Bee Data Monte Carlo
Group 2

Vasiliy Ostapenko (vostapenko)
Arthur Starodynov (astarodynov)
Jake Bentley (williambentley)

Contents

Useful Links for Background Knowledge:

- Link 1
- Link 2
- Link 3

Abstract

In this paper, we use Monte Carlo methods for parameter estimation in the context of a linear regression model of bee wing vein density on bee body size. Concretely, we use a dataset of 48 bee samples, fit a simple linear regression model to estimate b1, and then compare that estimate with those obtained from bootstrapping and Markov chain Monte Carlo, respectively. We are interested in this analysis because we seek to determine whether the slope parameter for the model is significant (nonzero). We first obtained an OLS estimate for b1, which was nonzero. We then used Monte Carlo resampling (bootstrapping) with varying sample sizes to build several confidence intervals for b1. Finally, using MCMC, we built another sampling distribution of b1. We found that, with a large enough sample size, both Monte Carlo methods produced estimates which were significant (nonzero), agreeing with the OLS estimate. These results matter, because we now know that we have significant impact on the response from the predictor variable.

Introduction

The motivation for this project came from wanting to extend the research of a fellow UCSB student, whose dataset we used. In his studies, he collected 48 bee samples, observing both categorical and numeric parameters. He then fit a simple linear regression model of vein density on body size, hypothesizing that there was a linear relationship between the two variables. The resulting low R-squared made us question whether there was indeed a significant relationship. Being in a computational statistics course, we wanted to move beyond a t-test for slope or a nested model F-test, and thus chose two Monte Carlo methods for estimating the slope parameter. In particular, we were acutely interested in whether it is significant (nonzero).

```r
df = read.csv("./data/bee_data.csv") # load the data
df$Ratio = 100 * df$Ratio # scale response by 100 to avoid small decimals
```

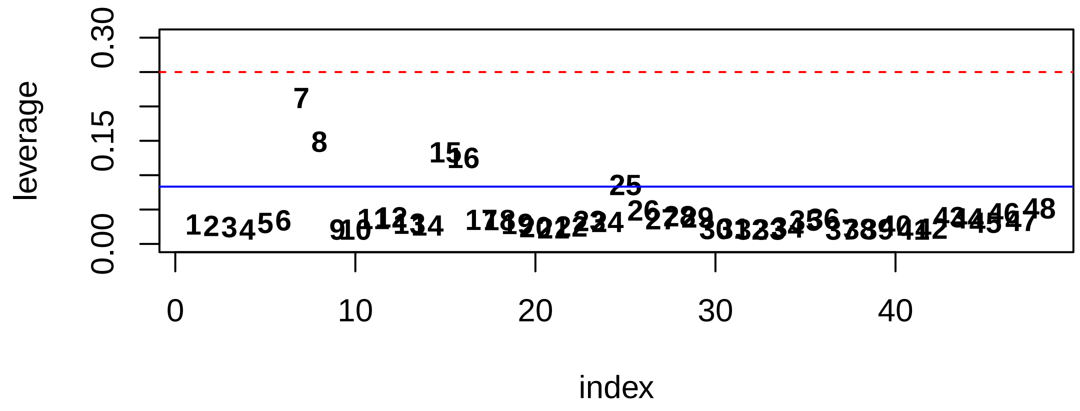
Methods

The first method we used to analyze the data and gain a baseline estimate for b1 was fitting a simple linear regression model and computing the slope through OLS. A simple linear regression model assumes that the response is a linear function of the predictors as well as noise, which follows a normal distribution. Concretely, in our implementation, we used the glm() function and specified the Gaussian family due to our data being continuous and unbounded. Once the model was fit, we obtained a nonzero estimate for b1. However, due to the low R-squared, we decided to explore the data further. It was clear that there were a few outliers in the data based on the Cook's distance criterion. Thus, we fit another linear regression model to the data without the outliers. Finally, we got a slightly better R-squared and a nonzero estimate for b1, still. We now decided to explore how these results stacked up to computational estimates.

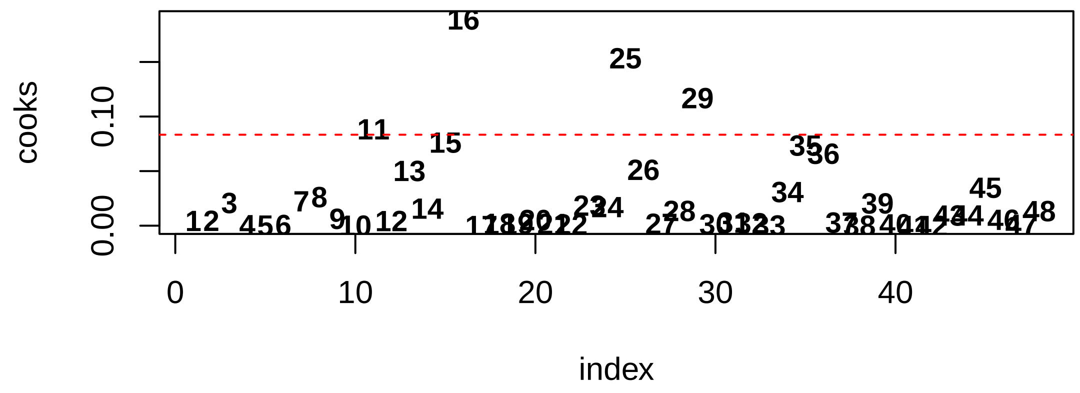
```
mod = glm(formula=Ratio ~ IT, family="gaussian", data=df) # fit simple OLS model
summary(mod)
```

```
##
## Call:
## glm(formula = Ratio ~ IT, family = "gaussian", data = df)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -3.365  -0.720  -0.047   0.911   3.004
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   13.754      0.376   36.54   <2e-16 ***
## IT             0.418      0.129    3.24   0.0023 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 2.016)
##
##     Null deviance: 113.810  on 47  degrees of freedom
## Residual deviance:  92.715  on 46  degrees of freedom
## AIC: 173.8
##
## Number of Fisher Scoring iterations: 2
```

```
lev = hatvalues(mod) # compute leverages

n = nrow(df)
p = 3
dat = data.frame(index=seq(length(lev)), leverage=lev)
plot(leverage~index, col="white", data=dat, pch=NULL, ylim=c(0, 0.3))
text(leverage~index, labels=index, data=dat, cex=0.9, font=2)
abline(h=(p+1)/n, col="blue")
abline(h=3*(p+1)/n, col="red", lty=2)
```



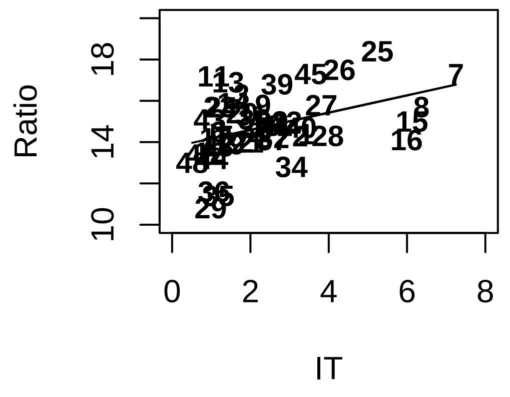
```r
d = cooks.distance(mod) # compute Cook's distance measure

dat2 = data.frame(index=seq(length(d)), cooks=d)
plot(cooks~index, col="white", data=dat2, pch=NULL)
text(cooks~index, labels=index, data=dat2, cex=0.9, font=2)
abline(h=4/n, col="red", lty=2)
```

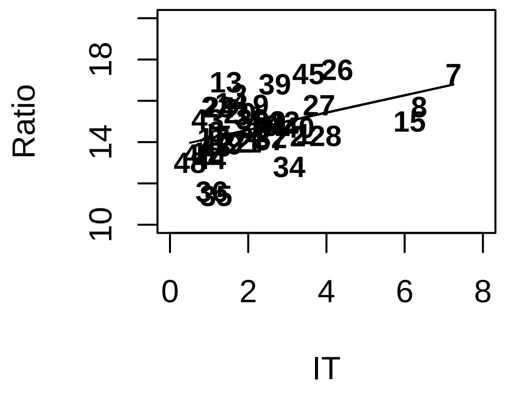


```r
mod2 = glm(formula=Ratio ~ IT, family="gaussian", data=df[-c(11, 16, 25, 29), ])
summary(mod2) # simple OLS fit to data without outliers
```

```
##
## Call:
## glm(formula = Ratio ~ IT, family = "gaussian", data = df[-c(11,
##     16, 25, 29), ])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8473  -0.6806  -0.0482   0.7498   2.5475
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   13.754      0.345   39.89   <2e-16 ***
## IT             0.419      0.124    3.38   0.0016 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.469)
##
##     Null deviance: 78.488  on 43  degrees of freedom
## Residual deviance: 61.684  on 42  degrees of freedom
## AIC: 145.7
##
## Number of Fisher Scoring iterations: 2
```

```r
par(mfrow=c(1, 2))
{
  {
    plot(Ratio~IT, data=df, col="white", pch=NULL,
         xlim=c(0, 8), ylim=c(10, 20))
    text(Ratio~IT, labels=rownames(df), data=df,
         cex=0.9, font=2)
    lines(x=df$IT, y=predict(mod, df))
  }
  {
    plot(Ratio~IT, data=df[-c(11, 16, 25, 29), ], col="white", pch=NULL,
         xlim=c(0, 8), ylim=c(10, 20))
    text(Ratio~IT, labels=rownames(df[-c(11, 16, 25, 29), ]), data=df[-c(11, 16, 25, 29), ],
         cex=0.9, font=2)
    lines(x=df[-c(11, 16, 25, 29), ]$IT, y=predict(mod2, df[-c(11, 16, 25, 29), ]))
  }
}
```





```r
print(paste0("mod R2: ", round(1 - mod$deviance/mod$null.deviance, 3)))
```

```
## [1] "mod R2: 0.185"
```

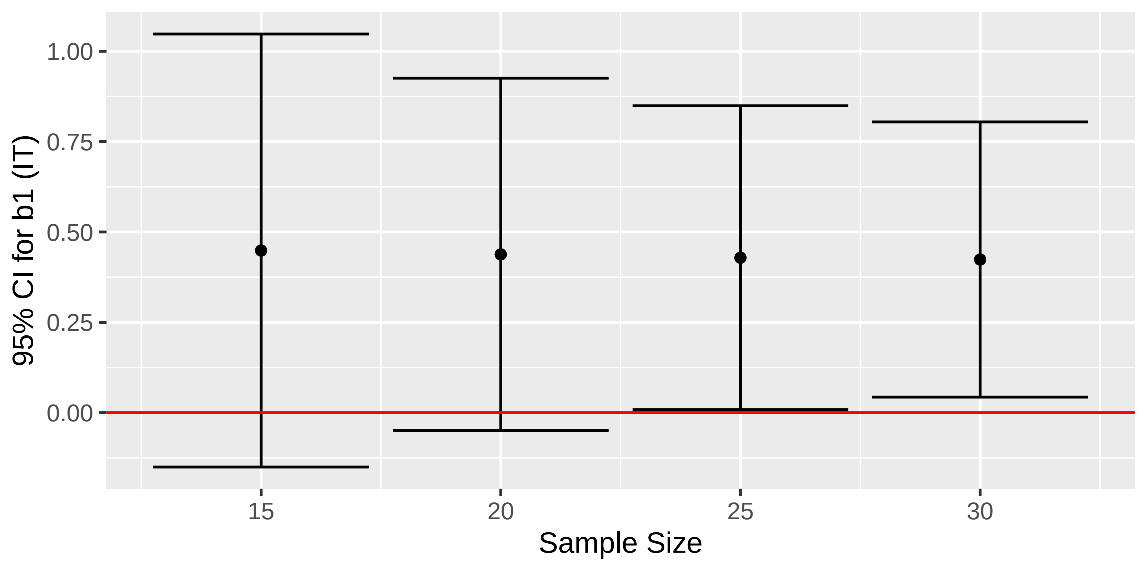
```r
print(paste0("mod2 R2: ", round(1 - mod2$deviance/mod2$null.deviance, 3)))
```

```
## [1] "mod2 R2: 0.214"
```

The first computational method that we used was Monte Carlo resampling (bootstrapping) with sample sizes of 15, 20, 25, and 30 to build several confidence intervals for b1. By repeatedly resampling from the data, bootstrapping enabled us to generate a novel sample from the data that can be used to estimate our parameter of interest. Bootstrapping relies on the Law of Large Numbers, a conclusion of which is that the average value of a parameter estimate from a great many trials would converge to the expected value of the actual parameter. Thus, we generated n=1000 samples for each sample size, refit the linear regression model with just that data, estimated the b1 parameter, and stored every estimate for each sample and each sample size group. After this, we essentially had four different sampling distributions of b1, using which we made four estimates and four different 95% confidence intervals for b1.

```r
boot_sim = function(nboot=10000, bootsizes=c(15, 20, 25, 30)) {
  container = list()
  j = 1
  for(bootsize in bootsizes) {
    b0_estimates = vector(mode="numeric", length=nboot)
    b1_estimates = vector(mode="numeric", length=nboot)
    for(i in seq(nboot)) {
      bdat = df[sample(nrow(df), size=bootsize, replace=TRUE), ]
      bfit = update(mod, data=bdat) # update model with new sample
      b0_estimates[i] = coef(bfit)[[1]]
      b1_estimates[i] = coef(bfit)[[2]]
    }
    results = list(size=bootsize,
                   b0_mu=mean(b0_estimates), b0_se=sd(b0_estimates),
                   b1_mu=mean(b1_estimates), b1_se=sd(b1_estimates))
    container[[j]] = results
    j = j+1
  }
  container = rbindlist(container)
  return(container)
}

data = boot_sim() # generate simulated data
```

```r
p = ggplot(data, aes(size, b1_mu)) +
  geom_point() +
  geom_errorbar(aes(ymin=b1_mu-1.96*b1_se, ymax=b1_mu+1.96*b1_se)) +
  geom_hline(yintercept=0, col="red") +
  labs(y="95% CI for b1 (IT)", x="Sample Size")

p
```



The second computational method we used was the Metropolis-Hastings algorithm, which falls under the realm of Markov chain Monte Carlo (MCMC). The algorithm works because of the Bayes theorem - we are able to draw from the distributions of b0 and b1 without knowing what those are. Instead, we define a "prior" distribution for the response variable, along with a likelihood function for b0 and b1, and use those in tandem to get samples for b0 and b1. First, we start with the prior distribution. Since we don't know much about the true distribution of this data, we set the prior to be a uniform distribution defined between 0 and close to the maximum of our predictor values. We then apply a log-transform, to remedy a possible convergence to zero for some small values involved. This action also changes the operation of multiplication into addition. Next, we create our likelihood function, which gives us the probability that our data was observed, given what we believe the parameters to be. For the likelihood, we use a normal distribution with mean b0 + b1x and a variance of nine. Then, for each step, we want to calculate the posterior probability (sample from our target distribution). This is just the sum of the likelihood and the prior (on the log-scale). One last step is necessary to generate a proper sample, which is to either accept or reject it based on some criteria.

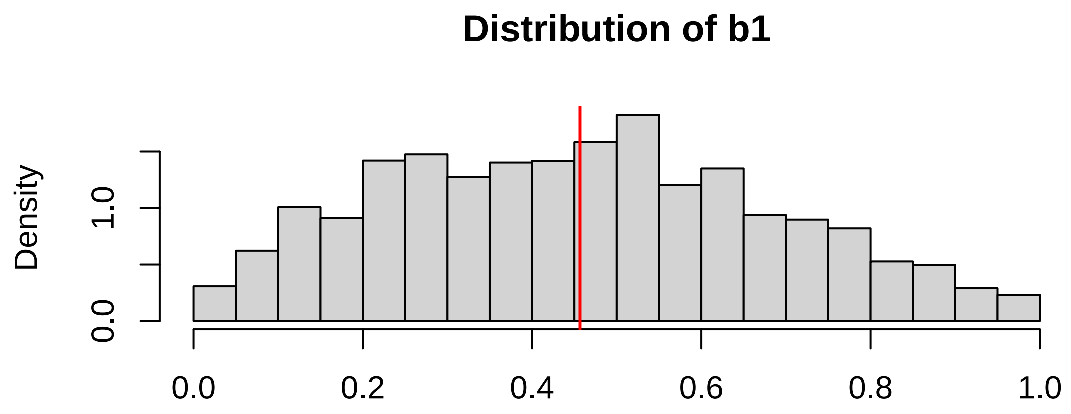
The AR step requires a proposal function, which we set to a normal distribution with variance 0.25 and means b0 and b1, respectively. Each transition of the Markov chain, we calculate our proposal given our previous state. Then, we calculate our acceptance ratio, using our posterior probability given our proposal minus our posterior probability given the previous step. We thus get the probability of accepting the proposal as the next state or rejecting this value and not transitioning in the next time-step. We repeat this ten thousand times to generate, after about two thousand steps, a distribution for b0 and b1. Finally, we can obtain an estimate for b1 by simply taking an average.

```r
prior_probability = function(beta) {
  a = beta[1]
  b = beta[2]
  a_prior = dunif(a, min=0, max=30) # naively assume uniform prior
  b_prior = dunif(b, min=0, max=1)
  return(log(a_prior)+log(b_prior))
}

likelihood_probability = function(beta, x, y) {
  a = beta[1]
  b = beta[2]
  y_predict = a+b*x
  single_likelihoods = dnorm(x=y, mean=y_predict, sd=3.0)
  return(sum(log(single_likelihoods)))
}

posterior_probability = function(beta, x, y) {
  return( likelihood_probability(beta, x, y) + prior_probability(beta) )
}

proposal_function = function(beta) {
  a = beta[1]
  b = beta[2]
  a_new = rnorm(n=1, mean=a, sd=0.5)
  b_new = rnorm(n=1, mean=b, sd=0.5)
  beta_new = c(a_new, b_new)
  return(beta_new)
}

mcmc_sim = function(x, y, n=10000) {
  container = data.frame(b0=vector(mode="numeric", length=n),
                         b1=vector(mode="numeric", length=n))
  beta_0 = c(0.5, 0.5)
  container[1, ] = beta_0

  for(step in 2:n) {
    beta_old = as.numeric(container[step-1, ])
    beta_proposal = proposal_function(beta_old)

    prob = exp(posterior_probability(beta_proposal, x, y) -
                 posterior_probability(beta_old, x, y))

    u = runif(n=1, min=0, max=1)

    if(is.na(u < prob)) {
      container[step, ] = beta_old
    }
    else {
      if(u < prob) {
        container[step, ] = beta_proposal # MC transition to new state
      }
      else {
        container[step, ] = beta_old # MC stays in old state
      }
    }
  }
  return(container)
}

data2 = mcmc_sim(x=df$IT, y=df$Ratio, n=10000) # generate simulated data
```

```
burn_in = 2000
beta_posterior = data2[burn_in:nrow(data2), ]
print(paste0("b0 estimate: ", round(mean(beta_posterior$b0), 3),
             "; b1 estimate: ",
             round(mean(beta_posterior$b1), 3)))
```

```
## [1] "b0 estimate: 13.649; b1 estimate: 0.457"
```

```r
{
  hist(beta_posterior$b1, breaks=30, freq=FALSE, xlab="",
       main=paste0("Distribution of b1"))
  abline(v=mean(beta_posterior$b1), col="red", lwd=1.5)
}
```



Results

Using OLS, we obtained a point estimate for the slope at around 0.42, a confidence interval which did not include zero, as well as a line with visually nonzero slope. We can now compare this baseline result with our computational results.

Using bootstrapping, each of the sample sizes produces a point estimate for the slope close to 0.44. We can also see that the two smaller sample sizes (15 and 20) give us confidence intervals which include zero. However, increasing the sample size to 25, 30, and beyond shrinks these bounds to be significantly nonzero. Thus, the Monte Carlo resampling method generally agrees with OLS in a significant slope for the model.

Using MCMC, we have a sampling distribution for b1 ranging between zero and one, centered at around 0.45. Thus, Metropolis-Hastings agrees with OLS, and produces an accurate estimate for the slope despite our broad assumptions about the underlying data.

Discussion

Our results matter because they confirm a significant relationship between the predictor and response variable in our linear model. Moreover, they demonstrate that estimating linear model parameters could be done using computation, instead of relying on linear algebra as well as F- and t-tests. Given our positive results, we still wish that there were more sample points in the data, which would make our estimates even more accurate. An interesting observation which arose during the running of MCMC was that small changes in the proposal function had meaningful impacts on the final parameter estimates. Looking at some other distributions besides normal as well as playing with the variance, we arrived at a satisfactory proposal function. Finally, a way in which the project could be extended would be to use other models besides simple linear to study the response variable. Given that there is categorical data in the dataset as well, it could be converted to numeric via dummy coding. Then, all the data could be fed into a neural net, for instance, which would train to make accurate predictions given new sample data. This would require an exploration of neural net architectures, gradient descent, and other relevant topics.

Appendix

Given a dataset of multiple predictors our first goal was to form some model that would show how a set of variables can have a direct impact on an outcome. Thinking of a simple method to do this was by looking at a linear regression model, as this model takes into account that an outcome, or response, is a linear function of some predictors and noise (a variation in y that is unrelated to the predictor). Since the goal was to look at the relationship between bee forewing density and body size, we used the glm function in the stats package of R. We set the model formula as Ratio on IT. Running glm() and print the summary, we saw that a nonzero estimate of b1 (the slope). However, there was a clear problem with this model, which was a low R-squared value. We saw that outliers could be playing a significant role. Using a criterion known as Cook's Distance, we identified the high leverage points and took them out of our model analysis, recognizing that the model fit should be better. Looking at the new summary statistics, we notice that, not only did the R-squared value improve, but also that the p value decreased as well. This means that getting rid of the outliers would increase the significance of IT predicting Ratio. Finally, plotted the fits of our linear model with the outliers as well as the fit without.

Next, we wanted to use computational methods to fully analyze if there was a linear relationship between the data. The first method used was bootstrapping. In bootstrapping we wanted to set a few sample sizes that would allow us to analyze if sample size had an effect on the estimation of b1. Therefore, we set the sample sizes to be 15, 20, 25, and 30. This would allow us to look at the confidence intervals of these samples, where the smallest confidence interval would be the most preferred sample size to predict an accurate b1 approximation. Knowing that bootstrapping relies on LLN, we wanted to resample the data a significant number of times, focusing on resampling it 1000 times within each sample size. Since the data was resampled 1000 times we predicted that the approximate value of b1 would be closer to the expected value and hence refit the linear model to now estimate this approximation. Within each sample size we approximated b1 and set it into its respective sample size group. From there we decided to use a 95% confidence interval instead of a 99% as we wanted to let our interval be as shallow as possible, as well as a 95% would be more accurate in predicting the approximation from larger sample sizes. Hence, once the confidence interval was plotted within each respective sample size, it was clear that with a larger sample size the prediction of b1 was more accurate and did not include 0 which is valuable. Having our b1 estimate be 0 represents that there is no linear correlation between our response and predictor variables, which is the opposite of our intended goal. Thus with bootstrapping it was figured out that with larger sample sizes b1 would be more indicative of figuring out the actual correlation. Looking at the results it was clear that our slope of each confidence interval brought in an estimate of b1 being close to .44.

With the higher value of b1 using bootstrapping we wanted to explore another computational method to compare b1 further. Since bootstrapping used resampling techniques, we wanted to use something similar that could be applied into further depth, which was the Metropolis-Hastings algorithm. The Metropolis-Hastings algorithm follows along with Markov Chain Monte Carlo methods, which is a deeper look then bootstrapping as markov chains is a stochastic model that describes a series of possible events where the probability of each event is only dependent on the state of the event attained in the previous state. We believed that using a Markov Chain sampling technique would be beneficial in exploring the value of b1 since with bootstrapping we were able to get a better approximation of b1 through 1000 samples, and a markov chain can have n states. In addition, this method was chosen as it uses a variation of an acceptance-rejection sampling method. An acceptance-rejection method was important to use as we can solidify which resampled data we would want based on some set of criteria. If the sample did not follow the criteria set upon it we could reject it, not factoring it into the approximation of b1. Although this method is great at creating a distribution based on prior distribution and creating a new distribution from which b1 can be approximated is beneficial, there are a few drawbacks to this algorithm. The Metropolis Hastings algorithm makes the samples correlated which over the long term will correctly follow a set of nearby samples to be correlated with each other but not correctly reflect the distribution. This represents that the sample sizes can be significantly smaller and due to that lead to large errors. In addition, eventually the markov chain will reach the desired distribution over time but the initial states can be far off the intended distribution, hence needing a burn in period. In our methods, we specifically had 10000 samples and we burned in the first 2000 thinking that these could skew our distribution, which would skew the approximation of b1.

Data source:

- Leonardo Eisner de Eisenhoff's GitHub repository: https://shorturl.at/htyDU

Data dictionary:

- Ratio: response variable (numeric, continuous)
- IT: predictor variable (numeric, continuous)
- other columns in data.frame "data": unused

References and Citations:

[1] Cook's Distance (2022) Retrieved from https://en.wikipedia.org/wiki/Cook%27s_distance

[2] Eisner de Eisenhoff L. 2022. Body Size and Taxonomic Influence on Bee Wing Vein Density. University of California Santa Barbara.

[3] Letian W. B. (2004) Markov Chain Monte Carlo Linear Regression. Retrieved from https://letianzj.githu b.io/mcmc-linear-regression.html

[4] Linear Regression (January 2010) Retrieved from https://en.wikipedia.org/wiki/Linear_regression

[5] Metropolis–Hastings algorithm (2022) Retrieved from https://en.wikipedia.org/wiki/Metropolis%E2%80%9 3Hastings_algorithm#:~:text=In%20statistics%20and%20statistical%20physics,which%20direct%20sampling%20is%20difficult

[6] Molina, E. (2021) Bootstrap Sampling in R. Retrieved from https://towardsdatascience.com/bootstrap-sampling-in-r-a7bc9d3ca14a

[7] Ross, S. M. (2002) Introduction to Probability Models (8th ed.). Amsterdam: Academic. Press