Homework 4 Solutions

BEE 4850/5850

Overview

Instructions

The goal of this homework assignment is to practice simulation-based uncertainty quantification, including the bootstrap and Markov chain Monte Carlo.

- Problem 1 asks you to use the parametric and non-parametric bootstrap to estimate the median of water level data.
- Problem 2 asks you to use the parametric bootstrap to estimate parameter uncertainty in a semi-empirical sea-level rise model.
- Problem 3 (only required for students in BEE 5850) asks you to use the bootstrap to quantify uncertainties in the gender-impact-on-hurricane-damages dataset from Homework 2.

Load Environment

The following code loads the environment and makes sure all needed packages are installed. This should be at the start of most Julia scripts.

```
import Pkg
Pkg.activate(@__DIR__)
Pkg.instantiate()
```

The following packages are included in the environment (to help you find other similar packages in other languages). The code below loads these packages for use in the subsequent notebook (the desired functionality for each package is commented next to the package).

```julia
using Random # random number generation and seed-setting
using DataFrames # tabular data structure
using DataFramesMeta # API which can simplify chains of DataFrames
↪ transformations
using CSV# reads/writes .csv files
using Distributions # interface to work with probability distributions
using Plots # plotting library
using StatsBase # statistical quantities like mean, median, etc
using StatsPlots # some additional statistical plotting tools
using Optim
using LaTeXStrings
using Dates

Random.seed!(1)
```

Problems

Scoring

- Problem 1 is worth 10 points;
- Problem 2 is worth 10 points;
- Problem 3 is worth 5 points.

Problem 1

Let's load the data from 'data/salamanders.csv'.

```julia
dat = CSV.read(joinpath("data", "salamanders.csv"), DataFrame, delim=";")
```

	SITE	SALAMAN	PCTCOVER	FORESTAGE
	Int64	Int64	Int64	Int64
1	1	13	85	316
2	2	11	86	88
3	3	11	90	548
4	4	9	88	64
5	5	8	89	43
6	6	7	83	368
7	7	6	83	200
8	8	6	91	71
9	9	5	88	42
10	10	5	90	551
11	11	4	87	675
12	12	3	83	217
13	13	3	87	212
14	14	3	89	398
15	15	3	92	357
16	16	3	93	478
17	17	2	2	5
18	18	2	87	30
19	19	2	93	551
20	20	1	7	3
21	21	1	16	15
22	22	1	19	31
23	23	1	29	10
24	24	1	34	49
…	…	…	…	…

Since we're using a Poisson model, the model specification is

$$\begin{aligned} y &\sim \text{Poisson}(\lambda) \\ \log(\lambda) &= ax + b, \end{aligned}$$

where x is the percent groundcover of the plot and y is the salamander count.

Let's find the maximum likelihood.

```julia
function salamander_pcover(p, counts, pctcover)
    a, b = p
    λ = exp.(a * pctcover .+ b)
    ll = sum(logpdf.(Poisson.(λ), counts))
    return ll
end
```

```julia
lb = [-10.0, -50.0]
ub = [10.0, 50.0]
p0 = [0.0, 0.0]

# function to make standardizing the predictor more convenient
stdz(x) = (x .- mean(x)) / std(x)

result = optimize(p -> -salamander_pcover(p, dat.SALAMAN,
↪  stdz(dat.PCTCOVER)), lb, ub, p0)
θ_mle = result.minimizer
```

```
2-element Vector{Float64}:
 1.1595061318756
 0.42951129115546005
```

Each bootstrap replicate consists of a new dataset formed by resampling plot datum, to which we repeat the above analysis to refit the model.

```julia
nboot = 1_000
θ_boot = zeros(nboot, 2) # storage for bootstrap replicates
for i = 1:nboot
    idx = sample(1:nrow(dat), nrow(dat), replace=true)
    boot_dat = dat[idx, :]
    result = optimize(p -> -salamander_pcover(p, boot_dat.SALAMAN,
↪   stdz(boot_dat.PCTCOVER)), lb, ub, p0)
    θ_boot[i, :] = result.minimizer
end
# show the bootstrap mean
θ̂ = mean(θ_boot; dims=1)
@show θ̂;
```

```
θ̂ = [1.1794723089593893 0.38099303881850943]
```

We can now visualize the bootstrapped sampling distributions and compare to the MLE.

```julia
p1 = histogram(θ_boot[:, 1], xlabel="a", ylabel="Count",
↪   fillcolor=:white, label=false)
vline!(p1, [θ_mle[1]], color=:red, lw=2, label="MLE")
vline!(p1, [θ̂[1]], color=:purple, lw=2, label="Bootstrap Mean")
plot!(p1, size=(400, 400))
```

```
p2 = histogram(θ_boot[:, 2], xlabel="b", ylabel="Count",
↪  fillcolor=:white, label=false)
vline!(p2, [θ_mle[2]], color=:red, lw=2, label="MLE")
vline!(p2, [θ̂[2]], color=:purple, lw=2, label="Bootstrap Mean")
plot!(p2, size=(400, 400))

display(p1)
display(p2)
```

We can see from Figure **??** that both of the bootstrapped distributions are skewed, but that the estimates only have a small bias (recall that the bootstrap estimate of bias of a statistic t is $\mathbb{E}[\tilde{t}] - \hat{t}$, where $\hat{t}$ is the MLE and $\tilde{t}$ is a bootstrap estimate); a has a bias of 0.02 and b has a bias of -0.05.

To find the 90% confidence intervals, we use the basic bootstrap formula

$$\left(\hat{t} - (Q_{\tilde{t}}(1 - \alpha/2) - \hat{t}), \hat{t} - (Q_{\tilde{t}}(\alpha/2) - \hat{t})\right),$$

where $\alpha = 0.1$.

```
a_q = quantile(θ_boot[:, 1], [0.95, 0.05])
b_q = quantile(θ_boot[:, 2], [0.95, 0.05])

a_ci = (2 * θ_mle[1] - a_q[1], 2 * θ_mle[1] - a_q[2])
b_ci = (2 * θ_mle[2] - b_q[1], 2 * θ_mle[2] - b_q[2])
@show round.(a_ci, digits=2);
@show round.(b_ci, digits=2);
```

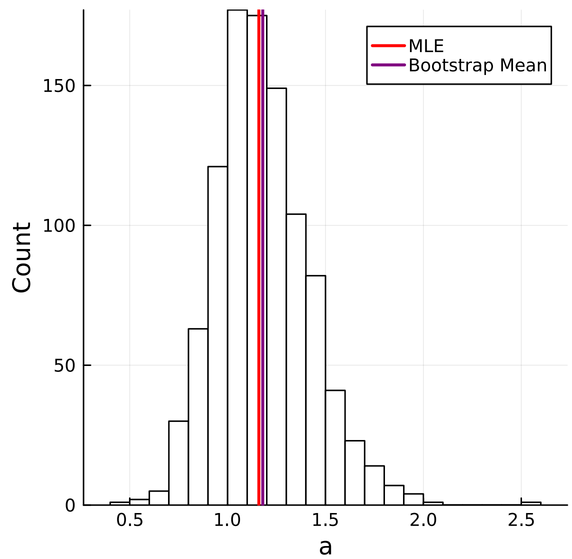
```
round.(a_ci, digits = 2) = (0.72, 1.49)
round.(b_ci, digits = 2) = (0.1, 0.91)
```

One important note is that your answers might differ a bit from these: the confidence interval estimates are a bit sensitive to the random bootstrap replicates with this number of replicates, but the answer should be roughly similar to this.

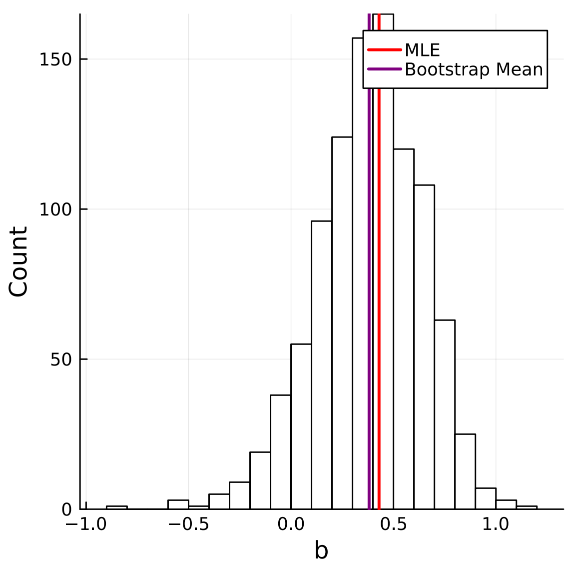
Problem 2

First, load the data

```
read: Connection reset by peer
send: Broken pipe
```

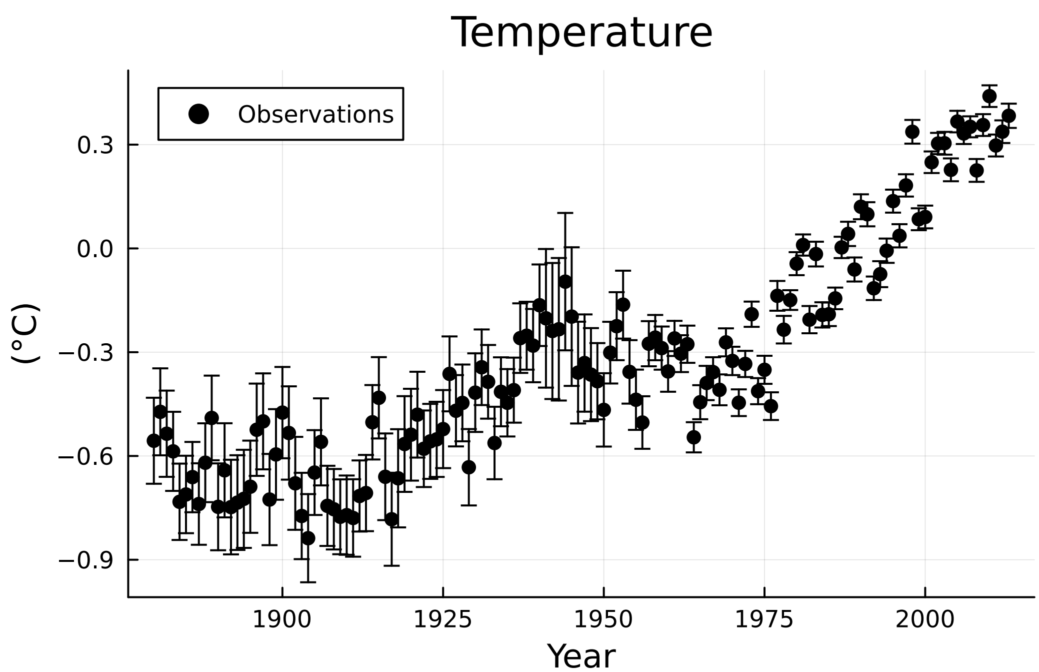


(a) Bootstrapped estimates of parameter values for Problem 1. The red line is the MLE for the given parameter and the purple line is the bootstrap mean.



(b)

```
norm_yrs = 1980:1999

sl_dat = DataFrame(CSV.File(joinpath("data",
↪  "CSIRO_Recons_gmsl_yr_2015.csv")))

rename!(sl_dat, [:Year, :GMSLR, :SD]) # rename to make columns easier to
↪  work with
sl_dat[!, :Year] .-= 0.5 # shift year to line up with years instead of
↪  being half-year
sl_dat[!, :GMSLR] .-= mean(filter(row -> row.Year ∈ norm_yrs, sl_dat)[!,
↪  :GMSLR]) # rescale to be relative to 1880-1900 mean for consistency
↪  with temperature anomaly

# load temperature data
temp_dat = DataFrame(CSV.File(joinpath("data",
↪  "HadCRUT.5.0.2.0.analysis.summary_series.global.annual.csv")))
rename!(temp_dat, [:Year, :Temp, :Lower, :Upper]) # rename to make
↪  columns easier to work with
filter!(row -> row.Year ∈ sl_dat[!, :Year], temp_dat) # reduce to the
↪  same years that we have SL data for
temp_normalize = mean(filter(row -> row.Year ∈ norm_yrs, temp_dat)[!,
↪  :Temp]) # get renormalization to rescale temperature to 1880-1900
↪  mean
temp_dat[!, :Temp] .-= temp_normalize
temp_dat[!, :Lower] .-= temp_normalize
temp_dat[!, :Upper] .-=  temp_normalize

sl_plot = scatter(sl_dat[!, :Year], sl_dat[!, :GMSLR], yerr=sl_dat[!,
↪  :SD], color=:black, label="Observations", ylabel="(mm)",
↪  xlabel="Year", title="Sea Level Anomaly")

temp_plot = scatter(temp_dat[!, :Year], temp_dat[!, :Temp],
↪  yerr=(temp_dat[!, :Temp] - temp_dat[!, :Lower], temp_dat[!, :Upper]
↪  - temp_dat[!, :Temp]), color=:black, label="Observations",
↪  ylabel="(°C)", xlabel="Year", title="Temperature")
```



The Grinsted model in code (skipping past the discretization since we did this in HW2):

```
function grinsted_slr(params, temps; Δt=1)
    a, b, τ, S₀ = params
    S = zeros(length(temps)) # initialize storage
    Seq = a * temps .+ b
    S[1] = S₀
    for i = 2:length(S)
        S[i] = S[i-1] +  Δt * (Seq[i] - S[i-1]) / τ
    end
    return S[1:end]
end
```

```
grinsted_slr (generic function with 1 method)
```

The log likelihood function with AR(1) residuals:

```
function ar1_loglik(params, temp_dat, slr_obs, Δt=1.0)
    a, b, τ, S₀, ρ, σ = params
    slr_sim = grinsted_slr((a, b, τ, S₀), temp_dat; Δt = Δt)
    # whiten residuals
    resids = slr_obs .- slr_sim
    ll = 0
```

```
    for t = 1:length(temp_dat) - 1
        if t == 1
            ll += logpdf(Normal(0, sqrt(σ^2 / (1 - ρ^2))), resids[t])
        else
            ll += sum(logpdf(Normal(ρ * resids[t], σ), resids[t+1]))
        end
    end
    return ll
end
```

```
ar1_loglik (generic function with 2 methods)
```

Now, let's find the MLE by optimizing the `ar1_loglik` function.

```
low_bds = [-2000.0, -2000.0, 0.1, sl_dat.GMSLR[1] - 1.96 * sl_dat.SD[1],
 ↪  -1.0, 0.1]
up_bds = [20_000.0, 30_000.0, 10_000.0, sl_dat.GMSLR[1] + 1.96 *
 ↪  sl_dat.SD[1], 0.99, 100.0]
p₀ = [10.0, 0.0, 1500.0, sl_dat.GMSLR[1], 0.0, 10.0]

mle_optim = optimize(p -> -ar1_loglik(p, temp_dat.Temp, sl_dat.GMSLR),
 ↪  low_bds, up_bds, p₀)
p_mle = mle_optim.minimizer
@show p_mle;
```

```
p_mle = [546.9549427127922, 410.3873816251446, 179.69518456677608,
-158.18775262248883, 0.5215568507538185, 4.960943964640367]
```

We can plot the fitted model and the residuals:

```
slr_fit = grinsted_slr(p_mle[1:end-2], temp_dat.Temp)
resids = sl_dat.GMSLR - slr_fit

pfit = plot(sl_dat.Year, slr_fit, color=:blue, xlabel="Year",
 ↪  ylabel="Global Mean Sea Level (mm)", label="Model Fit")
scatter!(pfit, sl_dat.Year, sl_dat.GMSLR, color=:black,
 ↪  label="Observations")
plot!(pfit, size=(400, 400))

presids = scatter(sl_dat.Year, resids, color=:black, xlabel="Year",
 ↪  ylabel="Model Residuals (mm)", legend=false)
plot!(presids, size=(400, 400))
```

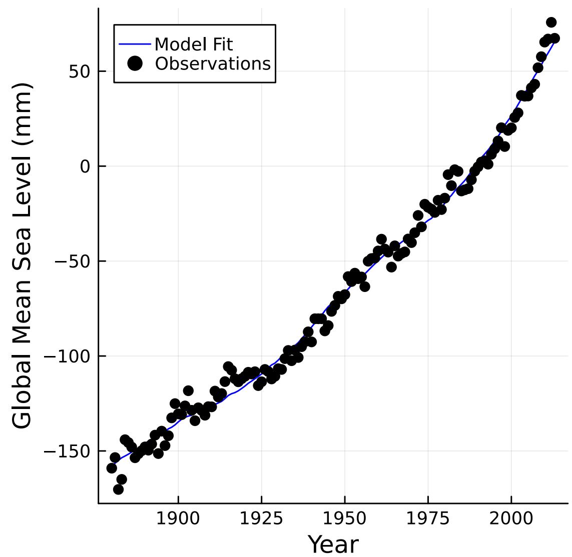
```julia
display(pfit)
display(presids)
```

Now we want to generate 1,000 replicates of the residuals using the fitted AR(1) process and add them back to the model hindcast.

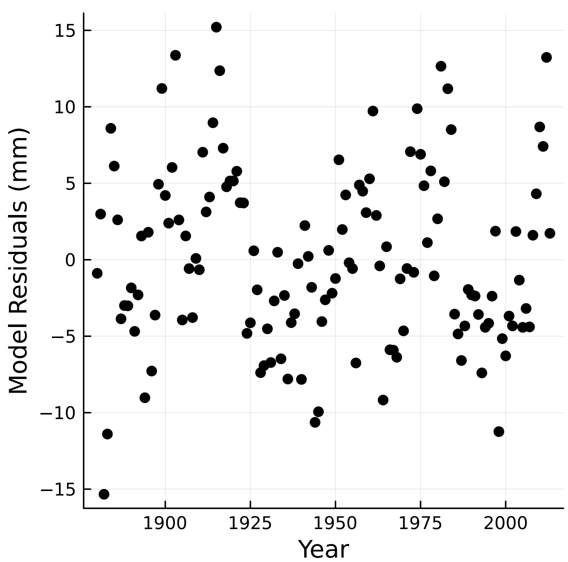
```julia
function ar1_sim(ρ, σ, n_sim, T)

    sim_out = zeros(n_sim, T)
    for t = 1:T
        if t == 1
           sim_out[:, t] = rand(Normal(0, sqrt(σ^2 / (1 - ρ^2))), n_sim)
        else
            sim_out[:, t] = rand.(Normal.(ρ * sim_out[:, t-1], σ))
        end
    end
    return sim_out
end

n_sim = 1_000
resids_boot = ar1_sim(p_mle[end-1], p_mle[end], n_sim, nrow(temp_dat))
slr_boot = mapslices(row -> row .+ slr_fit, resids_boot; dims=2)
```

```
1000×134 Matrix{Float64}:
 -155.578  -160.371  -150.263  …  65.6277  66.8024  60.9281  66.3092
 -166.817  -150.491  -144.177     50.7711  53.8146  58.418   54.321
 -159.91   -162.683  -163.252     50.7926  68.1767  61.0413  63.3705
 -162.234  -151.419  -150.336     56.3515  66.5042  65.2169  70.7606
 -154.785  -153.74   -152.474     53.0952  59.1195  62.5341  61.0469
 -158.985  -157.702  -154.061  …  53.8543  58.2929  60.5728  58.7454
 -161.992  -149.346  -157.096     57.0526  65.5743  61.8247  55.8163
 -157.933  -162.833  -164.287     52.874   55.6301  63.7023  70.2268
 -157.496  -149.485  -152.446     48.1722  47.95    52.4044  63.0504
 -145.989  -149.469  -149.711     64.1377  57.1585  53.9137  51.6984
    ⋮                          ⋱   ⋮
 -152.459  -161.553  -158.747     61.4694  70.3682  68.5933  74.4939
 -163.098  -154.776  -146.78      59.051   64.7032  62.6031  66.4458
 -156.917  -158.073  -143.377     64.4128  63.236   70.6126  71.2534
 -158.449  -159.636  -159.65      62.12    60.2227  61.7398  67.2607
 -151.149  -152.926  -151.828  …  55.7968  62.4473  62.726   64.2052
 -159.789  -161.955  -157.092     63.7249  59.6974  61.7888  64.13
```

```
read: Connection reset by peer
send: Broken pipe
```



(a) Sea-level hindcast using the fitted Grinsted model and the model residuals.



(b)

```
 -159.274  -154.538  -146.59      54.7705  62.0086  53.6441  63.3536
 -164.655  -157.275  -162.345     46.6267  49.4417  59.9164  62.2635
 -151.977  -153.05   -153.809     50.9142  62.5265  58.7918  65.9182
```

Now we refit the model to each replicate to get the estimates of the bootstrap parameters.

```
a_boot = zeros(n_sim)
b_boot = zeros(n_sim)
τ_boot = zeros(n_sim)
S₀_boot = zeros(n_sim)
ρ_boot = zeros(n_sim)
σ_boot = zeros(n_sim)

for i = 1:n_sim
    mle_optim = optimize(p -> -ar1_loglik(p, temp_dat.Temp, slr_boot[i,
 ↪  :]), low_bds, up_bds, p₀)
    a_boot[i], b_boot[i], τ_boot[i], S₀_boot[i], ρ_boot[i], σ_boot[i] =
 ↪  mle_optim.minimizer
end
```

Plotting the histograms:

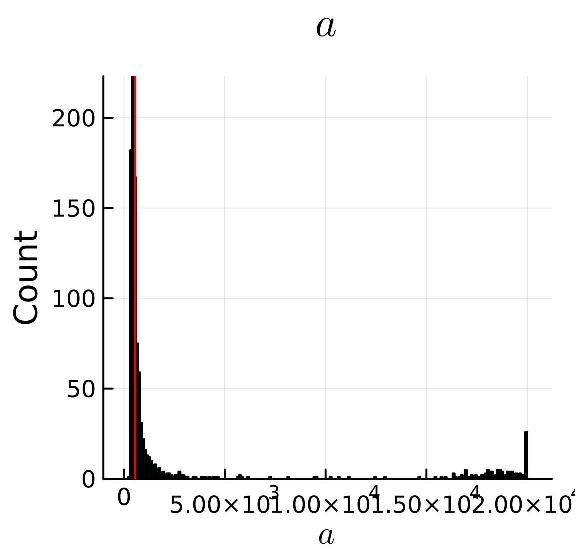
```
hist1 = histogram(a_boot, xlabel=L"$a$", ylabel="Count", legend=false,
 ↪  title=L"$a$")
vline!(hist1, [p_mle[1]], color=:red)
plot!(hist1, size=(300, 300))

hist2 = histogram(b_boot, xlabel=L"$b$", ylabel="Count", legend=false,
 ↪  title=L"$b$")
vline!(hist2, [p_mle[2]], color=:red)
plot!(hist2, size=(300, 300))

hist3 = histogram(τ_boot, xlabel=L"$\tau$", ylabel="Count",
 ↪  legend=false, title=L"$\tau$")
vline!(hist3, [p_mle[3]], color=:red)
plot!(hist3, size=(300, 300))

hist4 = histogram(S₀_boot, xlabel=L"$S_0$", ylabel="Count",
 ↪  legend=false, title=L"$S_0$")
vline!(hist4, [p_mle[4]], color=:red)
plot!(hist4, size=(300, 300))
```

```
hist5 = histogram(ρ_boot, xlabel=L"$\rho$", ylabel="Count",
 ↪  legend=false, title=L"$\rho$")
vline!(hist5, [p_mle[5]], color=:red)
plot!(hist5, size=(300, 300))

hist6 = histogram(σ_boot, xlabel=L"$\sigma$", ylabel="Count",
 ↪  legend=false, title=L"$\sigma$")
vline!(hist6, [p_mle[6]], color=:red)
plot!(hist6, size=(300, 300))

display(hist1)
display(hist2)
display(hist3)
display(hist4)
display(hist5)
display(hist6)
```

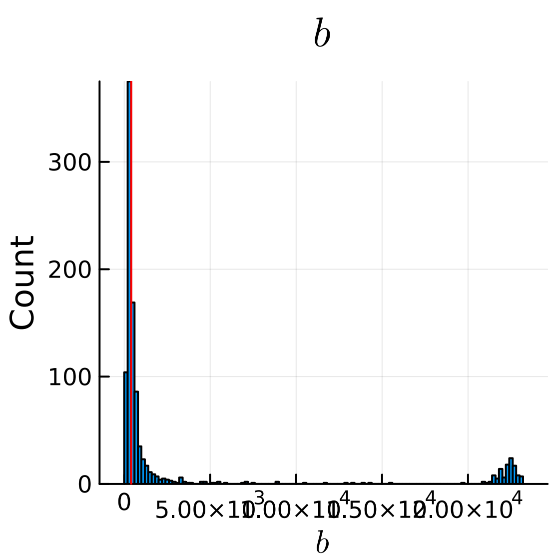
We can see that there are some outlying replicates which result in very high bootstrap samples for the model parameters a, b, and τ. But now we can calculate the sampling distribution of the sea-level sensitivity to temperature, which is a/τ. Plotting this distribution:

```
sens = a_boot ./ τ_boot

psens = histogram(sens, xlabel="Sea-Level Sensitivity to Temperature
 ↪  (mm/°C)", ylabel="Count", label=false)
vline!(psens, [p_mle[1] / p_mle[3]], color=:red, label="MLE")
plot!(psens, size=(600, 400))
```

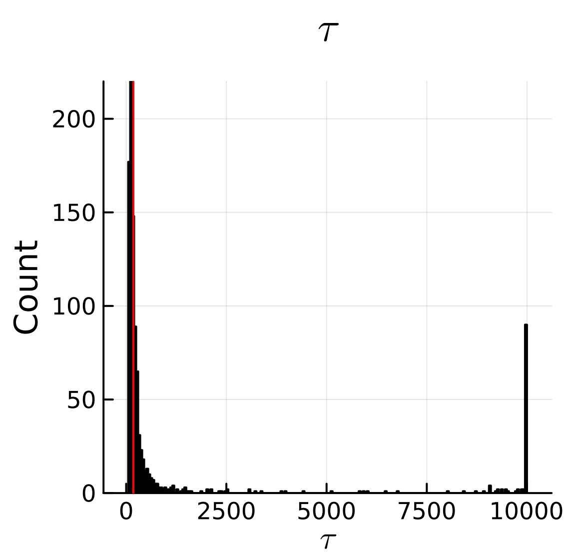
```
read: Connection reset by peer
send: Broken pipe
read: Connection reset by peer
send: Broken pipe
```



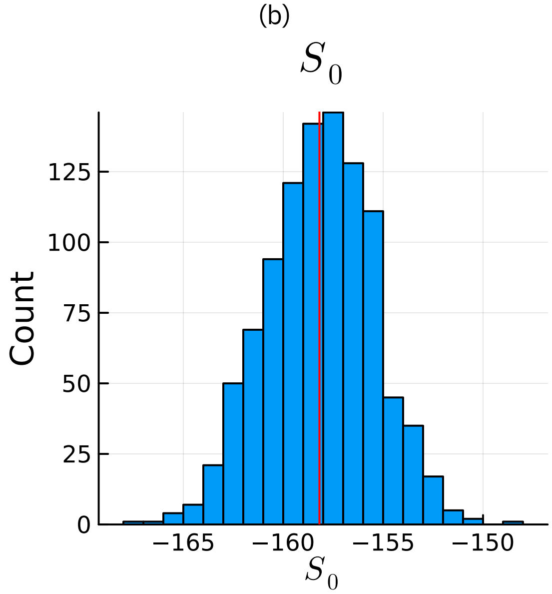
(a) Bootstrap samples for each parameter. The red line is the MLE.

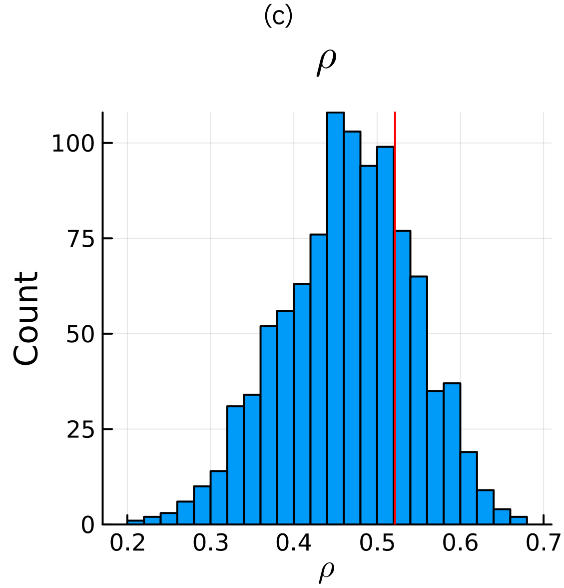


(b)



(c)





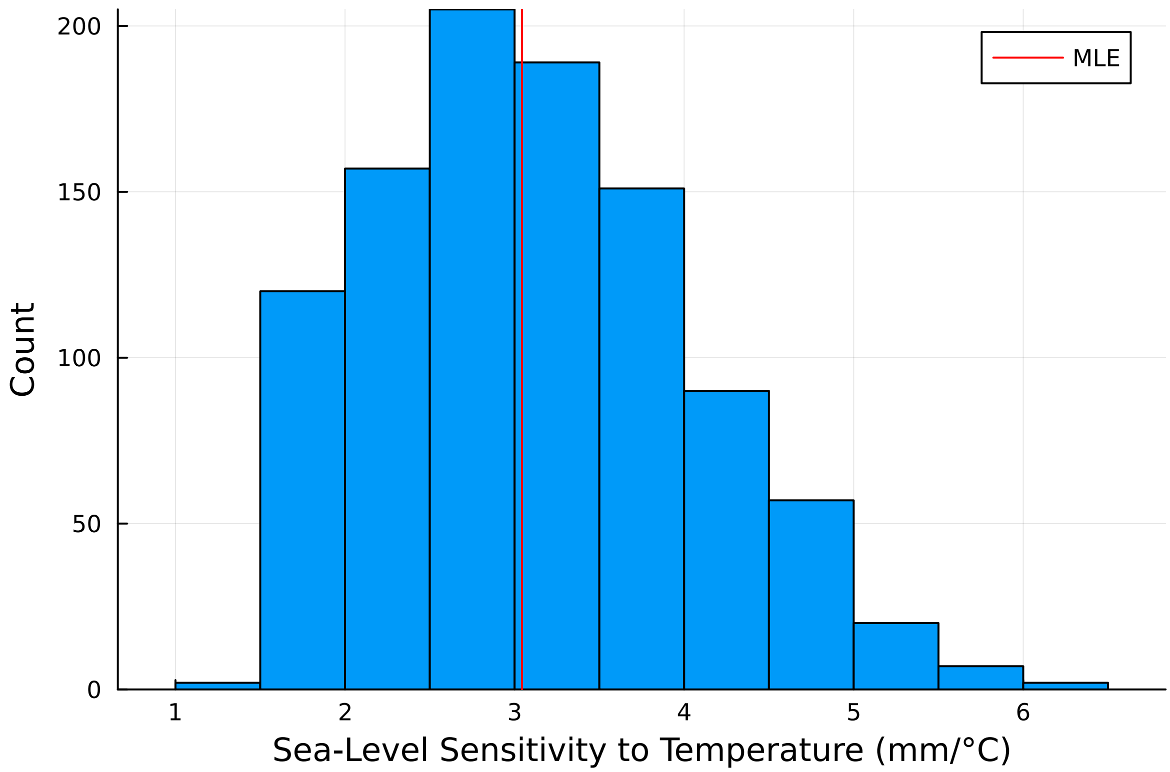


Figure 4: Bootstrap distribution for the sea-level sensitivity to temperature. The red line is the MLE.

We can see from Figure 4 that the large outlying samples cancel out in terms of the numerical effect, which is why the model fit still worked well.[1]

The 90% basic bootstrap confidence interval then becomes

```
sens_mle = p_mle[1] / p_mle[3]
q_boot = quantile(sens, [0.95, 0.05])
ci_boot = 2 * sens_mle .- q_boot
vspan!(psens, ci_boot, linecolor=:grey, fillcolor=:grey, alpha=0.3,
 ↪  fillalpha=0.3, label="90% CI")
```

[1]This also highlights a key point for model calibration and statistical inference: individual model parameters may not be directly interpretable due to these compensating effects, and it's valuable to look at physically-relevant statistics rather than specific parameters. If the individual model parameters are important for interpretation or extrapolation, they should be appropriately constrained or prior information should be used through a Bayesian analysis.

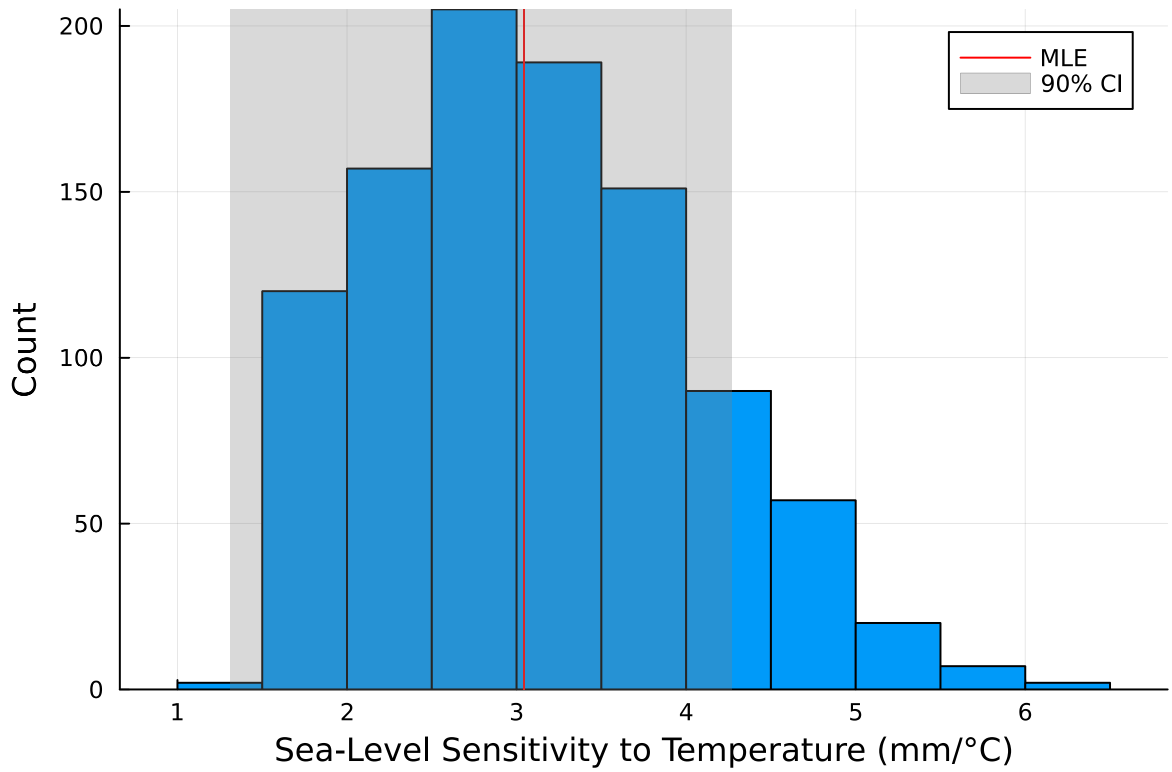


Figure 5: Bootstrap distribution for the sea-level sensitivity to temperature. The red line is the MLE. The grey region is the 90% confidence interval estimated using the basic bootstrap method.

The values for the 90% CI are [1.3, 4.3], and the MLE is 3.0. We can see the impact of the skewed distribution in Figure 5.

Problem 3

Loading the data and separating out the weather-induced variability:

```
function load_data(fname)
    date_format = "yyyy-mm-dd HH:MM"
    # this uses the DataFramesMeta package -- it's pretty cool
    return @chain fname begin
        CSV.File(; dateformat=date_format)
        DataFrame
        rename(
```

```
            "Time (GMT)" => "time", "Predicted (m)" => "harmonic",
             ↪  "Verified (m)" => "gauge"
        )
        @transform :datetime = (Date.(:Date, "yyyy/mm/dd") + Time.(:time))
        select(:datetime, :gauge, :harmonic)
        @transform :weather = :gauge - :harmonic
        @transform :month = (month.(:datetime))
    end
end

dat = load_data("data/norfolk-hourly-surge-2015.csv")

plot(dat.datetime, dat.weather; ylabel="Gauge Weather Variability (m)",
 ↪  label="Detrended Data", linewidth=1, legend=:topleft,
 ↪  xlabel="Date/Time")
```

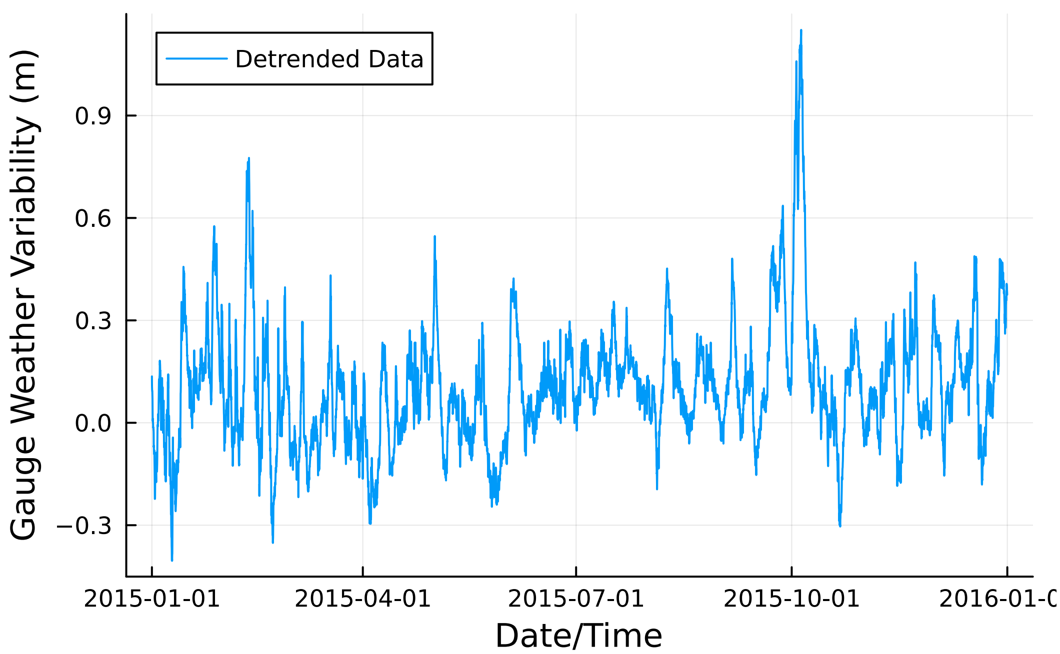


Figure 6: Weather-induced variability data from Problem 3.

Implementing the moving block bootstrap:

```
function generate_blocks(d, k)
    n_blocks = length(d) - k + 1
    blocks = zeros(Float64, (k, n_blocks))
    for i = 1:n_blocks
        blocks[:, i] = d[i:(k+i-1)]
    end
    return blocks
end

n_boot = 1_000
function moving_bootstrap(d, k, n_boot)
    blocks = generate_blocks(d, k)
    m = Int64(floor(length(d) / k))
    l = rem(length(d), k)
    n_blocks = size(blocks)[2]
    surge_bootstrap = zeros(length(d), n_boot)
    for i = 1:n_boot
        block_sample_idx = sample(1:n_blocks, m; replace=true)
        surge_bootstrap[1:length(d)-l, i] = reduce(vcat, blocks[:,
 ↪  block_sample_idx])
        if l > 0
            surge_bootstrap[length(d)-l+1:end, i] = blocks[:,
 ↪  sample(1:n_blocks, 1)][1:l]
        end
    end
    return surge_bootstrap
end

k = 20
weather_boot = moving_bootstrap(dat.weather, k, 1_000)
```

```
8760×1000 Matrix{Float64}:
 0.257   0.504   0.128  0.145   0.472  …  0.069  0.626  0.165  -0.07
 -0.172
 0.267   0.5     0.083  0.136   0.467     0.048  0.625  0.159  -0.088
 -0.147
 0.294   0.496   0.052  0.119   0.474     0.138  0.641  0.183  -0.094
 -0.132
 0.319   0.504   0.052  0.125   0.503     0.171  0.665  0.184  -0.085
 -0.091
 0.31    0.511   0.058  0.141   0.498     0.118  0.703  0.183  -0.125
 -0.097
```

```
 0.285   0.51    0.07   0.151   0.516  …  0.087  0.744  0.186  -0.153
 -0.123
 0.246   0.503   0.084  0.152   0.515     0.078  0.796  0.167  -0.139
 -0.128
 0.234   0.487   0.071  0.153   0.518     0.095  0.836  0.164  -0.12
 -0.146
 0.223   0.461   0.039  0.162   0.505     0.172  0.874  0.179  -0.108
 -0.165
 0.209   0.447   0.009  0.176   0.504     0.19   0.885  0.203  -0.117
 -0.177
 ⋮                                     ⋱  ⋮
 0.188  -0.113  -0.061  0.14    0.038     0.161  0.089  0.181   0.217
 0.068
 0.179  -0.11   -0.061  0.138   0.002     0.126  0.099  0.165   0.223
 0.057
 0.183  -0.114  -0.056  0.135  -0.001     0.102  0.103  0.129   0.212
 0.091
 0.148  -0.115  -0.057  0.128   0.01      0.093  0.102  0.102   0.195
 0.127
 0.097  -0.11   -0.054  0.116   0.004  …  0.1    0.095  0.085   0.169
 0.155
 0.087  -0.097  -0.04   0.119  -0.013     0.111  0.081  0.062   0.156
 0.117
 0.086  -0.102  -0.021  0.091  -0.012     0.137  0.061  0.047   0.161
 0.075
 0.091  -0.064  -0.022  0.085  -0.005     0.125  0.064  0.032   0.176
 0.084
 0.074  -0.052  -0.01   0.071  -0.024     0.135  0.076  0.03    0.152
 0.082
```

Now we add these replicates back to the harmonic prediction, then find the distribution of the medians.

```julia
gauge_boot = mapslices(col -> col + dat.harmonic, weather_boot; dims=1)
median_boot = median(gauge_boot, dims=1)'
pmed = histogram(median_boot, xlabel="Median Tide Gauge Reading (m)",
↪  ylabel="Count", label=false)
vline!(pmed, [median(dat.gauge)], label="Observation", color=:red)
```

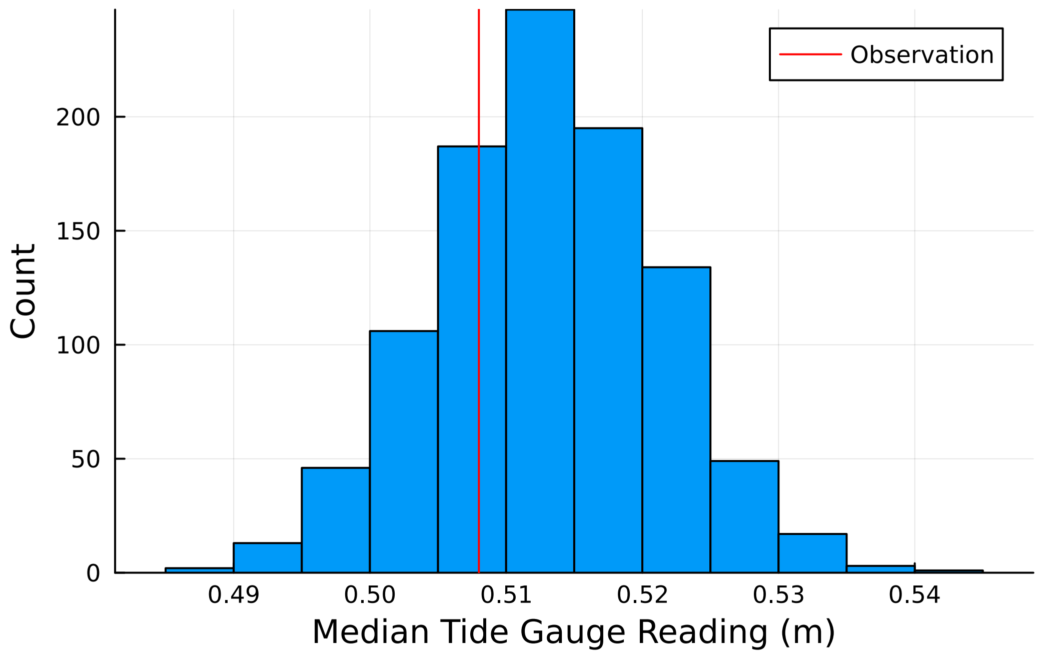


Figure 7: Histogram of the bootstrap distribution of median tide gauge values based on a block size of 20. The red line is the median of the sample.

The bias in the estimator of the median is the difference between the bootstrap mean and the sample estimate, which is about 5 mm. The 90% basic bootstrap confidence interval is [0.49, 0.517] m.

Now, let's repeat the analysis with a block length of 50.

```
k = 50
weather_boot = moving_bootstrap(dat.weather, k, 1_000)
gauge_boot = mapslices(col -> col + dat.harmonic, weather_boot; dims=1)
median_boot = median(gauge_boot, dims=1)'
pmed = histogram(median_boot, xlabel="Median Tide Gauge Reading (m)",
↪  ylabel="Count", label=false)
vline!(pmed, [median(dat.gauge)], label="Observation", color=:red)
```

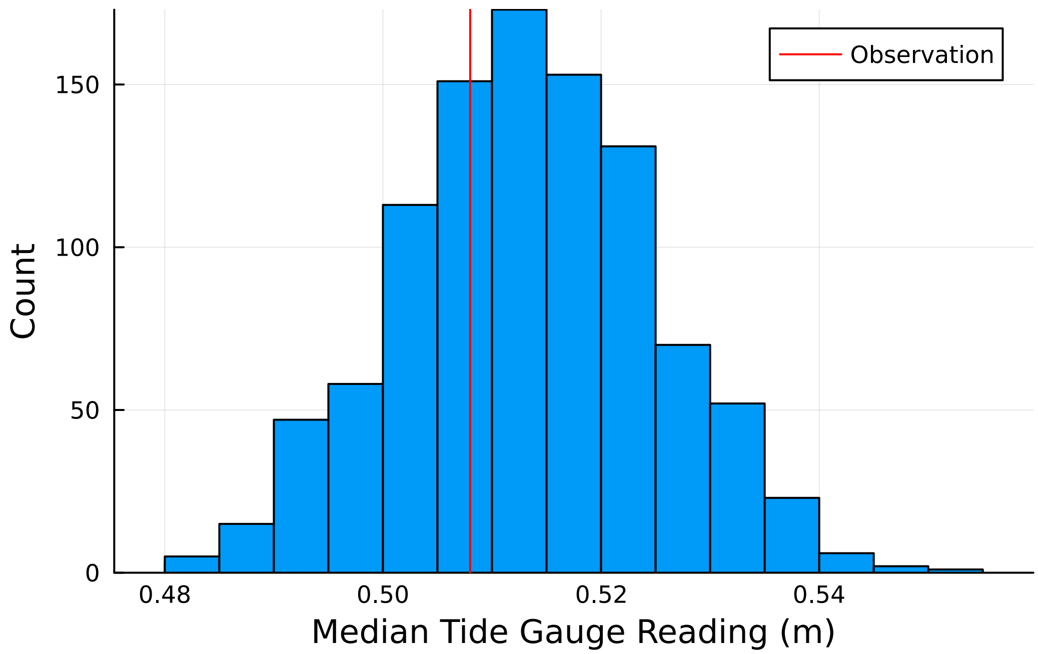


Figure 8: Histogram of the bootstrap distribution of median tide gauge values based on a block size of 50. The red line is the median of the sample.

Now, the bias is about 5 mm and the 90% basic bootstrap confidence interval is [0.483, 0.523] m.

The two different block sizes produced relatively similar results, with some slight variation for the confidence interval estimates, which are shifted slightly higher (but just by a few mm) with the larger block size. Larger blocks mean that there is a greater probability of sampling the (rare) larger values within any given block, which might be expected to produce more occurrences of these values even though the number of re-sampled blocks is smaller. This would have a result of increasing the median. But in general, the median estimator seems to be relatively stable across these block sizes, though even larger blocks might change this.