question One

1.1 a

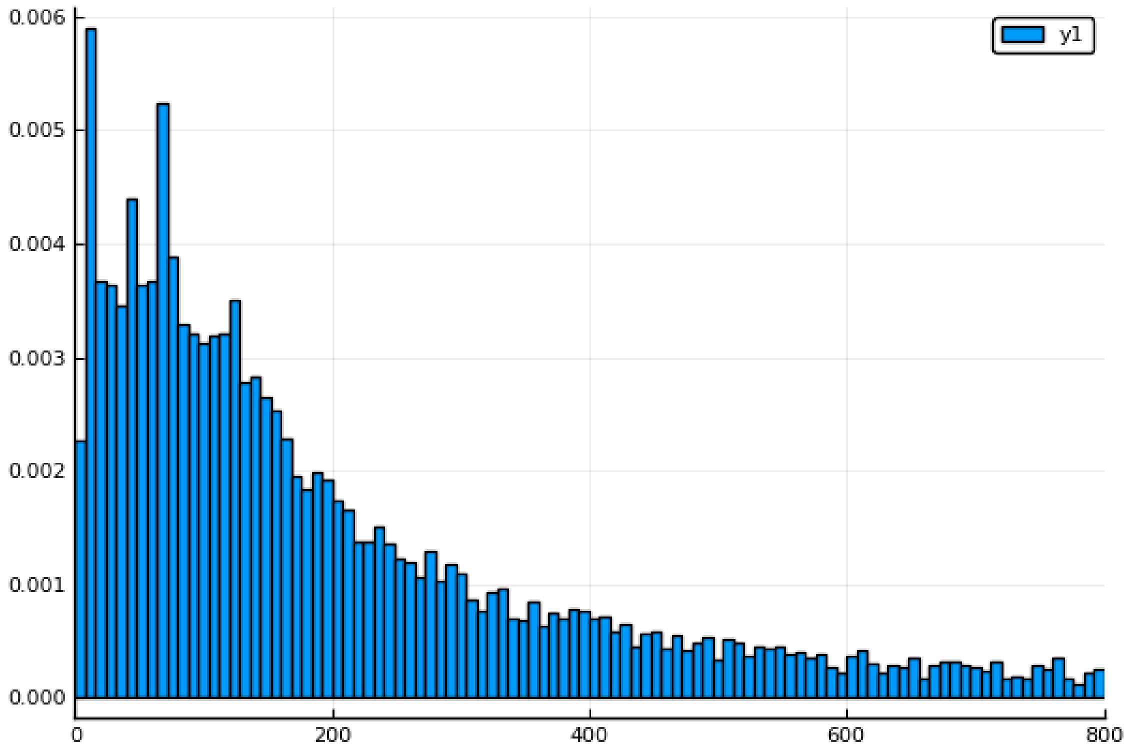
```
#healthClaims = CSV.read( "clms.txt", header=[:A] )
healthClaims = DataFrame(load("clms.csv", header_exists=false, colnames=["A"]))
#describe( healthClaims )

#println( "Standard Deviation: ", std(healthClaims[:A]))

results = [["mean", "min", "median", "max", "StdDev"] [mean(healthClaims[:A]), minimum(healthClaims[:A]),
    median(healthClaims[:A]), maximum(healthClaims[:A]), std(healthClaims[:A])]]
```

mean	720.2779753272437
min	0.01
median	172.21
max	227967.25
StdDev	3972.850824119446

```
histogram( healthClaims[:A], bins=1000, normalize = true)
savefig("histOne.png")
```




```
truncatedHealthClaims = healthClaims[healthClaims[:A] .<= 800, 1]

# Doing this will make them sum to one
#histogram( truncatedHealthClaims, bins = 100, normalize = true)
#We force all bins to have length 8, and allow for 100 of them.
histogram( healthClaims[:A], bins=0:8:800, normalize=true, xlims=(0,800))
savefig("histTwo.png")
```



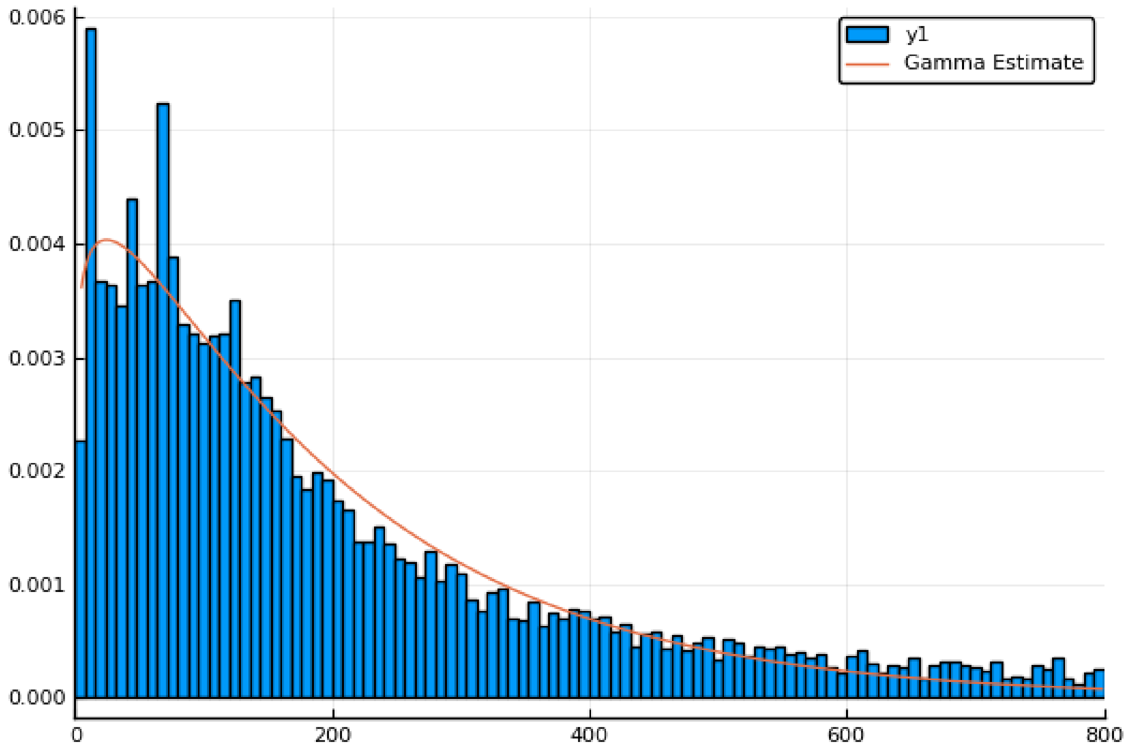
1.2 b

```
function GammaLogLikelihood( x::Vector{Float64}, α::Float64, β::Float64)
    #Yes I know I could get this using Distributions.jl which could
    #even do the MLE estimate But thats pretty much cheating, and
    #gamma is in the exponential family so using Newton's method will
    #cause no issues.

    #Pdf is: $\frac{1}{\Gamma(\alpha)\beta^\alpha} x^{\alpha-1} \exp\left(-\frac{x}{\beta}\right)$
    #Log-likelihood is: $-\alpha \log(\beta) - \log(\Gamma(\alpha)) + (\alpha - 1)\log x - \frac{x}{\beta}$

    return -α*log( β) - lgamma(α) + (α - 1)*mean(log.(x)) - mean(x) / β
end

function GammaGradient( x::Vector{Float64}, α::Float64, β::Float64)
    delA = -log(β) - digamma(α) + mean(log.(x))
    #delB = mean(x) / β - α
    delB = mean(x) / β^2 - α / β
    return [delA,delB]
end

function GammaHessian( x::Vector{Float64}, α::Float64, β::Float64)
    delAA = -trigamma(α)
    delAB = -1 / β
    delBB =( α / (β*β)) - ((2* mean(x)) / (β*β*β))
    return [delAA delAB; delAB delBB]
end

function GammaPDF( α::Float64, β::Float64, x::Float64)
    return  (1 / (gamma(α)*β^α))*x^(α-1)*exp( -x/β)
end

function EstimateGammaParameters( data::Vector{Float64}, guess::Vector{Float64}, gradientFun, hessianFun)

```

```julia
    θ = guess
    tol = 1e-10
    maxLoops = 100

    grad = gradientFun( data, θ... )
    hess = hessianFun( data, θ... )

    loopCounter = 0
    while( loopCounter < maxLoops && norm(grad) >= tol)
        θ = θ - hess \ grad
        grad = gradientFun( data, θ... )
        hess = hessianFun( data, θ... )

        loopCounter += 1
        # println( norm(grad))
        # println( θ)
        # println( " ")
    end
    #println( loopCounter)
    return θ
end
healthCosts = convert(  Vector{Float64}, truncatedHealthClaims )#healthClaims[:A] )

β₀ =  var(healthCosts) / mean(healthCosts)
α₀ = mean(healthCosts) / β₀

(Gamma_α̂, Gamma_β̂) = EstimateGammaParameters( healthCosts, [α₀, β₀], GammaGradient, GammaHessian)

likelihood = GammaLogLikelihood(  healthCosts, Gamma_α̂, Gamma_β̂)

result = [["\$\\est{\\alpha}\$: ", "\$\\est{\\beta}\$: ", "Likelihood: " ] [ Gamma_α̂,  Gamma_β̂, likelihood]]
```

$\widehat{\alpha}_n$:	1.1397564780585858
$\widehat{\beta}_n$:	174.8688733959653
Likelihood:	-6.28964508639924

```julia
histogram( healthClaims[:A], bins=0:8:800, normalize=true, xlims=(0,800))
pdfXVal = range( minimum(truncatedHealthClaims)+5, maximum(truncatedHealthClaims))
#pdfXVal = linspace( minimum(truncatedHealthClaims), maximum(truncatedHealthClaims))
pdfYVal = [GammaPDF( Gamma_α̂, Gamma_β̂, x ) for x in pdfXVal]


plot!( pdfXVal, pdfYVal, label="Gamma Estimate" )
savefig("histPDF_Gamma.png")
```



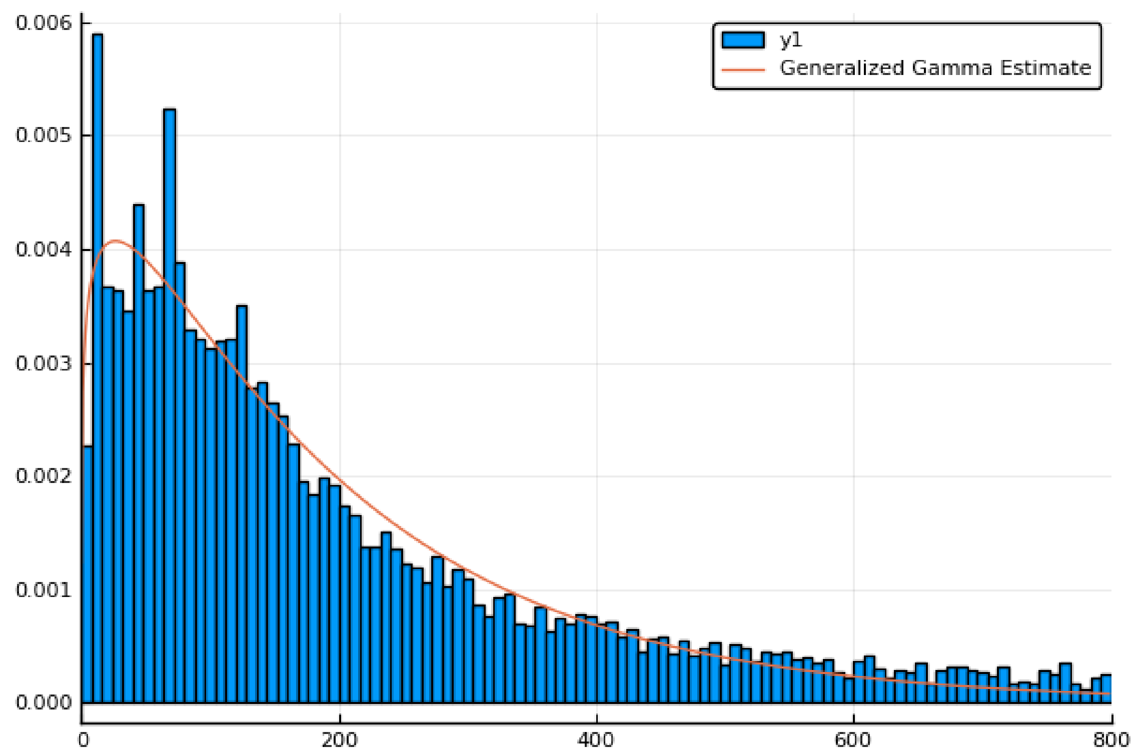
2 c

```
# (GG):  $f(x;\alpha,\beta,m) = \frac{m}{\beta^\alpha \Gamma\left(\frac{\alpha}{m}\right)} x^{\alpha-1} e^{-\left(\frac{x}{\beta}\right)^m}, \quad x \in [0,\infty),\ \alpha,\beta,m > 0$
function GGammaPDF( α::Float64, β::Float64, m::Float64, x::Float64)
    return ( (m / β^α) * x^(α-1) * exp( - (x / β)^m) ) / gamma( α / m)
end


function GGammaLikelihood( x::Vector{Float64}, α::Real, β::Real, m::Real)
    return log(m) - α*log(β) + (α - 1)*mean(log.(x)) - mean( (x ./ β).^m  ) - lgamma( α / m )
end

function EstimateGG( data::Vector{Float64}, guess::Vector{Float64})
    #To hard enforce that all of our parameters are positive, we
    #exponentiate them
    θ = log.(guess)
    fun(x::Vector) = -GGammaLikelihood( data, exp.(x)... )



    result = optimize(fun, θ, ConjugateGradient(), autodiff=:forward)
end

sln = EstimateGG( healthCosts, [Gamma_α̂, Gamma_β, 1.0])

GG_α̂ = exp(sln.minimizer[1])
GG_β = exp(sln.minimizer[2])
GG_m̂ = exp(sln.minimizer[3])
GG_LogLikelihood = -sln.minimum

println( "GG α̂ = ", GG_α̂)
println( "GG β = ", GG_β )
println( "GG m̂ = ", GG_m̂ )
println( "Likelihood Value: ", GG_LogLikelihood )
```

```julia

result = [["GG \$\\est{\\alpha}\$: ", "GG \$\\est{\\beta}\$: ", "GG \$\\est{m}\$: ","GG Likelihood: " ] [ GG_α̂,  GG_β, GG_m̂, GG_LogLikelihood]]
```

GG $\widehat{\alpha}_n$:	1.1755020098846642
GG $\widehat{\beta}_n$:	156.18446475134172
GG $\widehat{m}_n$:	0.9498167064643459
GG Likelihood:	-6.289560051458711

```julia
histogram( healthClaims[:A], bins=0:8:800, normalize=true, xlims=(0,800))
pdfXVal = range( minimum(truncatedHealthClaims), maximum(truncatedHealthClaims))
#pdfXVal = linspace( minimum(truncatedHealthClaims), maximum(truncatedHealthClaims))
pdfYVal = [GGammaPDF( GG_α̂, GG_β, GG_m̂, x ) for x in pdfXVal]

plot!( pdfXVal, pdfYVal, label="Generalized Gamma Estimate" )
savefig( "histPDF_GG.png" )
```



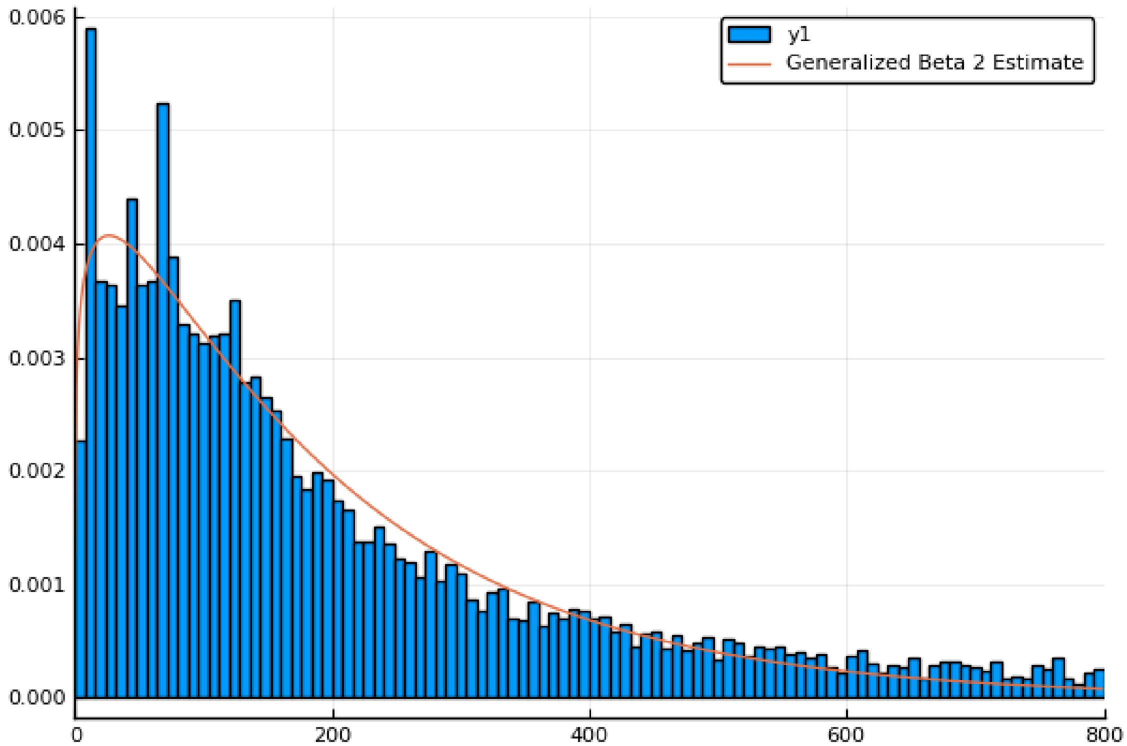
2.1 d

```julia
function GBetaTwoPDF( x::Float64, a::Real, b::Real, p::Real, q::Real)
    #We require all parameters to be positive, so abs(a) = a
    return a*x^(a*p -1) / (b^(a*p) *beta(p,q)*(1+(x/b)^a)^(p+q))
end

#$GG(\alpha,\beta,m) = \lim_{q\to\infty} GB2\left(a=m, b=q^{1/m}\beta, p=\frac{\alpha}{m}, q\right)$

function GBetaTwoLikelihood( x::Vector{Float64}, a::Real, b::Real, p::Real, q::Real)
    return log( a) + (a*p -1)*mean(log.(x)) - (a*p)*log(b) - log(beta(p,q)) - (p+q)*mean( log.( 1 .+(x ./ b).^a ))
end

```

```julia
function EstimateGBetaTwo( data::Vector{Float64}, guess::Vector{Float64})
      #To hard enforce that all of our parameters are positive, we
      #exponentiate them
    θ = log.(guess)
    #θ = guess
    fun(x::Vector) = -GBetaTwoLikelihood( data, exp.(x)... )


    #This guy is being fickle, and Newton() would not converge
    #LBFGS converges, but to a higher value than Newton()
    result = optimize(fun, θ, NewtonTrustRegion(), autodiff=:forward, Optim.Options(iterations=2000) )
end
sln = EstimateGBetaTwo( healthCosts, [GG_m̂, 10000^(1 / GG_m̂) * GG_β̂, GG_α̂ / GG_m̂, 10000])

GB2_α̂ = exp( sln.minimizer[1])
GB2_β̂ = exp( sln.minimizer[2])
GB2_p̂ = exp( sln.minimizer[3])
GB2_q̂ = exp( sln.minimizer[4])
GB2_LogLikelihood = -sln.minimum

result = [["GB2 \$\\est{\\alpha}\$: ", "GB2 \$\\est{\\beta}\$: ", "GB2 \$\\est{p}\$: ","GB2 \$\\est{q}\$: ","GB2 Likelihood: " ] [GB2_α̂, GB2_β̂,  GB2_p̂,  GB2_q̂, -sln.minimum]]
```

GB2 $\widehat{\alpha}_n$:	0.9498191942062975
GB2 $\widehat{\beta}_n$:	1.016136547549504 (9)
GB2 $\widehat{p}_n$:	1.2376044907191777
GB2 $\widehat{q}_n$:	2.960836571954795 (6)
GB2 Likelihood:	-6.289560054045967

```julia
histogram( healthClaims[:A], bins=0:8:800, normalize=true, xlims=(0,800))
pdfXVal = range( minimum(truncatedHealthClaims), maximum(truncatedHealthClaims))
#pdfXVal = linspace( minimum(truncatedHealthClaims), maximum(truncatedHealthClaims))
pdfYVal = [GBetaTwoPDF( x, GB2_α̂, GB2_β̂, GB2_p̂, GB2_q̂ ) for x in pdfXVal]

plot!( pdfXVal, pdfYVal, label="Generalized Beta 2 Estimate" )
savefig( "histPDF_GB2.png" )
```



2.2 e

Since the likelihood function values at the optimum for parts (b) and (c) are the constrained maximum likelihood estimators, the likelihood ratio test is simply:

$$2\left(f(\widehat{\theta}_n - \tilde{\theta}_n)\right) \sim \chi^2_p$$

Where p is the number of constraints in the estimation procedure.

```
# Gamma Has Two restrictions
tStatGamma = 2*(GB2_LogLikelihood - likelihood)
# Generalized Gamma Has One Restriction
tStatGG = 2*(GB2_LogLikelihood - GG_LogLikelihood)

results = [["", "Gamma", "Generalized Gamma"] [ "\$\\chi^{2}\$", tStatGamma, tStatGG] ["p-value",
↪  cdf(Chisq(2),tStatGamma), cdf( Chisq(1),tStatGG) ] ]
```

	χ^2	p-value
Gamma	0.00017006408454989241	8.502842715330726 (-5)
Generalized Gamma	-5.796508162347891 (-9)	0.0

2.3 f

The Probability that someone has a health care claim of more than \1000$isgivenby$:

$$\begin{aligned}\Pr(X > 1000) &= 1 - \Pr(X \leq 1000) \\ &= \int_0^{1000} f_X dx\end{aligned}$$

However, since the integral of a Generalized Beta 2 Distribution is quite nasty, we will compute it numerically.

```
f(x) = GBetaTwoPDF( x, GB2_α̂, GB2_β, GB2_p̂, GB2_q̂ )
area = quadgk( f, 0, 1000 )[1]
output = ["Probability of Having > 1000: " (1-area)]
```

Probability of Having > 1000: 0.00507829692428996

3 Question 2

3.1 a

Equations (3) and (5) tell us that

$$\begin{aligned}w_t - (1-\alpha)exp(z_t)(k_t)^{\alpha-1} &= 0 \\ z_t &= \rho z_{t-1} + (1-\rho)\mu + \epsilon_t\end{aligned}$$

Taking logs of equation (3):

$$\begin{aligned}\log w_t &= \log(1-\alpha) + z_t + (\alpha - 1)\log k_t \\ z_t &= \log w_t - \log(1-\alpha) - (\alpha-1)\log k_t\end{aligned}$$

This tells us that for $t > 1$

$$\begin{aligned}\log w_t - \log(1-\alpha) - (\alpha-1)\log k_t &\sim \mathcal{N}\left(\rho z_{t-1} + (1-\rho)\mu, \sigma^2\right) \\ &\sim \mathcal{N}\left(\rho\left(\log w_{t-1} - \log(1-\alpha) - (\alpha-1)\log k_{t-1}\right) + (1-\rho)\mu, \sigma^2\right)\end{aligned}$$

For $t = 1$

$$\log w_1 - \log(1-\alpha) - (\alpha-1)\log k_t \sim \mathcal{N}(\mu, \sigma^2)$$

We may now estimate this model using Maximum Likelihood Estimation

```
#𝒩 (ρ (log w_{t−1} − log(1 − α) − (α − 1) log k_{t−1}) + (1 − ρ)μ, σ^2)

#Clean it up when it exists, comes in the order: (c, k, w, r)
macroData = DataFrame(load("MacroSeries.csv", header_exists=false, colnames=["C", "K", "W", "R"]))

w = convert( Vector{Float64}, macroData[:W] )
k = convert( Vector{Float64}, macroData[:K] )

function LogLikelihood( N, w::Vector{Float64}, k::Vector{Float64}, α::Real, ρ::Real, μ::Real, σ²::Real  )
    #The pdf of a normal: 1/sqrt(2πσ^2) exp(−(x−μ)^2/(2σ^2))
```

```
    #Log Likelihood: $-\frac{1}{2}\log\sigma^2 - \frac{(x-\mu)^2}{2\sigma^2}$

    logLik = -.5*log(σ²)- ( log(w[1]) - log(1-α) - (1-α)*log(k[1]) - μ)^2 / (2*σ²)
    #Note the way that the model is structured is: F(...) = 0, so we
    #are maximizing the likelihood of getting a 0 returned for all the
    #moments

    #Note we do not have the -.5*log(2*pi)
    #Because that does not matter at all for MLE estimation.
    for i in 2:N
        mean = ρ*(log(w[i-1]) - log( 1 - α)  - (α-1)*log( k[i-1])) + (1-ρ)*μ
        logLik += -.5*log( σ² ) - (  (log(w[i]) - log(1-α) - (1-α)*log(k[i]) - mean)^2 / (2*σ²))
    end
    return logLik
end

N = length(w)

α₀ = .5
β = .99
μ₀ = 1.0
σ₀ = 1.0
ρ₀ = 0.0

#We parameterize each of the variables so that they meet their constraints.
# tanh is used to ensure that $\rho \in (-1,1)$
θ = zeros(4)
θ[1] = log( α₀ / ( 1 - α₀) )
θ[2] = atanh( ρ₀)
θ[3] = log( μ₀ )
θ[4] = log( σ₀)


fun(x::Vector) = -LogLikelihood( N, w, k, exp(x[1]) / (1 + exp(x[1])), tanh(x[2]), exp(x[3]), exp(x[4])  )

result = optimize(fun, θ, Newton(), autodiff=:forward)

model_θ = result.minimizer

model_α̂ = exp(model_θ[1]) / (1 + exp(model_θ[1]))
model_ρ̂ = tanh(model_θ[2])
model_μ̂ = exp(model_θ[3])
model_σ̂ = exp(model_θ[4])

output = [["\$\\est{\\alpha}\$:", "\$\\est{\\rho}\$:", "\$\\est{\\mu}\$:", "\$\\est{\\sigma^{2}}\$:"]  [model_α̂,
↪  model_ρ̂, model_μ̂, model_σ̂]]
```

$\widehat{\alpha}_n$:	0.112797360917888892
$\widehat{\rho}_n$:	0.0013757752571974219
$\widehat{\mu}_n$:	2.198742765991596
$\widehat{\sigma^2}_n$:	0.009500213046354935

4 b

Taking logs of equation (3):

$$\begin{aligned}\log w_t &= \log(1-\alpha) + z_t + (\alpha-1)\log k_t \\ z_t &= \log w_t - \log(1-\alpha) - (\alpha-1)\log k_t\end{aligned}$$

This tells us that for $t > 1$

$$\begin{aligned}\log w_t - \log(1-\alpha) - (\alpha-1)\log k_t &\sim \mathcal{N}\left(\rho z_{t-1} + (1-\rho)\mu, \sigma^2\right) \\ &\sim \mathcal{N}\left(\rho\left(\log w_{t-1} - \log(1-\alpha) - (\alpha-1)\log k_{t-1}\right) + (1-\rho)\mu, \sigma^2\right)\end{aligned}$$

For $t = 1$

$$\log w_1 - \log(1-\alpha) - (\alpha-1)\log k_t \sim \mathcal{N}(\mu, \sigma^2)$$

We may now estimate this model using Maximum Likelihood Estimation
Equations (4) and (5) read:

$$\begin{aligned} r_t - \alpha \exp(z_t) k_t^{\alpha-1} &= 0 \\ z_t &= \rho z_{t-1} + (1-\rho)\mu + \epsilon_t \\ \epsilon_t &\sim \mathcal{N}(0, \sigma^2) \end{aligned}$$

Taking logs and isolating z_t

$$\begin{aligned} \log r_t &= \log\alpha + (\alpha-1)\log k_t + z_t \\ z_t &= \log\alpha + (\alpha-1)\log k_t - \log r_t \end{aligned}$$

For $t > 1$:

$$\begin{aligned}\log\alpha + (\alpha-1)\log k_t - \log r_t &\sim \mathcal{N}\left(\rho z_{t-1} + (1-\rho)\mu, \sigma^2\right) \\ &\sim \mathcal{N}\left(\rho\left(\log\alpha + (\alpha-1)\log k_{t-1} - \log r_{t-1}\right) + (1-\rho)\mu, \sigma^2\right)\end{aligned}$$

For $t = 1$:

$$\log\alpha + (\alpha-1)\log k_1 - \log r_1 \sim \mathcal{N}(\mu, \sigma^2)$$

This can be estimated using an MLE.

```julia
r = convert( Vector{Float64}, macroData[:R] )
k = convert( Vector{Float64}, macroData[:K] )

#$\log r_t - \log\alpha - z_t - (\alpha-1)\log k_t = 0$

function LogLikelihood( N, w::Vector{Float64}, k::Vector{Float64}, α::Real, ρ::Real, μ::Real, σ²::Real  )
    #The pdf of a normal: $\frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{(x-\mu)^2}{2\sigma^2})$
    #Log Likelihood: $-\frac{1}{2}\log\sigma^2 - \frac{(x-\mu)^2}{2\sigma^2}$

    logLik = -.5*log(σ²) - (log(α) + (α-1)*log(k[1]) - log(r[1]) - μ)^2 / (2*σ² )
    #Note the way that the model is structured is: F(...) = 0, so we
    #are maximizing the likelihood of getting a 0 returned for all the
    #moments

    for i in 2:N
        mean = ρ*(log(α) + (α-1)*log(k[i-1]) - log(r[i-1])) + (1-ρ)*μ
        logLik += -.5*log( σ² ) - (  (log(α) + (α-1)*log(k[i]) - log(r[i]) - mean)^2 / (2*σ²))
    end
    return logLik
end

N = length(w)

# α₀ = .5
```

```
# β = .99
# μ₀ = 1.0
# σ₀ = 1.0
# ρ₀ = .99
  α₀ = .5
  β = .99
  μ₀ = 1.0
  σ₀ = 1.0
  ρ₀ = 0.0

# #We param
eterize each of the variables so that they meet their constraints.
# tanh is used to ensure that ρ ∈ (−1, 1)
θ = zeros(4)
θ[1] = log( α₀ / ( 1 - α₀) )
θ[2] = atanh( ρ₀)
θ[3] = log( μ₀ )
θ[4] = log( σ₀)


fun(x::Vector) = -LogLikelihood( N, w, k, exp(x[1]) / (1 + exp(x[1])), tanh(x[2]), exp(x[3]), exp(x[4])  )

result = optimize(fun, θ, Newton(), autodiff=:forward)

model_θ = result.minimizer

model_α̂ = exp(model_θ[1]) / (1 + exp(model_θ[1]))
model_ρ̂ = tanh(model_θ[2])
model_μ̂ = exp(model_θ[3])
model_σ̂ = exp(model_θ[4])

output = [["\$\\est{\\alpha}\$:", "\$\\est{\\rho}\$:", "\$\\est{\\mu}\$:", "\$\\est{\\sigma^{2}}\$:"]  [model_α̂,
↪  model_ρ̂, model_μ̂, model_σ̂]]
```

$\widehat{\alpha}_n$:	1
$\widehat{\rho}_n$:	0.26158802254436014
$\widehat{\mu}_n$:	9793456505444984 (-30)
$\widehat{\sigma^2}_n$:	0.009480777698471455

4.1 c

From the derivation of the distribution of $\log r_t$ in part (b):

$$
\begin{aligned}
\Pr(r_t > 1) &= \Pr(\log r_t > 0) \\
&= \Pr(\log \alpha + z_t + (\alpha - 1)\log k_t > 0) \\
&= \Pr(\log \alpha + \rho z_{t-1} + (1-\rho)\mu + \epsilon_t + (\alpha - 1)\log k_t > 0) \\
&= \Pr(\log(\alpha) + \rho z_{t-1} + (1-\rho)\mu + \frac{Z}{\sigma} + (\alpha - 1)\log k_t > 0) \\
&= \Pr(Z > -\sigma(\log(\alpha) + \rho z_{t-1} + (1-\rho)\mu + (\alpha - 1)\log k_t)) \\
&= 1 - \Pr(Z \leq -\sigma(\log(\alpha) + \rho z_{t-1} + (1-\rho)\mu + (\alpha - 1)\log k_t)) \\
&= \Phi^{-1}(-\sigma(\log(\alpha) + \rho z_{t-1} + (1-\rho)\mu + (\alpha - 1)\log k_t)) \\
&\approx \Phi^{-1}(-\widehat{\sigma}_n(\log \widehat{\alpha}_n + \widehat{\rho}_n 10 + (1-\widehat{\rho}_n)\widehat{\mu}_n + (\widehat{\alpha}_n - 1)\log(7,500,000)))
\end{aligned}
$$

```
  prob = cdf( Normal(), -sqrt(model_σ̂)*( log(model_α̂) + model_ρ̂*10 + (1-model_ρ̂)*model_μ̂ + (model_α̂-1)*log( 7500000)))
result = ["Prob" prob]
```

Prob	0.39947494113405524