

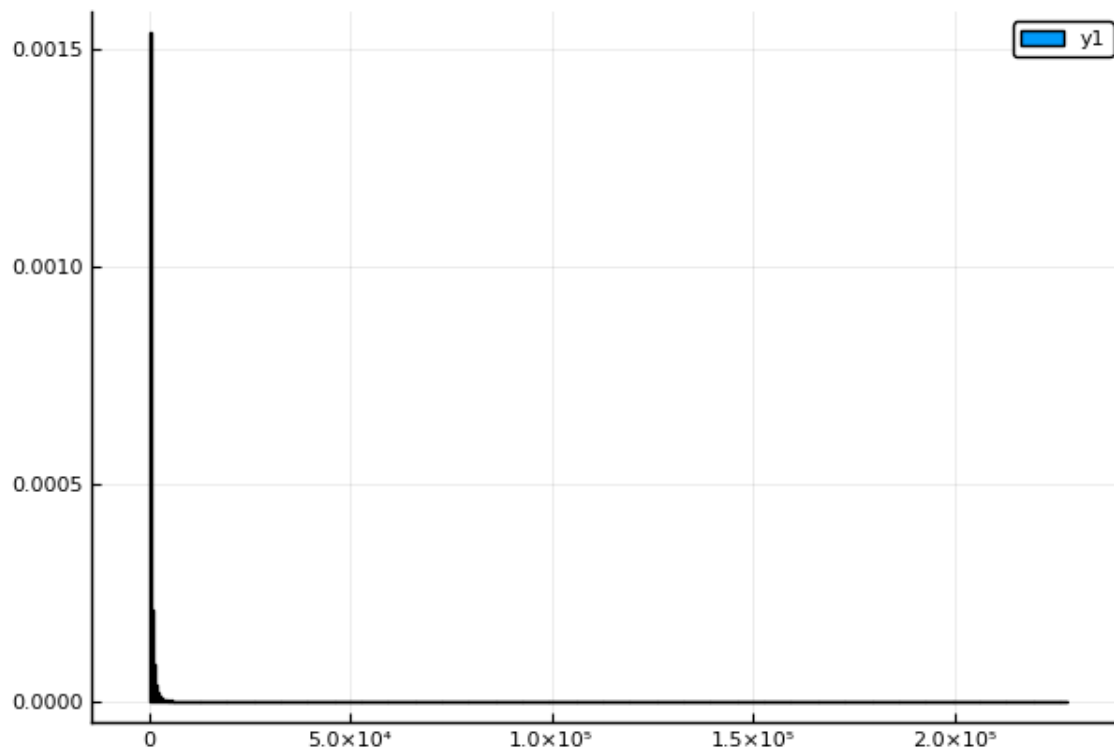
Structural Estimation Pset 2

Timothy Schwieg

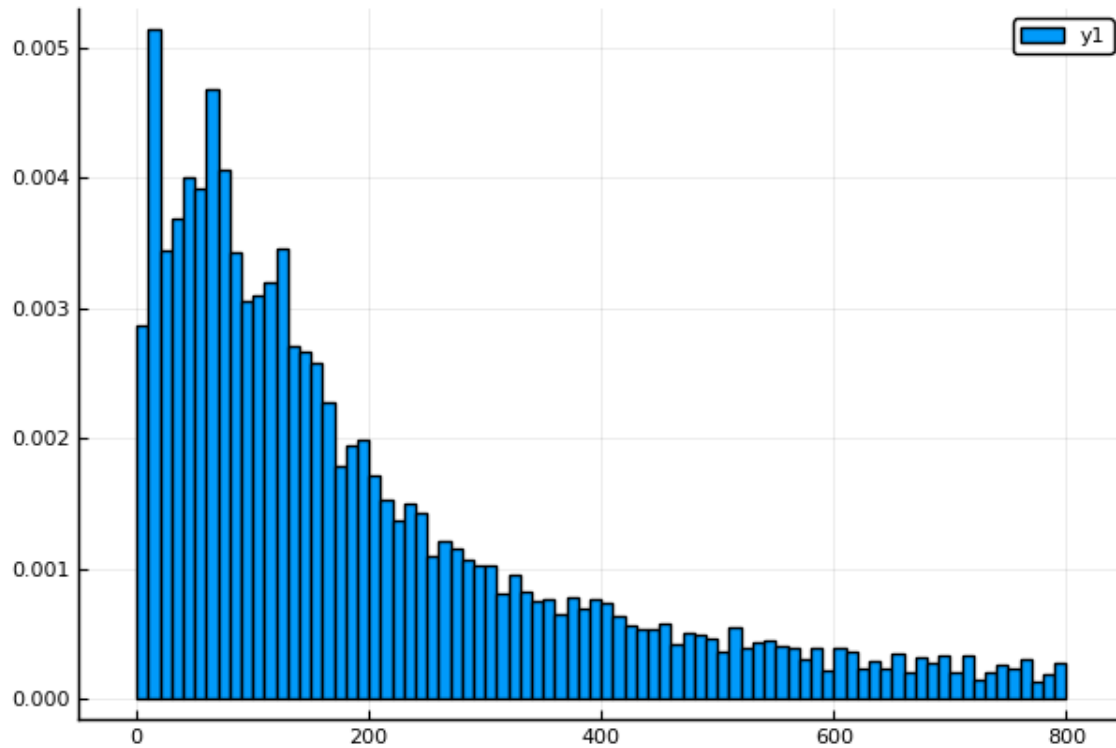
1 Question One

1.1 a

```
1 healthClaims = CSV.read( "clms.txt", header=:A )
2 describe( healthClaims )
3
4 println( "Standard Deviation: ", std(healthClaims[:A]))
5
6 histogram( healthClaims[:A], bins=1000, normalize = true)
7 savefig("histOne.png")
```



```
8 truncatedHealthClaims = healthClaims[healthClaims[:A] .<= 800, 1]
9
10
11 histogram( truncatedHealthClaims, bins = 100, normalize = true)
12 savefig("histTwo.png")
```



1.2 b

```

13 function GammaLogLikelihood( x::Vector{Float64}, α::Float64, β::Float64)
14     #Yes I know I could get this using Distributions.jl which could
15     #even do the MLE estimate But thats pretty much cheating, and
16     #gamma is in the exponential family so using Newton's method will
17     #cause no issues.
18
19     #Pdf is:  $\frac{1}{\Gamma(\alpha)\beta^\alpha} x^{\alpha-1} \exp\left(-\frac{x}{\beta}\right)$ 
20     #Log-likelihood is:  $-\alpha \log(\beta) - \log(\Gamma(\alpha)) + (\alpha - 1) \log x - \frac{x}{\beta}$ 
21
22     return -α*log( β) - lgamma(α) + (α - 1)*mean(log.(x)) - mean(x) / β
23 end
24
25 function GammaGradient( x::Vector{Float64}, α::Float64, β::Float64)
26     delA = -log(β) - digamma(α) + mean(log.(x))
27     #delB = mean(x) / β - α
28     delB = mean(x) / β^2 - α / β
29     return [delA,delB]
30 end
31
32 function GammaHessian( x::Vector{Float64}, α::Float64, β::Float64)
33     delAA = -trigamma(α)
34     delAB = -1 / β
35     delBB = ( α / (β*β)) - ((2* mean(x)) / (β*β*β))
36     return [delAA delAB; delAB delBB]
37 end
38
39 function GammaPDF( α::Float64, β::Float64, x::Float64)
40     return (1 / (gamma(α)*β^α))*x^(α-1)*exp( -x/β)
41 end
42
43 function EstimateGammaParameters( data::Vector{Float64}, guess::Vector{Float64}, gradientFun, hessianFun)
44

```

```

45     θ = guess
46     tol = 1e-10
47     maxLoops = 100
48
49     grad = gradientFun( data, θ... )
50     hess = hessianFun( data, θ... )
51
52     loopCounter = 0
53     while( loopCounter < maxLoops && norm(grad) >= tol)
54         θ = θ - hess \ grad
55         grad = gradientFun( data, θ... )
56         hess = hessianFun( data, θ... )
57
58         loopCounter += 1
59         # println( norm(grad))
60         # println( θ)
61         # println( " ")
62     end
63     #println( loopCounter)
64     return θ
65 end
66 healthCosts = convert( Vector{Float64}, truncatedHealthClaims )#healthClaims[:A] )
67
68 β₀ = var(healthCosts) / mean(healthCosts)
69 α₀ = mean(healthCosts) / β₀
70
71 (Gamma_α, Gamma_β) = EstimateGammaParameters( healthCosts, [α₀, β₀], GammaGradient, GammaHessian)
72
73 likelihood = GammaLogLikelihood( healthCosts, Gamma_α, Gamma_β)
74
75 result = [["\\est{\\alpha}\\$:", "\\$\\est{\\beta}\\$:", "Likelihood: " ] [ Gamma_α, Gamma_β, likelihood]]

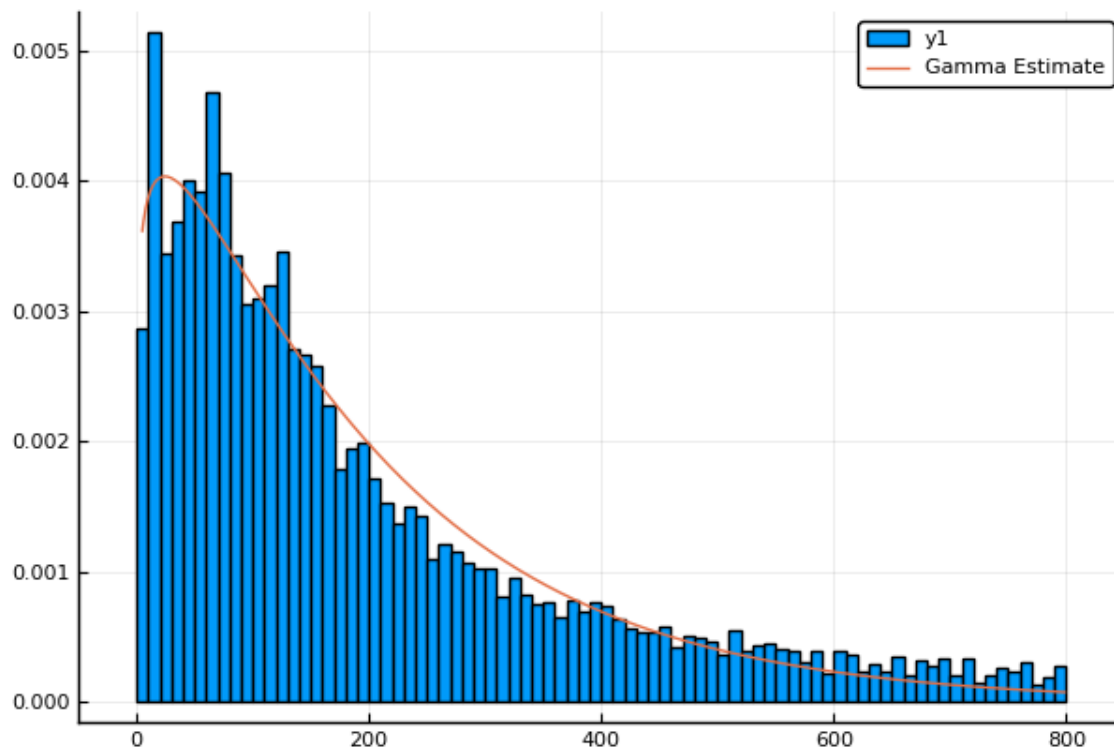
```

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

```

76 histogram( truncatedHealthClaims, bins = 100, normalize = true)
77 pdfXVal = range( minimum(truncatedHealthClaims)+5, maximum(truncatedHealthClaims))
78 #pdfXVal = linspace( minimum(truncatedHealthClaims), maximum(truncatedHealthClaims))
79 pdfYVal = [GammaPDF( Gamma_α, Gamma_β, x ) for x in pdfXVal]
80
81
82 plot!( pdfXVal, pdfYVal, label="Gamma Estimate" )
83 savefig("histPDF_Gamma.png")

```



2 c

```

84 #I don't think this is the correct pdf?
85 function GGammaPDF( α::Float64, β::Float64, m::Float64, x::Float64)
86     return ( (m / β^α) * x^(α-1) * exp( - (x / β)^m ) ) / gamma( α / m)
87
88     #return (m * x^(m*β - 1) * exp( - (x / α)^m )) / (α^(m*β) * gamma( β ) )
89 end
90
91
92 function GGammaLikelihood( x::Vector{Float64}, α::Real, β::Real, m::Real)
93     return log(m) - α*log(β) + (α - 1)*mean(log.(x)) - mean( (x ./ β).^m ) - lgamma( α / m )
94 end
95
96 function EstimateGG( data::Vector{Float64}, guess::Vector{Float64})
97     #To hard enforce that all of our parameters are positive, we
98     #exponentiate them
99     θ = log.(guess)
100     fun(x::Vector) = -GGammaLikelihood( data, exp.(x)... )
101
102
103
104     result = optimize(fun, θ, ConjugateGradient(), autodiff=:forward)
105 end
106
107 sln = EstimateGG( healthCosts, [Gamma_α, Gamma_β, 1.0])
108
109 GG_α = exp(sln.minimizer[1])
110 GG_β = exp(sln.minimizer[2])
111 GG_m = exp(sln.minimizer[3])
112 GG_LogLikelihood = -sln.minimum
113
114 println( "GG α = ", GG_α)
115 println( "GG β = ", GG_β )
116 println( "GG m = ", GG_m )

```

```

117 println( "Likelihood Value: ", GG_LogLikelihood )
118
119 result = [ ["GG \${est}\alpha}\$: ", "GG \${est}\beta}\$: ", "GG \${est}m}\$: ", "GG Likelihood: " ] [ GG_α, GG_β,
↪ GG_m, GG_LogLikelihood]]

```

```

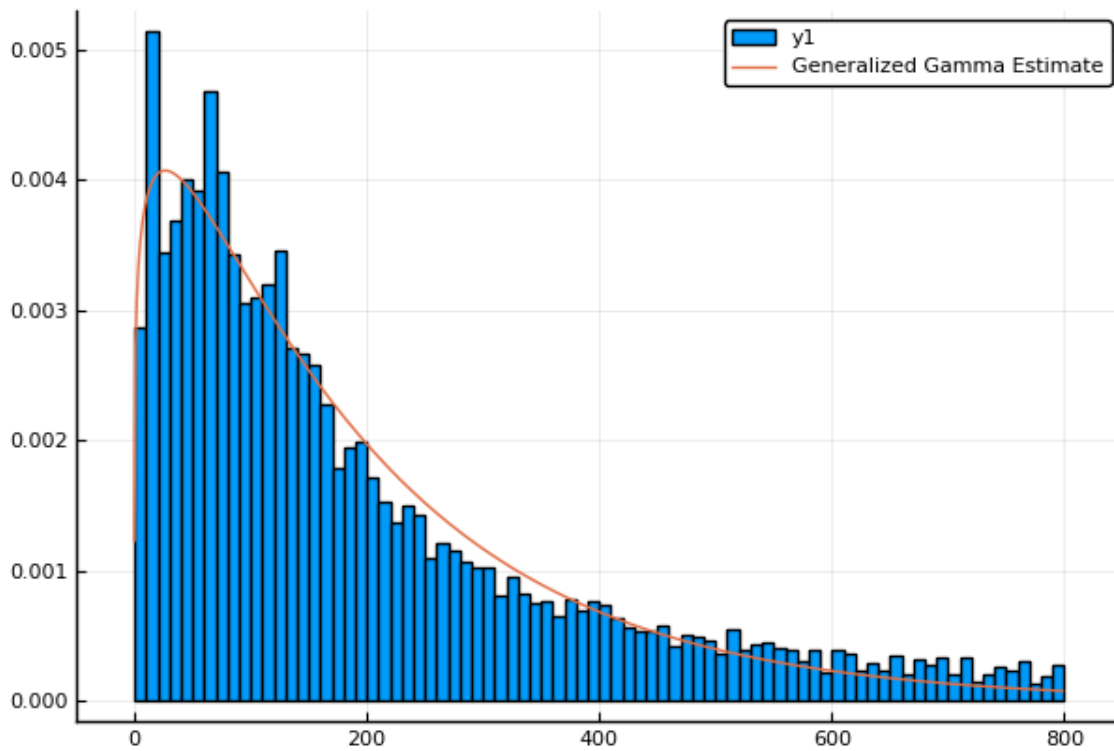
GG  $\hat{\alpha}_n$ :          1.1755020098846642
GG  $\hat{\beta}_n$ :          156.18446475134172
GG  $\hat{m}_n$ :           0.9498167064643459
GG Likelihood:    -6.289560051458711

```

```

120 histogram( truncatedHealthClaims, bins = 100, normalize = true)
121 pdfXVal = range( minimum(truncatedHealthClaims), maximum(truncatedHealthClaims))
122 #pdfXVal = linspace( minimum(truncatedHealthClaims), maximum(truncatedHealthClaims))
123 pdfYVal = [GGammaPDF( GG_α, GG_β, GG_m, x ) for x in pdfXVal]
124
125 plot!( pdfXVal, pdfYVal, label="Generalized Gamma Estimate" )
126 savefig( "histPDF_GG.png" )

```



2.1 d

```

127 function GBetaTwoPDF( x::Float64, a::Real, b::Real, p::Real, q::Real)
128     #We require all parameters to be positive, so abs(a) = a
129     return a*x^(a*p - 1) / (b^(a*p) * beta(p,q)*(1+(x/b)^a)^(p+q))
130 end
131
132 function GBetaTwoLikelihood( x::Vector{Float64}, a::Real, b::Real, p::Real, q::Real)
133     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 ))
134 end
135
136 function EstimateGBetaTwo( data::Vector{Float64}, guess::Vector{Float64})

```

```

137     #To hard enforce that all of our parameters are positive, we
138     #exponentiate them
139     θ = log.(guess)
140     #θ = guess
141     fun(x::Vector) = -GBetaTwoLikelihood( data, exp.(x)... )
142
143
144     #This guy is being fickle, and Newton() would not converge
145     #LBFGS converges, but to a higher value than Newton()
146     result = optimize(fun, θ, NewtonTrustRegion(), autodiff=:forward, Optim.Options(iterations=2000) )
147 end
148
149 sln = EstimateGBetaTwo( healthCosts, [GG_α, GG_β, GG_ρ, 10000])
150
151 GB2_α = exp( sln.minimizer[1])
152 GB2_β = exp( sln.minimizer[2])
153 GB2_ρ = exp( sln.minimizer[3])
154 GB2_θ = exp( sln.minimizer[4])
155 GB2_LogLikelihood = -sln.minimum
156
157 result = [ "GB2 \${est{\alpha}}\$ : ", "GB2 \${est{\beta}}\$ : ", "GB2 \${est{p}}\$ : ", "GB2 \${est{q}}\$ : ", "GB2
↳ Likelihood: " ] [GB2_α, GB2_β, GB2_ρ, GB2_θ, -sln.minimum]

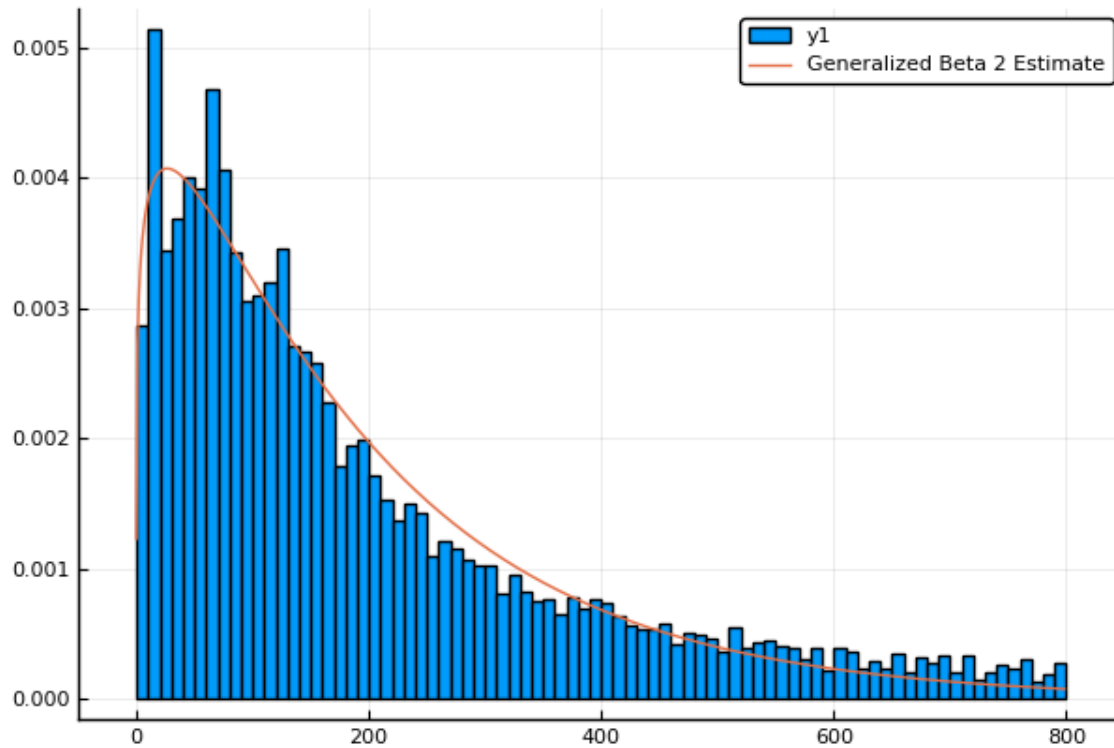
```

GB2 $\hat{\alpha}_n$:	0.9498180950429491
GB2 $\hat{\beta}_n$:	1.0983701276884081 (9)
GB2 \hat{p}_n :	1.2376067626960379
GB2 \hat{q}_n :	3.187929333688613 (6)
GB2 Likelihood:	-6.289560054356965

```

158 histogram( truncatedHealthClaims, bins = 100, normalize = true)
159 pdfXVal = range( minimum(truncatedHealthClaims), maximum(truncatedHealthClaims))
160 #pdfXVal = linspace( minimum(truncatedHealthClaims), maximum(truncatedHealthClaims))
161 pdfYVal = [GBetaTwoPDF( x, GB2_α, GB2_β, GB2_ρ, GB2_θ ) for x in pdfXVal]
162
163 plot!( pdfXVal, pdfYVal, label="Generalized Beta 2 Estimate" )
164 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:

$$2 \left(f(\hat{\theta}_n - \tilde{\theta}_n) \right) \sim \chi_p^2$$

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

```

165 # Gamma Has Two restrictions
166 tStatGamma = 2*(GB2_LogLikelihood - likelihood)
167 # Generalized Gamma Has One Restriction
168 tStatGG = 2*(GB2_LogLikelihood - GG_LogLikelihood)
169
170 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 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, we will compute it numerically.

```

171 f(x) = GBetaTwoPDF( x, GB2_α, GB2_β, GB2_ρ, GB2_η )
172 area = quadgk( f, 0, 1000 )[1]
173 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

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

Note that: $z_0 = \mu$ Therefore:

$$\begin{aligned}z_1 &= \mu + \epsilon_1 \\ z_2 &= \mu + \rho \epsilon_1 + \epsilon_2 \\ z_t &= \mu + \sum_{i=0}^{t-1} \rho^i \epsilon_{t-i}\end{aligned}$$

Combining these two together:

$$w_t - (1 - \alpha) \exp \left(\mu + \sum_{i=0}^{t-1} \rho^i \epsilon_{t-i} \right) k_t^\alpha = 0$$

Taking logs and isolating the random component:

$$\log w_t - \log(1 - \alpha) - \mu - \alpha \log k_t = \sum_{i=0}^{t-1} \rho^i \epsilon_{t-i}$$

Note that the sum of iid distributed normal random variables is distributed normal, where the variance is given by the sum of the variances.

Thus

$$\sum_{i=0}^{t-1} \rho^i \epsilon_{t-i} \sim \mathcal{N}(0, \sigma^2 \sum_{i=0}^{t-1} \rho^{2i}) = \mathcal{N} \left(0, \sigma^2 \frac{1 - \rho^{2t}}{1 - \rho^2} \right)$$

We may now estimate this model using Maximum Likelihood Estimation

```

174 #log w_t - log(1 - alpha) - mu - alpha log k_t = sum_{i=0}^{t-1} rho^i epsilon_{t-i}
175 # Variance of error: sigma^2 * (1 - rho^{2t}) / (1 - rho)
176
177 #Clean it up when it exists, comes in the order: (c, k, w, r)
178 macroData = CSV.read( "MacroSeries.txt", header=[:C,:K,:W,:R])
179
180 w = convert( Vector{Float64}, macroData[:W] )
181 k = convert( Vector{Float64}, macroData[:K] )
182
183 function LogLikelihood( N, w::Vector{Float64}, k::Vector{Float64}, alpha::Real, rho::Real, mu::Real, sigma2::Real )
184     #The pdf of a normal: 1 / (sqrt(2*pi*sigma2) * exp(-(x-mu)^2 / (2*sigma2)))
185     #Log Likelihood: -1/2 log sigma2 - (x-mu)^2 / (2*sigma2)
186
187     logLik = 0.0
188     #Note the way that the model is structured is: F(...) = 0, so we
189     #are maximizing the likelihood of getting a 0 returned for all the
190     #moments
191
192     #Note we do not have the -.5*log(2*pi)
193     #Because that does not matter at all for MLE estimation.
194     for i in 1:N
195         mean = log(w[i]) - log( 1 - alpha ) - mu - alpha*log( k[i] )
196         var = sigma2 * ( 1 - rho^(2*i) ) / ( 1 - rho )
197         logLik += -.5*log( sigma2 ) - ( mean*mean / (2*sigma2) )
198     end
199     return logLik
200 end
201
202 N = length(w)
203
204 alpha_0 = .5
205 beta_0 = .99
206 mu_0 = 1.0
207 sigma_0 = 1.0
208 rho_0 = 0.0
209
210 #We parameterize each of the variables so that they meet their constraints.
211 # tanh is used to ensure that rho in (-1,1)
212 theta = zeros(4)
213 theta[1] = log( alpha_0 / ( 1 - alpha_0 ) )
214 theta[2] = atanh( rho_0 )
215 theta[3] = log( mu_0 )
216 theta[4] = log( sigma_0 )
217
218
219 fun(x::Vector) = -LogLikelihood( N, w, k, exp(x[1]) / (1 + exp(x[1])), tanh(x[2]), exp(x[3]), exp(x[4]) )
220
221 result = optimize(fun, theta, LBFGS(), autodiff=:forward)
222
223 model_theta = result.minimizer
224
225 model_alpha_hat = exp(model_theta[1]) / (1 + exp(model_theta[1]))
226 model_rho_hat = tanh(model_theta[2])
227 model_mu_hat = exp(model_theta[3])
228 model_sigma2_hat = exp(model_theta[4])
229
230 output = [ ["\\est{\\alpha}\\$:", "\\est{\\rho}\\$:", "\\est{\\mu}\\$:", "\\est{\\sigma^2}\\$:" ] [model_alpha_hat,
    ↪ model_rho_hat, model_mu_hat, model_sigma2_hat]

```

$$\begin{aligned}
 \hat{\alpha}_n: & \quad 0.9999999999985967 \\
 \hat{\rho}_n: & \quad 0.0 \\
 \hat{\mu}_n: & \quad 27.626774841787046 \\
 \hat{\sigma}_n^2: & \quad 0.01003725876812115
 \end{aligned}$$

4 b

Equations (4) and (5) read:

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

From part (a) we know that (5) can be recursively solved to yield:

$$z_t \sim \mathcal{N}\left(\mu, \sigma^2 \frac{1 - \rho^{2i}}{1 - \rho}\right)$$

Solving for r_t then taking logs in equation (4)

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

This can be written as:

$$F(r_t, k_t, \alpha, \mu, \sigma, \rho) = 0$$

where the variance of the random variable described by F is known, and the same as the variance of z_t . Thus this system can be estimated by MLE.

```

231 r = convert( Vector{Float64}, macroData[:R] )
232 k = convert( Vector{Float64}, macroData[:K] )
233
234 #log r_t - log alpha - z_t - (alpha - 1) log k_t = 0
235
236 function LogLikelihood( N, w::Vector{Float64}, k::Vector{Float64}, alpha::Real, rho::Real, mu::Real, sigma2::Real )
237     #The pdf of a normal: 1/sqrt(2*pi*sigma^2) * exp(-(x-mu)^2/(2*sigma^2))
238     #Log Likelihood: -1/2 log sigma^2 - (x-mu)^2/(2*sigma^2)
239
240     logLik = 0.0
241     #Note the way that the model is structured is: F(...) = 0, so we
242     #are maximizing the likelihood of getting a 0 returned for all the
243     #moments
244
245     for i in 1:N
246         mean = log(r[i]) - log( alpha ) - mu - (alpha - 1)*log( k[i] )
247         var = sigma^2 * ( 1 - rho^(2*i) ) / ( 1 - rho )
248         logLik += -.5*log( sigma^2 ) - ( mean*mean / (2*sigma^2) )
249     end
250     return logLik
251 end
252
253 N = length(w)
254
255 alpha = .5
256 rho = .99
257 mu = 1.0
258 sigma = 1.0
259 rho = .99
260
261 #We parameterize each of the variables so that they meet their constraints.
262 # tanh is used to ensure that rho in (-1,1)
263 theta = zeros(4)

```

```

264  θ[1] = log( α₀ / ( 1 - α₀ ) )
265  θ[2] = atanh( ρ₀ )
266  θ[3] = log( μ₀ )
267  θ[4] = log( σ₀ )
268
269
270  fun(x::Vector) = -LogLikelihood( N, w, k, exp(x[1]) / (1 + exp(x[1])), tanh(x[2]), exp(x[3]), exp(x[4]) )
271
272  result = optimize(fun, θ, Newton(), autodiff=:forward)
273
274  model_θ = result.minimizer
275
276  model_α̂ = exp(model_θ[1]) / (1 + exp(model_θ[1]))
277  model_ρ̂ = tanh(model_θ[2])
278  model_μ̂ = exp(model_θ[3])
279  model_σ̂ = exp(model_θ[4])
280
281  output = [["\\est{\\alpha}\\$:", "\\est{\\rho}\\$:", "\\est{\\mu}\\$:", "\\est{\\sigma^{2}}\\$:"] [model_α̂,
↪ model_ρ̂, model_μ̂, model_σ̂]]

```

$\hat{\alpha}_n$:	0.8887650406380285
$\hat{\rho}_n$:	0.99
$\hat{\mu}_n$:	1.8877579805483233
$\hat{\sigma}_n^2$:	0.009515136163054447

4.1 c

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

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

```

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

```

Prob 0.21644022445230773