1. See the QuadLL2 function in the julia code attached below.
2. For $T_i = 1$, the probability can be written as follows

$$Pr\left(\eta_{i0} < \frac{-\alpha_0 - X_i\beta - Z_{it}\gamma}{\sigma_0}\right) \tag{1}$$

For $T_i = 2$, the probability is

$$Pr\left(\epsilon_{i0} < \alpha_0 + X_i\beta + Z_{it}\gamma, \epsilon_{i1} < -\alpha_1 - X_i\beta - Z_{it}\gamma\right) \tag{2}$$

The conditional probability can be written as follows

$$\begin{aligned}
&Pr\left(\epsilon_{i1} < -\alpha_1 - X_i\beta - Z_{it}\gamma | \epsilon_{i0} < \alpha_0 + X_i\beta + Z_{it}\gamma\right) \\
&= Pr\left(\eta_{i0} + \rho\sigma_0\eta_{i1} < -\alpha_1 - X_i\beta - Z_{it}\gamma | \eta_{i0} < \frac{\alpha_0 + X_i\beta + Z_{it}\gamma}{\sigma_0}\right) \\
&= Pr\left(\eta_{i1} < \frac{-\alpha_1 - X_i\beta - Z_{it}\gamma - \eta_{i0}^*}{\rho\sigma_0}\middle| \eta_{i0} < \frac{\alpha_0 + X_i\beta + Z_{it}\gamma}{\sigma_0}\right)
\end{aligned}$$

where η^* is a random variable from $\eta_{i0} < \frac{\alpha_0 + X_i\beta + Z_{it}\gamma}{\sigma_0}$. Then, the probability of $T_i = 2$ is

$$Pr\left(\eta_{i1} < \frac{-\alpha_1 - X_i\beta - Z_{it}\gamma - \eta_{i0}^*}{\rho\sigma_0}\middle| \eta_{i0} < \frac{\alpha_0 + X_i\beta + Z_{it}\gamma}{\sigma_0}\right) Pr\left(\eta_{i0} < \frac{\alpha_0 + X_i\beta + Z_{it}\gamma}{\sigma_0}\right) \tag{3}$$

Similarly, we can define probability in $T_i = 3, 4$. Based on these probabilities, we can define the log likelihood. The funciton is GHKLL2.

3. See the function AcceptRejectLL()
4. We got the following likelihoods:
   - Quadrature method:-40992
   - GHK method:-60572 (Changes somewhat each time the function is called)
   - Accept/Reject method:-75167 (Changes somewhat each time the function is called)

   The calculated likelihood is different under each method, but the order (-1e5) is the same.
5. We got the following result.

$$\begin{aligned}
\alpha_0 = 3.12, \alpha_1 = 0.88, \alpha_2 = 2.29 \\
score_0 = 0.00, ratespread = -0.25, largeloan = -0.79, mediumloan = -0.39 \\
irefinance = -0.04, ager = -0.42, cltv = 0.34, dti = 0.67, cu = -0.43 \\
firstmort = 0.59, iFHA = -0.05, openyear2 = -0.70, openyear3 = 0.07 \\
openyear4 = 0.12, openyear5 = 0.02 \\
score0 = 0.30, score1 = -0.16, score2 = 0.35, \rho = 0.57
\end{aligned}$$

Appendix

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

```
#==
    This file conducts the analyses for JF's PS2
==#

using StatFiles, DataFrames, Optim, BenchmarkTools, Latexify, CSV
# We can use BenchmarkTools for better precision. Just need to replace time
# with btime. The runtime of the overall code get longer as btime runs the
# code multiple times to reduce noise

#include("./PS2b/JuliaCode/functions.jl")
include("./functions.jl")

## load the mortgage data and sparse grid weights as a DataFrames (and
## convert weights to matrices
df = DataFrame(StatFiles.load("PS2b/data/Mortgage_performance_data.dta"))
w1 = DataFrame(CSV.File("PS2b/data/KPU_d1_l20.csv")) |> Matrix
w2 = DataFrame(CSV.File("PS2b/data/KPU_d2_l20.csv")) |> Matrix

# Use this if you are loading data from the root folder.
df = DataFrame(StatFiles.load("../data/Mortgage_performance_data.dta"))
w1 = DataFrame(CSV.File("../data/KPU_d1_l20.csv")) |> Matrix
w2 = DataFrame(CSV.File("../data/KPU_d2_l20.csv")) |> Matrix

#df = DataFrame(CSV.File("C:/Users/ryana/OneDrive/Documents/School/PhD Economics/Research/GitHub/ECON-899/PS1b/

## Separate data into independent variable matrices X and Z and
## dependent variable vector Y
X = df[!, [:score_0, :rate_spread, :i_large_loan, :i_medium_loan,
    :i_refinance, :age_r, :cltv, :dti, :cu,
    :first_mort_r, :i_FHA, :i_open_year2,
    :i_open_year3, :i_open_year4, :i_open_year5]] |> Matrix;

Z = df[!, [:score_0, :score_1, :score_2]] |> Matrix;

Y = df[!, :duration]; #|> Matrix


for name in names(df)
    println(name)
end
## 1. Evaluate log-likelihood using the quadrature method
println("See QuadLL() function")

## 2. Evaluate simulated log-likelihood function using GHK
println("see GHKLL() function")

## 3. Evaluate simulated log-likelihood function using accept/reject
println("see AcceptRejectLL() function")

## 4. Compare predicted choice probabilities for each of the above methods
θ₀ = [0, -1, -1, 0 * ones(size(X, 2), 1), 0.3 * ones(size(Z, 2), 1), 0.5]

ll_quad=QuadLL2(Y, X, Z, w1, w2, θ₀)
ll_ghk=GHKLL2(Y, X, Z, θ₀)
ll_ar=AcceptRejectLL(Y, X, Z, θ₀)

## 5. Maximize quadrature log-likelihood function using BFGS
θ₀ = vcat([0, -1, -1], 0 * ones(size(X, 2), 1), 0.3 * ones(size(Z, 2), 1), [0.5])

θ = optimize(t -> -QuadLL2(Y, X, Z, w1, w2,
```

```
                              [t[1],t[2],t[3],t[4:(3+size(X,2))],
                               t[(4+size(X,2)):(3+size(X,2)+size(Z,2))],
                               t[(4+size(X,2)+size(Z,2))]]), θ₀,
               method = BFGS(), f_tol = 1e-5, g_tol = 1e-5).minimizer
```

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

```
#==
This file defines functions used in JF's PS2
==#
using Optim, Distributions, Parameters, LinearAlgebra

# structure of model parameters
mutable struct ModelParameters
    α₀::Float64
    α₁::Float64
    α₂::Float64
    β::Array{Float64}
    γ::Array{Float64}
    ρ::Float64
end # parameters struct

# Calculate log-likelihood using quadrature method
function QuadLL2(Y, X, Z, W1, W2, θ)

    u = W1[:, 1]; w = W1[:, 2]
    μ₀ = W2[:, 1]; μ₁ = W2[:, 2]; ω = W2[:, 3]

    param = ModelParameters(θ[1], θ[2], θ[3], θ[4], θ[5], θ[6])
    @unpack α₀, α₁, α₂, β, γ, ρ = param

    # Calculate σ₀ and σ₀²
    σ₀² = 1/(1-ρ)^2
    σ₀  = 1/(1-ρ)

    tmp = α₀ .+ X*β .+ Z*γ;
    mρ = zeros(size(X,1), size(u,1));

    for i in 1:size(X,1) # For each observation, get range based on domain (0,1) at t = 0
        mρ[i,:] = log.(u') .+ tmp[i]
    end

    tmp = α₁ .+ X*β .+ Z*γ
    mρ1 = zeros(size(X,1), size(u,1));

    for i in 1:size(X,1) # For each observation, get range based on domain (0,1) at t = 1
        mρ1[i,:] = log.(u') .+ tmp[i]
    end

    mdρ = ones(size(mρ,1), size(mρ,2));

    for i in 1:size(X,1) # For each observation, get Jacobian 1/u
        mdρ[i,:] = mdρ[i,:] ./u
    end

    L1 = cdf.(Normal(), (-α₀ .- X*β .- Z*γ)./σ₀)

    density = pdf.(Normal(), mρ./σ₀)./σ₀

    L2 = (cdf.(Normal(), - α₁ .- X*β .- Z*γ .- ρ .* mρ) .* density.* mdρ) * w

    density = pdf.(Normal(), mρ1 - ρ*mρ) .* pdf.(Normal(), mρ./σ₀) ./ σ₀
```

```
    L3 = (cdf.(Normal(), - α₂ .- X*β .- Z*γ .- ρ .* mρ1) .* density .* mdρ .* mdρ) *
w

    L4 = 1 .- L1 .- L2 .- L3

    ll = 0

    for i = 1:size(Y, 1)

        if Y[i] == 1

            ## If the likelihood becomes minus, I evaluate this value as 1e-10.
            if L1[i] < 0
                L1[i]=1e-10
            else
            end

            ll = ll + log(L1[i])

        elseif Y[i] == 2

            if L2[i] < 0
                L2[i]=1e-10
            else

            end

            ll = ll + log(L2[i])

        elseif Y[i] == 3

            if L3[i] < 0
                L3[i]=1e-10
            else
            end

            ll = ll + log(L3[i])

        elseif Y[i] == 4

            if L4[i] < 0
                L4[i]=1e-10
            else
            end

            ll = ll + log(L4[i])
        end

    end # for i

    return(ll)

end

# Calculate log-likelihood using quadrature method
function QuadLL(Y, X, Z, W1, W2, θ)
    # separate weights and nodes from W1 and W2
    u = W1[:, 1]; w = W1[:, 2]
    μ₀ = W2[:, 1]; μ₁ = W2[:, 2]; ω = W2[:, 3]

    # unpack model parameters
    param = ModelParameters(θ[1], θ[2], θ[3], θ[4], θ[5], θ[6])
    @unpack α₀, α₁, α₂, β, γ, ρ = param

    # Calculate σ₀ and σ₀^2
    σ₀^2 = 1/(1-ρ)^2
    σ₀  = 1/(1-ρ)
```

```
# map integral bounds to [0, 1] Upper bound (-Infinity, α + Xβ + Zγ)
# ρ(u) = ln(u) + α + Xβ + Zγ

b₀ = (a, x, z) -> log.(a) .+ (α₀ + dot(x, β) + dot(z, γ))
b₁ = (a, x, z) -> log.(a) .+ (α₁ + dot(x, β) + dot(z, γ))

# per-observation likelihood:

L1 = (x, z) -> log(cdf(Normal(), (-α₀ - dot(x, β) - dot(z, γ))/σ₀))
# I think this should just be σ₀ not σ₀²

# I think the location of the () about σ₀ wrong.
#L2 = (x, z) -> log(sum(w.*
#    (cdf.(Normal(), (-ρ)*b₀(u, x, z) .- (α₁ .+ dot(x, β) + dot(z, γ)))./σ₀).*
#    pdf.(Normal(), b₀(u./σ₀, x, z)) ./ u))

function L2(x, z)
    out=0
    try
        out=log(sum(w.*
            (cdf.(Normal(), -α₁ .- dot(x, β) .- dot(z, γ) .-ρ*b₀(u, x, z))).* # Function
            (pdf.(Normal(), b₀(σ₀ .* u, x, z)./σ₀)./σ₀) .* # Density
            (1 ./u))) # Jacobian

    catch #Sometimes parameters will be tried that make the above try to take the log
        #of a negative number. This is an attempted fix for that which just returns something
        #awful in that case so that it won't be picked
        out=log(1e-5)
        print("Attempted a point that does not work.")
    end
    return out
end


function L3(x, z)
    out=0
    try
        out=log(sum(ω.*((cdf.(Normal(), (-ρ)*b₁(μ₁, x, z) .- (α₂ .+ dot(x, β)
+
            dot(z, γ)))./σ₀).*pdf.(Normal(), b₀(μ₀, x, z)./σ₀).*pdf.(Normal(), b₁(μ₁, x, z).-
            ρ*b₀(μ₀, x, z)) ./ (μ₀ .* μ₁)))
    catch
        out=log(1e-5)
        print("Attempted a point that does not work.")
    end
    return out
end
function L4(x, z)
    out=0
    try
        out=log(sum(ω.*(cdf.(Normal(), -(ρ)*b₁(μ₁, x, z) .+ (α₂ + dot(x, β) +
            dot(z, γ)))./σ₀).*pdf.(Normal(), b₁(μ₁./σ₀, x, z)).*pdf.(Normal(), b₁(μ₁, x, z) .-
            ρ*b₀(μ₀, x, z)) ./ (μ₀ .* μ₁)))
    catch
        out=log(1e-5)
        print("Attempted a point that does not work.")
    end
    return out
end

# calculate the log-likelihood for all observations
ll = 0
for i = 1:size(Y, 1)
    if Y[i] == 1
        ll = ll + L1(X[i, :], Z[i, :])
    elseif Y[i] == 2
```

```
            ll = ll + L2(X[i, :], Z[i, :])
        elseif Y[i] == 3
            ll = ll + L3(X[i, :], Z[i, :])
        elseif Y[i] == 4
            ll = ll + L4(X[i, :], Z[i, :])
        end
    end # for i

    return ll
end # quadrature log-likelihood function



# Calculate log-likelihood using quadrature method
function GHKLL(Y, X, Z, θ; sims = 100)

    # unpack model parameters
    param = ModelParameters(θ[1], θ[2], θ[3], θ[4], θ[5], θ[6])
    @unpack α0, α1, α2, β, γ, ρ = param
    σ0  = 1/(1-ρ)

    ll = 0
    for i=1:size(Y, 1)
        ll_i = 1
        ϵ_draws = zeros(sims, 3)
        if Y[i] > 1 # Need to draw from a distribution which won't make the borrower repay in period 1
            ϵ_draws[:, 1] = rand.(truncated(Normal(0, σ0), -Inf, -α0 - dot(X[i, :],
β) - dot(Z[i, :], γ)), sims)
        elseif Y[i] == 1 #Find the probability that this draw would have occured
            ll_i=1-cdf(Normal(), (-α0 - dot(X[i, :], β) - dot(Z[i, :], γ))/σ0)
        end
        if Y[i] > 2 # Need to draw from a distribution which won't make the borrower repay in period 2
            ϵ_draws[:, 2] = [rand(truncated(Normal(0, σ0), -Inf, -α0 - dot(X[i, :],
β) - dot(Z[i, :], γ) - ρ*ϵ_draws[si, 1])) for si = 1:sims]
        elseif Y[i] == 2 # Find the probability that this draw would have occured
            ll_i = (1/sims)*cdf(Normal(), (-α0 - dot(X[i, :], β) - dot(Z[i, :], γ))/σ0)*
                sum(1 .- cdf.(Normal(), (-α0 - dot(X[i, :], β) - dot(Z[i, :], γ))
.- ρ*ϵ_draws[:, 1]))
        end
        if Y[i] > 3 # Need to draw from a distribution which won't make the borrower repay in period 3
            ϵ_draws[:, 3]=[rand(truncated(Normal(0, σ0), -Inf, -α0 - dot(X[i, :],
β) - dot(Z[i, :], γ) - (ρ^(2))*ϵ_draws[si, 1] - ρ*ϵ_draws[si, 2])) for si = 1:sims]
            # Find the probability that Y[i]==4 would have occured
            ll_i = (1/sims)*cdf(Normal(), (-α0 - dot(X[i, :], β) - dot(Z[i, :], γ))/σ0)*
                sum((cdf.(Normal(), (-α0 - dot(X[i, :], β) - dot(Z[i, :], γ)) .-
ρ*ϵ_draws[:, 1])).*
                cdf.(Normal(), (-α0 - dot(X[i, :], β) - dot(Z[i, :], γ)) .- (ρ^(2))*ϵ_draws[:, 1] .-
ρ*ϵ_draws[:, 2]))
        elseif Y[i] == 3 # Find the probability that this draw would have occured
            ll_i = (1/sims)*cdf(Normal(), (-α0 - dot(X[i, :], β) - dot(Z[i, :], γ))/σ0)*
                sum(  (cdf.(Normal(), (-α0 - dot(X[i, :], β) - dot(Z[i, :], γ)) .-
ρ*ϵ_draws[:, 1])).*
                (1 .- cdf.(Normal(), (-α0 - dot(X[i, :], β) - dot(Z[i, :], γ)) .-
(ρ^(2))*ϵ_draws[:, 1] .- ρ*ϵ_draws[:, 2]))    )
        end
        ll += log(ll_i)
    end
    return ll
end # GHK log-likelihood function

function GHKLL2(Y, X, Z, θ; sims = 100)
    #ref: http://fmwww.bc.edu/repec/bocode/g/GHK_note.pdf
    param = ModelParameters(θ[1], θ[2], θ[3], θ[4], θ[5], θ[6])
    @unpack α0, α1, α2, β, γ, ρ = param
    σ0  = 1/(1-ρ)

    ll_store = 0
```

```
    for i=1:size(Y, 1)

        if Y[i] == 1
            ll = cdf(Normal(0, 1), (-α0 - dot(X[i, :], β) - dot(Z[i, :], γ))/σ0)
        elseif Y[i] == 2
            η0 = rand.(truncated(Normal(0, 1), -Inf, (α0 + dot(X[i, :], β) + dot(Z[i, :],
γ))./σ0), 100)
            ll = mean(cdf(Normal(0, 1), (-α1 - dot(X[i, :], β) - dot(Z[i, :], γ)
.- η0)./(ρ .* σ0)) .*
                cdf(Normal(0, 1), (α0 + dot(X[i, :], β) + dot(Z[i, :], γ))./(σ0)))
        elseif Y[i] == 3
            η0 = rand.(truncated(Normal(0, 1), -Inf, (α0 + dot(X[i, :], β) + dot(Z[i, :],
γ))./σ0), 100)
            tmp = ((α1 + dot(X[i, :], β) + dot(Z[i, :], γ) .- η0)./σ0)
            η1 = zeros(100)
            for i in 1:100
                η1[i] = (rand.(truncated(Normal(0, 1), -Inf, (tmp[i])[1]), 1))[1]
            end
            ll = mean(cdf(Normal(0, 1), (-α2 - dot(X[1, :], β) - dot(Z[i, :], γ)
.- η0 .- ρ.* η1)./((ρ.^2) .* σ0)) .*
                cdf(Normal(0, 1), (α1 + dot(X[i, :], β) + dot(Z[i, :], γ) .- η1)./(ρ
.* σ0)) .*
                cdf(Normal(0, 1), (α0 + dot(X[i, :], β) + dot(Z[i, :], γ))./(σ0)))
        else

            ll1 = cdf(Normal(0, 1), (-α0 - dot(X[i, :], β) - dot(Z[i, :], γ))/σ0)

            η0 = rand.(truncated(Normal(0, 1), -Inf, (α0 + dot(X[i, :], β) + dot(Z[i, :],
γ))./σ0), 100)
            ll2 = mean(cdf(Normal(0, 1), (-α1 - dot(X[i, :], β) - dot(Z[i, :], γ)
.- η0)./(ρ .* σ0)) .*
                cdf(Normal(0, 1), (α0 + dot(X[i, :], β) + dot(Z[i, :], γ))./(σ0)))

            η0 = rand.(truncated(Normal(0, 1), -Inf, (α0 + dot(X[i, :], β) + dot(Z[i, :],
γ))./σ0), 100)
            tmp = ((α1 + dot(X[i, :], β) + dot(Z[i, :], γ) .- η0)./σ0)
            η1 = zeros(100)
            for j in 1:100
                η1[j] = (rand.(truncated(Normal(0, 1), -Inf, (tmp[j])[1]), 1))[1]
            end
            ll3 = mean(cdf(Normal(0, 1), (-α2 - dot(X[i, :], β) - dot(Z[i, :], γ)
.- η0 .- ρ.* η1)./((ρ.^2) .* σ0)) .*
                cdf(Normal(0, 1), (α1 + dot(X[i, :], β) + dot(Z[i, :], γ) .- η1)./(ρ
.* σ0)) .*
                cdf(Normal(0, 1), (α0 + dot(X[i, :], β) + dot(Z[i, :], γ))./(σ0)))

            ll = 1 - ll1 - ll2 - ll3

        end

        ll_store += log.(ll)

    end

    return(ll_store)

end


# Calculate log-likelihood using accept-reject method
function AcceptRejectLL(Y, X, Z, θ; sims = 100, k = maximum(Y))

    # unpack model parameters
    param = ModelParameters(θ[1], θ[2], θ[3], θ[4], θ[5], θ[6])
    @unpack α0, α1, α2, β, γ, ρ = param
```

```
    σ0 = 1 / (1 - ρ)

    ll = 0 # initialize log-likelihood

    # Define index functions for each outcome
    I1 = (x, z, ε) -> ε .< -(α0 .+ x * β .+ z * γ)
    I2 = (x, z, ε) -> (ε[:, 1] .< -(α0 .+ x * β .+ z * γ)) .& (ε[:, 2] .< -(α1 .+ x * β .+ z * γ) .- ρ * ε[:, 1])
    I3 = (x, z, ε) -> (ε[:, 1] .< -(α0 .+ x * β .+ z * γ)) .& (ε[:, 2] .< -(α1 .+ x * β .+ z * γ) .- ρ * ε[:, 1]) .& (
        ε[:, 3] .< -(α1 .+ x * β .+ z * γ) .- (ρ^2) * ε[:, 1] .- ρ * ε[:, 2])
    I4 = (x, z, ε) -> (ε[:, 1] .< -(α0 .+ x * β .+ z * γ)) .& (ε[:, 2] .< -(α1 .+ x * β .+ z * γ) .- ρ * ε[:, 1]) .& (
        ε[:, 3] .< α1 .+ x * β .+ z * γ .- (ρ^2) * ε[:, 1] .- ρ * ε[:, 2])

    # Calculate log-likelihood for Y = 1 observations
    x, z = repeat(X[Y.==1, :], inner = [sims, 1]), repeat(Z[Y.==1, :], inner = [sims, 1])
    ε = rand.(Normal(0, σ0), size(x, 1))
    for i = 1:sum(Y .== 1)
        ind = ((i-1)*sims+1):(i*sims)
        if sum(I1(x[ind, :], z[ind, :], ε[ind, :]))!=0
            ll += log(1/sum(I1(x[ind, :], z[ind, :], ε[ind, :])))
            #ll += log(sum(I1(x[ind, :], z[ind, :], ε[ind, :])) / sims)
        else
            ll+=log(1/sims) #Act as though it happened at least once to avoid -infinity
        end
    end

    # Calculate log-likelihood for Y = 2 observations
    x, z = repeat(X[Y.==2, :], inner = [sims, 1]), repeat(Z[Y.==2, :], inner = [sims, 1])
    ε = [rand.(Normal(0, σ0), size(x, 1)) rand.(Normal(), size(x, 1))]
    for i = 1:sum(Y .== 2)
        ind = ((i-1)*sims+1):(i*sims)
        if sum(I2(x[ind, :], z[ind, :], ε[ind, :]))!=0
            ll += log(1/sum(I2(x[ind, :], z[ind, :], ε[ind, :])))
            #ll += log(sum(I2(x[ind, :], z[ind, :], ε[ind, :])) / sims)
        else
            ll+=log(1/sims) #Act as though it happened at least once to avoid -infinity
        end
    end
    # Calculate log-likelihood for Y = 3 observations
    x, z = repeat(X[Y.==3, :], inner = [sims, 1]), repeat(Z[Y.==3, :], inner = [sims, 1])
    ε = [rand.(Normal(0, σ0), size(x, 1)) rand.(Normal(), size(x, 1)) rand.(Normal(), size(x, 1))]
    for i = 1:sum(Y .== 3)
        ind = ((i-1)*sims+1):(i*sims)
        if sum(I3(x[ind, :], z[ind, :], ε[ind, :]))!=0
            #ll += log(1/sum(I3(x[ind, :], z[ind, :], ε[ind, :])))
            ll += log(sum(I3(x[ind, :], z[ind, :], ε[ind, :])) / sims)
        else
            ll+=log(1/sims) #Act as though it happened at least once to avoid -infinity
        end
    end
    # Calculate log-likelihood for Y = 4 observations
    x, z = repeat(X[Y.==4, :], inner = [sims, 1]), repeat(Z[Y.==4, :], inner = [sims, 1])
    ε = [rand.(Normal(0, σ0), size(x, 1)) rand.(Normal(), size(x, 1)) rand.(Normal(), size(x, 1))]
    for i = 1:sum(Y .== 4)
        ind = ((i-1)*sims+1):(i*sims)
        if sum(I4(x[ind, :], z[ind, :], ε[ind, :]))!=0
            #ll += log(1/sum(I4(x[ind, :], z[ind, :], ε[ind, :])))
            ll += log(sum(I4(x[ind, :], z[ind, :], ε[ind, :])) / sims)
        else
            ll+=log(1/sims) #Act like it happened at least once to avoid -infinity
        end
    end
    return ll
end # accept-reject log-likelihood function
```