Structural Estimation Pset 2

Timothy Schwieg

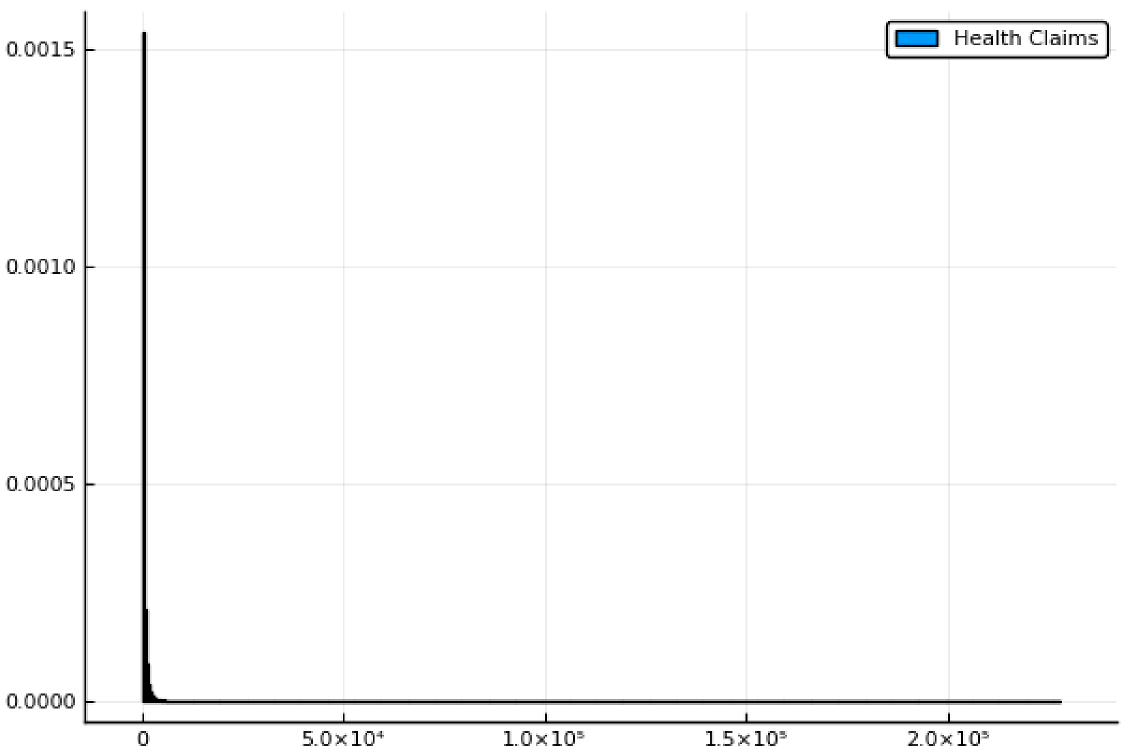
1 Question One

1.1 a

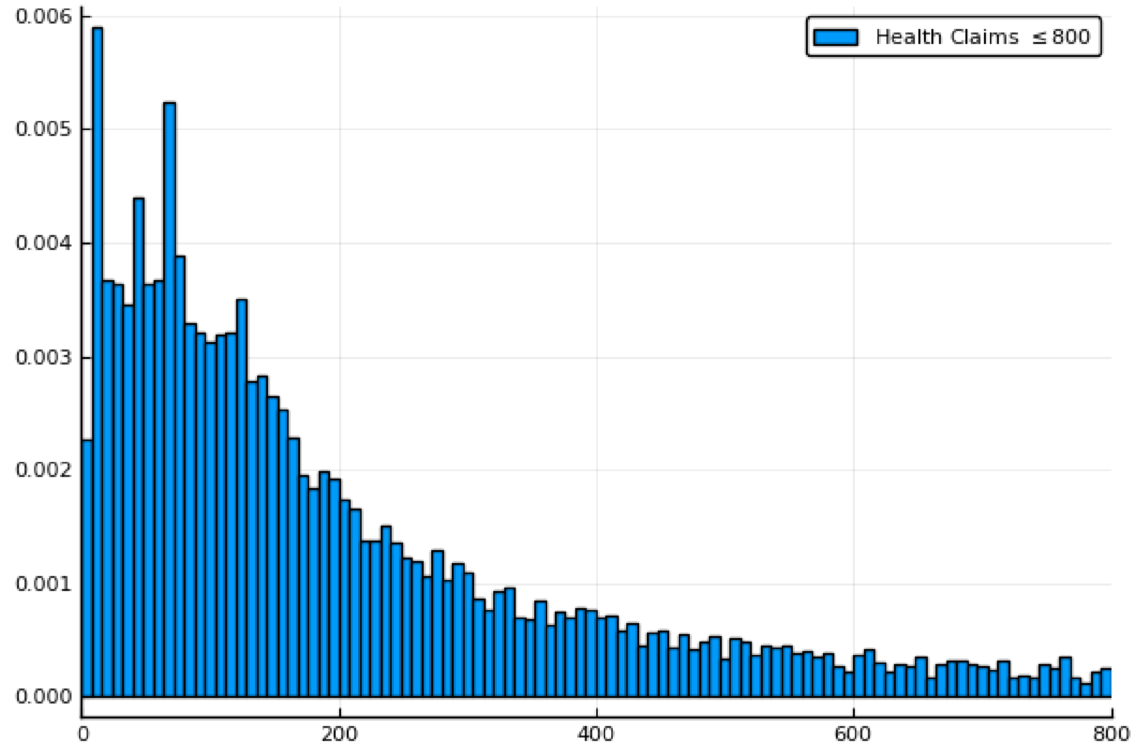
```
healthClaims = DataFrame(load("clms.csv", header_exists=false, colnames=["A"]))

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

mean	720.28
min	0.01
median	172.21
max	2.2797×10^{05}
StdDev	3972.9

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



```julia
  #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),label="Health Claims \$\\leq 800\$")
savefig("histTwo.png")
```



We can see the shape of the distribution for majority of the data points lie below 800. There is a very large tail that distorts the histogram, preventing anything from being seen on the first one. All we are able to see is that there is a large amount of mass somewhere slightly above zero in the first one. The second distribution shows the mode, and indicates the very long tail that the distribution is likely to contain.

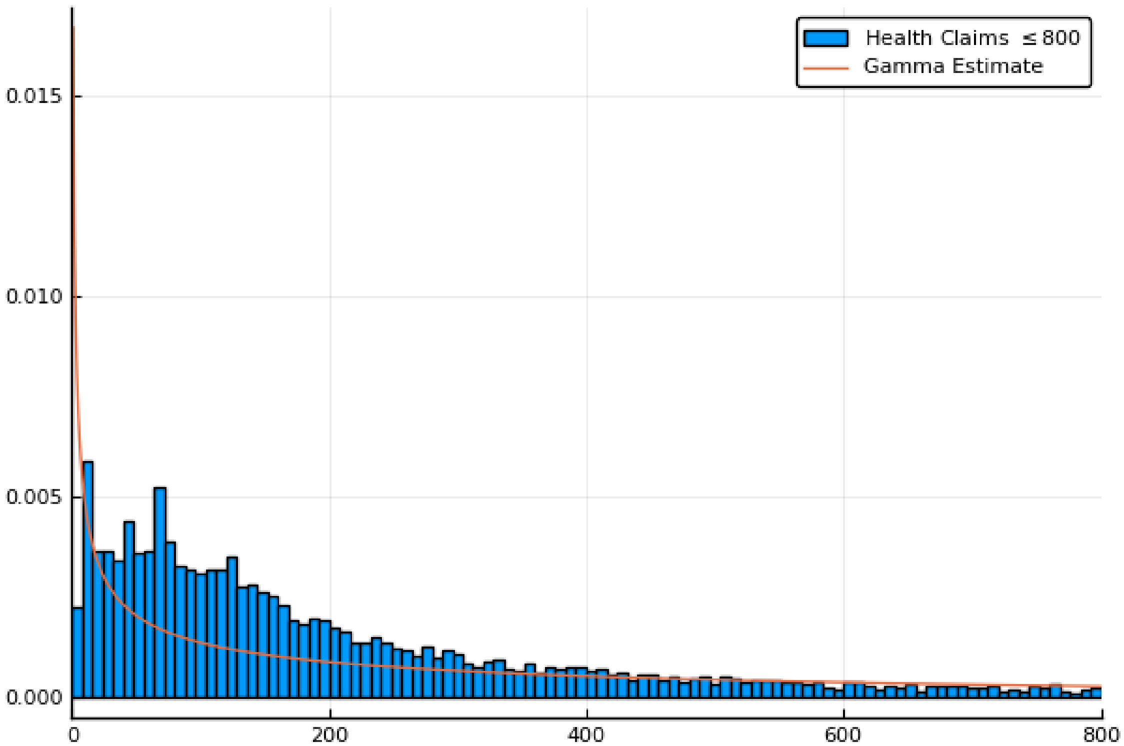
1.2 b

```julia
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)

    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}, healthClaims[:A] )

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

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

likelihood = GammaLogLikelihood(  healthCosts, Gamma_α̂, Gamma_β)

result = [["\$\\est{\\alpha}\$: ", "\$\\est{\\beta}\$: ", "Likelihood: " ] cln.([ Gamma_α̂,  Gamma_β, likelihood])]
```

$\widehat{\alpha}_n$:	0.47251
$\widehat{\beta}_n$:	1524.4
Likelihood:	-7.3193

```
histogram( healthClaims[:A], bins=0:8:800, normalize=true, xlims=(0,800),label="Health Claims \$\\leq 800\$")
pdfXVal = range( 0.0, 800.0)

pdfYVal = [GammaPDF( Gamma_α̂, Gamma_β, x ) for x in pdfXVal]

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



We can see that this fit over-fits the tail of the distribution at the cost of the bulk of the mass. It places a relatively high probability of being at a very small value, when the distribution appears to have a hump.

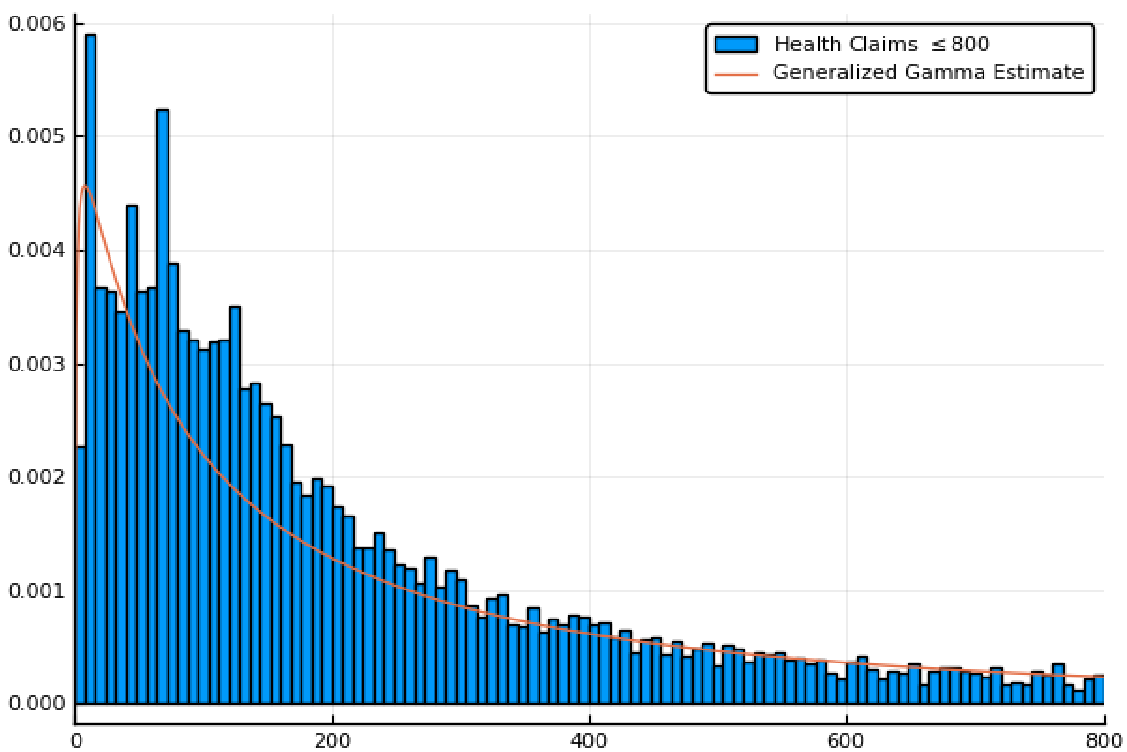
2 c

```
# (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. Limit them to .1 as the lower bound for
    #numerics sake
    θ = log.(guess .- .1)
    fun(x::Vector) = -GGammaLikelihood( data, (exp.(x).+ .1)... )



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


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

GG_α̂ = exp(sln.minimizer[1]) + .1
```

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

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

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

GG $\widehat{\alpha}_n$:	1.7396
GG $\widehat{\beta}_n$:	0.1
GG $\widehat{m}_n$:	0.24872
GG Likelihood:	-7.0746

```julia
histogram( healthClaims[:A], bins=0:8:800, normalize=true, xlims=(0,800),label="Health Claims \$\\leq 800\$")
pdfXVal = range(0.0, 800.0)
#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" )
```



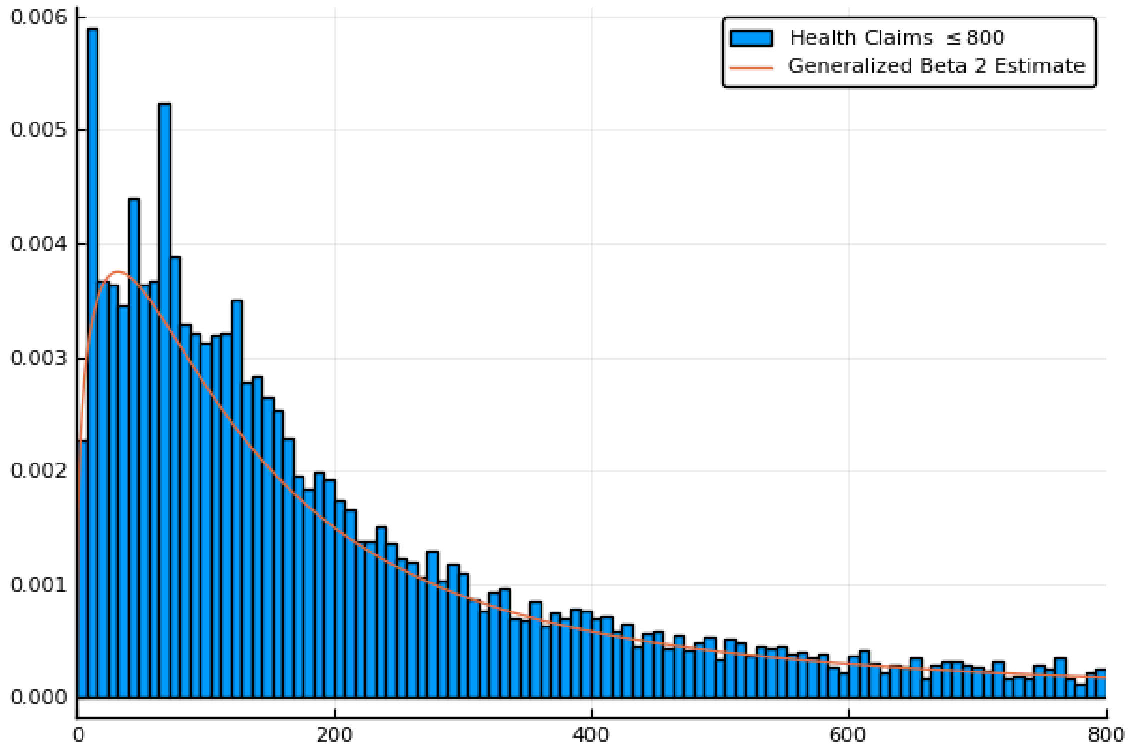
This distribution captures the mode of the distribution being greater than zero, and while the hump is still occurring too early in order to fit the long tail of the distribution; it appears to fit the histogram much better than the Gamma Distribution fit.

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

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

function EstimateGBetaTwo( data::Vector{Float64}, guess::Vector{Float64})
      #To hard enforce that all of our parameters are positive, we
      #exponentiate them
    θ = log.(guess .- .1)
    #θ = guess
    fun(x::Vector) = -GBetaTwoLikelihood( data, (exp.(x) .+ .1)... )


    #This guy is being fickle, Newton() struggles a little bit, but
    #NewtonTrust seems to outperform LBFGS
    result = optimize(fun, θ, NewtonTrustRegion(), autodiff=:forward, Optim.Options(iterations=2000) )
end

#$GG(\alpha, \beta, m) = \lim_{q \to \infty} GB2\left(a = m, b = q^{1/m}\beta, p = \frac{\alpha}{m}, q\right)$
sln = EstimateGBetaTwo( healthCosts, [GG_m̂, 10000^(1 / GG_m̂) * GG_β, GG_α̂ / GG_m̂, 10000])

GB2_α̂ = exp( sln.minimizer[1]) + .1
GB2_β = exp( sln.minimizer[2]) + .1
GB2_p̂ = exp( sln.minimizer[3]) + .1
GB2_q̂ = exp( sln.minimizer[4]) + .1
GB2_LogLikelihood = -sln.minimum

result = [["GB2 \$\\est{\\alpha}\$: ", "GB2 \$\\est{\\beta}\$: ", "GB2 \$\\est{p}\$: ","GB2 \$\\est{q}\$: ","GB2 Likelihood: " ] cln.([GB2_α̂, GB2_β,  GB2_p̂,  GB2_q̂, -sln.minimum])]
```

GB2 $\widehat{\alpha}_n$:	1.2714
GB2 $\widehat{\beta}_n$:	143.23
GB2 $\widehat{p}_n$:	1.0299
GB2 $\widehat{q}_n$:	0.84852
GB2 Likelihood:	-7.0354

```julia
histogram( healthClaims[:A], bins=0:8:800, normalize=true, xlims=(0,800),label="Health Claims \$\\leq 800\$")
pdfXVal = range( 0.0, 800.0)
#pdfXVal = linspace( minimum(truncatedHealthClaims), maximum(truncatedHealthClaims))
pdfYVal = [GBetaTwoPDF( x, GB2_α̂, GB2_β, GB2_p̂, GB2_q̂ ) for x in pdfXVal]

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



We can see that the Generalized Beta 2 Distribution has fit the distribution near 0 slightly better than the Generalized Gamma Distribution did. It still captures the long tail of the distribution relatively well, though the fit is only slightly better than the previous one.

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:

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

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

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

results = [["", "Gamma", "Generalized Gamma"] [ "\$\\chi^{2}\$", cln(tStatGamma), cln(tStatGG)] ["p-value",  cln(1.0 -
↪  cdf(Chisq(4),tStatGamma)), cln(1.0 - cdf( Chisq(4),tStatGG)) ] ]
```

	χ^2	p-value
Gamma	56.771	1.382×10^{-11}
Generalized Gamma	7.8294	0.098033

We find that we can reject the Null Hypothesis that the parameters of the Generalized Beta 2 are consistent with the Gamma Distribution at pretty much any significance level.

We find that the probability that this data could be generated by a Gamma Distribution is virtually zero.

For the Generalized Gamma Distribution, we find that it is possible that these parameters are consistent with the Generalized Gamma Distribution. To be willing to reject this hypothesis, we must be willing to accept a 10% chance of being incorrect. Since we are not psychologists, we will fail to reject this hypothesis.

2.3 f

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

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

However, since the integral of a Generalized Beta 2 Distribution is quite nasty, I shall compute it numerically. We ignore more complicated methods of quadrature and brute force rhomboid quadrature.

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

Probability of Having > 1000:   0.11766

We would like to do the same for the Gamma Distribution as well.

```
f(x) = GammaPDF( Gamma_α̂, Gamma_β, x )
area = quadgk(f, 0, 1000)[1]
output = ["Gamma Probability of Having > 1000: " cln(1-area)]
```

Gamma Probability of Having > 1000:   0.23678

We can see that the Gamma Distribution overstates the long tail of the distribution, as it is difficult for this distribution to fit a large amount of data very far away from the mean.

3 Question 2

3.1 a

Equations (3) and (5) tell us that

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

Taking logs of equation (3):

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

This tells us that for $t > 1$

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

For $t = 1$

$$\log w_1 - \log(1-\alpha) - \alpha \log k_1 \sim \mathcal{N}(\mu, \sigma^2)$$

We may now estimate this model using Maximum Likelihood Estimation

```julia
#\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)

#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: \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(w[1]) - log(1-α) - (α)*log(k[1]) - μ)^2 / (2*σ²))

    #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)*log( k[i-1])) + (1-ρ)*μ
        logLik += -.5*log( σ² ) - (  (log(w[i]) - log(1-α) - (α)*log(k[i]) - mean)^2 / (2*σ²))
    end
    return logLik
end


N = length(w)

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

#We parameterize 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
```

```julia

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}}\$:"]  cln.([model_α̂,
↪  model_ρ̂, model_μ̂, model_σ̂])]
```

$$\begin{array}{lr} \widehat{\alpha}_n\text{:} & 0.70216 \\ \widehat{\rho}_n\text{:} & 0.47972 \\ \widehat{\mu}_n\text{:} & 6.2533 \\ \widehat{\sigma^2}_n\text{:} & 0.0084723 \end{array}$$

```julia
#Sadly Optim.jl does not automatically report the hessian, though I am
#sure it is obtainable. So we will use forward-mode automatic
#differentiation to obtain this hessian. However it does not always
#return symmetric matrices, so we will make the matrix symmetric then
#invert it using the cholesky decomposition to be numerically stable.
hess = ForwardDiff.hessian(fun, result.minimizer)

F = cholesky(Hermitian(hess))
F.L * F.U = H
hessInv = cln.(F.U \ (F.L \ I))
#This is for version .6 rather than the 1.0 running above.
#F = chol(Hermitian(hess))
#hessInv = cln.(F \ (F' \ I))
result = hessInv
```

$$H^{-1} = \begin{pmatrix} 1.2234 & -0.38792 & -0.50942 & -2.1141 \times 10^{-12} \\ -0.38792 & 0.1361 & 0.16153 & 2.6498 \times 10^{-12} \\ -0.50942 & 0.16153 & 0.21213 & 3.384 \times 10^{-13} \\ -2.1141 \times 10^{-12} & 2.6498 \times 10^{-12} & 3.384 \times 10^{-13} & 0.02 \end{pmatrix}$$

We can see that the model believes that there is almost no co-variance between the σ^2 and the other parameters. There is a high standard error for α and σ^2 relative to the magnitude of the point estimate.

4 b

Equations (4) and (5) read:

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

Taking logs and isolating z_t

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

For $t > 1$:

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

For $t = 1$:

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

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, r::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(r[1]) - log(α) - (α-1)*log(k[1]) - μ)^2 / (2*σ²))

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

    return logLik
end

N = size(macroData)[1]

α₀ = .5
β = .99
μ₀ = .5
σ₀ = .5
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)

function limitedLogistic( unbounded::Real )
    return ((exp(unbounded)) / ( 1 + exp(unbounded)))*.99 + .005
end

#This clamp on the logistic function is quite the hack, since this
#function shouldn't get to 0 or 1, but it was getting stuck at 1
fun(x::Vector) = -LogLikelihood( N, r, k, limitedLogistic(x[1]), tanh(x[2]), exp(x[3]), exp(x[4])  )

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

bmodel_θ = result.minimizer

bmodel_α̂ = limitedLogistic(bmodel_θ[1])
bmodel_ρ̂ = tanh(bmodel_θ[2])
bmodel_μ̂ = exp(bmodel_θ[3])
bmodel_σ̂ = exp(bmodel_θ[4])
```

```

output = [["\$\\est{\\alpha}\$:", "\$\\est{\\rho}\$:", "\$\\est{\\mu}\$:", "\$\\est{\\sigma^{2}}\$:"]  cln.([bmodel_α̂,
↪  bmodel_ρ̂, bmodel_μ̂, bmodel_σ̂])]
```

$\widehat{\alpha}_n$:	0.70216
$\widehat{\rho}_n$:	0.47972
$\widehat{\mu}_n$:	5.0729
$\widehat{\sigma^2}_n$:	0.0084723

```
#Sadly Optim.jl does not automatically report the hessian, though I am
#sure it is obtainable. So we will use forward-mode automatic
#differentiation to obtain this hessian. However it does not always
#return symmetric matrices, so we will make the matrix symmetric then
#invert it using the cholesky decomposition to be numerically stable.
hess = ForwardDiff.hessian(fun, result.minimizer)

F = cholesky(Hermitian(hess))
#F.U' * F.U = H
hessInv = cln.(F.U \ (F.L \ I))
# F = chol(Hermitian(hess))
# hessInv = cln.(F \ (F' \ I))
result = hessInv
```

$$H^{-1} = \begin{pmatrix} 1.2582 & -0.3934 & -0.88139 & -3.7224 \times 10^{-13} \\ -0.3934 & 0.1361 & 0.27559 & 1.1806 \times 10^{-13} \\ -0.88139 & 0.27559 & 0.61745 & 2.6018 \times 10^{-13} \\ -3.7224 \times 10^{-13} & 1.1806 \times 10^{-13} & 2.6018 \times 10^{-13} & 0.02 \end{pmatrix}$$

We find nearly the same results for the point estimates, and the diagonal elements of the inverse Hessian, modulo some noise. We find that the off-diagonal elements are less consistent between the two estimates, though these co-variances are quite small relative to the measurements. To really tell the difference between the point estimates, we would have to compare the overlap of the confidence sets.

4.1 c

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

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

Where $Z \sim \mathcal{N}(0, 1)$

```
prob = 1 - cdf( Normal(), -(1.0 / sqrt(model_σ̂))*( log(model_α̂) + model_ρ̂*10 + (1-model_ρ̂)*model_μ̂ + (model_α̂-1)*log(7500000)))
result = ["\\Pr( r_t > 1) = " cln(prob)]
```

$$\Pr(\ \mathrm{r_t} > 1) = \quad 1$$