

```

1 | # -----
2 | # Author: Philip Coyle
3 | # Date Created: 06/26/2020
4 | # ps4_Q1.jl
5 | # -----
6 | using Distributed
7 |
8 | # Add cores for parallization
9 | if Sys.isapple()
10 |     addprocs(5)
11 | elseif Sys.islinux()
12 |     addprocs(19)
13 | end
14 |
15 | @everywhere using CSV, Optim, Random, LinearAlgebra, Statistics, SharedArrays
16 |
17 | # Get & Sort data
18 | function get_data()
19 |     # Get & Sort data
20 |     data = CSV.read("/home/p/pcoyle/CodingBootcamp/ProblemSets/PS4/lwage.csv")
21 |
22 |     lwage = data[:,1]
23 |     college = data[:,2]
24 |     experience = data[:,3]
25 |     experience2 = experience.^2
26 |     n = length(lwage)
27 |
28 |     Y = lwage
29 |     X = [ones(n) college experience experience2]
30 |
31 |     return X, Y, n
32 | end
33 |
34 | @everywhere function solve_mle(X::Array{Float64, 2}, Y::Array{Float64, 1},
35 | •   β_in::Array{Float64, 1}, n::Int64)
36 |     nvar = size(X,2)
37 |     func(vars) = Log_Likelihood(X, Y, vars[1:nvar], vars[nvar + 1], n)
38 |     opt = optimize(func, β_in, NelderMead())
39 |     β_opt = opt.minimizer
40 |     β_opt[end] = exp(β_opt[end])
41 |     return β_opt
42 | end
43 |
44 | @everywhere function Log_Likelihood(X, Y, β, log_σ, n)
45 |     σ = exp(log_σ)
46 |     llike = -n/2*log(2*π) - n*log(σ) - sum((Y - X * β).^2)/(2σ^2)
47 |     llike = -llike
48 |
49 |     return llike
50 | end
51 |
52 | function bootstrap_mle(X::Array{Float64, 2}, Y::Array{Float64, 1}, β::Array{Float64,
53 | •   1}, n::Int64, nsim::Int64)
54 |     nvar = size(X,2)
55 |     n_boot = convert{Int64,floor(n/2)}
56 |     β_boot = zeros(nvar+1, nsim)
57 |     β_in = β
58 |
59 |     for i = 1:nsim
60 |         println("At bootstrap simulation ", i, " of ", nsim)
61 |         boot_inx = randperm(n)[1:n_boot]
62 |         Y_boot = Y[boot_inx]
63 |         X_boot = X[boot_inx,:]
64 |         β_boot[i, :] = solve_mle(X_boot, Y_boot, β_in, log_σ)

```

```

64          $\beta_{mle} = \text{solve\_mle}(X\_boot, Y\_boot, \beta\_in, n\_boot)$ 
65          $\beta\_boot[:,i] = \beta_{mle}$ 
66
67          $\beta\_in = \beta_{mle}$ 
68     end
69
70     return  $\beta\_boot$ 
71 end
72
73 function bootstrap_mle_parallel(X::Array{Float64, 2}, Y::Array{Float64, 1},
74 •  $\beta::\text{Array}\{\text{Float64}, 1\}$ , n::Int64, nsim::Int64)
75     nvar = size(X,2)
76     n_boot = convert(Int64,floor(n/2))
77      $\beta\_boot = \text{SharedArray}\{\text{Float64}\}(nvar+1,nsim)$ 
78      $\beta\_in = \beta$ 
79
80     @sync @distributed for i = 1:nsim
81         println("At bootstrap simulation ", i, " of ", nsim)
82         boot_inx = randperm(n)[1:n_boot]
83         Y_boot = Y[boot_inx]
84         X_boot = X[boot_inx,:]
85
86          $\beta_{mle} = \text{solve\_mle}(X\_boot, Y\_boot, \beta\_in, n\_boot)$ 
87          $\beta\_boot[:,i] = \beta_{mle}$ 
88
89          $\beta\_in = \beta_{mle}$ 
90     end
91
92     return  $\beta\_boot$ 
93 end
94
95 X, Y, n = get_data()
96  $\beta_{mle} = \text{solve\_mle}(X, Y, \text{ones}(5), n)$ 
97
98 println("Running Bootstrap Simulation on 1 core")
99 @time  $\beta\_boot = \text{bootstrap\_mle}(X, Y, \beta_{mle}, n, 100)$ 
100
101 println("Running Simulation on 20 cores")
102 @time  $\beta\_boot = \text{bootstrap\_mle\_parallel}(X, Y, \beta_{mle}, n, 100)$ 
103
104  $\sigma = \text{std}(\beta\_boot, \text{dims}=2)$ 
105

```