Applied Statistics II

Vasileios (Bill) Katsianos

August 2023

Contents

1 Advanced Linear Models

Simpson's Paradox

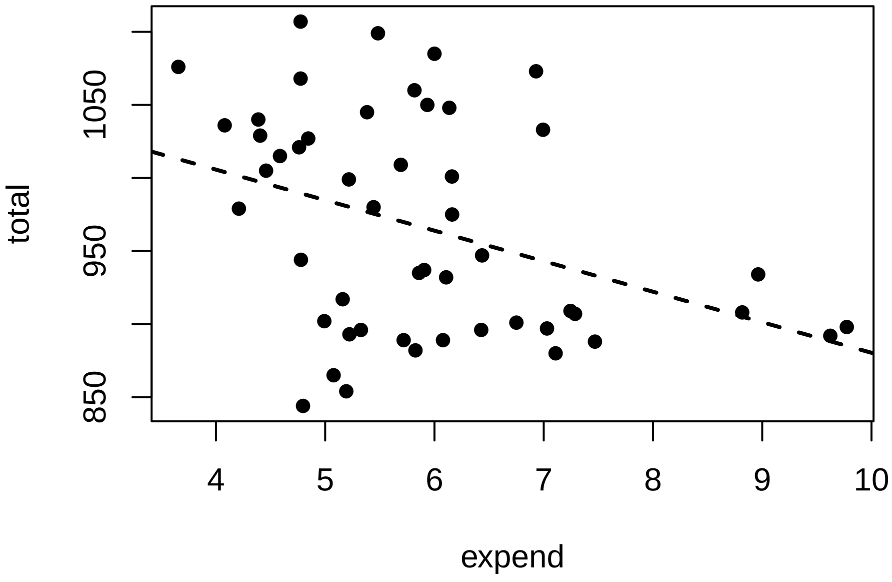
Simpson's paradox arises when two variables appear to be negatively (or positively) correlated when they are regarded by themselves, but their true positive (or negative) correlation is uncovered after taking another confounding variable into account. This phenomenon can be illustrated through the sat data set from the faraway package, which contains information on public school expenditure and SAT test scores for each of the 50 US states. After regressing the average total SAT score per state on the daily expenditure per pupil in US public schools, we observe that the amount of expenditure has a statistically significant negative effect on average total SAT score. This fact is quite surprising since one would normally expect that states which allocate more money on public education would boast better SAT scores.

```
library(faraway)
library(xtable)
```

```r
reduced = lm(total ~ expend, sat)
print(xtable(summary(reduced)), comment = FALSE)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1089.2937	44.3900	24.54	0.0000
expend	-20.8922	7.3282	-2.85	0.0064

```r
plot(total ~ expend, sat, pch = 16)
abline(reduced, lty = 2, lwd = 2)
```



It turns out that after taking the percentage of per state eligible students taking the SAT into account, the true positive effect of expenditure on average total SAT scores reveals itself. The percentage of eligible students taking the SAT obviously has a negative effect on average total SAT scores. In states with a low percentage of takers, only the well prepared students end up taking the SAT, and their scores are accordingly higher than average. On the other hand, in states with a high percentage of takers, a lot of unprepared students are encouraged to take the SAT anyway, which leads to a dilution of the average total scores. The reason why the correlation between expenditure and average total SAT score appears to be negative, when ignoring the percentage of eligible students taking the test, is that expenditure is positively correlated with the percentage of eligible students taking the test. In states which allocate a lot of money on public education, more students are encouraged to take the SAT and vice versa. By ignoring the percentage of eligible students taking the SAT, some of the effect of that variable on the average total SAT score is passed along to the expenditure variable instead, reversing its perceived effect on the response variable.

```r
full = lm(total ~ expend + takers, sat)
print(xtable(summary(full)), comment = FALSE)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	993.8317	21.8332	45.52	0.0000
expend	12.2865	4.2243	2.91	0.0055
takers	-2.8509	0.2151	-13.25	0.0000

```
print(xtable(cor(sat[, c(1, 4, 7)])), comment = FALSE)
```

	expend	takers	total
expend	1.00	0.59	-0.38
takers	0.59	1.00	-0.89
total	-0.38	-0.89	1.00

We can simulate a scenario which gives rise to this phenomenon in order to gain more insight into it. First, we simulate a normally distributed explanatory variable X. Then, we simulate a binomial confounding variable Z, whose probability of success is an increasing function of X. Lastly, we simulate a response variable Y on which X has a positive effect while Z has a negative effect.

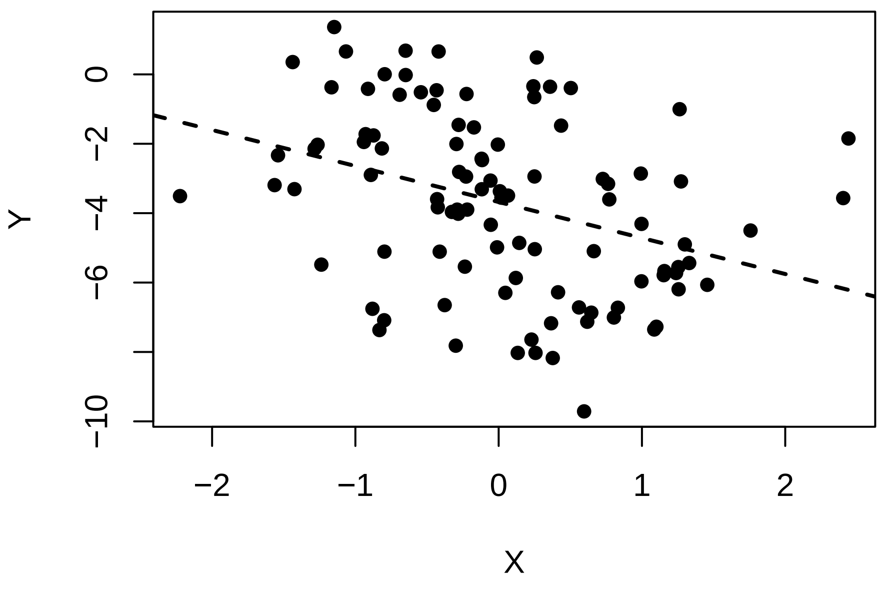
```
n = 100
X = rnorm(n)
Z = rbinom(n, 3, pnorm(X))
Y = 1 + 2 * X - 3 * Z + rnorm(n)
```

We can see from the regression summary, as well as the scatter plot, that X appears to have a statistically significant negative effect on Y when the confounding variable Z is ignored. This is because the explanatory variable X, which is positively correlated with the confounding variable Z, takes on some of the effect of Z on the response variable Y, reversing its individual effect.

```
library(xtable)
reduced = lm(Y ~ X)
print(xtable(summary(reduced)), comment = FALSE)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-3.6782	0.2404	-15.30	0.0000
X	-1.0376	0.2737	-3.79	0.0003

```
plot(X, Y, pch = 16)
abline(reduced, lty = 2, lwd = 2)
```



After taking the effect of the confounding variable Z into account, the true positive effect of the explanatory variable X on the response variable Y is restored.

```r
full = lm(Y ~ X + Z)
print(xtable(summary(full)), comment = FALSE)
```

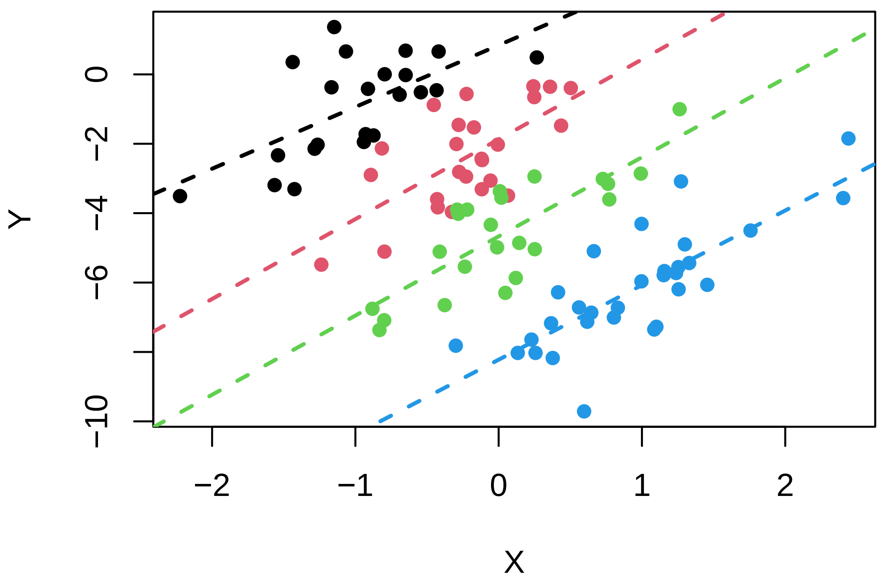
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.1881	0.2616	4.54	0.0000
X	2.0660	0.1942	10.64	0.0000
Z	-3.0662	0.1509	-20.33	0.0000

```r
print(xtable(cor(cbind(X, Z, Y))), comment = FALSE)
```

	X	Z	Y
X	1.00	0.79	-0.36
Z	0.79	1.00	-0.80
Y	-0.36	-0.80	1.00

The confounding effect of the Z variable can also be illustrated by stratifying the sample according to Z and running separate regressions on each subset of the sample.

```r
plot(X, Y, col = Z + 1, pch = 16)
for (i in 1:4) {
    fit = lm(Y ~ X, subset = Z == i - 1)
    abline(fit, col = i, lty = 2, lwd = 2)
    print(xtable(summary(fit)), comment = FALSE)
}
```



	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.8407	0.5361	1.57	0.1325
X	1.7800	0.4844	3.67	0.0015

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-1.8713	0.2469	-7.58	0.0000
X	2.3009	0.5303	4.34	0.0002

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-4.6698	0.2045	-22.84	0.0000
X	2.2818	0.3670	6.22	0.0000

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-8.2223	0.3553	-23.14	0.0000
X	2.1494	0.3170	6.78	0.0000

Hypothesis Testing for Linear Constraints

Suppose we have a linear model $Y = X\beta + \varepsilon$, where $X \in \mathbb{R}^{n\times p}$, $\beta \in \mathbb{R}^p$, $\varepsilon \sim \mathcal{N}_n\left(\mathbf{0}_n, \sigma^2 I_n\right)$, and we want to perform the hypothesis test $H_0 : R\beta = r$ vs. $H_1 : R\beta = r$, where $R \in \mathbb{R}^{k\times p}$ and $r \in \mathbb{R}^k$. Then, there are a number of different statistical tests which we might employ.

First, we might observe that $R\widehat{\beta} \sim \mathcal{N}_k\left(r, \sigma^2 R\left(X^{\mathrm{T}}X\right)^{-1} R^{\mathrm{T}}\right)$ under the null hypothesis, where $\widehat{\beta} = \left(X^{\mathrm{T}}X\right)^{-1} X^{\mathrm{T}}Y$. This implies that:

$$Q_1 = \frac{1}{\sigma^2}\left(R\widehat{\beta} - r\right)^{\mathrm{T}}\left[R\left(X^{\mathrm{T}}X\right)^{-1}R^{\mathrm{T}}\right]^{-1}\left(R\widehat{\beta} - r\right) \sim \chi^2_k.$$

Additionally, we know that:

$$Q_2 = \frac{n-p}{\sigma^2}S^2 = \frac{1}{\sigma^2}\left\|Y - X\widehat{\beta}\right\|_2^2 \sim \chi^2_{n-p}.$$

According to Cochran's theorem, Q_1 and Q_2 are mutually independent. Under the null hypothesis, we infer that:

$$F = \frac{Q_1/k}{Q_2/(n-p)} = \frac{1}{kS^2}\left(R\widehat{\beta}-r\right)^{\mathrm{T}}\left[R\left(X^{\mathrm{T}}X\right)^{-1}R^{\mathrm{T}}\right]^{-1}\left(R\widehat{\beta}-r\right) \sim F_{k,n-p}.$$

This is an exact F test statistic for the given hypotheses. We reject the null hypothesis when the observed value f of the test statistic is larger than the quantile $F_{k,n-p;\alpha}$ or, equivalently, when p-value $= P(F \geqslant f) < \alpha$.

According to the strong law of large numbers, we could make use of the fact that:

$$\frac{Q_2}{n-p} \xrightarrow{\text{a.s.}} 1.$$

According to Slutsky's theorem, we infer that:

$$W = \frac{Q_1}{Q_2/(n-p)} = kF = \frac{1}{S^2}\left(R\widehat{\beta}-r\right)^{\mathrm{T}}\left[R\left(X^{\mathrm{T}}X\right)^{-1}R^{\mathrm{T}}\right]^{-1}\left(R\widehat{\beta}-r\right) \xrightarrow{d} \chi^2_k.$$

This is an asymptotic Wald test statistic for the given hypotheses. We reject the null hypothesis when the observed value w of the test statistic is larger than the quantile $\chi^2_{k;\alpha}$ or, equivalently, when p-value $= P(W \geqslant w) < \alpha$.

Alternatively, we could calculate the MLE $\widehat{\beta}_0$ of β under the null hypothesis. We want to maximize the log-likelihood function $\ell\left(\beta,\sigma^2 \mid Y\right)$ with respect to β under the constraint $R\beta = r$, so we are going to utilize the method of Lagrange multipliers. Let $\lambda \in \mathbb{R}^k$. Then, we want to maximize the following function:

$$\mathcal{L}(\beta,\lambda) = -\frac{n}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\left\|Y - X\beta\right\|_2^2 - \lambda^{\mathrm{T}}\left(R\beta - r\right).$$

First, we differentiate with respect to the vector β:

$$\frac{\partial\mathcal{L}(\beta,\lambda)}{\partial\beta} = \frac{1}{\sigma^2}X^{\mathrm{T}}(Y - X\beta) - R^{\mathrm{T}}\lambda$$

Hence, we infer that $\sigma^2R^{\mathrm{T}}\lambda = X^{\mathrm{T}}Y - X^{\mathrm{T}}X\widehat{\beta}_0$. Left-multiply both sides of this equation by $R\left(X^{\mathrm{T}}X\right)^{-1}$ to get that:

$$\sigma^2R\left(X^{\mathrm{T}}X\right)^{-1}R^{\mathrm{T}}\lambda = R\left(X^{\mathrm{T}}X\right)^{-1}X^{\mathrm{T}}Y - R\cancel{\left(X^{\mathrm{T}}X\right)^{-1}}\,\cancel{X^{\mathrm{T}}X}\widehat{\beta}_0.$$

We observe that $\left(X^{\mathrm{T}}X\right)^{-1}X^{\mathrm{T}}Y = \widehat{\beta}$ and $R\widehat{\beta}_0 = r$, so we infer that:

$$\lambda = \frac{1}{\sigma^2}\left[R\left(X^{\mathrm{T}}X\right)^{-1}R^{\mathrm{T}}\right]^{-1}\left(R\widehat{\beta}-r\right).$$

Substitute this expression for λ into the initial equation involving $\widehat{\beta}_0$ to get that:

$$X^{\mathrm{T}}X\widehat{\beta}_0 = X^{\mathrm{T}}Y - R^{\mathrm{T}}\left[R\left(X^{\mathrm{T}}X\right)^{-1}R^{\mathrm{T}}\right]^{-1}\left(R\widehat{\beta}-r\right) \Rightarrow$$

$$\widehat{\beta}_0 = \widehat{\beta} - \left(X^{\mathrm{T}}X\right)^{-1}R^{\mathrm{T}}\left[R\left(X^{\mathrm{T}}X\right)^{-1}R^{\mathrm{T}}\right]^{-1}\left(R\widehat{\beta}-r\right).$$

Additionally, we define the following MLEs of σ^2 under the full and the reduced model respectively:

$$\widehat{\sigma}^2 = \frac{1}{n}\left\|Y - X\widehat{\beta}\right\|_2^2, \quad \widehat{\sigma}_0^2 = \frac{1}{n}\left\|Y - X\widehat{\beta}_0\right\|_2^2.$$

According to Wilks' theorem, we infer that:

$$\mathrm{LR} = -2\left[\ell\left(\widehat{\beta}_0, \widehat{\sigma}_0^2 \middle| Y\right) - \ell\left(\widehat{\beta}, \widehat{\sigma}^2 \middle| Y\right)\right] = -2n\log\frac{\widehat{\sigma}}{\widehat{\sigma}_0} \sim \chi^2_k.$$

This is a likelihood ratio test statistic for the given hypotheses. We reject the null hypothesis when the observed value LR_0 of the test statistic is larger than the quantile $\chi^2_{k;\alpha}$ or, equivalently, when p-value $= P(\mathrm{LR} \geqslant \mathrm{LR}_0) < \alpha$.

Lastly, we could calculate the score function and the Fisher information matrix of β as follows:

$$U_{\sigma^2}(\beta) = \frac{\partial \ell\left(\beta, \sigma^2 \mid Y\right)}{\partial \beta} = \frac{1}{\sigma^2} X^{\mathrm{T}}(Y - X\beta),$$

$$\frac{\partial^2 \ell\left(\beta, \sigma^2 \mid Y\right)}{\partial \beta \partial \beta} = -\frac{1}{\sigma^2} X^{\mathrm{T}} X,$$

$$\mathcal{I}_{\sigma^2}(\beta) = -\mathbb{E}\left[\frac{\partial^2 \ell\left(\beta, \sigma^2 \mid Y\right)}{\partial \beta \partial \beta}\right] = \frac{1}{\sigma^2} X^{\mathrm{T}} X.$$

Therefore, we get the following Score (or Lagrange Multiplier) test statistic:

$$\begin{aligned}\mathrm{ST} = U_{\widehat{\sigma}_0^2}\left(\widehat{\beta}_0\right)^{\mathrm{T}} \mathcal{I}_{\widehat{\sigma}_0^2}\left(\widehat{\beta}_0\right)^{-1} U_{\widehat{\sigma}_0^2}\left(\widehat{\beta}_0\right) &= \frac{1}{\widehat{\sigma}_0^2}\left(Y - X\widehat{\beta}_0\right)^{\mathrm{T}} X\left(X^{\mathrm{T}} X\right)^{-1} X^{\mathrm{T}}\left(Y - X\widehat{\beta}_0\right) \\ &= \frac{1}{\widehat{\sigma}_0^2}\widehat{\varepsilon}_0^{\mathrm{T}} P \widehat{\varepsilon}_0 = \frac{1}{\widehat{\sigma}_0^2}\left\|P\widehat{\varepsilon}_0\right\|_2^2 \sim \chi^2_k,\end{aligned}$$

where $\widehat{\varepsilon}_0 = Y - X\widehat{\beta}_0$ is the residual vector of the reduced model and $P = X\left(X^{\mathrm{T}} X\right)^{-1} X^{\mathrm{T}}$ is the orthogonal projection matrix corresponding to the design matrix X. We reject the null hypothesis when the observed value ST_0 of the test statistic is larger than the quantile $\chi^2_{k;\alpha}$ or, equivalently, when p-value $= P(\mathrm{ST} \geqslant \mathrm{ST}_0) < \alpha$.

Now, we illustrate these various test statistics on the prostate data set from the faraway package. Let:

$$\mathrm{lpsa}_i = \beta_0 + \beta_1 \mathrm{lcavol}_i + \beta_2 \mathrm{lweight}_i + \beta_3 \mathrm{svi}_i + \varepsilon_i,$$

where $\varepsilon_i \sim \mathcal{N}\left(0, \sigma^2\right)$ are independent for $i = 1, 2, \ldots, n$. We observe that the estimated coefficients for lcavol, lweight and svi are pretty close numerically, while their corresponding standard errors are also pretty high.

```r
library(faraway)
library(xtable)
n = dim(prostate)[1]
p = 4
full = lm(lpsa ~ lcavol + lweight + svi, prostate)
betafull = full$coefficients
S = summary(full)$sigma
sigmafull = sqrt(mean(full$residuals^2))
Y = full$model[, 1]
X = model.matrix(full)
print(xtable(summary(full)), comment = FALSE)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-0.2681	0.5435	-0.49	0.6230
lcavol	0.5516	0.0747	7.39	0.0000
lweight	0.5085	0.1502	3.39	0.0010
svi	0.6662	0.2098	3.18	0.0020

Suppose we want to perform the hypothesis test $H_0 : \beta_1 = \beta_2 = \beta_3$ vs. every possible alternative. Under the null hypothesis, we observe that:

$$\text{lpsa}_i = \beta_0 + \beta_1 \left(\text{lcavol}_i + \text{lweight}_i + \text{svi}_i\right) + \varepsilon_i.$$

We define:

$$R = \begin{bmatrix} 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \in \mathbb{R}^{2\times 4}, \quad r = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \in \mathbb{R}^2.$$

Then, we can calculate the MLE of β under the null hypothesis $H_0 : R\beta = r$.

```
k = 2
R = rbind(c(0, 1, -1, 0), c(0, 0, 1, -1))
r = c(0, 0)
betareduced = betafull - drop(solve(crossprod(X), t(R)) %*% solve(R %*% solve(crossprod(X),
    t(R)), R %*% betafull - r))
residuals = prostate$lpsa - drop(X %*% betareduced)
sigmareduced = sqrt(mean(residuals^2))
print(xtable(t(data.frame(Estimate = betareduced)), digits = c(0, rep(4, 4))),
    comment = FALSE)
```

	(Intercept)	lcavol	lweight	svi
Estimate	-0.4725	0.5654	0.5654	0.5654

```
reduced = lm(lpsa ~ I(lcavol + lweight + svi), prostate)
print(xtable(summary(reduced)), comment = FALSE)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-0.4725	0.2454	-1.93	0.0572
I(lcavol + lweight + svi)	0.5654	0.0449	12.58	0.0000

First, we perform the F test. The observed value of the test statistic is 0.19 and the corresponding p-value is 0.83, which implies failure to reject the null hypothesis $H_0 : \beta_1 = \beta_2 = \beta_3$, as expected. We verify our calculations by using R's built-in anova function to compare the reduced against the full model.

```
FT = drop(crossprod(R %*% betafull - r, solve(R %*% solve(crossprod(X), t(R)),
    R %*% betafull - r)))/(k * S^2)
print(FT)
```

```
[1] 0.186172
```

```r
pf(FT, k, n - p, lower.tail = FALSE)
```

[1] 0.8304394

```r
print(xtable(anova(reduced, full)), comment = FALSE)
```

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	95	47.98				
2	93	47.78	2	0.19	0.19	0.8304

Next, we perform the Wald test. The observed value of the test statistic is 0.37 and the corresponding p-value is 0.83, which similarly implies failure to reject the null hypothesis $H_0 : \beta_1 = \beta_2 = \beta_3$. We verify our calculations by again using R's built-in anova function and specifying test = "Chisq" to compare the reduced against the full model.

```r
WT = k * FT
print(WT)
```

[1] 0.372344

```r
pchisq(WT, k, lower.tail = FALSE)
```

[1] 0.8301308

```r
print(xtable(anova(reduced, full, test = "Chisq")), comment = FALSE)
```

	Res.Df	RSS	Df	Sum of Sq	Pr(>Chi)
1	95	47.98			
2	93	47.78	2	0.19	0.8301

Subsequently, we perform the likelihood ratio test. The observed value of the test statistic is 0.39 and the corresponding p-value is 0.82, which similarly implies failure to reject the null hypothesis $H_0 : \beta_1 = \beta_2 = \beta_3$. We verify our calculations by using the lrtest function from the lmtest package to compare the reduced against the full model.

```r
library(lmtest)
LR = -2 * (logLik(reduced)[1] - logLik(full)[1])
print(LR)
```

[1] 0.3875834

```r
LR = -2 * n * log(sigmafull/sigmareduced)
print(LR)
```

[1] 0.3875834

```r
pchisq(LR, k, lower.tail = FALSE)
```

[1] 0.8238295

```r
print(xtable(lrtest(reduced, full)), comment = FALSE)
```

	#Df	LogLik	Df	Chisq	Pr(>Chisq)
1	3	-103.49			
2	5	-103.30	2	0.39	0.8238

Lastly, we perform the score test. The observed value of the test statistic is 0.39 and the corresponding p-value is 0.82, which similarly implies failure to reject the null hypothesis $H_0 : \beta_1 = \beta_2 = \beta_3$. We observe that the results of all the test statistics concur. Furthermore, the results of the asymptotic Wald, likelihood ratio and score tests are perfectly in line with the results of the exact F test, even with a moderate sample size of $n = 97$ observations.

```r
SF = crossprod(X, residuals)/sigmareduced^2
FI = crossprod(X)/sigmareduced^2
ST = drop(crossprod(SF, solve(FI, SF)))
print(ST)
```

```
[1] 0.3868101
```

```r
P = X %*% solve(crossprod(X), t(X))
ST = sum((P %*% residuals)^2)/sigmareduced^2
print(ST)
```

```
[1] 0.3868101
```

```r
pchisq(ST, k, lower.tail = FALSE)
```

```
[1] 0.8241481
```

Bias-Variance Trade-off

We know that the MSE of an estimator $\widehat{\beta}$ of a parameter vector $\beta \in \mathbb{R}^p$ can be decomposed in the following manner:

$$\operatorname{MSE}\left(\widehat{\beta}\right) = \mathbb{E}\left\|\widehat{\beta} - \beta\right\|_2^2 = \mathbb{E}\left\|\widehat{\beta} - \mathbb{E}\left(\widehat{\beta}\right)\right\|_2^2 + \left\|\mathbb{E}\left(\widehat{\beta}\right) - \beta\right\|_2^2 = \operatorname{tr}\left[\operatorname{Var}\left(\widehat{\beta}\right)\right] + \left\|\operatorname{Bias}\left(\widehat{\beta}\right)\right\|_2^2.$$

This decomposition of the MSE of an estimator is known as the bias-variance decomposition. In linear regression we know that the least squares estimator $\widehat{\beta}$ is the best linear unbiased estimator (BLUE) of β. However, this is only the case if the fitted linear model coincides with the true linear model which generated our response variable. If the fitted linear model is missing some important predictors, then our least squares estimator becomes biased. Conversely, if our fitted linear model includes some redundant predictors, then the variance of our least squares estimator gets inflated.

Now, we illustrate this bias-variance trade-off on simulated data. Suppose we have a linear model $Y = X\beta + \varepsilon$, where $X \in \mathbb{R}^{n\times p}$, $\beta \in \mathbb{R}^p$ and $\varepsilon \sim \mathcal{N}_n\left(\mathbf{0}_n, \sigma^2 I_n\right)$. First, we simulate $p = 20$ normally distributed predictors $X_1, X_2, \ldots, X_p$ of size $n = 1000$ and normalize them so that their Euclidean norm is equal to 1. The effect of the first 10 predictors on the response variable is equal to 2, while the other 10 have no effect on the response variable. Then, we simulate $n_{\text{sim}} = 10000$ samples $Y^{(1)}, Y^{(2)}, \ldots, Y^{(n_{\text{sim}})}$ of size $n = 1000$ from this linear model with $\sigma^2 = 1$.

```r
n = 1000
p = 20
beta = c(rep(2, p/2), numeric(p/2))
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
nsim = 10000
Y = drop(X %*% beta) + matrix(rnorm(n * nsim), n)
```

For $j = 1, 2, \ldots, p$, we define the linear model $Y = X^{(j)}\beta^{(j)} + \varepsilon$, where $X^{(j)} = \begin{bmatrix} X_1 & X_2 & \cdots & X_j \end{bmatrix} \in \mathbb{R}^{n\times j}$ and $\beta^{(j)} \in \mathbb{R}^j$. For $k = 1, 2, \ldots, n_{\text{sim}}$, we fit this linear model to $Y^{(k)}$ and calculate the corresponding least squares estimator as follows:

$$\widehat{\beta}^{(j,k)} = \begin{bmatrix} \left({X^{(j)}}^{\mathrm{T}} X^{(j)}\right)^{-1} {X^{(j)}}^{\mathrm{T}} Y^{(k)} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^p.$$

Then, we estimate the bias, the variance and the MSE of $\widehat{\beta}^{(j)}$ as follows:

$$\widehat{\text{Bias}}\left[\widehat{\beta}^{(j)}\right] = \left\| \frac{1}{n_{\text{sim}}} \sum_{k=1}^{n_{\text{sim}}} \widehat{\beta}^{(j,k)} - \beta \right\|_2,$$

$$\widehat{\text{Var}}\left[\widehat{\beta}^{(j)}\right] = \frac{1}{n_{\text{sim}}} \sum_{k=1}^{n_{\text{sim}}} \left\| \widehat{\beta}^{(j,k)} - \frac{1}{n_{\text{sim}}} \sum_{\ell=1}^{n_{\text{sim}}} \widehat{\beta}^{(j,\ell)} \right\|_2^2,$$

$$\widehat{\text{MSE}}\left[\widehat{\beta}^{(j)}\right] = \frac{1}{n_{\text{sim}}} \sum_{k=1}^{n_{\text{sim}}} \left\| \widehat{\beta}^{(j,k)} - \beta \right\|_2^2.$$

We verify that the estimated MSE of $\widehat{\beta}^{(j)}$ satisfies the bias-variance decomposition for $j = 1, 2, \ldots, p$.

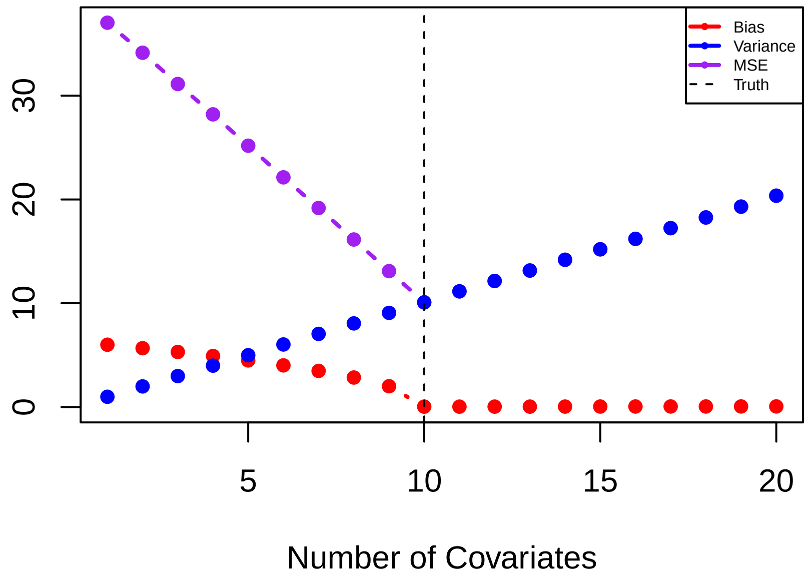
```r
Bias = numeric(p)
Var = numeric(p)
MSE = numeric(p)
for (j in 1:p) {
    betahat = rbind(solve(crossprod(X[, 1:j]), crossprod(X[, 1:j], Y)), matrix(0,
        p - j, nsim))
    Bias[j] = sqrt(sum((rowMeans(betahat) - beta)^2))
    Var[j] = mean(colSums((betahat - rowMeans(betahat))^2))
    MSE[j] = mean(colSums((betahat - beta)^2))
}
all.equal(MSE, Var + Bias^2)
```

```
## [1] TRUE
```

We observe that the estimated bias of $\widehat{\beta}^{(j)}$ is initially severely inflated for $j = 1$ and decreases as we keep adding important predictors to the linear model for $j = 2, 3, \ldots, 10$. For $j = 10$, the estimator $\widehat{\beta}^{(j)}$ becomes an unbiased estimator of β and remains so after adding other redundant predictors to the linear model for $j = 11, 12, \ldots, 20$.

On the other hand, the variance of $\widehat{\beta}^{(j)}$ steadily increases for every additional predictor we add to the linear model. Naturally, the MSE of $\widehat{\beta}^{(j)}$ strikes a balance between the bias and the variance of $\widehat{\beta}^{(j)}$. It is minimized by the true linear model with the first 10 predictors, since this linear model leads to an unbiased estimator of β without any redundant predictors to unnecessarily increase its variance. For $j = 10, 11, \ldots, 20$, the estimated MSE of $\widehat{\beta}^{(j)}$ is approximately equal to its estimated variance, since $\widehat{\beta}^{(j)}$ is unbiased.

```r
plot(MSE, type = "b", ylim = c(0, max(MSE)), xlab = "Number of Covariates",
    ylab = NA, col = "purple", pch = 16, lwd = 2)
lines(Bias, type = "b", col = "red", pch = 16, lwd = 2)
lines(Var, type = "b", col = "blue", pch = 16, lwd = 2)
abline(v = which.min(MSE), lty = 2)
legend("topright", c("Bias", "Variance", "MSE", "Truth"), col = c("red", "blue",
    "purple", "black"), lty = c(rep(1, 3), 2), lwd = c(rep(2, 3), 1), pch = c(rep(16,
    3), NA), cex = 0.5)
```



Model Selection

One of the most commonly utilized model selection methods for linear regression is step-wise regression, since it provides a computationally feasible alternative to best subset selection. Step-wise regression methods produce a sequence of linear models with increasing or decreasing number of predictors. The predictor to be added or removed at each step of the step-wise selection procedure is selected based on some pre-specified criterion. Then, the best linear model is picked out of the sequence of models produced by step-wise regression based again on some pre-specified (possibly the same) criterion.

One of the many issues with step-wise regression methods is that the same data set is usually utilized for both the initial predictor selection and the final model selection. If the set of available predictors is fairly large, then a lot of truly redundant predictors will falsely appear to have a statistically significant effect on the response variable simply due to random chance. In the forward selection procedure, the best predictor out of those predictors will be selected to be added to the linear model at each step. Since the final model selection criterion is calculated on the same set of data, it will be biased towards selecting a linear model which includes a lot of those redundant predictors, even if the penalty for each additional predictor is harsh. Consequently, these step-wise regression methods often lead to a severe over-fitting of the linear model.

One easy workaround is to split the data set into a training set, a validation set and a test set, provided that the sample size is large enough. First, step-wise regression is performed on the training set to produce a sequence of candidate linear models. Then, the model selection criterion is computed based on the validation set for each of the candidate linear models, and the best model out of them is selected. Finally, inference is made on the test set based on the selected linear model.

Now, we illustrate this over-fitting phenomenon and the proper way to address it based on simulated data. Suppose we have a linear model $Y = X\beta + \varepsilon$, where $X \in \mathbb{R}^{n \times p}$, $\beta \in \mathbb{R}^p$ and $\varepsilon \sim \mathcal{N}_n\left(\mathbf{0}_n, \sigma^2 I_n\right)$. First, we simulate $p = 2000$ normally distributed predictors $X_1, X_2, \ldots, X_p$ of size $n = 3000$ and normalize them so that their Euclidean norm is equal to 1. The effect of the first 10 predictors on the response variable is equal to 10, while the other 1990 have no effect on the response variable. Then, we take a sample Y of size $n = 3000$ from this linear model with $\sigma^2 = 1$.

```r
n = 3000
p = 2000
beta = c(rep(10, 10), numeric(p - 10))
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
Y = X %*% beta + rnorm(n)
```

First, we apply the forward selection procedure to produce a sequence of linear models with up to 20 predictors. We use a t test of statistical significance to select the best available predictor at each step of the selection procedure. In other words, we in turn add each available predictor to the previously selected linear model and calculate the p-value of the t test of statistical significance for its coefficient. Then, the predictor with the smallest possible p-value is selected to be added to the linear model. Finally, we calculate the Bayesian information criterion for each of the linear models produced by the step-wise regression method. We see that the linear model which minimizes the value of BIC makes use of at least 20 predictors, even though only 10 of them have an actual effect on the response variable.

```r
bic = numeric(20)
for (k in 1:20) {
    pval = rep(1, p)
    for (j in k:p) {
        fit = lm(Y ~ X[, c(seq_len(k - 1), j)])
        pval[j] = summary(fit)$coefficients[k + 1, 4]
    }
    ind = which.min(pval)
    X[, c(k, ind)] = X[, c(ind, k)]
    fit = lm(Y ~ X[, seq_len(k)])
    bic[k] = (k + 2) * log(n) - 2 * logLik(fit)[1]
}
which.min(bic)
```

```
## [1] 20
```

In order to rectify this, we split the data set into a training, a validation and a test set of equal sizes. First, we perform exactly the same forward selection procedure, but only on the training set. Then, we calculate the mean squared prediction error for each of the linear models produced by the step-wise regression method on the validation set. We see that the linear model which minimizes the value of MSPE makes use of exactly 10 predictors, which coincides with the true number of predictors in the simulated linear model. After selecting that linear model, we can perform inference by fitting it on the test set.

```r
train = sample(n, n/3)
valid = sample(setdiff(1:n, train), n/3)
test = setdiff(1:n, c(train, valid))
mspe = numeric(20)
for (k in 1:20) {
    pval = rep(1, p)
    for (j in k:p) {
        fit = lm(Y ~ X[, c(seq_len(k - 1), j)], subset = train)
        pval[j] = summary(fit)$coefficients[k + 1, 4]
    }
    ind = which.min(pval)
    X[, c(k, ind)] = X[, c(ind, k)]
    fit = lm(Y ~ X[, seq_len(k)], subset = train)
    mspe[k] = mean((Y[valid] - cbind(1, X[valid, seq_len(k)]) %*% fit$coefficients)^2)
}
which.min(mspe)
```

```
## [1] 10
```

```r
fit = lm(Y ~ X[, seq_len(which.min(mspe))], subset = test)
```

Box-Cox Transformation

In order to eliminate non-linearity, non-constant variance or non-normality, a transformation of the response variable is often required. In the case where the response variable is strictly positive, the most commonly used method is the Box-Cox transformation:

$$y_i^{(\lambda)} = \begin{cases} \frac{y_i^\lambda - 1}{\lambda}, & \lambda \neq 0 \\ \log y_i, & \lambda = 0 \end{cases}.$$

Then, we assume that $Y^{(\lambda)} = X\beta_\lambda + \varepsilon$, where $X \in \mathbb{R}^{n\times p}$, $\beta_\lambda \in \mathbb{R}^p$ and $\varepsilon \sim \mathcal{N}_n\left(\mathbf{0}_n, \sigma_\lambda^2 I_n\right)$.

In order to select the optimal value of λ, we first fit this linear model for a grid of different values of λ and calculate the corresponding MLEs $\widehat{\beta}_\lambda$, $\widehat{\sigma}_\lambda^2$ of β_λ and σ_λ^2 respectively. The goal is to maximize the profile likelihood $L\left(\lambda \middle| y, \widehat{\beta}_\lambda, \widehat{\sigma}_\lambda^2\right)$ of λ given the original response variable y with respect to λ. In order to calculate the profile likelihood of λ, we first need to calculate the Jacobian of the transformation:

$$\frac{\partial y_i^{(\lambda)}}{\partial y_i} = y_i^{\lambda-1}.$$

Therefore, we get that:

$$L\left(\lambda \,\middle|\, y, \widehat{\beta}_\lambda, \widehat{\sigma}^2_\lambda\right) = \prod_{i=1}^{n} f_{Y_i}\left(y_i; \lambda, \widehat{\beta}_\lambda, \widehat{\sigma}^2_\lambda\right) = \prod_{i=1}^{n} f_{Y_i^{(\lambda)}}\left(y_i^{(\lambda)}; \widehat{\beta}_\lambda, \widehat{\sigma}^2_\lambda\right) \left|\frac{\partial y_i^{(\lambda)}}{\partial y_i}\right|$$
$$= \left(2\pi\widehat{\sigma}^2_\lambda\right)^{-n/2} \exp\left\{-\frac{1}{2\widehat{\sigma}^2_\lambda}\left[y^{(\lambda)} - X\widehat{\beta}_\lambda\right]^{\mathrm{T}}\left[y^{(\lambda)} - X\widehat{\beta}_\lambda\right]\right\} \prod_{i=1}^{n} y_i^{\lambda-1}.$$

In practice, we want our linear regression model to be interpretable, so we never use the actual optimal value $\widehat{\lambda}$ yielded by the Box-Cox analysis. Instead, we use a generalized likelihood ratio test to construct a $100(1-\alpha)\%$ asymptotic confidence interval for λ. Then, we select a value of λ within the bounds of that CI which will lead to a meaningful transformation, such as a log transformation, a square root transformation or a reciprocal transformation. According to Wilks' theorem, we know that:

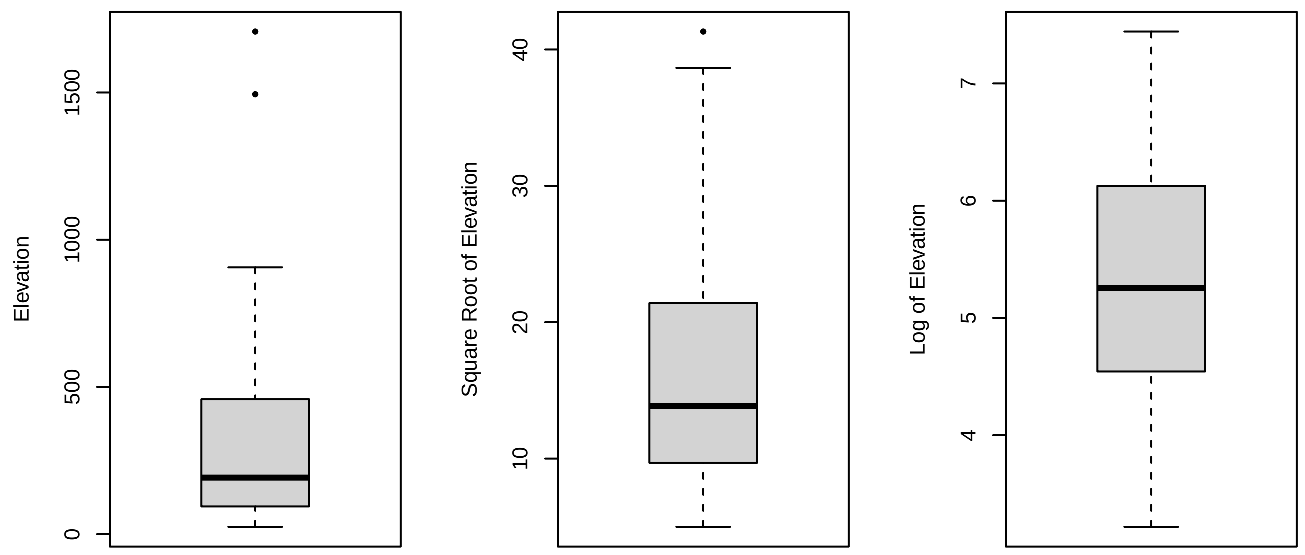
$$\mathrm{LR}_{\lambda_0} = -2\left[\ell\left(\lambda_0 \,\middle|\, y, \widehat{\beta}_{\lambda_0}, \widehat{\sigma}^2_{\lambda_0}\right) - \ell\left(\widehat{\lambda} \,\middle|\, y, \widehat{\beta}_{\widehat{\lambda}}, \widehat{\sigma}^2_{\widehat{\lambda}}\right)\right] \sim \chi^2_1,$$

under the null hypothesis $H_0 : \lambda = \lambda_0$. Then, a $100(1-\alpha)\%$ asymptotic CI for λ is given by the set of all λ_0 values which lead to a failure to reject the null hypothesis of the likelihood ratio test against the alternative hypothesis $H_1 : \lambda \neq \lambda_0$. In other words,

$$I_{1-\alpha}(\lambda) = \left\{\lambda \in \mathbb{R} : \mathrm{LR}_\lambda \leqslant \chi^2_{1;\alpha}\right\} = \left\{\lambda \in \mathbb{R} : \ell\left(\lambda \,\middle|\, y, \widehat{\beta}_\lambda, \widehat{\sigma}^2_\lambda\right) \geqslant \ell\left(\widehat{\lambda} \,\middle|\, y, \widehat{\beta}_{\widehat{\lambda}}, \widehat{\sigma}^2_{\widehat{\lambda}}\right) - \frac{1}{2}\chi^2_{1;\alpha}\right\}.$$

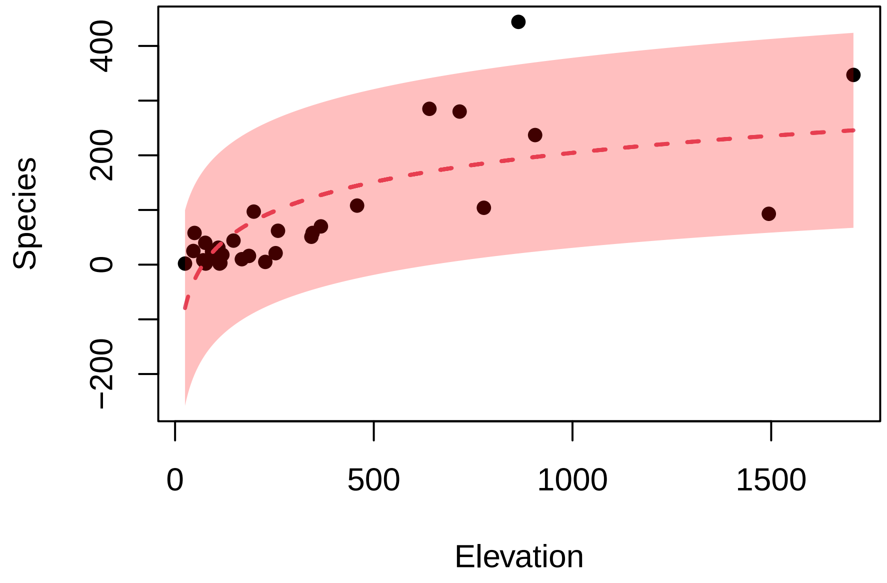
Now, we illustrate the Box-Cox transformation on the gala data set from the faraway package. First, we observe that the distribution of the elevation predictor is severely skewed. After attempting a square root and a log transformation of the predictor, we observe that the distribution of elevation appears to be approximately normal under the log transformation.

```r
library(faraway)
X = gala$Elevation
par(mfrow = c(1, 3))
boxplot(X, ylab = "Elevation", pch = 16)
boxplot(sqrt(X), ylab = "Square Root of Elevation", pch = 16)
boxplot(log(X), ylab = "Log of Elevation", pch = 16)
```

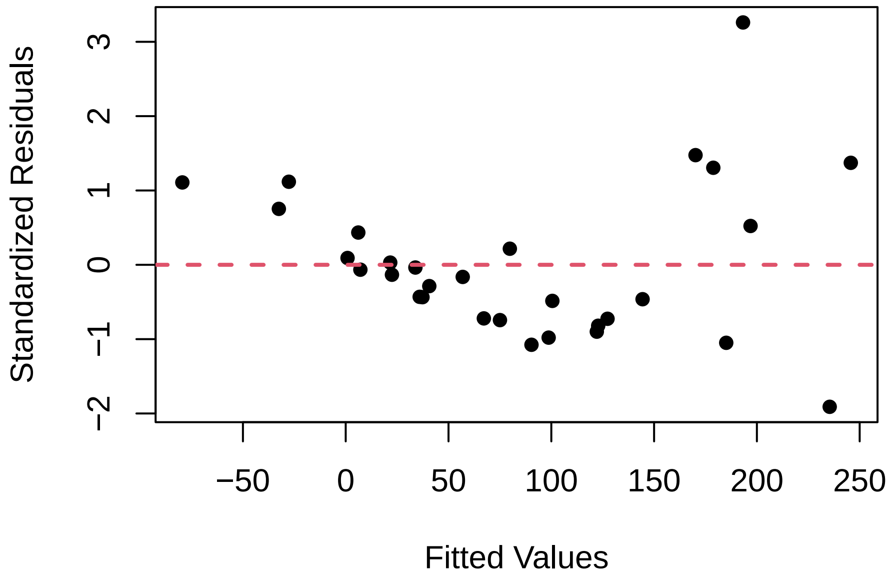


Hence, we opt to fit a linear model with log-elevation as a predictor. However, the resulting regression curve and its corresponding prediction region don't appear to capture the trend in the response variable very well. We also observe that the variation in the standardized residuals appears to increase proportionally to the fitted values, which is a clear sign of heteroscedasticity. We also see evidence of a non-linear relationship between the predictor and the response variable.

```
fit = lm(Species ~ log(Elevation), gala)
Y = fit$model[, 1]
x = seq(min(X), max(X), 0.1)
predictions = predict(fit, data.frame(Elevation = x), interval = "prediction")
plot(Species ~ Elevation, gala, ylim = c(min(predictions), max(Y)), pch = 16)
lines(predictions[, 1] ~ x, gala, "l", col = 2, lty = 2, lwd = 2)
polygon(c(x, rev(x)), c(predictions[, 2], rev(predictions[, 3])), border = NA,
    col = rgb(1, 0, 0, 0.25))
```



```r
plot(fit$fitted.values, rstandard(fit), xlab = "Fitted Values", ylab = "Standardized Residuals",
    pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```

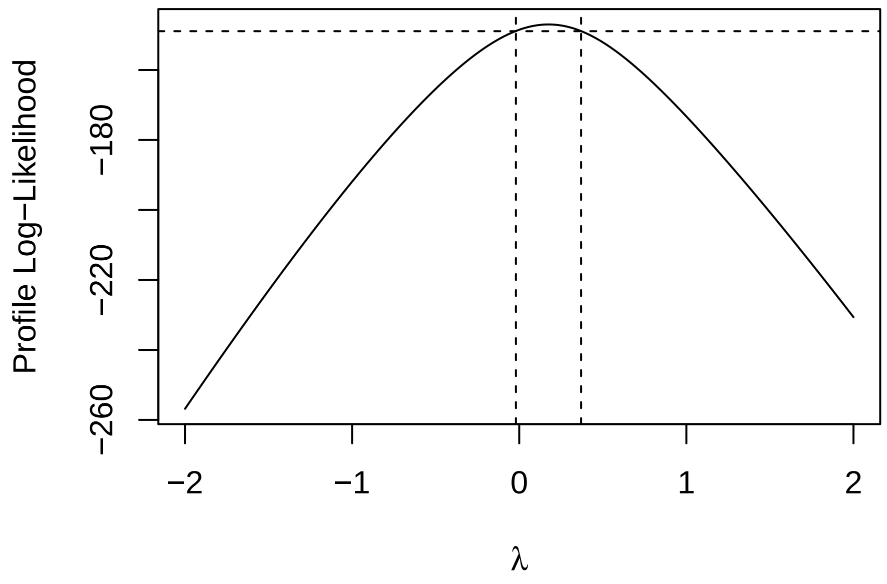


In order to rectify this, we try applying a Box-Cox transformation. We fit the linear model $Y_i^{(\lambda)} = \beta_0 + \beta_1 \log X_i + \varepsilon_i$ for a range of λ values from -2 to 2 and calculate the corresponding values of the profile log-likelihood. The resulting 95% asymptotic CI for λ ranges from -0.02 to 0.37. Since $\lambda = 0$ and $\lambda = 1/3$ lie within this CI, we conclude that a log or a cube root transformation of the response variable would be most appropriate. We can verify the validity of our calculations by using the boxcox function from the MASS package. We see that this approach yields an identical 95% CI for λ.

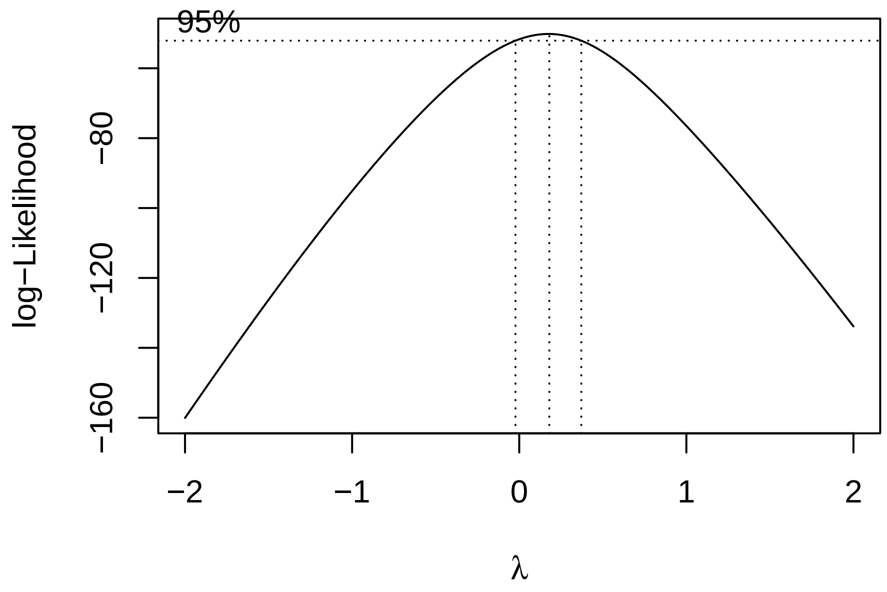
```r
library(MASS)
alpha = 0.05
lambda = seq(-2, 2, 0.01)
loglik = numeric(401)
for (i in 1:401) {
    if (lambda[i] == 0) {
        Ypower = log(Y)
    } else {
        Ypower = (Y^lambda[i] - 1)/lambda[i]
    }
    power = lm(Ypower ~ log(Elevation), gala)
    loglik[i] = logLik(power)[1] + (lambda[i] - 1) * sum(log(Y))
}
CI = range(lambda[loglik > max(loglik) - qchisq(alpha, 1, lower.tail = FALSE)/2])
print(CI)
```

```
## [1] -0.02  0.37
```

```
plot(lambda, loglik, "l", xlab = expression(lambda), ylab = "Profile Log-Likelihood")
abline(h = max(loglik) - qchisq(alpha, 1, lower.tail = FALSE)/2, lty = 2)
abline(v = CI, lty = 2)
```



```
BC = boxcox(fit, lambda)
```



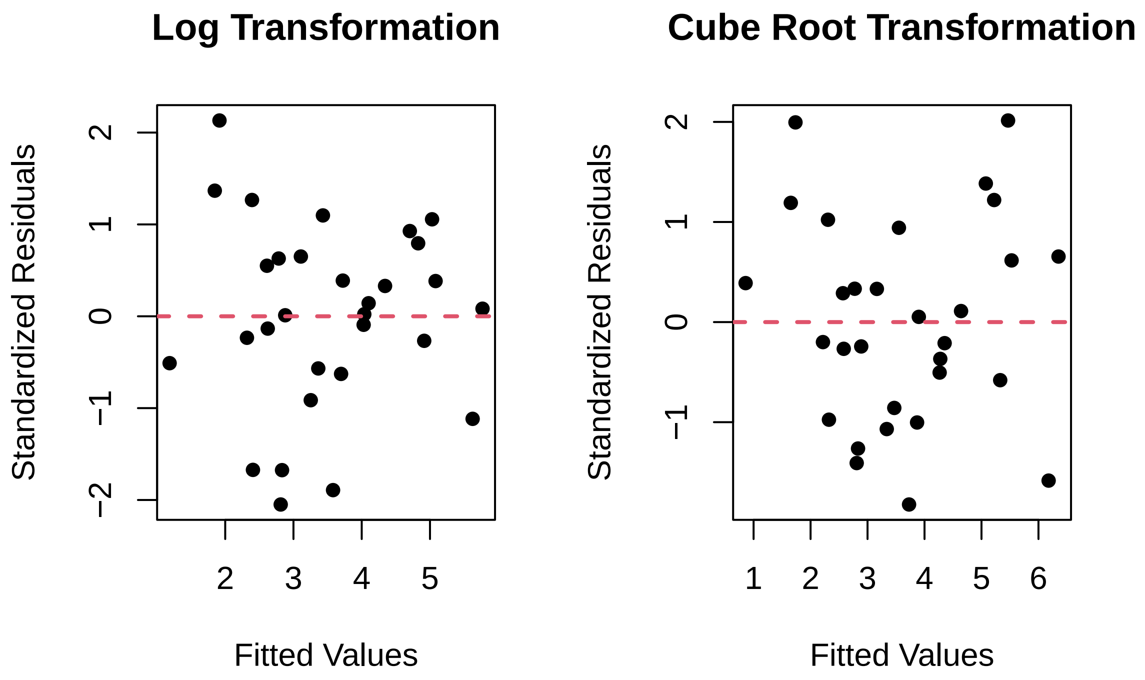
```
range(BC$x[BC$y > max(BC$y) - qchisq(alpha, 1, lower.tail = FALSE)/2])
```

```
## [1] -0.02  0.37
```

After applying either the log or the cube root transformation to the response variable, the heteroscedasticity in the standardized residuals has been eliminated, and the non-linear pattern has completely vanished.

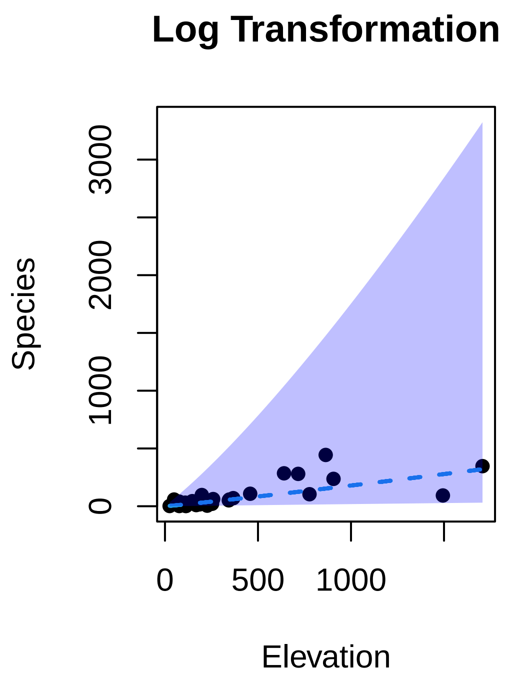
```
par(mfrow = c(1, 2))
power = lm(log(Species) ~ log(Elevation), gala)
plot(power$fitted.values, rstandard(power), main = "Log Transformation", xlab = "Fitted Values",
    ylab = "Standardized Residuals", pch = 16)
```

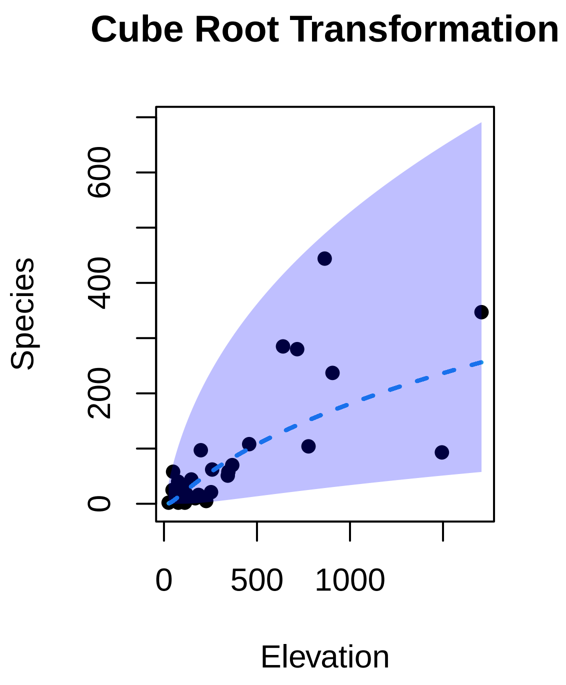
```
abline(h = 0, col = 2, lty = 2, lwd = 2)
power = lm(Species^(1/3) ~ log(Elevation), gala)
plot(power$fitted.values, rstandard(power), main = "Cube Root Transformation",
    xlab = "Fitted Values", ylab = "Standardized Residuals", pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



Since we have used a transformation on the response variable, we have to reverse the transformation on the predictions we make in order to predict the response variable on its original scale. We observe that the regression curve resulting from the cube root transformation is doing a much better job of capturing the trend in the response variable than both the original regression line the the one resulting from the log transformation.

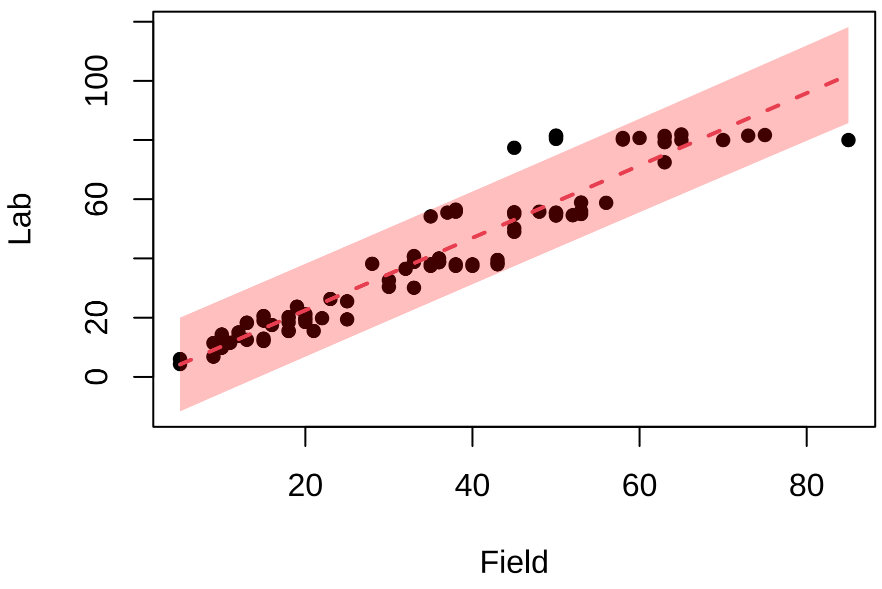
```
par(mfrow = c(1, 2))
power = lm(log(Species) ~ log(Elevation), gala)
predictions = exp(predict(power, data.frame(Elevation = x), interval = "prediction"))
plot(Species ~ Elevation, gala, main = "Log Transformation", ylim = range(predictions),
    pch = 16)
lines(predictions[, 1] ~ x, gala, "l", col = 4, lty = 2, lwd = 2)
polygon(c(x, rev(x)), c(predictions[, 2], rev(predictions[, 3])), border = NA,
    col = rgb(0, 0, 1, 0.25))
power = lm(Species^(1/3) ~ log(Elevation), gala)
predictions = predict(power, data.frame(Elevation = x), interval = "prediction")^3
plot(Species ~ Elevation, gala, main = "Cube Root Transformation", ylim = range(predictions),
    pch = 16)
lines(predictions[, 1] ~ x, gala, "l", col = 4, lty = 2, lwd = 2)
polygon(c(x, rev(x)), c(predictions[, 2], rev(predictions[, 3])), border = NA,
    col = rgb(0, 0, 1, 0.25))
```



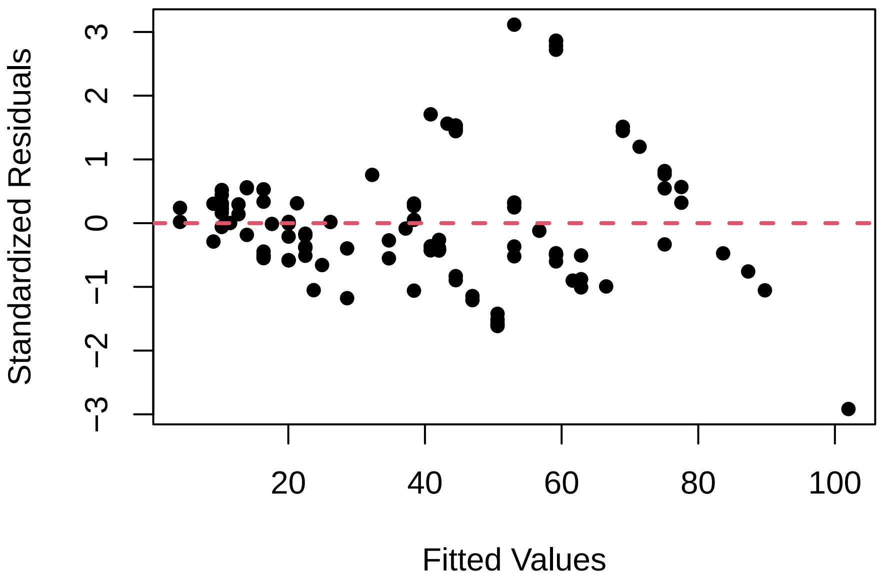


Next, we illustrate the Box-Cox transformation on the pipeline data set from the faraway package. The resulting regression line and its corresponding prediction region don't appear to capture the trend in the response variable very well, and the variation in the standardized residuals again appears to increase proportionally to the fitted values, which is a clear sign of heteroscedasticity. After applying the suggested square root transformation, the new residuals vs. fitted plot suggests that the relationship between the predictor and the response variable is no longer linear.

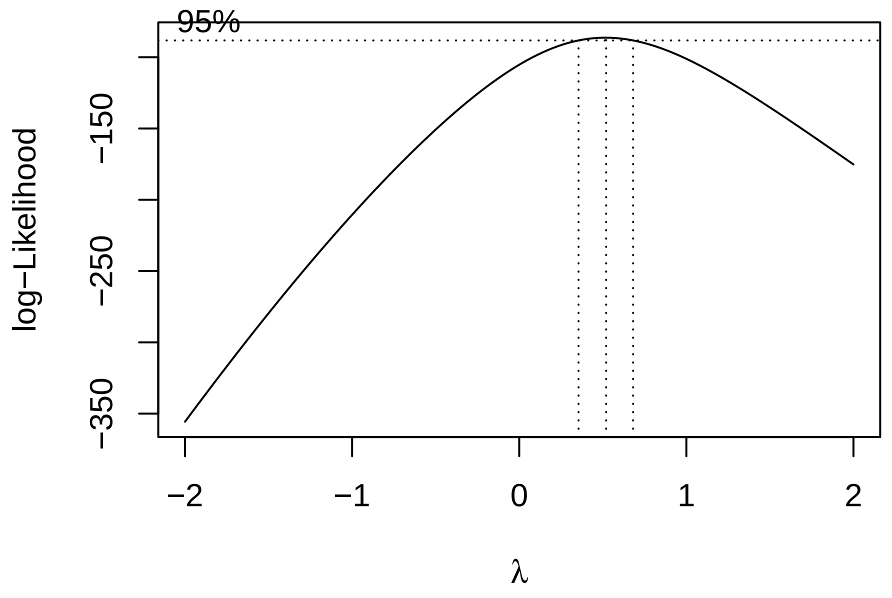
```r
fit = lm(Lab ~ Field, pipeline)
Y = fit$model[, 1]
X = model.matrix(fit)[, -1]
x = seq(min(X), max(X), 0.1)
predictions = predict(fit, data.frame(Field = x), interval = "prediction")
plot(Lab ~ Field, pipeline, ylim = range(predictions), pch = 16)
lines(predictions[, 1] ~ x, gala, "l", col = 2, lty = 2, lwd = 2)
polygon(c(x, rev(x)), c(predictions[, 2], rev(predictions[, 3])), border = NA,
    col = rgb(1, 0, 0, 0.25))
```



```
plot(fit$fitted.values, rstandard(fit), xlab = "Fitted Values", ylab = "Standardized Residuals",
    pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



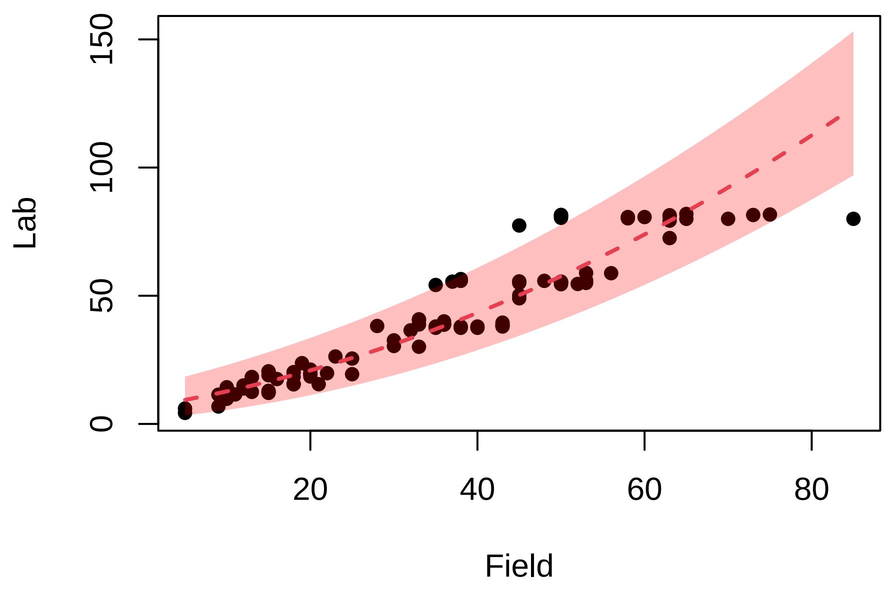
```
BC = boxcox(fit, lambda)
```



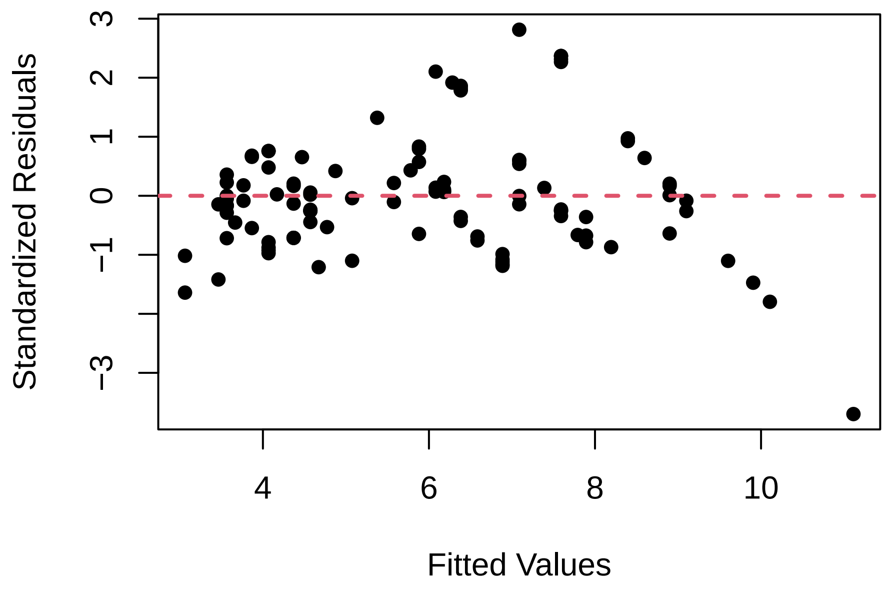
```
range(BC$x[BC$y > max(BC$y) - qchisq(alpha, 1, lower.tail = FALSE)/2])
```

```
## [1] 0.36 0.68
```

```
power = lm(sqrt(Lab) ~ Field, pipeline)
predictions = predict(power, data.frame(Field = x), interval = "prediction")^2
plot(Lab ~ Field, pipeline, ylim = range(predictions), pch = 16)
lines(predictions[, 1] ~ x, gala, "l", col = 2, lty = 2, lwd = 2)
polygon(c(x, rev(x)), c(predictions[, 2], rev(predictions[, 3])), border = NA,
    col = rgb(1, 0, 0, 0.25))
```



```
plot(power$fitted.values, rstandard(power), xlab = "Fitted Values", ylab = "Standardized Residuals",
    pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```

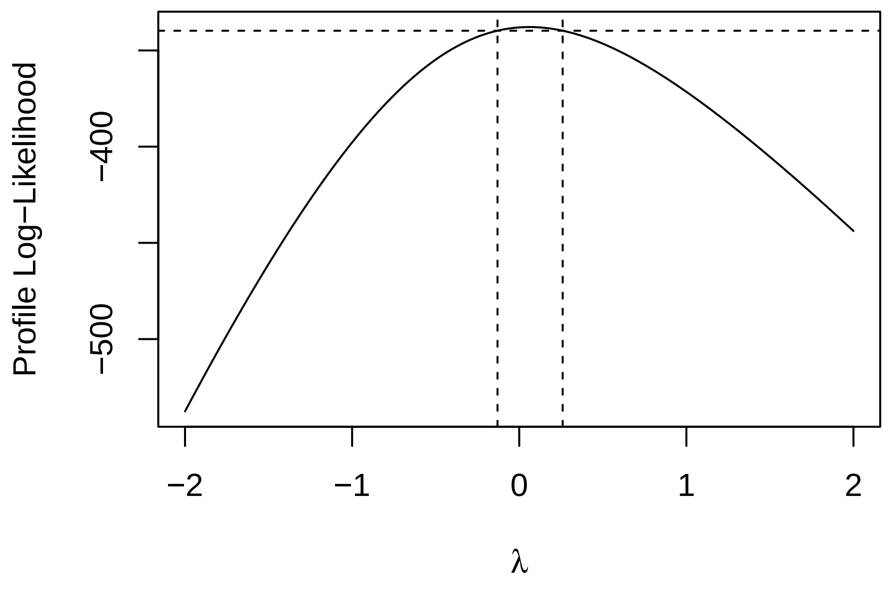


Since the predictor is measured in the same units of measurement as the response variable, we attempt simultaneously applying the same power transformation on both the response variable and the predictor. We fit the linear model $Y_i^{(\lambda)} = \beta_0 + \beta_1 X_i^{(\lambda)} + \varepsilon_i$ for a range of λ values from -2 to 2 and calculate the corresponding values of the profile log-likelihood. The resulting 95% asymptotic CI for λ ranges from -0.13 to 0.26. Since $\lambda = 0$ lies within this CI, we conclude that a log transformation of both the response and the predictor variable would be most appropriate.

```
for (i in 1:401) {
    if (lambda[i] == 0) {
        Ypower = log(Y)
        Xpower = log(X)
    } else {
        Ypower = (Y^lambda[i] - 1)/lambda[i]
        Xpower = (X^lambda[i] - 1)/lambda[i]
    }
    power = lm(Ypower ~ Xpower)
    loglik[i] = logLik(power)[1] + (lambda[i] - 1) * sum(log(Y))
}
CI = range(lambda[loglik > max(loglik) - qchisq(alpha, 1, lower.tail = FALSE)/2])
print(CI)
```

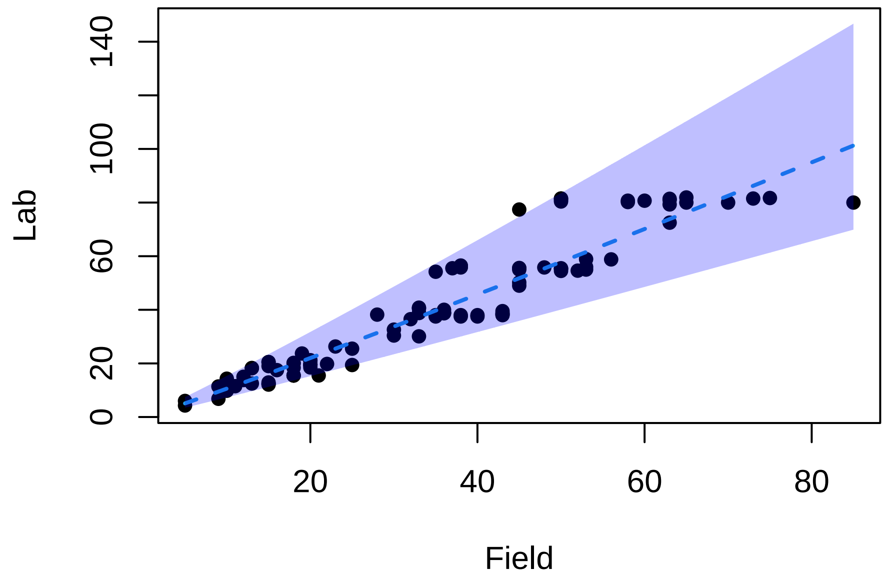
```
## [1] -0.13  0.26
```

```
plot(lambda, loglik, "l", xlab = expression(lambda), ylab = "Profile Log-Likelihood")
abline(h = max(loglik) - qchisq(alpha, 1, lower.tail = FALSE)/2, lty = 2)
abline(v = CI, lty = 2)
```

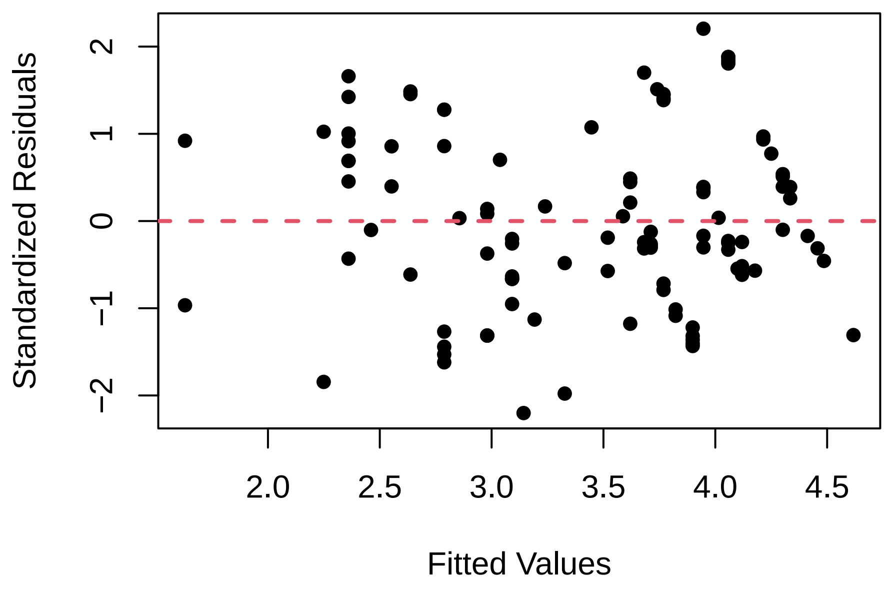


Since we have used a log-transformation on the response variable, we have to exponentiate any predictions we make in order to predict the response variable on its original scale. We observe that the resulting regression line and its corresponding prediction region appear to be doing a much better job of capturing the trend in the response variable than the original regression line, and the heteroscedasticity in the standardized residuals appears to have been eliminated.

```
power = lm(log(Lab) ~ log(Field), pipeline)
predictions = exp(predict(power, data.frame(Field = x), interval = "prediction"))
plot(Lab ~ Field, pipeline, ylim = range(predictions), pch = 16)
lines(predictions[, 1] ~ x, gala, "l", col = 4, lty = 2, lwd = 2)
polygon(c(x, rev(x)), c(predictions[, 2], rev(predictions[, 3])), border = NA,
    col = rgb(0, 0, 1, 0.25))
```



```
plot(power$fitted.values, rstandard(power), xlab = "Fitted Values", ylab = "Standardized Residuals",
    pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



Bootstrap

The least squares method leads to unbiased estimates of the regression coefficients without any distributional assumption on the error terms. On the other hand, accurate estimation of the residual variance and the standard errors of the coefficient estimators hinges on the assumptions of normality, homoscedasticity and independence of the error terms. In cases where some of these assumptions are somehow violated, the bootstrap method presents a viable non-parametric or semi-parametric alternative.

Suppose we are interested in the linear model $Y = X\beta + \varepsilon$, where $X \in \mathbb{R}^{n \times p}$, $\beta \in \mathbb{R}^p$ and $\varepsilon \sim \mathcal{N}_n\left(\mathbf{0}_n, \sigma^2 I_n\right)$. The semi-parametric bootstrap approach is also referred to as bootstrapping the residuals.

Algorithm 1.1 Semi-Parametric Bootstrap

```
    Input: Random sample $(Y, X)$ and bootstrap sample size $n_{\text{boot}}$.
1:  We calculate the least squares estimate $\widehat{\beta}$ of the regression coefficient $\beta$ and the residual vector $\widehat{\varepsilon} = Y - X\widehat{\beta}$.
2:  For $k = 1, 2, \ldots, n_{\text{boot}}$, we iterate the following steps:
    i: We take a bootstrap sample $\widehat{\varepsilon}^{(k)}$ with replacement from the residual vector $\widehat{\varepsilon}$;
    ii: We define the bootstrapped response variable $Y^{(k)} = X\widehat{\beta} + \widehat{\varepsilon}^{(k)}$;
    iii: We regress $Y^{(k)}$ on $X$ and calculate the bootstrapped least squares estimate $\widehat{\beta}^{(k)}$ of $\beta$.
3:  We calculate the sample standard deviation of the vector of bootstrapped least squares estimates. This is a
    valid estimate of the standard error of $\widehat{\beta}$.
    Output: Vector of bootstrapped least squares estimates and its sample standard deviation.
```

It should be noted that the semi-parametric bootstrap approach still assumes that the standard errors are homoscedastic and independent. If those assumptions are also in doubt, then it is better to adopt a fully non-parametric bootstrap approach.

Now, we illustrate this semi-parametric bootstrap approach on simulated data. First, we simulate a sample of size

$n = 1000$ with $p = 2$ predictors distributed in the following way:

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \sim \mathcal{N}_2\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}\right).$$

The effect of the first predictor on the response variable is 2, while the effect of the second predictor is 3. Then, we take a sample Y of size $n = 1000$ from this linear model with $\sigma^2 = 100$.

```r
library(MASS)
library(xtable)
n = 1000
p = 2
beta = c(2, 3)
X = mvrnorm(n, numeric(p), matrix(c(3, 1, 1, 2), p))
Y = X %*% beta + rnorm(n, sd = 10)
fit = lm(Y ~ X)
betahat = fit$coefficients
residuals = fit$residuals
print(xtable(summary(fit)), comment = FALSE)
```

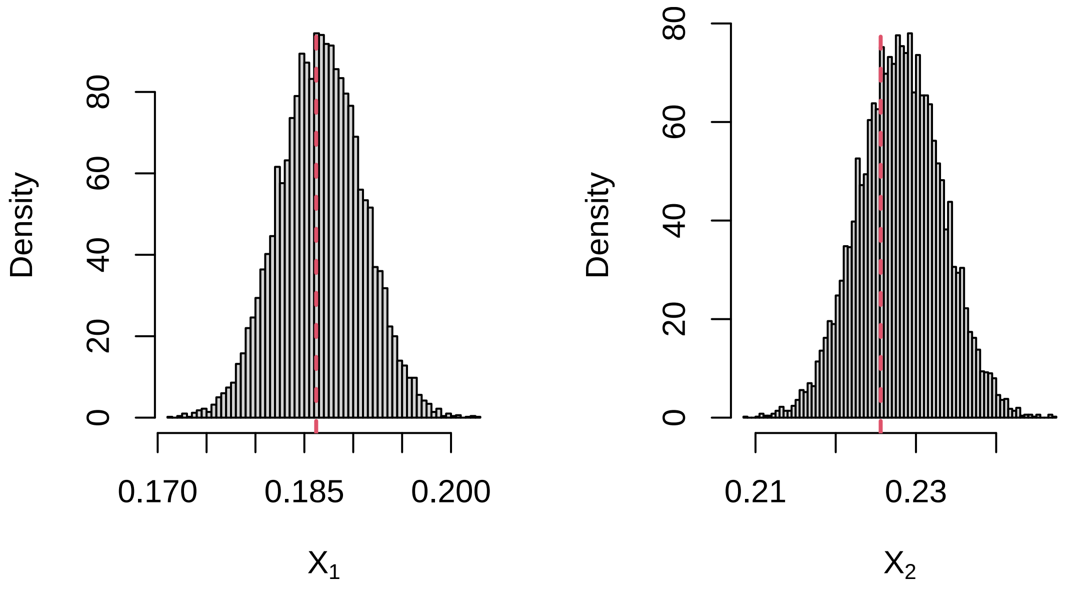
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.6880	0.3021	2.28	0.0230
X1	2.0332	0.1870	10.87	0.0000
X2	2.8289	0.2285	12.38	0.0000

First, we take $n_{\text{boot}} = 10000$ bootstrapped residual samples and calculate the corresponding bootstrapped response variables. Then, we calculate the bootstrapped least squares estimates of β and their corresponding standard errors. Finally, we calculate the medians of the bootstrapped least squares estimates, the medians of the bootstrapped standard errors and the standard deviations of the bootstrapped least squares estimates. We observe that the medians of the bootstrapped least squares estimates lie very close to the estimated regression coefficients. We can also see that the standard deviations of the bootstrapped least squares estimates are very close to the medians of the bootstrapped standard errors, which is a sign that normal linear regression accurately estimates the magnitude of the standard errors. This is not surprising, since none of the assumptions of normal linear regression are violated in this simulated data set.

```r
nboot = 10000
betaboot = matrix(0, nboot, p)
SEboot = matrix(0, nboot, p)
boot = matrix(0, p, 3)
rownames(boot) = colnames(X)[-1]
colnames(boot) = c("Median Coefficient", "Median Bootstrap S.E.", "SD of Bootstrap Coefficients")
for (i in 1:nboot) {
    Yboot = cbind(1, X) %*% betahat + sample(residuals, replace = TRUE)
    fit = lm(Yboot ~ X)
    betaboot[i, ] = fit$coefficients[-1]
    SEboot[i, ] = summary(fit)$coefficients[-1, 2]
}
boot[, 1] = apply(betaboot, 2, median)
boot[, 2] = apply(SEboot, 2, median)
boot[, 3] = apply(betaboot, 2, sd)
print(xtable(boot, digits = c(0, rep(4, 3))), comment = FALSE)
```

	Median Coefficient	Median Bootstrap S.E.	SD of Bootstrap Coefficients
1	2.0310	0.1867	0.1862
2	2.8333	0.2281	0.2256

```
par(mfrow = c(1, 2))
hist(SEboot[, 1], "FD", freq = FALSE, main = NA, xlab = expression(X[1]))
abline(v = boot[1, 3], col = 2, lty = 2, lwd = 2)
hist(SEboot[, 2], "FD", freq = FALSE, main = NA, xlab = expression(X[2]))
abline(v = boot[2, 3], col = 2, lty = 2, lwd = 2)
```



The fully non-parametric approach makes absolutely no distributional assumptions on the regression standard errors, so it is valid under any setting.

We apply this fully non-parametric approach on the same simulated data set. We can see that the standard deviations of the bootstrapped least squares estimates are very close to the medians of the bootstrapped standard errors, which is a sign that normal linear regression accurately estimates the magnitude of the standard errors. Since none of the normal linear regression assumptions are actually violated, the preceding semi-parametric approach performs slightly better than this fully non-parametric approach for the same number of bootstrapped samples.

```
for (i in 1:nboot) {
    fit = lm(Y ~ X, subset = sample(n, replace = TRUE))
    betaboot[i, ] = fit$coefficients[-1]
    SEboot[i, ] = summary(fit)$coefficients[-1, 2]
```

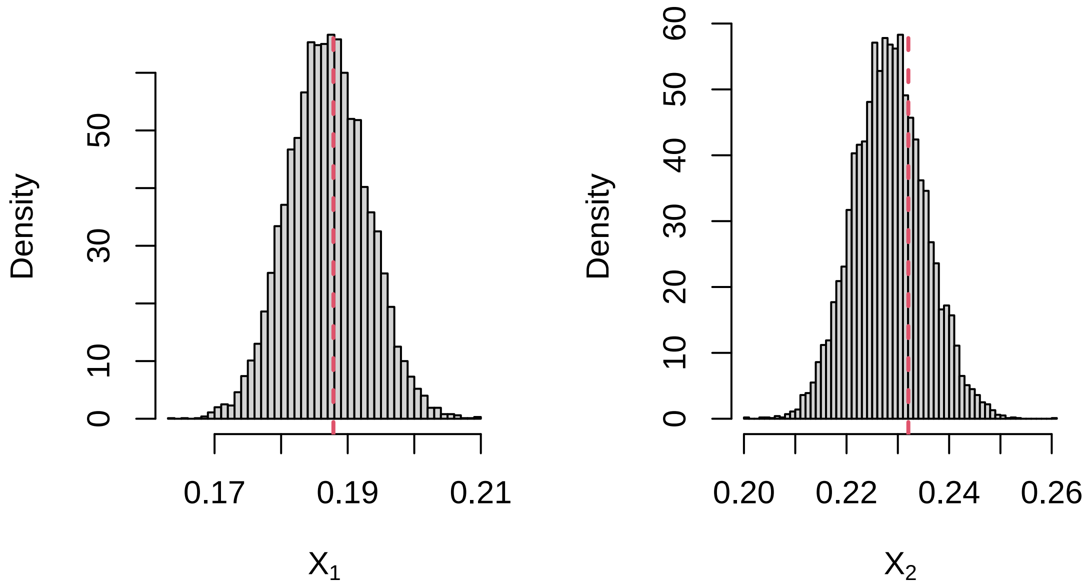
Algorithm 1.2 Non-Parametric Bootstrap

```
    **Input**: Random sample $(Y, X)$ and bootstrap sample size $n_{\text{boot}}$.
1:  For $k = 1, 2, \ldots, n_{\text{boot}}$, we iterate the following steps:
    i:  We take a bootstrap sample $\left(Y^{(k)}, X^{(k)}\right)$ with replacement from the sample $(Y, X)$;
    ii: We regress $Y^{(k)}$ on $X^{(k)}$ and calculate the bootstrapped least squares estimate $\widehat{\beta}^{(k)}$ of $\beta$.
2:  We calculate the sample standard deviation of the vector of bootstrapped least squares estimates. This is a
    valid estimate of the standard error of $\widehat{\beta}$.
    **Output**: Vector of bootstrapped least squares estimates and its sample standard deviation.
```

```r
}
boot[, 1] = apply(betaboot, 2, median)
boot[, 2] = apply(SEboot, 2, median)
boot[, 3] = apply(betaboot, 2, sd)
print(xtable(boot, digits = c(0, rep(4, 3))), comment = FALSE)
```

	Median Coefficient	Median Bootstrap S.E.	SD of Bootstrap Coefficients
1	2.0306	0.1869	0.1879
2	2.8300	0.2283	0.2320

```r
par(mfrow = c(1, 2))
hist(SEboot[, 1], "FD", freq = FALSE, main = NA, xlab = expression(X[1]))
abline(v = boot[1, 3], col = 2, lty = 2, lwd = 2)
hist(SEboot[, 2], "FD", freq = FALSE, main = NA, xlab = expression(X[2]))
abline(v = boot[2, 3], col = 2, lty = 2, lwd = 2)
```

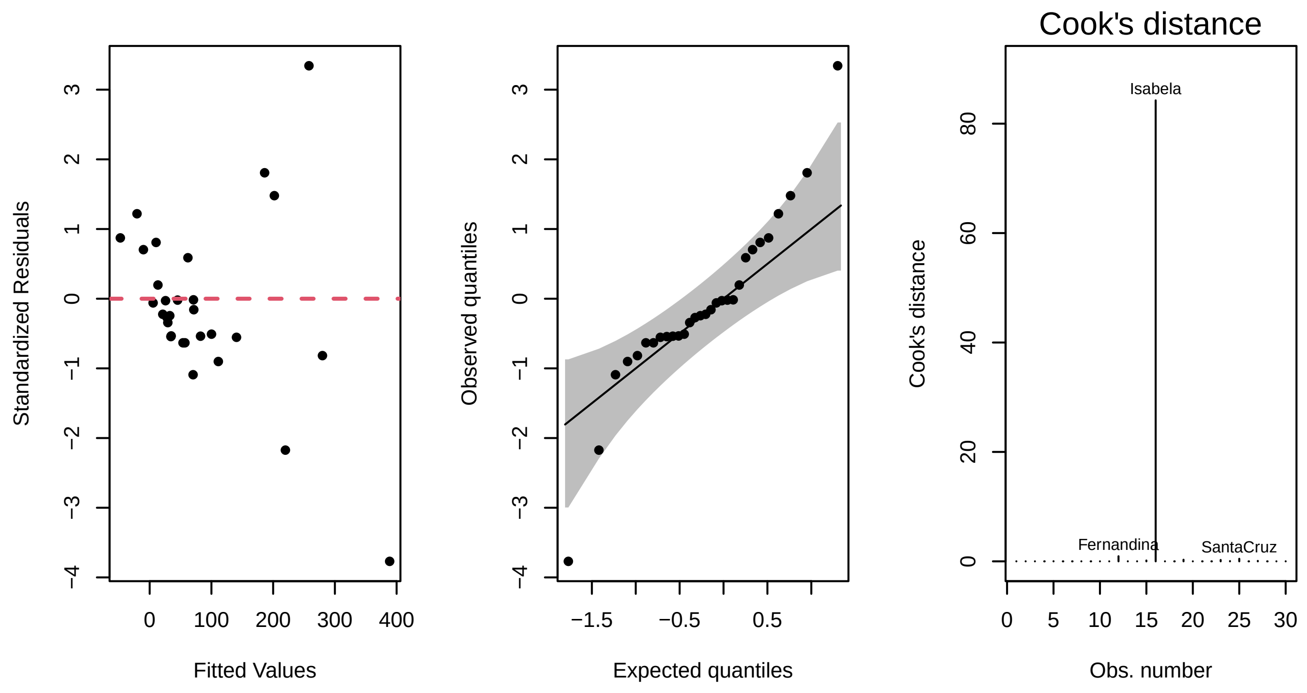


Afterwards, we consider applying these bootstrap approaches on the gala data set from the faraway package. We remark the existence of one extremely influential point as well as the fact that the variation of the standardized residuals increases proportionally with the fitted values. Thus, we expect that the usual parametric standard error estimates are going to be extremely unreliable.

```r
library(faraway)
library(qqconf)
n = dim(gala)[1]
p = 4
fit = lm(Species ~ Area + Elevation + Nearest + Adjacent, gala)
betahat = fit$coefficients
residuals = fit$residuals
X = model.matrix(fit)
print(xtable(summary(fit)), comment = FALSE)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-0.1792	18.1098	-0.01	0.9922
Area	-0.0249	0.0225	-1.11	0.2796
Elevation	0.3254	0.0537	6.06	0.0000
Nearest	-0.7273	0.8264	-0.88	0.3872
Adjacent	-0.0786	0.0175	-4.50	0.0001

```r
par(mfrow = c(1, 3))
plot(fit$fitted.values, rstandard(fit), xlab = "Fitted Values", ylab = "Standardized Residuals",
    pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
invisible(capture.output(qq_conf_plot(rstandard(fit), qnorm, points_params = list(pch = 16))))
plot(fit, 4)
```

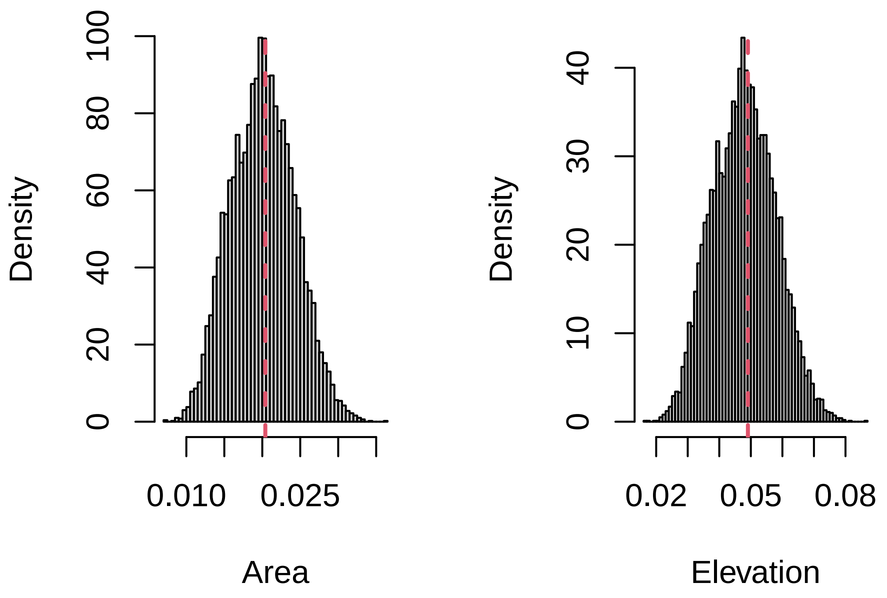


We start with the semi-parametric approach of bootstrapping the residuals. We can see that the standard deviations of the bootstrapped least squares estimates are very close to the medians of the bootstrapped standard errors, which is surprising considering the large deviations from the assumptions of normal linear regression. This is a sign that the semi-parametric approach has completely failed at capturing the true variability in the least squares estimator, so we should abandon it and move on to the fully non-parametric approach.

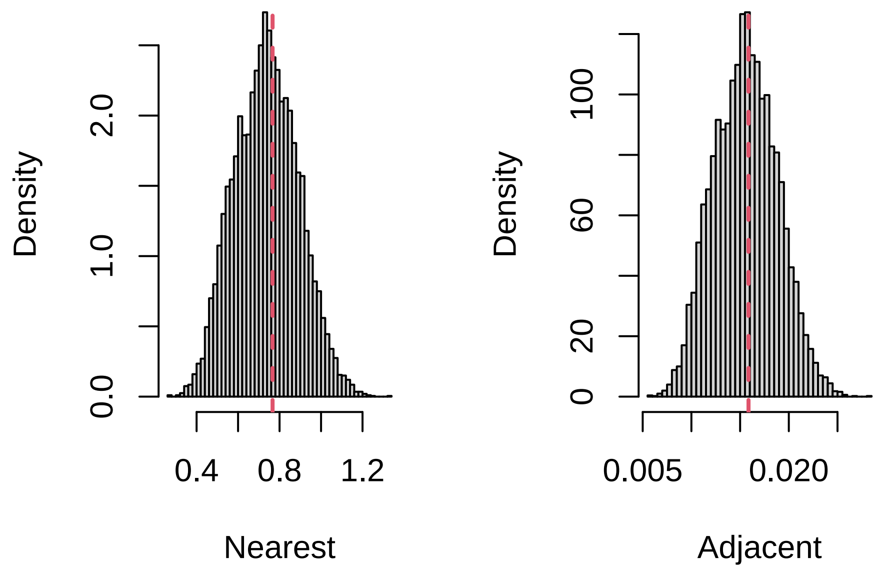
```r
betaboot = matrix(0, nboot, p)
SEboot = matrix(0, nboot, p)
boot = matrix(0, p, 3)
rownames(boot) = colnames(X)[-1]
colnames(boot) = c("Median Coefficient", "Median Bootstrap S.E.", "SD of Bootstrap Coefficients")
for (i in 1:nboot) {
    Yboot = X %*% betahat + sample(residuals, replace = TRUE)
    fit = lm(Yboot ~ Area + Elevation + Nearest + Adjacent, gala)
    betaboot[i, ] = fit$coefficients[-1]
    SEboot[i, ] = summary(fit)$coefficients[-1, 2]
}
boot[, 1] = apply(betaboot, 2, median)
boot[, 2] = apply(SEboot, 2, median)
boot[, 3] = apply(betaboot, 2, sd)
print(xtable(boot, digits = c(0, rep(4, 3))), comment = FALSE)
```

	Median Coefficient	Median Bootstrap S.E.	SD of Bootstrap Coefficients
Area	-0.0259	0.0201	0.0204
Elevation	0.3245	0.0478	0.0491
Nearest	-0.7834	0.7368	0.7663
Adjacent	-0.0794	0.0156	0.0159

```r
par(mfrow = c(1, 2))
hist(SEboot[, 1], "FD", freq = FALSE, main = NA, xlab = rownames(boot)[1])
abline(v = boot[1, 3], col = 2, lty = 2, lwd = 2)
hist(SEboot[, 2], "FD", freq = FALSE, main = NA, xlab = rownames(boot)[2])
abline(v = boot[2, 3], col = 2, lty = 2, lwd = 2)
```



```r
hist(SEboot[, 3], "FD", freq = FALSE, main = NA, xlab = rownames(boot)[3])
abline(v = boot[3, 3], col = 2, lty = 2, lwd = 2)
hist(SEboot[, 4], "FD", freq = FALSE, main = NA, xlab = rownames(boot)[4])
abline(v = boot[4, 3], col = 2, lty = 2, lwd = 2)
```

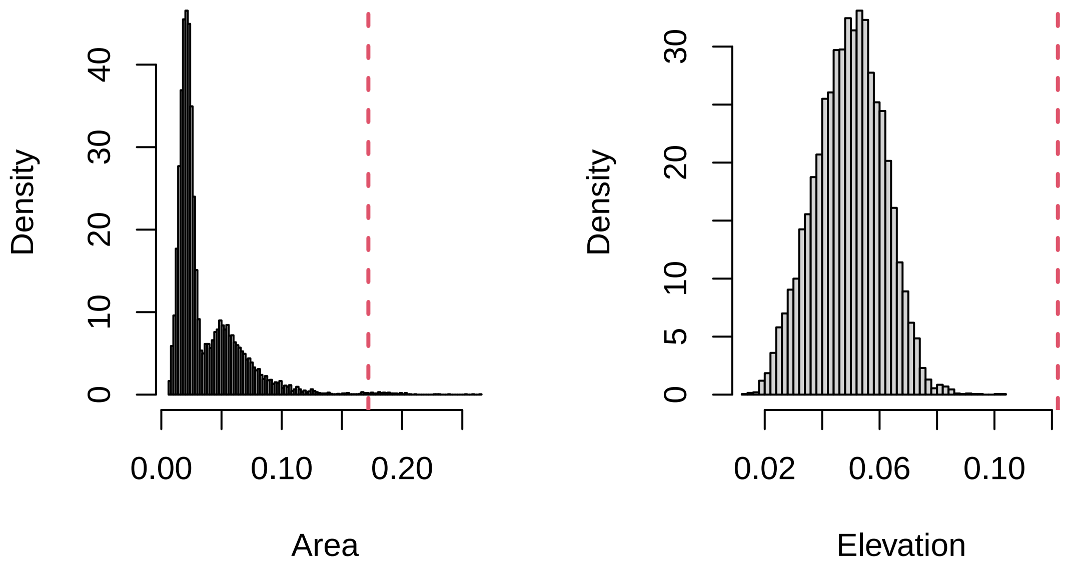


Then, we apply the non-parametric approach of bootstrapping the entire sample. We can now see that the standard deviations of the bootstrapped least squares estimates massively deviate from the medians of the bootstrapped standard errors. More precisely, we observe that normal linear regression severely underestimates the true standard errors of our least squares estimators. On the other hand, the medians of the bootstrapped least squares estimates lie really close to the estimated regression coefficients, even though the assumptions of normal linear regression are violated.

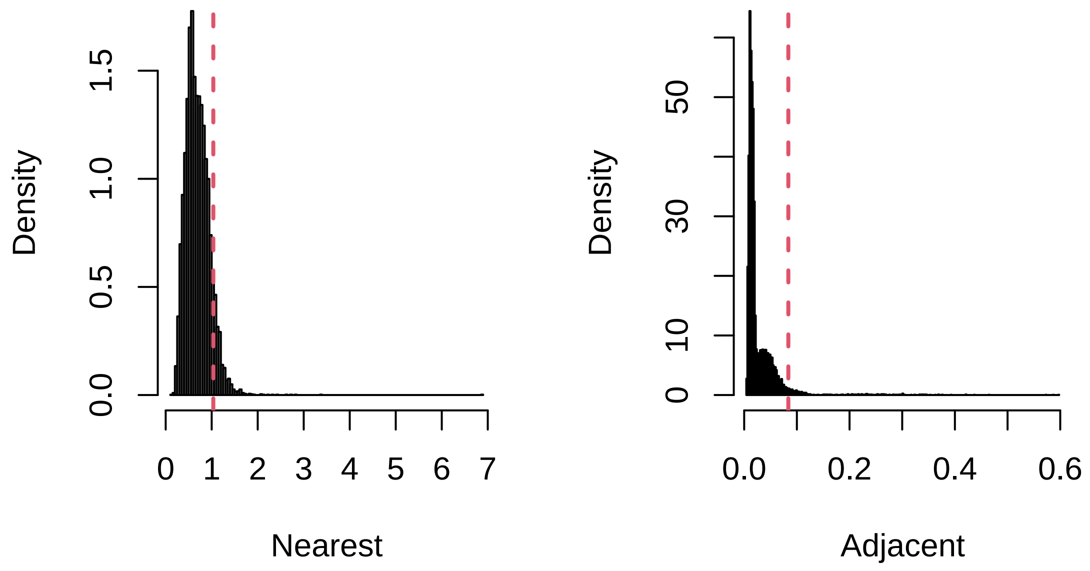
```r
for (i in 1:nboot) {
    fit = lm(Species ~ Area + Elevation + Nearest + Adjacent, gala, sample(n,
        replace = TRUE))
    betaboot[i, ] = fit$coefficients[-1]
    SEboot[i, ] = summary(fit)$coefficients[-1, 2]
}
boot[, 1] = apply(betaboot, 2, median)
boot[, 2] = apply(SEboot, 2, median)
boot[, 3] = apply(betaboot, 2, sd)
print(xtable(boot, digits = c(0, rep(4, 3))), comment = FALSE)
```

	Median Coefficient	Median Bootstrap S.E.	SD of Bootstrap Coefficients
Area	-0.0116	0.0247	0.1720
Elevation	0.2768	0.0499	0.1221
Nearest	-0.6172	0.6644	1.0351
Adjacent	-0.0657	0.0164	0.0838

```
par(mfrow = c(1, 2))
hist(SEboot[, 1], "FD", freq = FALSE, main = NA, xlab = rownames(boot)[1])
abline(v = boot[1, 3], col = 2, lty = 2, lwd = 2)
hist(SEboot[, 2], "FD", freq = FALSE, main = NA, xlim = c(min(SEboot[, 2]),
    boot[2, 3]), xlab = rownames(boot)[2])
abline(v = boot[2, 3], col = 2, lty = 2, lwd = 2)
```



```
hist(SEboot[, 3], "FD", freq = FALSE, main = NA, xlab = rownames(boot)[3])
abline(v = boot[3, 3], col = 2, lty = 2, lwd = 2)
hist(SEboot[, 4], "FD", freq = FALSE, main = NA, xlab = rownames(boot)[4])
abline(v = boot[4, 3], col = 2, lty = 2, lwd = 2)
```



Permutation Tests

The classical test statistics for linear regression heavily rely on the assumption of normality of the response variable. In cases where this assumption is somehow violated, permutation tests present a viable non-parametric alternative. Suppose we are interested in the linear model $Y_i = \beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip} + \varepsilon_i$, where $\varepsilon_i \sim \mathcal{N}\left(0, \sigma^2\right)$ are independent. First, we want to perform an overall test of statistical significance $H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$ vs. every possible alternative. The classical approach is to use the overall F test of statistical significance to draw inference. In the absence of normality, we need to resort to a non-parametric approach. Under the null hypothesis, we observe that the original sample $\{(Y_i, X_{i1}, \ldots, X_{ip})\}$ has the same distribution as a permuted sample $\left\{\left(Y_{\pi(i)}, X_{i1}, \ldots, X_{ip}\right)\right\}$ for any random permutation $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$.

Algorithm 1.3 Permutation F Test of Overall Statistical Significance

```
Input: Random sample $(Y, X)$ and permutation sample size $n_{\text{perm}}$.
1: We calculate the observed value $f^{\text{obs}}$ of the $F$ test statistic on the original sample $(Y, X)$.
2: For $k = 1, 2, \ldots, n_{\text{perm}}$, we iterate the following steps:
   i: We take a random permutation $Y^{(k)}$ of the response variable $Y$;
   ii: We calculate the observed value $f_k^{\text{perm}}$ of the $F$ test statistic on the permuted sample $\left(Y^{(k)}, X\right)$.
3: We can estimate the p-value of the $F$ test as follows:
   $$\text{p-value} = \frac{1 + \sum_{k=1}^{n_{\text{perm}}} \mathbb{1}_{\{f_k^{\text{perm}} > f^{\text{obs}}\}}}{1 + n_{\text{perm}}}.$$
Output: Estimated $F$ test p-value.
```

Although this approach makes use of the exact same test statistic as the parametric approach, the calculation of the p-value makes absolutely no distributional assumptions on the data, so it constitutes a fully non-parametric approach.

Now, we illustrate this permutation test on simulated data. First, we simulate $p = 5$ normally distributed predictors

$X_1, X_2, \dots, X_p$ of size $n = 100$ and normalize them so that their Euclidean norm is equal to 1. The effect of all p predictors on the response variable is equal to 1. Then, we take a sample Y of size $n = 100$ from this linear model with $\sigma^2 = 1$.

```r
n = 100
p = 5
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
beta = rep(1, p)
Y = X %*% beta + rnorm(n)
fit = lm(Y ~ X)
```

To start with, we calculate the observed value of the F test statistic and the corresponding p-value yielded by the usual parametric approach, which hypothesizes that $F \sim F_{p,n-p-1}$ under the null hypothesis. Then, we take $n_{\text{perm}} = 10000$ random permutations of the response variable and calculate the observed value of the F test statistic for each of them. The estimated p-value of this approach is almost equal to the p-value of the parametric approach, since there is no violation of the normality assumption in the simulated data set. Furthermore, the distribution of the permuted F statistic values closely agrees with the theoretical $F_{p,n-p-1}$ distribution of the parametric approach under the null hypothesis.

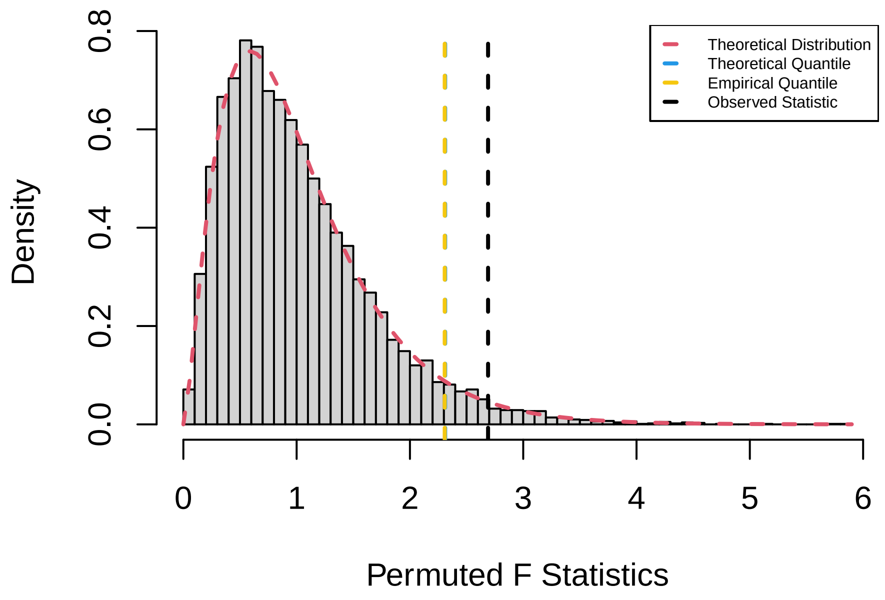
```r
alpha = 0.05
f = as.vector(summary(fit)$fstatistic)[1]
pf(f, p, n - p - 1, lower.tail = FALSE)
```

```
## [1] 0.02567136
```

```r
nperm = 10000
Fperm = numeric(nperm)
for (i in 1:nperm) {
    Yperm = sample(Y)
    perm = lm(Yperm ~ X)
    Fperm[i] = summary(perm)$fstatistic[1]
}
print((1 + sum(Fperm > f))/(1 + nperm))
```

```
## [1] 0.02409759
```

```r
hist(Fperm, "FD", freq = FALSE, main = NA, xlab = "Permuted F Statistics")
curve(df(x, p, n - p - 1), add = TRUE, col = 2, lty = 2, lwd = 2)
abline(v = qf(1 - alpha, p, n - p - 1), col = 4, lty = 2, lwd = 2)
abline(v = quantile(Fperm, 1 - alpha), col = 7, lty = 2, lwd = 2)
abline(v = f, lty = 2, lwd = 2)
legend("topright", c("Theoretical Distribution", "Theoretical Quantile", "Empirical Quantile",
    "Observed Statistic"), col = c(2, 4, 7, 1), lty = rep(2, 4), lwd = rep(2,
    4), cex = 0.5)
```



Afterwards, we endeavor to apply the same logic to a test of statistical significance for a single regression coefficient, let's say $H_0 : \beta_1 = 0$ vs. $H_1 : \beta_1 \neq 0$. Before we can permute the response variable, we first need to regress out the effect of all other predictors except for X_1, since their effect on the response variable is generally non-zero under the null hypothesis.

Algorithm 1.4 Permutation t Test of Statistical Significance

```
**Input**: Random sample $(Y, X)$ and permutation sample size $n_{\text{perm}}$.
1: We calculate the observed value $t^{\text{obs}}$ of the $t$ test statistic for the statistical significance of $\beta_1$ on the original sample $(Y, X)$.
2: We regress $Y$ on $X_2, X_3, \dots, X_p$. We calculate the vectors $\widehat{Y}$ of fitted values and $\widehat{\varepsilon}$ of residuals from this linear regression.
3: For $k = 1, 2, \dots, n_{\text{perm}}$, we iterate the following steps:
    i: We take a random permutation $\widehat{\varepsilon}^{(k)}$ of the residual vector $\widehat{\varepsilon}$;
    ii: We define the permuted response variable $Y^{(k)} = \widehat{Y} + \widehat{\varepsilon}^{(k)}$;
    iii: We regress $Y^{(k)}$ on $X_1, X_2, \dots, X_p$;
    iv: We calculate the observed value $t_k^{\text{perm}}$ of the $t$ test statistic for the statistical significance of $\beta_1$ on the permuted sample $\left(Y^{(k)}, X\right)$.
4: We can estimate the p-value of the $t$ test as follows:
       $$\text{p-value} = \frac{1 + \sum_{k=1}^{n_{\text{perm}}} \mathbb{1}_{\{|t_k^{\text{perm}}| > |t^{\text{obs}}|\}}}{1 + n_{\text{perm}}}.$$
**Output**: Estimated $t$ test p-value for $\beta_1$.
```

Now, we apply this permutation test on the same simulated data set. To start with, we calculate the observed value of the t test statistic and the corresponding p-value yielded by the parametric approach, which hypothesizes that $t \sim t_{n-p-1}$ under the null hypothesis $H_0 : \beta_1 = 0$. Then, we take $n_{\text{perm}} = 10000$ random permutations of the response variable according to this approach and calculate the observed value of the t test statistic for each of them.

The estimated p-value of this approach is almost equal to the p-value of the parametric approach since there is no violation of the normality assumption in the simulated data set. Furthermore, the distribution of the permuted t statistic values closely agrees with the theoretical t_{n-p-1} distribution of the parametric approach under the null hypothesis.

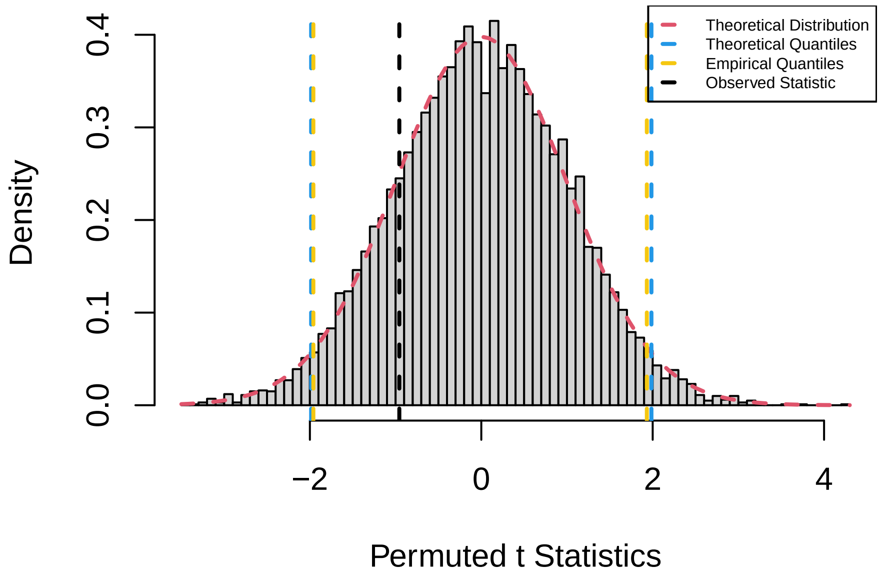
```r
t = summary(fit)$coefficients[2, 3]
print(2 * pt(abs(t), n - p - 1, lower.tail = FALSE))
```

```
## [1] 0.3412679
```

```r
aux = lm(Y ~ X[, -1])
Yhat = aux$fitted.values
Yres = aux$residuals
tperm = numeric(nperm)
for (i in 1:nperm) {
    Yperm = Yhat + sample(Yres)
    perm = lm(Yperm ~ X)
    tperm[i] = summary(perm)$coefficients[2, 3]
}
print((1 + sum(abs(tperm) > abs(t)))/(1 + nperm))
```

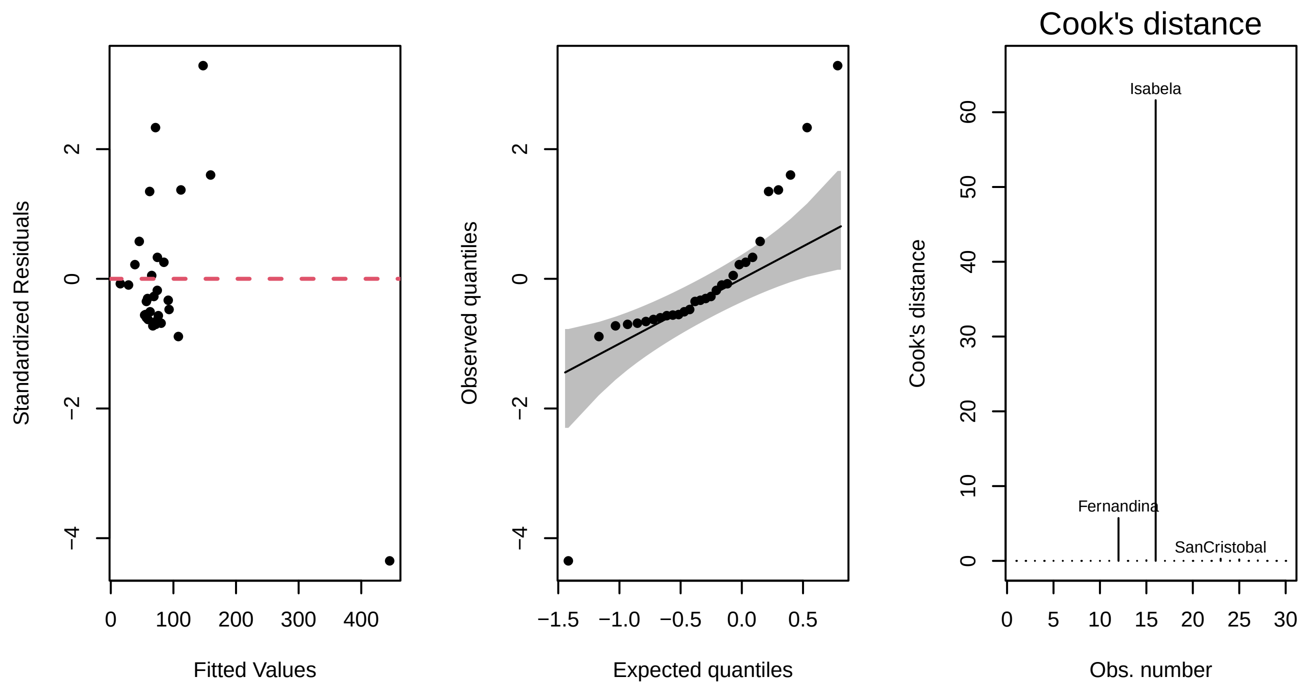
```
## [1] 0.3465653
```

```r
hist(tperm, "FD", freq = FALSE, main = NA, xlab = "Permuted t Statistics")
curve(dt(x, n - p - 1), add = TRUE, col = 2, lty = 2, lwd = 2)
abline(v = qt(c(alpha/2, 1 - alpha/2), n - p - 1), col = 4, lty = 2, lwd = 2)
abline(v = quantile(tperm, c(alpha/2, 1 - alpha/2)), col = 7, lty = 2, lwd = 2)
abline(v = t, lty = 2, lwd = 2)
legend("topright", c("Theoretical Distribution", "Theoretical Quantiles", "Empirical Quantiles",
    "Observed Statistic"), col = c(2, 4, 7, 1), lty = rep(2, 4), lwd = rep(2,
    4), cex = 0.5)
```



Afterwards, we apply these permutation tests on the gala data set from the faraway package. We remark the existence of some extreme outliers which clash with the normality assumption. Thus, we expect that the usual parametric testing approaches are going to be extremely unreliable.

```
library(faraway)
library(qqconf)
n = dim(gala)[1]
p = 4
fit = lm(Species ~ Area + Nearest + Scruz + Adjacent, gala)
Y = fit$model[, 1]
X = model.matrix(fit)[, -1]
par(mfrow = c(1, 3))
plot(fit$fitted.values, rstandard(fit), xlab = "Fitted Values", ylab = "Standardized Residuals",
    pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
invisible(capture.output(qq_conf_plot(rstandard(fit), qnorm, points_params = list(pch = 16))))
plot(fit, 4)
```



We start with the overall F test of statistical significance. Unsurprisingly, the estimated p-value of the permutation approach is far off from the p-value computed on the basis of the hypothesized parametric distribution. Additionally, the distribution of the permuted F statistic values, which is an approximation of the true null distribution of the test statistic, has a significantly different shape from the theoretical $F_{p,n-p-1}$ distribution. In particular, the theoretical distribution has a much lighter tail than the empirical distribution, which implies that the parametric approach has a much higher than nominal type I error.

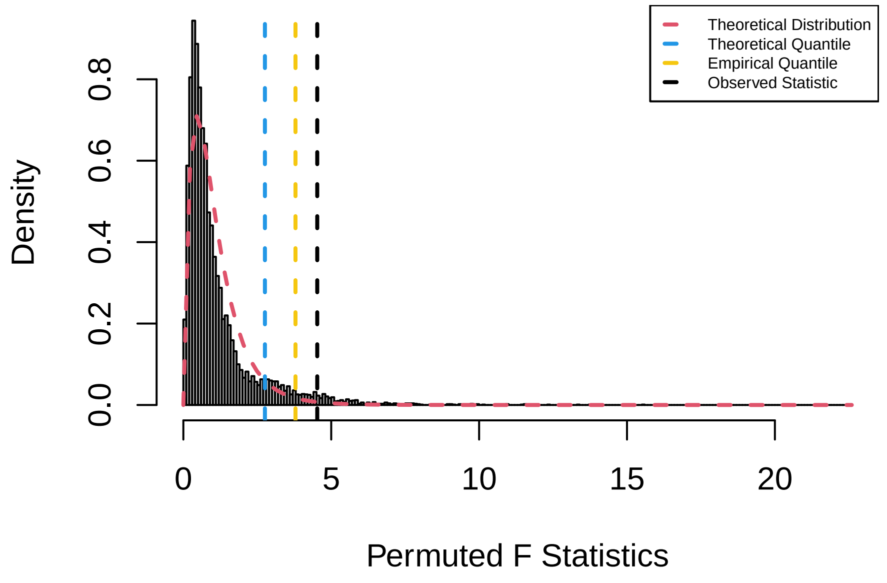
```
f = as.vector(summary(fit)$fstatistic)[1]
pf(f, p, n - p - 1, lower.tail = FALSE)
```

```
## [1] 0.006887787
```

```
for (i in 1:nperm) {
    Yperm = sample(Y)
    perm = lm(Yperm ~ X)
    Fperm[i] = summary(perm)$fstatistic[1]
}
print((1 + sum(Fperm > f))/(1 + nperm))
```

```
## [1] 0.03089691
```

```
hist(Fperm, "FD", freq = FALSE, main = NA, xlab = "Permuted F Statistics")
curve(df(x, p, n - p - 1), add = TRUE, col = 2, lty = 2, lwd = 2)
abline(v = qf(1 - alpha, p, n - p - 1), col = 4, lty = 2, lwd = 2)
abline(v = quantile(Fperm, 1 - alpha), col = 7, lty = 2, lwd = 2)
abline(v = f, lty = 2, lwd = 2)
legend("topright", c("Theoretical Distribution", "Theoretical Quantile", "Empirical Quantile",
    "Observed Statistic"), col = c(2, 4, 7, 1), lty = rep(2, 4), lwd = rep(2,
    4), cex = 0.5)
```



Then, we perform a test of statistical significance for the area predictor. Unsurprisingly, the estimated p-value of the permutation approach is far off from the p-value computed on the basis of the hypothesized parametric distribution. Additionally, the distribution of the permuted t statistic values, which is an approximation of the true null distribution of the test statistic, has a wildly different shape from the theoretical t_{n-p-1} distribution. In particular, the theoretical distribution has a much lighter right tail and a much heavier left tail than the empirical distribution, which implies that the parametric approach has a poorly calibrated type I error as well a severe loss in power compared to the permutation approach.

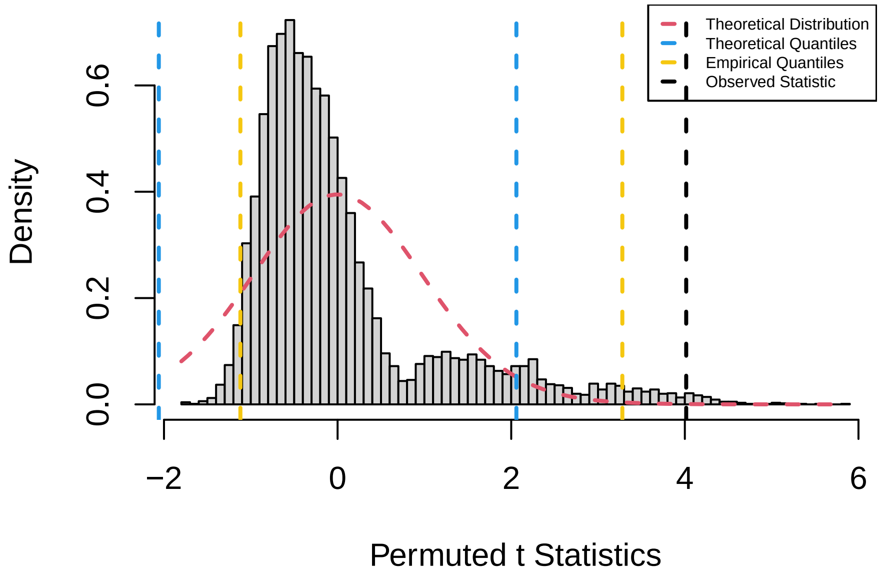
```
t = summary(fit)$coefficients[2, 3]
print(2 * pt(abs(t), n - 2, lower.tail = FALSE))
```

```
## [1] 0.0004030351
```

```r
aux = lm(Y ~ X[, -1])
Yhat = aux$fitted.values
Yres = aux$residuals
for (i in 1:nperm) {
    Yperm = Yhat + sample(Yres)
    perm = lm(Yperm ~ X)
    tperm[i] = summary(perm)$coefficients[2, 3]
}
print((1 + sum(abs(tperm) > abs(t)))/(1 + nperm))
```

```
## [1] 0.00829917
```

```r
hist(tperm, "FD", freq = FALSE, main = NA, xlab = "Permuted t Statistics")
curve(dt(x, n - p - 1), add = TRUE, col = 2, lty = 2, lwd = 2)
abline(v = qt(c(alpha/2, 1 - alpha/2), n - p - 1), col = 4, lty = 2, lwd = 2)
abline(v = quantile(tperm, c(alpha/2, 1 - alpha/2)), col = 7, lty = 2, lwd = 2)
abline(v = t, lty = 2, lwd = 2)
legend("topright", c("Theoretical Distribution", "Theoretical Quantiles", "Empirical Quantiles",
    "Observed Statistic"), col = c(2, 4, 7, 1), lty = rep(2, 4), lwd = rep(2,
    4), cex = 0.5)
```



Missing Data

Any data entries whose response value is missing from the data are generally discarded during statistical analysis. On the other hand, the handling of missing values in the predictors of a linear model depends on the type of missingness. There are 3 main types of missingness:

- **Missing completely at random**: For any predictor j, values are missing uniformly at random with probability p_j.
- **Missing at random**: For any predictor j, value x_{ij} is missing with probability p_{ij} which is a function of

other observed predictor values.

- **Missing not at random**: For any predictor j, value x_{ij} is missing with probability p_{ij} which is a function of unobserved data.

There are 4 main ways of handling missing values in data:

- **Discarding**: Ignore all data entries with a missing value in at least one predictor, and perform inference using the rest of the data set.
- **Mean Imputation**: Calculate the sample average $\overline{x}_j$ of X_j by ignoring the missing values, and substitute any missing values x_{ij} for $\overline{x}_j$.
- **Deterministic Regression Imputation**: Regress X_j on the rest of the predictors, and substitute and missing values x_{ij} for the point predictions given by the fitted model.
- **Stochastic Regression Imputation**: Regress X_j on the rest of the predictors, and substitute and missing values x_{ij} for new simulated values from the fitted model.

If predictor values are missing completely at random, then discarding data entries with at least one missing value is a viable option. However, as the number of available predictors increases, the probability of a data entry not having any missing value becomes really small, even if the individual probabilities of missingness for each predictor are really small. Hence, some imputation method needs to be applied when the fraction of data entries with at least one missing value is fairly large.

If predictor values are missing at random, then discarding data entries with at least one missing value may lead to severe estimation bias, so it is important to utilize some method of imputation. Mean imputation has the drawback of not preserving the initial relationship among predictors. On the other hand, deterministic regression imputation may significantly increase the collinearity among predictors. Stochastic regression imputation solves this problem by adding some random noise to the predictions made by the auxiliary regression model.

If predictor values are missing not at random, then any approach may lead to severely biased results. The only surefire way of dealing with this type of missingness is to make sure that this phenomenon doesn't occur in the duration of the study.

We illustrate these methods on the sat data set from the faraway package. we want to estimate the effect of expend and takers on the average total SAT scores.

```
library(faraway)
library(xtable)
n = dim(sat)[1]
fit = lm(total ~ expend + takers, sat)
betahat = fit$coefficients[-1]
print(xtable(cor(sat[, c(1, 4, 7)])), comment = FALSE)
```

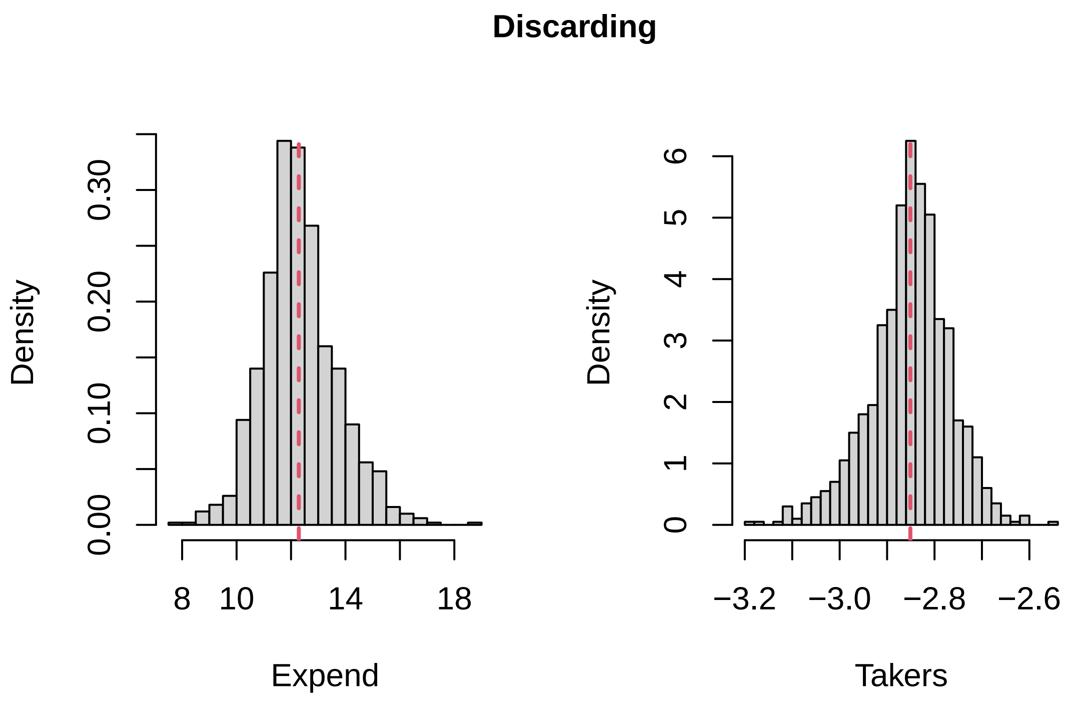
	expend	takers	total
expend	1.00	0.59	-0.38
takers	0.59	1.00	-0.89
total	-0.38	-0.89	1.00

First, we introduce missing values completely at random with probability 0.1 into the expend variable. We repeat this experiment 1000 times, handle the missing values in 4 different ways and calculate the resulting regression coefficients.

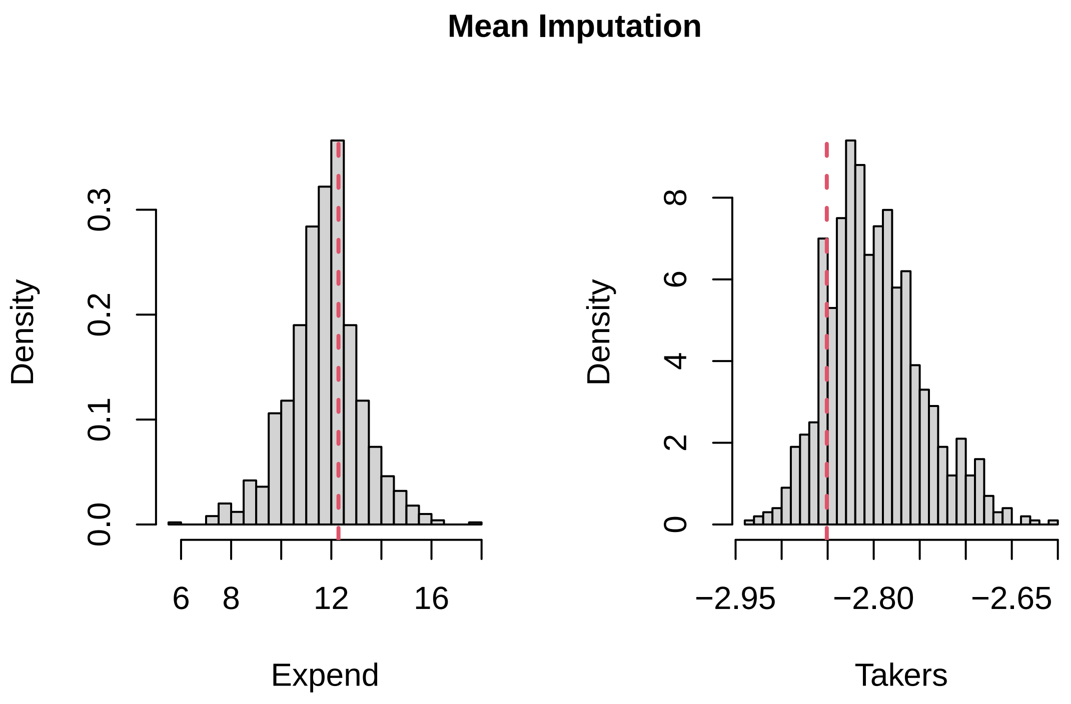
```r
nsim = 1000
betasim = matrix(0, nsim, 8)
for (i in 1:nsim) {
    Xmiss = sat$expend
    Xmiss[rbinom(n, 1, 0.1) == 1] = NA
    miss = lm(total ~ Xmiss + takers, sat)
    betasim[i, 1:2] = miss$coefficients[-1]
    Ximpute = Xmiss
    Ximpute[is.na(Xmiss)] = mean(Xmiss, na.rm = TRUE)
    miss = lm(total ~ Ximpute + takers, sat)
    betasim[i, 3:4] = miss$coefficients[-1]
    impute = lm(Xmiss ~ takers, sat)
    Ximpute[is.na(Xmiss)] = predict(impute, data.frame(takers = sat$takers[is.na(Xmiss)]))
    miss = lm(total ~ Ximpute + takers, sat)
    betasim[i, 5:6] = miss$coefficients[-1]
    Ximpute[is.na(Xmiss)] = cbind(1, sat$takers[is.na(Xmiss)]) %*% impute$coefficients +
        rnorm(sum(is.na(Xmiss)), sd = summary(impute)$sigma)
    miss = lm(total ~ Ximpute + takers, sat)
    betasim[i, 7:8] = miss$coefficients[-1]
}
```

We compare the histograms of the estimated regression coefficients after introduction of the missing values against the initially estimated regression coefficients without missing values. When a fairly small fraction of values are missing completely at random, we can see that any method of dealing with these missing values yields satisfactory results. The histograms of estimated regression coefficients display low variation, and the initially estimated regression coefficients lie close to the modes of the histograms.

```r
par(mfrow = c(1, 2))
hist(betasim[, 1], "FD", freq = FALSE, main = NA, xlab = "Expend")
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 2], "FD", freq = FALSE, main = NA, xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Discarding", line = -2, outer = TRUE, font = 2)
```

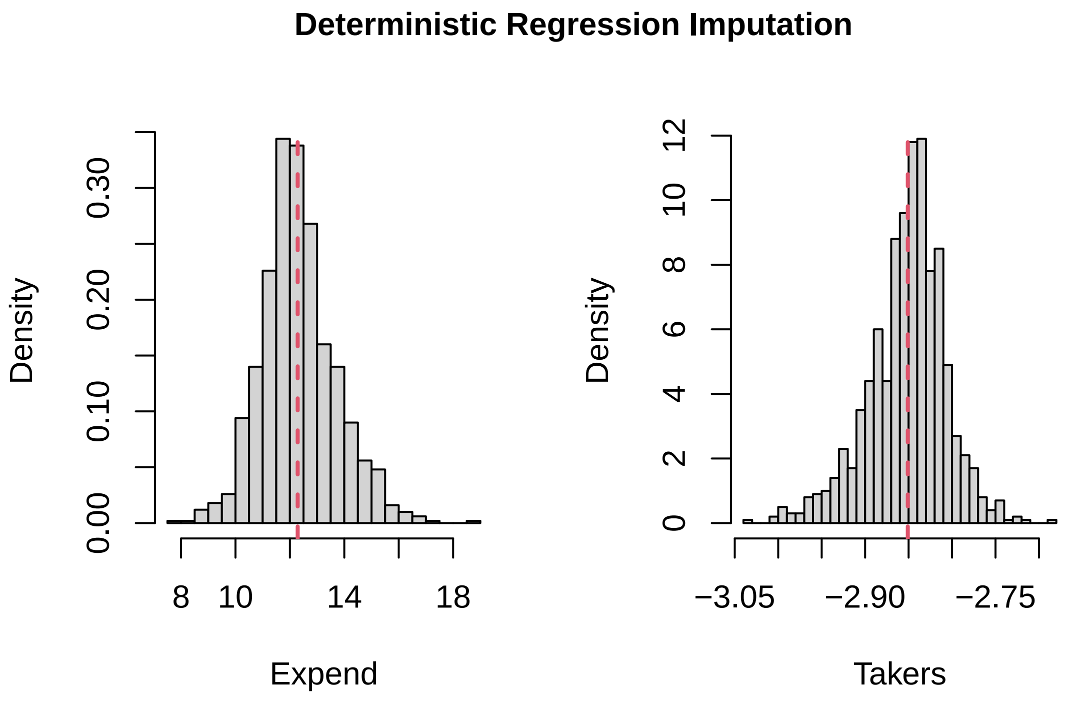


```
par(mfrow = c(1, 2))
hist(betasim[, 3], "FD", freq = FALSE, main = NA, xlab = "Expend")
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 4], "FD", freq = FALSE, main = NA, xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Mean Imputation", line = -2, outer = TRUE, font = 2)
```

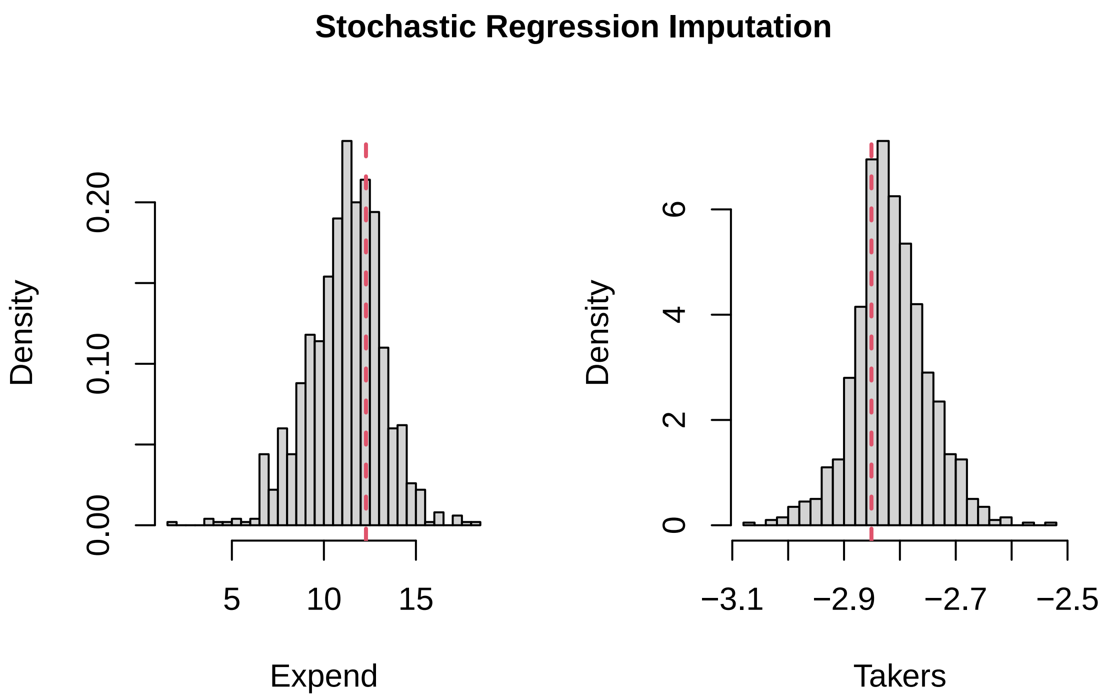


```
par(mfrow = c(1, 2))
hist(betasim[, 5], "FD", freq = FALSE, main = NA, xlab = "Expend")
```

```
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 6], "FD", freq = FALSE, main = NA, xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Deterministic Regression Imputation", line = -2, outer = TRUE, font = 2)
```



```
par(mfrow = c(1, 2))
hist(betasim[, 7], "FD", freq = FALSE, main = NA, xlab = "Expend")
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 8], "FD", freq = FALSE, main = NA, xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Stochastic Regression Imputation", line = -2, outer = TRUE, font = 2)
```

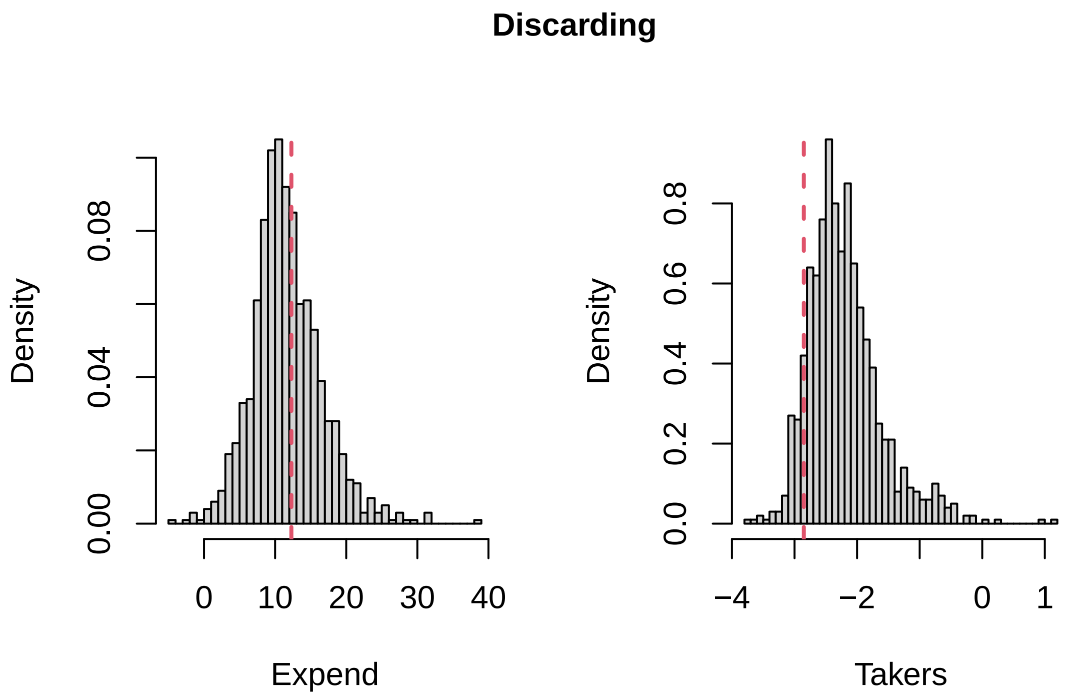


Then, we assume that values are missing in the expend variable as a function of the percentage of students taking the SAT in each US state. More specifically, the probability that the daily public school expenditure of state i is missing is equal to the percentage of students not taking the SAT in state i. Thus, states with a low percentage of students taking the SAT have a high chance of not reporting their public school expenditure. We repeat this experiment 1000 times, handle the missing values in 4 different ways and calculate the resulting regression coefficients.

```
for (i in 1:nsim) {
    Xmiss = sat$expend
    Xmiss[rbinom(n, 1, 1 - sat$takers/100) == 1] = NA
    miss = lm(total ~ Xmiss + takers, sat)
    betasim[i, 1:2] = miss$coefficients[-1]
    Ximpute = Xmiss
    Ximpute[is.na(Xmiss)] = mean(Xmiss, na.rm = TRUE)
    miss = lm(total ~ Ximpute + takers, sat)
    betasim[i, 3:4] = miss$coefficients[-1]
    impute = lm(Xmiss ~ takers, sat)
    Ximpute[is.na(Xmiss)] = predict(impute, data.frame(takers = sat$takers[is.na(Xmiss)]))
    miss = lm(total ~ Ximpute + takers, sat)
    betasim[i, 5:6] = miss$coefficients[-1]
    Ximpute[is.na(Xmiss)] = cbind(1, sat$takers[is.na(Xmiss)]) %*% impute$coefficients +
        rnorm(sum(is.na(Xmiss)), sd = summary(impute)$sigma)
    miss = lm(total ~ Ximpute + takers, sat)
    betasim[i, 7:8] = miss$coefficients[-1]
}
```

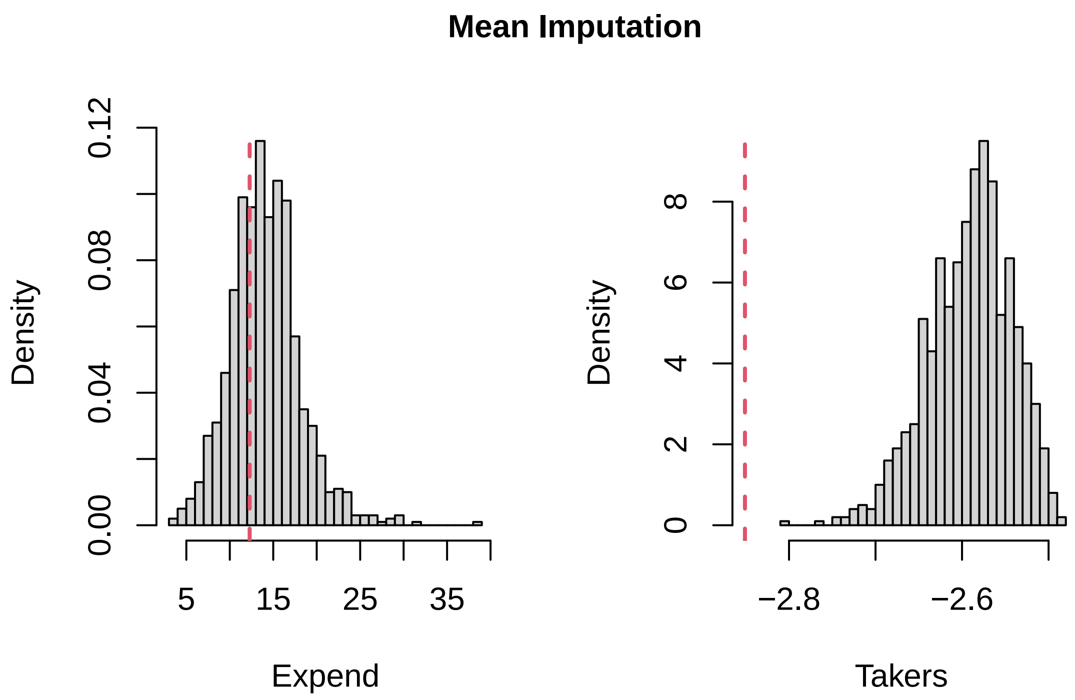
We compare the histograms of the estimated regression coefficients after introduction of the missing values against the initially estimated regression coefficients without missing values. When values are missing at random, the method of discarding data entries with at least one missing value leads to histograms with extremely high variation, which indicates a general inability to recover the true effects of the predictors on the response variable with just the fraction of available data. This is not surprising since the estimation of the regression coefficients is essentially based only on the data from states with a high percentage of students taking the SAT, while mostly ignoring the rest.

```r
par(mfrow = c(1, 2))
hist(betasim[, 1], "FD", freq = FALSE, main = NA, xlab = "Expend")
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 2], "FD", freq = FALSE, main = NA, xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Discarding", line = -2, outer = TRUE, font = 2)
```



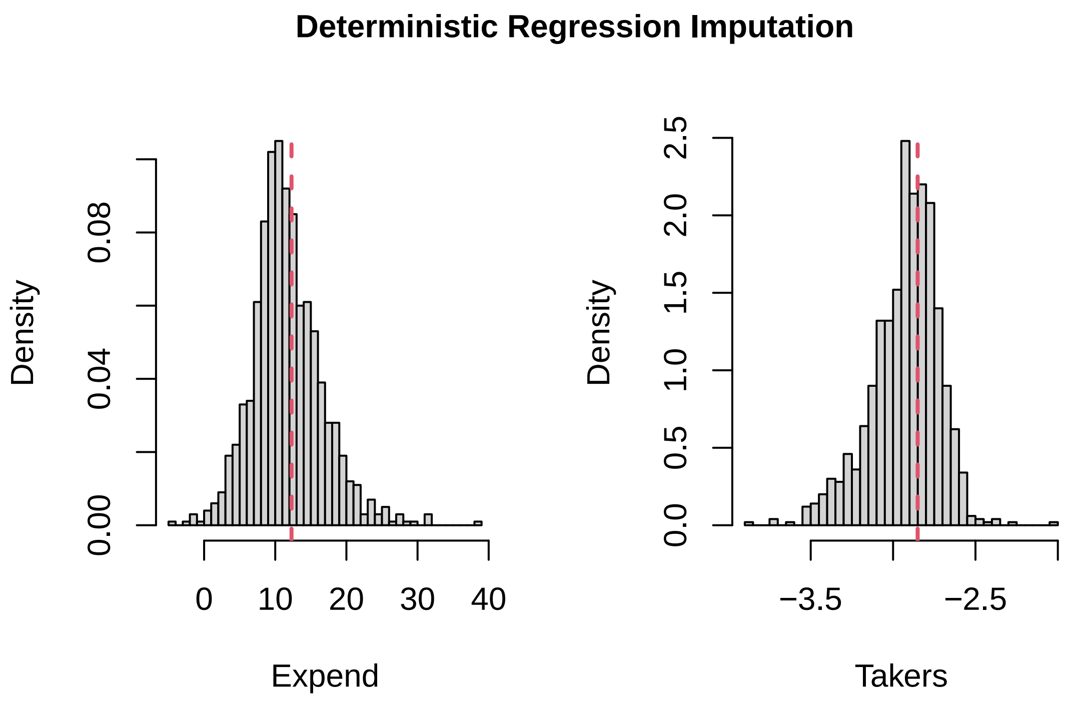
Mean imputation significantly dilutes the strong linear relationship between the 2 predictors. This leads to a falsely confident estimation of the takers coefficient, while the original coefficient lies quite far off the range of estimated coefficients.

```r
par(mfrow = c(1, 2))
hist(betasim[, 3], "FD", freq = FALSE, main = NA, xlab = "Expend")
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 4], "FD", freq = FALSE, main = NA, xlim = c(betahat[2], max(betasim[,
    4])), xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Mean Imputation", line = -2, outer = TRUE, font = 2)
```

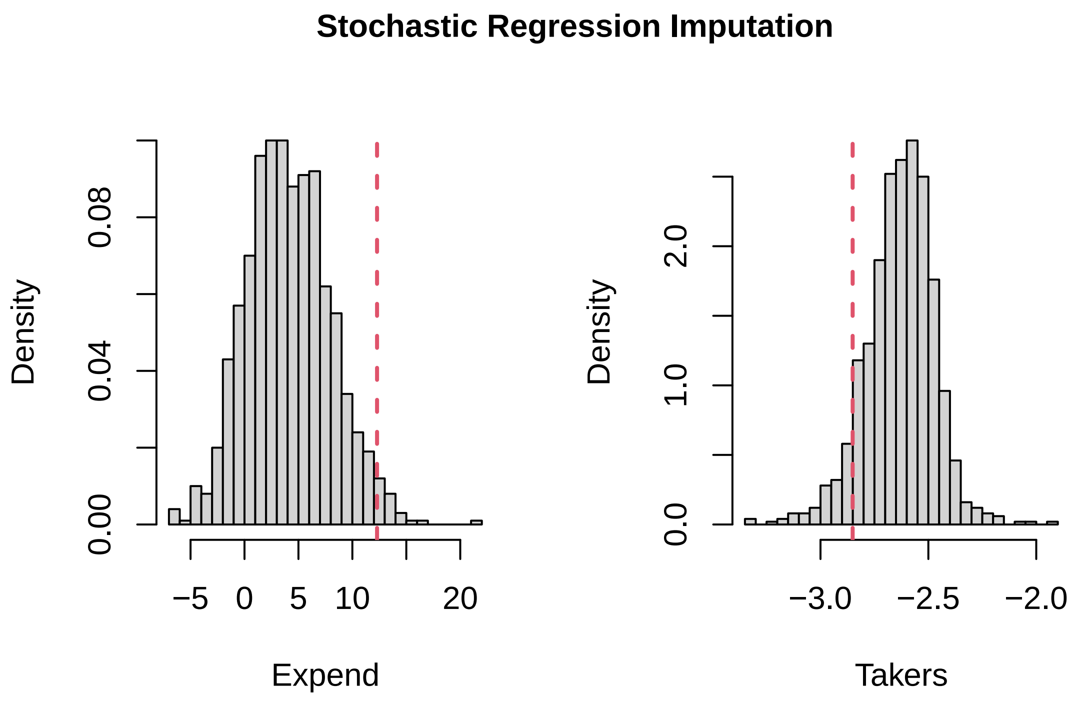


Regression imputation generally does a good job of estimating the original regression coefficients. Deterministic regression imputation leads to more precise estimates, but significantly increases the collinearity between the 2 predictors. Stochastic regression imputation avoids this problems, but leads to significantly more noisy estimates of the regression coefficients.

```r
par(mfrow = c(1, 2))
hist(betasim[, 5], "FD", freq = FALSE, main = NA, xlab = "Expend")
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 6], "FD", freq = FALSE, main = NA, xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Deterministic Regression Imputation", line = -2, outer = TRUE, font = 2)
```



```r
par(mfrow = c(1, 2))
hist(betasim[, 7], "FD", freq = FALSE, main = NA, xlab = "Expend")
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 8], "FD", freq = FALSE, main = NA, xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Stochastic Regression Imputation", line = -2, outer = TRUE, font = 2)
```

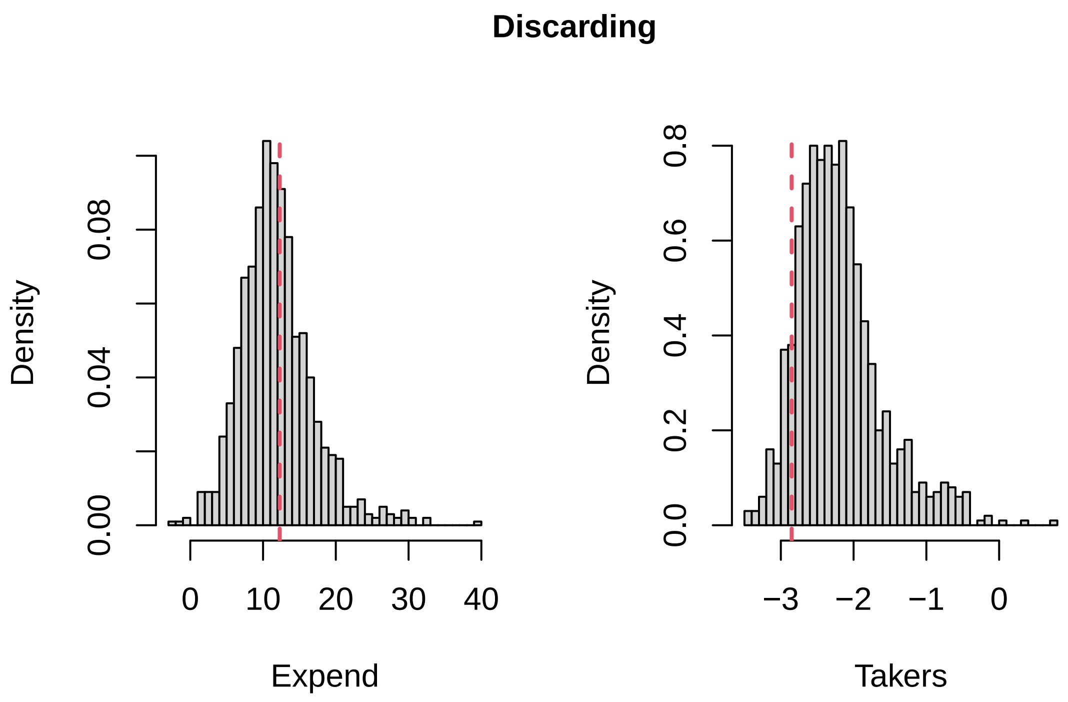


Lastly, we assume that values are missing in the takers variable as a function of itself. More specifically, the probability that the percentage of students taking the SAT in state i is missing is equal to the percentage of students not taking the SAT in state i. Thus, states with a low percentage of students taking the SAT have a high chance of not reporting it. We repeat this experiment 1000 times, handle the missing values in 4 different ways and calculate the resulting regression coefficients.

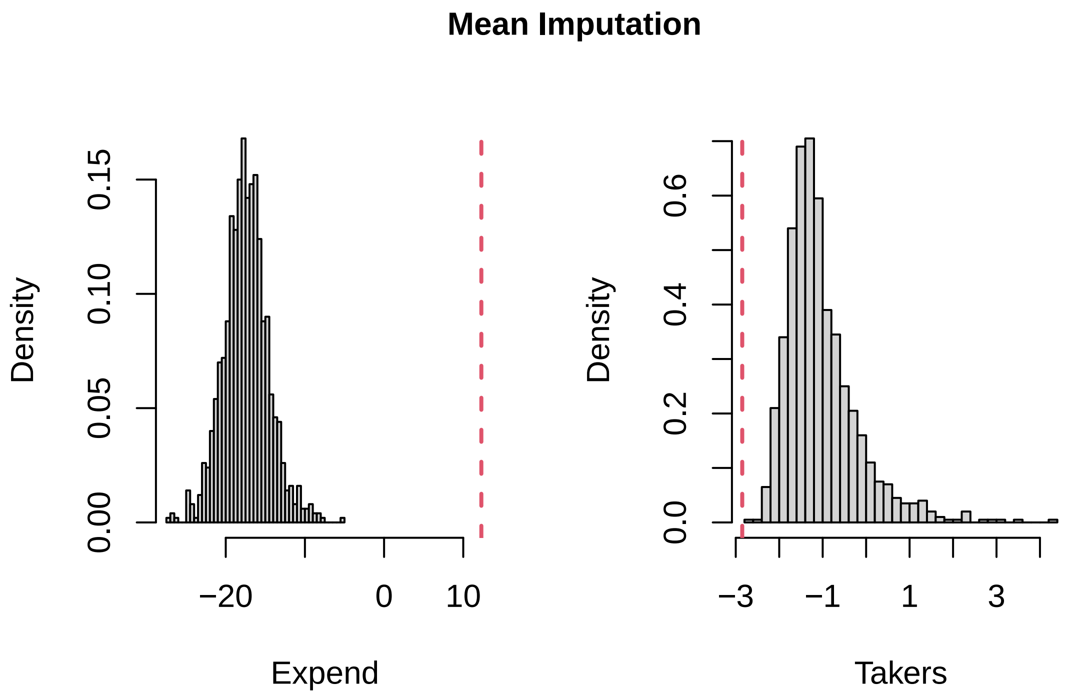
```r
for (i in 1:nsim) {
    Xmiss = sat$takers
    Xmiss[rbinom(n, 1, 1 - sat$takers/100) == 1] = NA
    miss = lm(total ~ expend + Xmiss, sat)
    betasim[i, 1:2] = miss$coefficients[-1]
    Ximpute = Xmiss
    Ximpute[is.na(Xmiss)] = mean(Xmiss, na.rm = TRUE)
    miss = lm(total ~ expend + Ximpute, sat)
    betasim[i, 3:4] = miss$coefficients[-1]
    impute = lm(Xmiss ~ expend, sat)
    Ximpute[is.na(Xmiss)] = predict(impute, data.frame(expend = sat$expend[is.na(Xmiss)]))
    miss = lm(total ~ expend + Ximpute, sat)
    betasim[i, 5:6] = miss$coefficients[-1]
    Ximpute[is.na(Xmiss)] = cbind(1, sat$expend[is.na(Xmiss)]) %*% impute$coefficients +
        rnorm(sum(is.na(Xmiss)), sd = summary(impute)$sigma)
    miss = lm(total ~ expend + Ximpute, sat)
    betasim[i, 7:8] = miss$coefficients[-1]
}
```

We compare the histograms of the estimated regression coefficients after introduction of the missing values against the initially estimated regression coefficients without missing values. When values are missing not at random, we can see that any method of dealing with these missing values fails to provide satisfactory results. The histograms of estimated regression coefficients display very high variation, and the initially estimated regression coefficients generally lie far off the modes of the histograms.

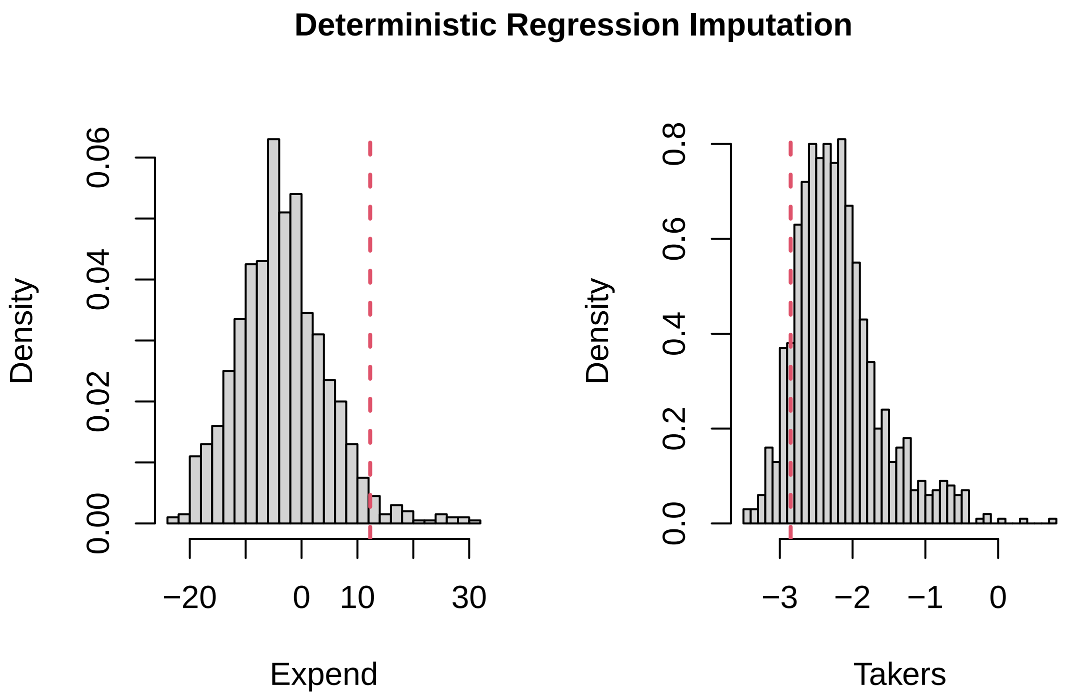
```r
par(mfrow = c(1, 2))
hist(betasim[, 1], "FD", freq = FALSE, main = NA, xlab = "Expend")
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 2], "FD", freq = FALSE, main = NA, xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Discarding", line = -2, outer = TRUE, font = 2)
```



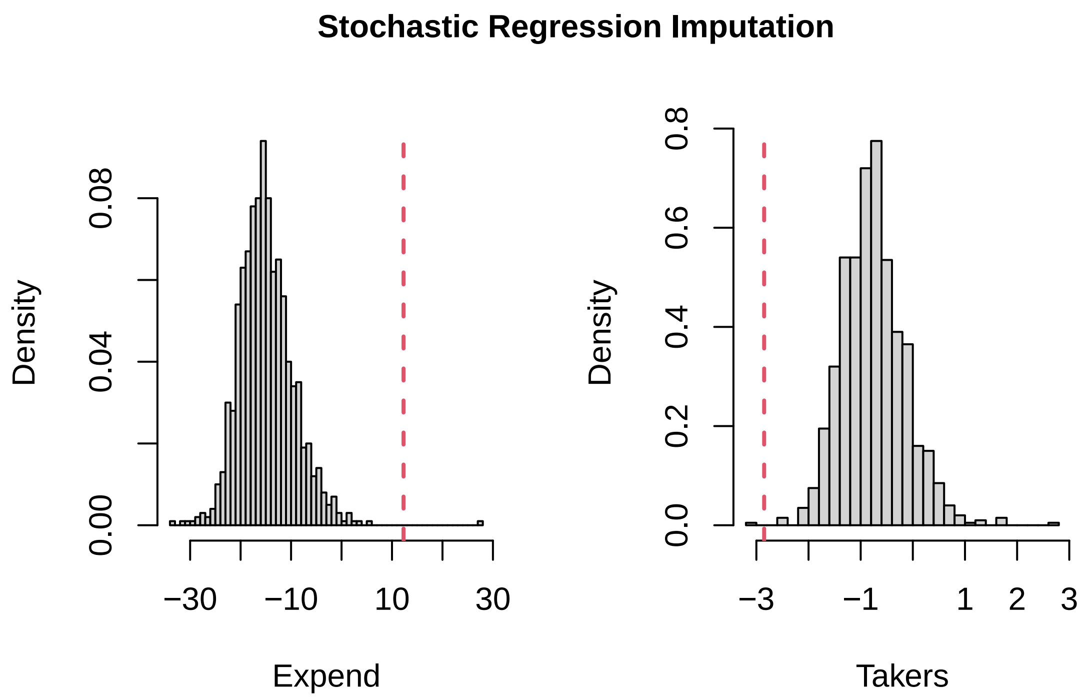
```
par(mfrow = c(1, 2))
hist(betasim[, 3], "FD", freq = FALSE, main = NA, xlim = c(min(betasim[, 3]),
    betahat[1]), xlab = "Expend")
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 4], "FD", freq = FALSE, main = NA, xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Mean Imputation", line = -2, outer = TRUE, font = 2)
```



```
par(mfrow = c(1, 2))
hist(betasim[, 5], "FD", freq = FALSE, main = NA, xlab = "Expend")
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 6], "FD", freq = FALSE, main = NA, xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Deterministic Regression Imputation", line = -2, outer = TRUE, font = 2)
```



```
par(mfrow = c(1, 2))
hist(betasim[, 7], "FD", freq = FALSE, main = NA, xlab = "Expend")
abline(v = betahat[1], col = 2, lty = 2, lwd = 2)
hist(betasim[, 8], "FD", freq = FALSE, main = NA, xlab = "Takers")
abline(v = betahat[2], col = 2, lty = 2, lwd = 2)
mtext("Stochastic Regression Imputation", line = -2, outer = TRUE, font = 2)
```



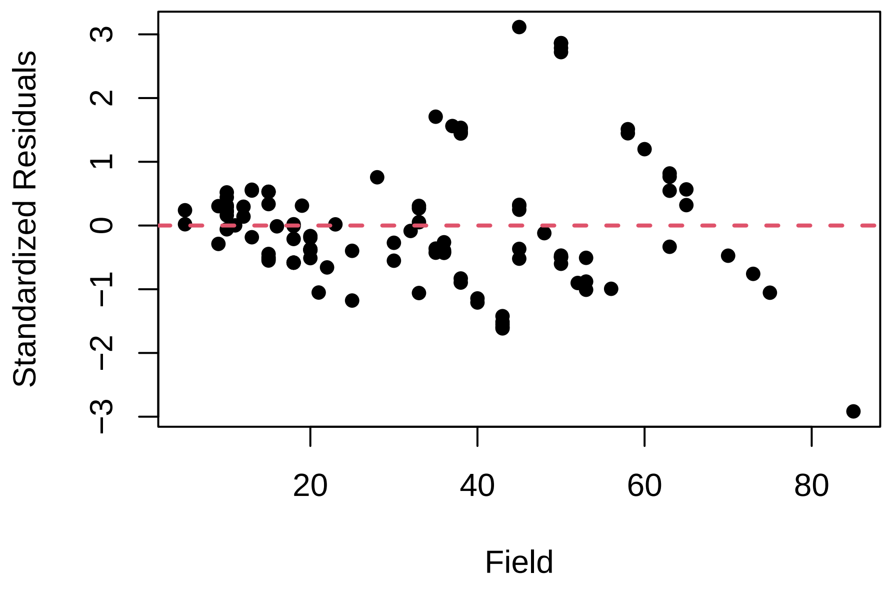
2 Heteroscedasticity and Autocorrelation

Heteroscedasticity Tests

Inference on linear models heavily relies on the assumption of homoscedasticity of the error terms. Hence, it is important to be able to test for departures from this assumption when it is in doubt. One possible form of heteroscedasticity appears when the variance of the error terms is a function of some of the available predictors. There are a number of different tests which can be used to check for this form of heteroscedasticity.

An example of this form of heteroscedasticity appears in the pipeline data set from the faraway package. We can clearly see that the variation in the standardized residuals increases in a cone shape as a function of the field predictor.

```
library(faraway)
n = dim(pipeline)[1]
p = 1
fit = lm(Lab ~ Field, pipeline)
residuals = fit$residuals
plot(rstandard(fit) ~ Field, pipeline, ylab = "Standardized Residuals", pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



Algorithm 2.1 Breusch-Pagan Test

```
    Input: Random sample $(Y, X)$.
1:  We regress $Y$ on $X \in \mathbb{R}^{n\times p}$, not including the intercept, and calculate the residual vector $\widehat{\varepsilon}$.
2:  We define the following auxiliary variable:
        $Y_i^{\text{aux}} = \frac{n}{\|\widehat{\varepsilon}\|_2^2}\widehat{\varepsilon}_i^2.$
3:  We regress $Y^{\text{aux}}$ on $X \in \mathbb{R}^{n\times p}$, not including the intercept, and calculate the sum of squares due to regression:
        $\text{SSR}_{\text{aux}} = \left\|\widehat{Y}^{\text{aux}} - 1\right\|^2.$
    Note that the sample average of $Y^{\text{aux}}$ is equal to 1 by construction.
4:  We calculate the observed value $\text{BP}_0$ of the Breusch-Pagan test statistic BP as follows:
        $\text{BP}_0 = \frac{1}{2}\text{SSR}_{\text{aux}}.$
    We know that $\text{BP} \xrightarrow{d} \chi_p^2$ under the null hypothesis of homoscedasticity.
5:  We calculate the p-value of the test as $P(\text{BP} \geqslant \text{BP}_0)$.
    Output: Observed test statistic $\text{BP}_0$ and p-value.
```

The observed value of the test statistic is 29.59 and the corresponding p-value is $5 \cdot 10^{-8}$, which implies the rejection of the null hypothesis of homoscedasticity against the alternative that the variance of the error terms is a linear function of the field predictor, as expected. We verify our calculations by using the bptest function from the lmtest package with the argument "studentize = FALSE".

```r
library(lmtest)
Yaux = n * residuals^2/sum(residuals^2)
aux = lm(Yaux ~ Field, pipeline)
BP = sum((aux$fitted.values - 1)^2)/2
print(BP)
```

```
## [1] 29.58568
```

```
pchisq(BP, p, lower.tail = FALSE)
```

```
## [1] 5.349868e-08
```

```
bptest(fit, studentize = FALSE)
```

```
##
##  Breusch-Pagan test
##
## data:  fit
## BP = 29.586, df = 1, p-value = 5.35e-08
```

Algorithm 2.2 Studentized Breusch-Pagan Test

```
    Input: Random sample $(Y, X)$.
1:  We regress $Y$ on $X \in \mathbb{R}^{n \times p}$, not including the intercept, and calculate the residual vector $\widehat{\varepsilon}$.
2:  We define the auxiliary variable $Y_i^{\text{aux}} = \widehat{\varepsilon}_i^2$.
3:  We regress $Y^{\text{aux}}$ on $X \in \mathbb{R}^{n \times p}$, not including the intercept, and calculate the coefficient of determination:
        $$R^2_{\text{aux}} = \frac{\text{SSR}_{\text{aux}}}{\text{SST}_{\text{aux}}}.$$
4:  We calculate the observed value $\text{SBP}_0$ of the studentized Breusch-Pagan test statistic SBP as $\text{SBP}_0 = nR^2_{\text{aux}}$.
    We know that $\text{SBP} \xrightarrow{d} \chi^2_p$ under the null hypothesis of homoscedasticity.
5:  We calculate the p-value of the test as $P(\text{SBP} \geqslant \text{SBP}_0)$.
    Output: Observed test statistic $\text{SBP}_0$ and p-value.
```

The studentized version of the Breusch-Pagan test was proposed by Roger Koenker as a more robust alternative to the original test in the case of departures from the normality assumption of the error terms. While the studentized version of the test has closer to nominal asymptotic type I error, its power is much lower under departures from normality.

The observed value of the test statistic is 16.05 and the corresponding p-value is $6 \cdot 10^{-5}$, which implies the rejection of the null hypothesis of homoscedasticity against the alternative that the variance of the error terms is a linear function of the field predictor, as expected. We verify our calculations by again using the bptest function from the lmtest package.

```
Yaux = residuals^2
aux = lm(Yaux ~ Field, pipeline)
BP = n * summary(aux)$r.squared
print(BP)
```

```
## [1] 16.04506
```

```
pchisq(BP, p, lower.tail = FALSE)
```

```
## [1] 6.185266e-05
```

```
bptest(fit)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  fit
## BP = 16.045, df = 1, p-value = 6.185e-05
```

Algorithm 2.3 White Test

```
**Input**: Random sample $(Y, X)$.
1: We regress $Y$ on $X \in \mathbb{R}^{n \times p}$, not including the intercept, and calculate the residual vector $\widehat{\varepsilon}$.
2: We define the auxiliary variable $Y_i^{\text{aux}} = \widehat{\varepsilon}_i^2$.
3: We regress $Y^{\text{aux}}$ on the predictors $X_1, X_2, \ldots, X_p$, the squared predictors $X_1^2, X_2^2, \ldots, X_p^2$ and the pairwise products of the predictors $X_1X_2, X_1X_3, \ldots, X_{p-1}X_p$. We calculate the coefficient of determination of this auxiliary regression:
       $R^2_{\text{aux}} = \frac{\text{SSR}_{\text{aux}}}{\text{SST}_{\text{aux}}}.$
4: We calculate the observed value $w$ of the White test statistic $W$ as $w = nR^2_{\text{aux}}$. Under the null hypothesis of homoscedasticity, we know that:
       $W \xrightarrow{d} \chi^2_{\frac{1}{2}p(p+3)}.$
5: We calculate the p-value of the test as $P(W \geqslant w)$.
**Output**: Observed test statistic $w$ and p-value.
```

The observed value of the test statistic is 16.22 and the corresponding p-value is $3 \cdot 10^{-4}$, which implies the rejection of the null hypothesis of homoscedasticity against the alternative that the variance of the error terms is a quadratic function of the field predictor, as expected. We verify our calculations by again using the bptest function from the lmtest package and specifying a second degree polynomial of field as the variance formula.

```
aux = lm(Yaux ~ poly(Field, 2), pipeline)
BP = n * summary(aux)$r.squared
print(BP)
```

```
## [1] 16.22166
```

```
pchisq(BP, p * (p + 3)/2, lower.tail = FALSE)
```

```
## [1] 0.0003002696
```

```
bptest(fit, ~poly(Field, 2), data = pipeline)
```

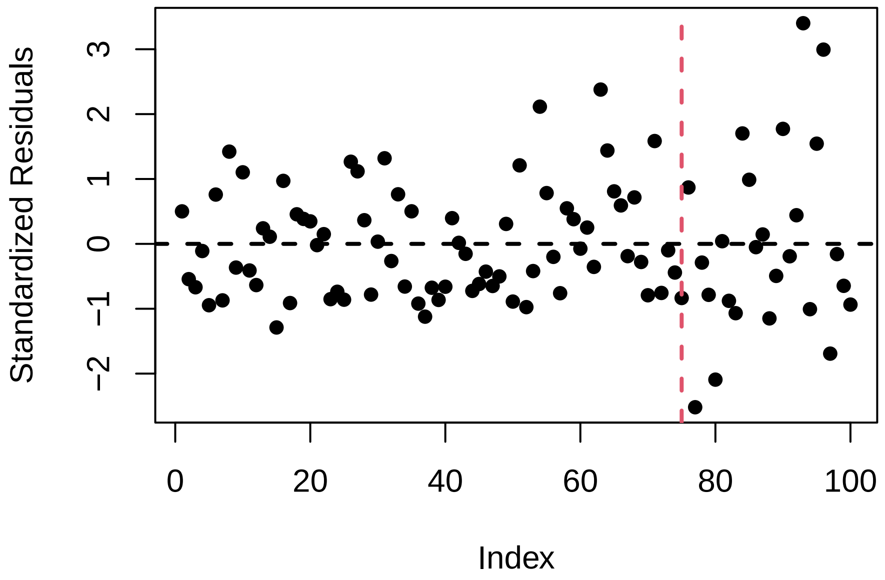
```
##
##  studentized Breusch-Pagan test
##
## data:  fit
## BP = 16.222, df = 2, p-value = 0.0003003
```

Another possible form of heteroscedasticity appears when the variance of the error terms differs between 2 groups of observations defined by some binary predictor. First, we simulate a normally distributed predictor X of size $n = 100$. Then, we simulate a binary predictor Z whose first $n^* = 75$ values are equal to 0 and the last $n - n^*$ are equal to 1. The effect of both predictors on the response variable is equal to 2. Lastly, we simulate a sample Y of size $n = 100$ from this linear model with:

$$\sigma_i^2 = \begin{cases} 1, & i = 1, 2, \dots, n^* \\ 4, & i = n^* + 1, n^* + 2, \dots, n \end{cases}.$$

We can clearly see that the variation in the residuals of the fitted linear model is slightly higher for the last $n - n^*$ observations.

```r
n = 100
nstar = 75
p = 1
X = rnorm(n)
Z = c(numeric(nstar), rep(1, n - nstar))
sigma = c(1, 2)
Y = 2 * X + 2 * Z + rnorm(n, sd = sigma[Z + 1])
fit = lm(Y ~ X + Z)
residuals = fit$residuals
plot(rstandard(fit), ylab = "Standardized Residuals", pch = 16)
abline(h = 0, lty = 2, lwd = 2)
abline(v = nstar, col = 2, lty = 2, lwd = 2)
```



The observed value of the GQ$^{(12)}$ test statistic is 0.31, while the observed value of the reciprocal GQ$^{(21)}$ test statistic is 3.18. The corresponding p-value of both equivalent test statistics is $2 \cdot 10^{-4}$, which implies the rejection of the null hypothesis of homoscedasticity against the alternative that the variance of the error terms is unequal between the 2 groups, as expected. We verify our calculations by using the gqtest function from the lmtest package with the argument alternative = "two.sided".

Algorithm 2.4 Goldfeld-Quandt Test

```
Input: Random sample $(Y, X)$ and change point $n^*$.
1: We regress $Y$ on $X \in \mathbb{R}^{n\times p}$, not including an intercept, separately for the first $n^*$ and the last $n - n^*$ observations.
2: We calculate the estimated residual variances $S_1^2$ and $S_2^2$ of the 2 linear models.
3: We calculate the observed value $\mathrm{GQ}_0^{(12)}$ of the Goldfeld-Quandt test statistic $\mathrm{GQ}^{(12)}$ as:
       $\mathrm{GQ}_0^{(12)} = \frac{S_1^2}{S_2^2}.$
   Under the null hypothesis of homoscedasticity, we know that $\mathrm{GQ}^{(12)} \sim F_{n^*-p-1,n-n^*-p-1}$. Equivalently, under the null hypothesis, note that:
       $\mathrm{GQ}^{(21)} = \frac{S_2^2}{S_1^2} \sim F_{n-n^*-p-1,n^*-p-1}.$
4: We calculate the p-value of the two-sided test as:
       $2 \cdot \min\left\{P\left(\mathrm{GQ}^{(12)} \leqslant \mathrm{GQ}_0^{(12)}\right), P\left(\mathrm{GQ}^{(12)} \geqslant \mathrm{GQ}_0^{(12)}\right)\right\}.$
Output: Observed test statistic $\mathrm{GQ}_0^{(12)}$ and p-value.
```

```r
aux1 = lm(Y ~ X, subset = Z == 0)
aux2 = lm(Y ~ X, subset = Z == 1)
GQ = summary(aux1)$sigma^2/summary(aux2)$sigma^2
print(GQ)
```

```
## [1] 0.3149394
```

```r
2 * min(pf(GQ, nstar - p - 1, n - nstar - p - 1), pf(GQ, nstar - p - 1, n -
    nstar - p - 1, lower.tail = FALSE))
```

```
## [1] 0.0001862302
```

```r
print(GQ^(-1))
```

```
## [1] 3.175214
```

```r
2 * min(pf(GQ^(-1), n - nstar - p - 1, nstar - p - 1), pf(GQ^(-1), n - nstar -
    p - 1, nstar - p - 1, lower.tail = FALSE))
```

```
## [1] 0.0001862302
```

```r
gqtest(Y ~ X, nstar, alternative = "two.sided")
```

```
##
##  Goldfeld-Quandt test
##
## data:  Y ~ X
## GQ = 3.1752, df1 = 23, df2 = 73, p-value = 0.0001862
## alternative hypothesis: variance changes from segment 1 to 2
```

Algorithm 2.5 Generalized F Test of Equality of Variances

```
Input: Random sample $(Y, X, Z)$.
1: We regress $Y$ on $X \in \mathbb{R}^{n\times p}$ and $Z \in \mathbb{R}^n$, not including an intercept, and calculate the residual vector $\widehat{\varepsilon}$.
2: Let $n^* = \sum_{i=1}^n \mathbb{1}_{\{z_i=0\}}$. We calculate the following residual variances:
   $$\widetilde{\sigma}_1^2 = \frac{1}{n^* \frac{n-p}{n} - 1} \sum_{i=1}^n \widehat{\varepsilon}_i^2 \mathbb{1}_{\{z_i=0\}}, \quad \widetilde{\sigma}_2^2 = \frac{1}{(n-n^*)\frac{n-p}{n} - 1} \sum_{i=1}^n \widehat{\varepsilon}_i^2 \mathbb{1}_{\{z_i=1\}}$$
3: We calculate the observed value $f^{(12)}$ of the generalized $F$ test statistic $F^{(12)}$ as:
   $$f^{(12)} = \frac{\widetilde{\sigma}_1^2}{\widetilde{\sigma}_2^2}.$$
   Under the null hypothesis of homoscedasticity, we know that:
   $$F^{(12)} \sim F_{n^* \frac{n-p}{n} - 1, (n-n^*)\frac{n-p}{n} - 1}.$$
4: We calculate the p-value of the two-sided test as:
   $$2 \cdot \min\left\{P\left(F^{(12)} \leqslant f^{(12)}\right), P\left(F^{(12)} \geqslant f^{(12)}\right)\right\}.$$
Output: Observed test statistic $f^{(12)}$ and p-value.
```

The observed value of the test statistic is 0.32 and the corresponding p-value is $2 \cdot 10^{-4}$, which implies the rejection of the null hypothesis of homoscedasticity against the alternative that the variance of the error terms is unequal between the 2 groups, as expected. We observe that the results of the 2 tests closely agree with each other.

```
df1 = nstar * (n - p)/n - 1
df2 = (n - nstar) * (n - p)/n - 1
var1 = sum(residuals[Z == 0]^2)/df1
var2 = sum(residuals[Z == 1]^2)/df2
FT = var1/var2
print(FT)
```

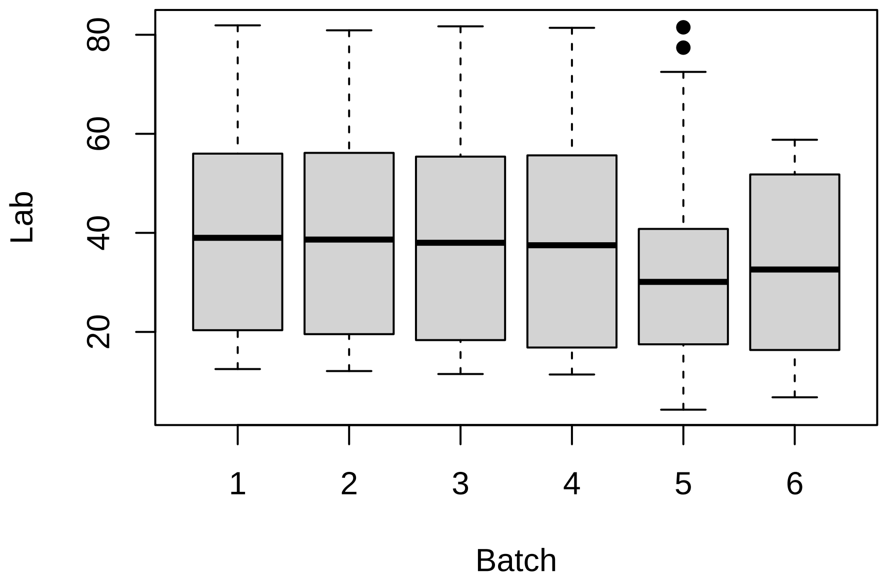
```
## [1] 0.321077
```

```
2 * min(pf(FT, df1, df2), pf(FT, df1, df2, lower.tail = FALSE))
```

```
## [1] 0.000209213
```

Another possible form of heteroscedasticity appears when the variance of the error terms differs across multiple groups defined by some categorical predictor. We can see that the variation in the lab variable appears to be fairly constant across the 6 different batches in the pipeline data set.

```
n = dim(pipeline)[1]
Y = pipeline$Lab
X = pipeline$Batch
fit = lm(Lab ~ Batch, pipeline)
boxplot(Lab ~ Batch, pipeline, pch = 16)
```



Algorithm 2.6 Levene's Test

```
    **Input**: Random sample $(Y, X)$, where $X \in \mathbb{R}^n$ is a categorical predictor with $k$ levels.
1: We regress $Y$ on $X$ and calculate the vector of fitted values $\widehat{Y}$.
2: We define the auxiliary variable $Y_i^{\text{aux}} = \left|Y_i - \widehat{Y}_i\right|$.
3: We regress $Y^{\text{aux}}$ on $X$ and perform an overall $F$ test of significance. Levene's test statistic coincides with the overall $F$ test statistic.
    **Output**: Observed test statistic $f$ and p-value.
```

The observed value of the test statistic is 0.82 and the corresponding p-value is 0.54, which implies a failure to reject the null hypothesis of homoscedasticity, as expected. We verify our calculations by using the leveneTest function from the car package with the argument "mean".

```
library(car)
Yaux = abs(Y - fit$fitted.values)
aux = lm(Yaux ~ Batch, pipeline)
anova(aux)
```

```
## Analysis of Variance Table
##
## Response: Yaux
##            Df  Sum Sq Mean Sq F value Pr(>F)
## Batch       5   253.9  50.775  0.3136 0.9038
## Residuals 101 16353.4 161.915
```

```r
leveneTest(fit, "mean")
```

```
## Levene's Test for Homogeneity of Variance (center = "mean")
##        Df F value Pr(>F)
## group   5  0.3136 0.9038
##       101
```

Algorithm 2.7 Brown-Forsythe Test

```
    Input: Random sample $(Y, X)$, where $X \in \mathbb{R}^n$ is a categorical predictor with levels $1, 2, \ldots, k$.
1:  We calculate the median $Y$ value $\text{med}_h(Y)$ within level $h = 1, 2, \ldots, k$.
2:  We define the auxiliary variable $Y_i^{\text{aux}} = |Y_i - \text{med}_{X_i}(Y)|$.
3:  We regress $Y^{\text{aux}}$ on $X$ and perform an overall $F$ test of significance. The Brown-Forsythe test statistic coincides with the overall $F$ test statistic.
    Output: Observed test statistic $f$ and p-value.
```

The Brown-Forsythe test is much more robust than Levene's test to departures from the normality assumption of the error terms, while maintaining high power. However, Levene's test boasts higher power if there exists no noticeable departure from normality.

The observed value of the test statistic is 0.84 and the corresponding p-value is 0.52, which implies a failure to reject the null hypothesis of homoscedasticity, as expected. We verify our calculations by using the leveneTest function from the car package.

```r
Yaux = abs(Y - aggregate(Lab ~ Batch, pipeline, median)[pipeline$Batch, 2])
aux = lm(Yaux ~ Batch, pipeline)
anova(aux)
```

```
## Analysis of Variance Table
##
## Response: Yaux
##            Df  Sum Sq Mean Sq F value Pr(>F)
## Batch       5   235.7  47.132  0.2471 0.9404
## Residuals 101 19263.2 190.725
```

```r
leveneTest(fit)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##        Df F value Pr(>F)
## group   5  0.2471 0.9404
##       101
```

It should be noted that Bartlett's test is generally more sensitive than Levene's test to departures from normality. The observed value of the test statistic is 0.63 and the corresponding p-value is 0.99, which implies a failure to reject the null hypothesis of homoscedasticity, as expected. We verify our calculations by using R's built-in bartlett.test function.

Algorithm 2.8 Bartlett's Test

```
    **Input**: Random sample $(Y, X)$, where $X \in \mathbb{R}^n$ is a categorical predictor with levels $1, 2, \ldots, k$.
1:  Let $n_h = \sum_{i=1}^n \mathbb{1}_{\{x_i = h\}}$ and $S_h^2$ be the sample variance of $Y$ within level $h = 1, 2, \ldots, k$.
2:  We define the pooled estimate of the variance of $Y$ as follows:
        $S_p^2 = \frac{1}{n-k} \sum_{h=1}^k (n_h - 1) S_h^2.$
3:  We calculate the observed value $\text{BT}_0$ of Bartlett's test statistic BT as:
        $\text{BT}_0 = \frac{(n-k)\log S_p^2 - \sum_{h=1}^k (n_h - 1)\log S_h^2}{1 + \left(\sum_{h=1}^k \frac{1}{n_h - 1} - \frac{1}{n-k}\right)\frac{1}{3(k-1)}}.$
    Under the null hypothesis of homoscedasticity, we know that $\text{BT} \xrightarrow{d} \chi_{k-1}^2$.
4:  We calculate the p-value of the test as $P(\text{BT} \geqslant \text{BT}_0)$.
    **Output**: Observed test statistic $\text{BT}_0$ and p-value.
```

```r
k = length(unique(X))
ns = table(X)
S2 = aggregate(Lab ~ Batch, pipeline, var)[, 2]
Sp2 = sum((ns - 1) * S2)/(n - k)
BT = ((n - k) * log(Sp2) - sum((ns - 1) * log(S2)))/(1 + (sum(1/(ns - 1)) -
    1/(n - k))/(3 * (k - 1)))
print(BT)
```

```
## [1] 0.6312727
```

```r
pchisq(BT, k - 1, lower.tail = FALSE)
```

```
## [1] 0.9865267
```

```r
bartlett.test(Lab ~ Batch, pipeline)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  Lab by Batch
## Bartlett's K-squared = 0.63127, df = 5, p-value = 0.9865
```

White's Heteroscedasticity-Consistent Estimator

The formula $\text{Var}\left(\widehat{\beta}\right) = \sigma^2 \left(X^\text{T} X\right)^{-1}$ for the variance of the least squares estimator is no longer true under the presence of heteroscedasticity in the error terms. More precisely, under the assumption that $Y_i = X_i^\text{T}\beta + \varepsilon_i$ with $\varepsilon_i \sim \mathcal{N}\left(0, \sigma_i^2\right)$ and $\Sigma = \text{diag}\left\{\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2\right\}$, the formula for the variance of the ordinary least squares estimator

$\widehat{\beta} = \left(X^{\mathrm{T}}X\right)^{-1} X^{\mathrm{T}}Y$ takes the following form:

$$\operatorname{Var}\left(\widehat{\beta}\right) = \left(X^{\mathrm{T}}X\right)^{-1} X^{\mathrm{T}}\Sigma X \left(X^{\mathrm{T}}X\right)^{-1}.$$

White proposed to regress Y on X, calculate the residual vector $\widehat{\varepsilon}$ and estimate the covariance matrix Σ by $\widehat{\Sigma} = \operatorname{diag}\left\{\widehat{\varepsilon}_1^2, \widehat{\varepsilon}_2^2, \ldots, \widehat{\varepsilon}_n^2\right\}$. Then, White's heteroscedasticity-consistent estimator for the variance of the ordinary least squares estimator is given by:

$$\widehat{\operatorname{Var}}\left(\widehat{\beta}\right) = \left(X^{\mathrm{T}}X\right)^{-1} X^{\mathrm{T}}\widehat{\Sigma} X \left(X^{\mathrm{T}}X\right)^{-1} = \left(\sum_{i=1}^{n} X_i X_i^{\mathrm{T}}\right)^{-1} \sum_{i=1}^{n} \widehat{\varepsilon}_i^2 X_i X_i^{\mathrm{T}} \left(\sum_{i=1}^{n} X_i X_i^{\mathrm{T}}\right)^{-1}.$$

Now, we illustrate White's estimator on the pipeline data set from the faraway package. We know that the variation in the standardized residuals increases in a cone shape as a function of the field predictor. We calculate the ordinary least squares estimate for the variance of the least squares estimator, the t test statistics and 95% confidence intervals for the regression coefficients.

```r
library(faraway)
library(xtable)
fit = lm(Lab ~ Field, pipeline)
betahat = fit$coefficients
X = model.matrix(fit)
n = dim(X)[1]
p = dim(X)[2]
varOLS = summary(fit)$sigma^2 * summary(fit)$cov.unscaled
print(xtable(varOLS, digits = c(0, 4, 4)), comment = FALSE)
```

	(Intercept)	Field
(Intercept)	2.4800	-0.0566
Field	-0.0566	0.0017

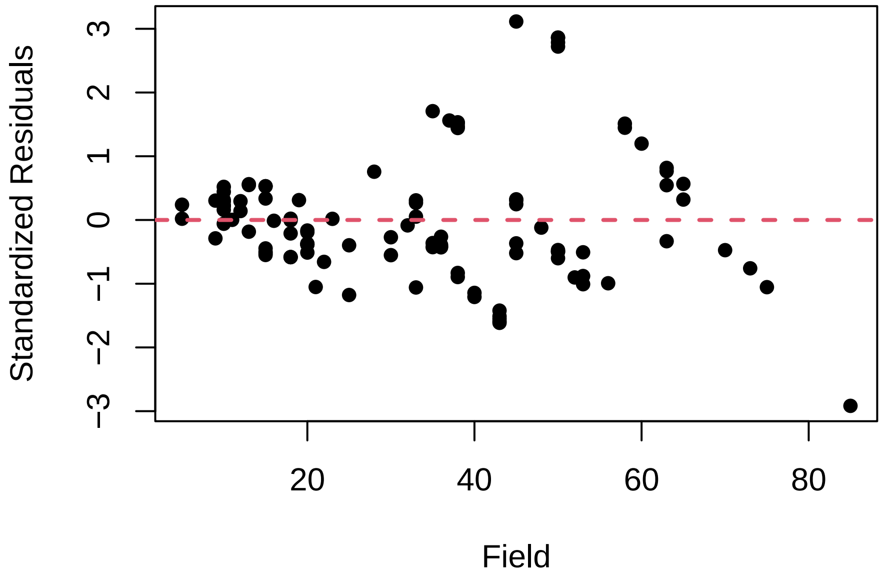
```r
print(xtable(summary(fit)), comment = FALSE)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-1.9675	1.5748	-1.25	0.2143
Field	1.2230	0.0411	29.78	0.0000

```r
print(xtable(confint(fit)), comment = FALSE)
```

	2.5 %	97.5 %
(Intercept)	-5.09	1.16
Field	1.14	1.30

```r
plot(rstandard(fit) ~ Field, pipeline, ylab = "Standardized Residuals", pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



Then, we calculate White's estimator and use it to construct heteroscedasticity-consistent confidence intervals for the regression coefficients. We observe that the heteroscedasticity-consistent standard error of the intercept is much smaller, so the corresponding confidence interval is much tighter.

```r
alpha = 0.05
Sigma = diag(fit$residuals^2)
varWhite = summary(fit)$cov.unscaled %*% crossprod(X, Sigma) %*% X %*% summary(fit)$cov.unscaled
print(xtable(varWhite, digits = c(0, 4, 4)), comment = FALSE)
```

	(Intercept)	Field
(Intercept)	1.3220	-0.0453
Field	-0.0453	0.0020

```r
White = matrix(0, p, 4)
rownames(White) = names(fit$coefficients)
colnames(White) = colnames(summary(fit)$coef)
White[, 1] = betahat
White[, 2] = sqrt(diag(varWhite))
White[, 3] = White[, 1]/White[, 2]
White[, 4] = 2 * pt(abs(White[, 3]), n - p, lower.tail = FALSE)
print(xtable(White), comment = FALSE)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-1.97	1.15	-1.71	0.09
Field	1.22	0.05	27.17	0.00

```r
White = White[, 1:2]
colnames(White) = colnames(confint(fit))
White[, 1] = betahat - qt(1 - alpha/2, n - p) * sqrt(diag(varWhite))
White[, 2] = betahat + qt(1 - alpha/2, n - p) * sqrt(diag(varWhite))
print(xtable(White), comment = FALSE)
```

	2.5 %	97.5 %
(Intercept)	-4.25	0.31
Field	1.13	1.31

We verify our calculation of White's estimator using the vcovHC function from the sandwich package with the argument "HC". We also verify our computation of the heteroscedasticity-consistent t test statistics by using the coeftest function from the lmtest packages and specifying the computed White's estimator as the covariance matrix of the estimated coefficients.

```r
library(sandwich)
library(lmtest)
varWhite = vcovHC(fit, "HC")
print(xtable(varWhite, digits = c(0, 4, 4)), comment = FALSE)
```

	(Intercept)	Field
(Intercept)	1.3220	-0.0453
Field	-0.0453	0.0020

```r
print(xtable(coeftest(fit, vcov. = varWhite)[, ]), comment = FALSE)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-1.97	1.15	-1.71	0.09
Field	1.22	0.05	27.17	0.00

```r
print(xtable(confint(coeftest(fit, vcov. = varWhite))), comment = FALSE)
```

	2.5 %	97.5 %
(Intercept)	-4.25	0.31
Field	1.13	1.31

Weighted Least Squares

Suppose that $Y_i = X_i^{\mathrm{T}}\beta + \varepsilon_i$, where $\beta \in \mathbb{R}^p$, $\varepsilon_i \sim \mathcal{N}\left(0, \sigma_i^2\right)$, $\sigma_i^2 = w_i^{-1}\sigma^2$ and $W = \operatorname{diag}\{w_1, w_2, \ldots, w_n\}$. We can equivalently write that $Y = X\beta + \varepsilon$, where $\varepsilon \sim \mathcal{N}_n\left(\mathbf{0}_n, \sigma^2 W^{-1}\right)$. Let $W^{1/2} = \operatorname{diag}\left\{\sqrt{w_1}, \sqrt{w_2}, \ldots, \sqrt{w_n}\right\}$ denote the square root of the diagonal weight matrix W. We define:

$$\widetilde{Y} = W^{1/2}Y, \quad \widetilde{X} = W^{1/2}X, \quad \widetilde{\varepsilon} = W^{1/2}\varepsilon \sim \mathcal{N}_n\left(\mathbf{0}_n, \sigma^2 I_n\right).$$

Then, we observe that the transformed linear model $\widetilde{Y} = \widetilde{X}\beta + \widetilde{\varepsilon}$ is homoscedastic and can be fitted using ordinary least squares. The ordinary least squares estimator of the transformed linear model is equal to the weighted least squares estimator of the original linear model:

$$\widehat{\beta}_{\mathrm{WLS}} = \left(\widetilde{X}^{\mathrm{T}}\widetilde{X}\right)^{-1}\widetilde{X}^{\mathrm{T}}\widetilde{Y} = \left(X^{\mathrm{T}}WX\right)^{-1}X^{\mathrm{T}}WY.$$

We calculate that:

$$E\left(\widehat{\beta}_{\mathrm{WLS}}\right) = \cancel{\left(X^{\mathrm{T}}WX\right)^{-1}}\,\cancel{X^{\mathrm{T}}WX}\beta = \beta,$$

$$\mathrm{Var}\left(\widehat{\beta}_{\mathrm{WLS}}\right) = \sigma^2\left(X^{\mathrm{T}}WX\right)^{-1}X^{\mathrm{T}}\cancel{W}\,\cancel{W^{-1}}WX\left(X^{\mathrm{T}}WX\right)^{-1} = \sigma^2\left(X^{\mathrm{T}}WX\right)^{-1}.$$

Let $\widehat{Y} = X\widehat{\beta}_{\mathrm{WLS}}$ and $\widehat{\varepsilon} = Y - \widehat{Y}$. We also define the weighted average of Y as follows:

$$\overline{Y}_{\mathrm{WLS}} = \frac{\sum_{i=1}^{n} w_i Y_i}{\sum_{i=1}^{n} w_i}.$$

Then, the sums of squares in weighted least squares regression are given by:

$$\mathrm{SSE} = \sum_{i=1}^{n} w_i\widehat{\varepsilon}_i^2 = \widehat{\varepsilon}^{\mathrm{T}}W\widehat{\varepsilon},$$

$$\mathrm{SSR} = \sum_{i=1}^{n} w_i\left(\widehat{Y}_i - \overline{Y}_{\mathrm{WLS}}\right)^2,$$

$$\mathrm{SST} = \sum_{i=1}^{n} w_i\left(Y_i - \overline{Y}_{\mathrm{WLS}}\right)^2.$$

It still holds that $\mathrm{SST} = \mathrm{SSR} + \mathrm{SSE}$, so we can define the R^2 coefficient of determination and its adjusted counterpart in the same manner as in ordinary least squares regression.

The true weights are obviously unknown, but a good choice for them may be deduced from an exploratory data analysis. For example, the variance of the response variable might appear to be a function of some of the available predictors. If the response y_i for individual i is an average over a sample of size n_i, then a logical choice of weight is $w_i = n_i$. If the response y_i for individual i is a sum over a sample of size n_i, then a logical choice of weight is $w_i = n_i^{-1}$.

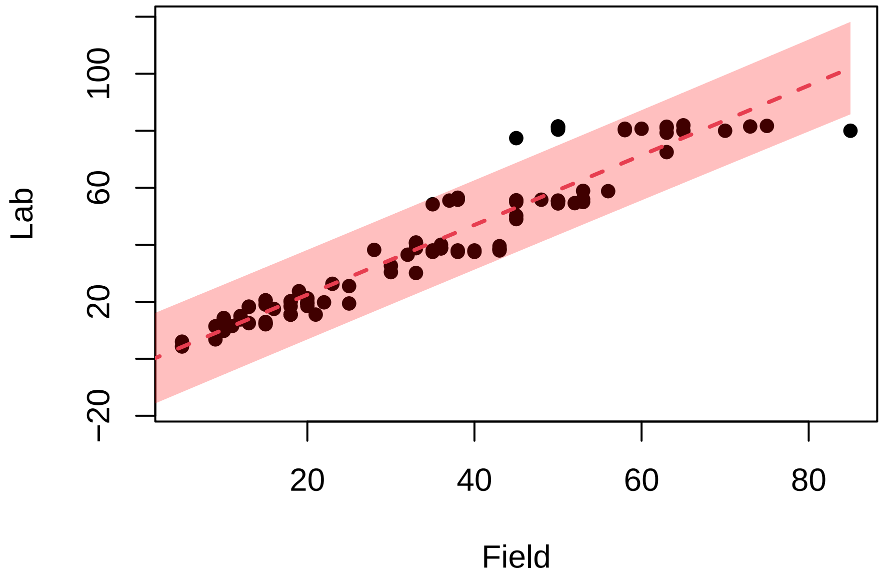
Now, we illustrate the weighted least squares method on the pipeline data set from the faraway package. We know that the variation in the standardized residuals increases in a cone shape as a function of the field variable.

```r
library(faraway)
fit = lm(Lab ~ Field, pipeline)
residuals = fit$residuals
Y = fit$model[, 1]
X = model.matrix(fit)
n = dim(X)[1]
p = dim(X)[2]
summary(fit)
```

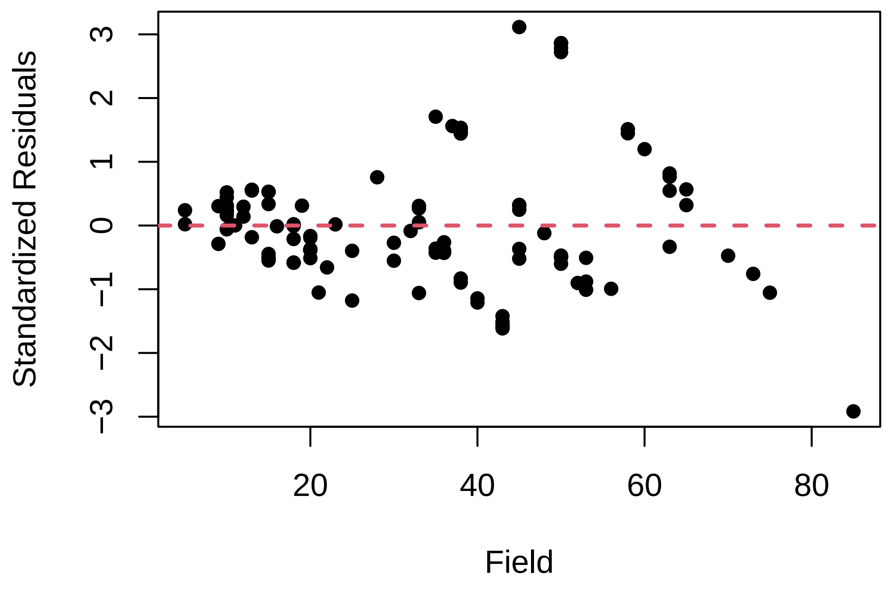
```
##
```

```
## Call:
## lm(formula = Lab ~ Field, data = pipeline)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -21.985  -4.072  -1.431   2.504  24.334
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.96750    1.57479  -1.249    0.214
## Field        1.22297    0.04107  29.778   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.865 on 105 degrees of freedom
## Multiple R-squared:  0.8941, Adjusted R-squared:  0.8931
## F-statistic: 886.7 on 1 and 105 DF,  p-value: < 2.2e-16
```

```
x = seq(min(X), max(X), 0.1)
predictions = predict(fit, data.frame(Field = x), interval = "prediction")
plot(Lab ~ Field, pipeline, ylim = range(predictions), pch = 16)
lines(predictions[, 1] ~ x, gala, "l", col = 2, lty = 2, lwd = 2)
polygon(c(x, rev(x)), c(predictions[, 2], rev(predictions[, 3])), border = NA,
    col = rgb(1, 0, 0, 0.25))
```



```
plot(rstandard(fit) ~ Field, pipeline, ylab = "Standardized Residuals", pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



The variance of the response variable appears to be a quadratic function of the field variable. Hence, a good choice of weights would be for them to be equal to the inverse of the square of the field variable. This leads to weights which become smaller as the predictor takes larger values, so the corresponding variances become larger. Any diagnostic checks for the weighted least squares model have to be based on the residuals multiplied by the square roots of the weights, since the unweighted residuals don't have constant variance by design. Hence, a good choice of weights would be for them to be equal to the inverse of the square of the field variable. The standard errors of the regression coefficients become much smaller after applying the weighted least squares method, leading to more precise estimates. Note that we can only compare models with the same response variable and the same choice of weights by using the coefficient of determination and its adjusted counterpart.

```
w = 1/pipeline$Field^2
W = diag(w)
betaWLS = drop(solve(crossprod(X, W) %*% X, crossprod(X, W) %*% Y))
YWLS = sum(w * Y)/sum(w)
fitted = drop(X %*% betaWLS)
residuals = (Y - fitted) * sqrt(w)
summary(residuals)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.40003 -0.15467 -0.04935  0.00000  0.13453  0.56117
```

```
SSE = sum(residuals^2)
SSR = sum(w * (fitted - YWLS)^2)
SST = sum(w * (Y - YWLS)^2)
S = sqrt(SSE/(n - p))
varWLS = S^2 * solve(crossprod(X, W) %*% X)
WLS = matrix(0, p, 4)
rownames(WLS) = names(fit$coefficients)
colnames(WLS) = colnames(summary(fit)$coef)
WLS[, 1] = betaWLS
```

```
WLS[, 2] = sqrt(diag(varWLS))
WLS[, 3] = WLS[, 1]/WLS[, 2]
WLS[, 4] = 2 * pt(abs(WLS[, 3]), n - p, lower.tail = FALSE)
print(WLS)
```

```
##                Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -0.8155332 0.60176881 -1.355227 1.782536e-01
## Field        1.1769553 0.03400955 34.606614 3.109939e-59
```

```
print(S)
```

```
## [1] 0.2138649
```

```
R2 = SSR/SST
print(R2)
```

```
## [1] 0.9193931
```

```
R2adj = 1 - (1 - R2) * (n - 1)/(n - p)
print(R2adj)
```

```
## [1] 0.9186254
```

```
f = (SSR/(p - 1))/S^2
print(f)
```

```
## [1] 1197.618
```

Alternatively, we can transform the response and the design matrix, including the intercept, by multiplying them by the square roots of the weights. Then, we can regress the transformed Y on the transformed X without an intercept, since a transformed intercept is included in the transformed X. We can see that all of the weighted least squares results expect for the coefficients of determination and the overall F test statistic agree with our previous results after following this method of fitting the regression model. The fitted values of the original response variable are obtained by dividing the fitted values by the square roots of the weights. We can see that the variation in the standardized residuals is fairly constant after using weighted least squares.

```
Yweight = Y * sqrt(w)
Xweight = X * sqrt(w)
WLS = lm(Yweight ~ 0 + Xweight)
summary(WLS$residuals)
```

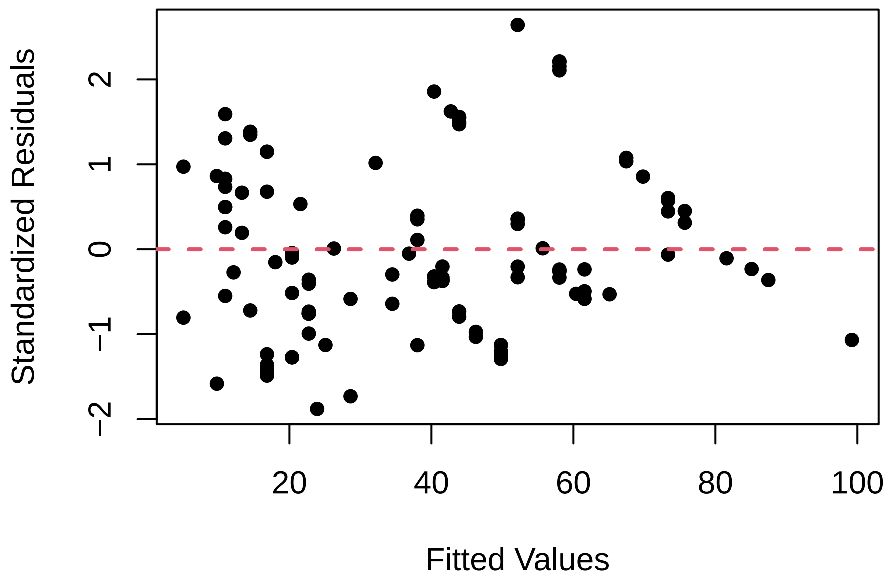
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.40003 -0.15467 -0.04935  0.00000  0.13453  0.56117
```

```
summary(WLS)
```

```
##
## Call:
## lm(formula = Yweight ~ 0 + Xweight)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.40003 -0.15467 -0.04935  0.13453  0.56117
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## Xweight(Intercept) -0.81553    0.60177  -1.355    0.178
## XweightField        1.17696    0.03401  34.607   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2139 on 105 degrees of freedom
## Multiple R-squared:  0.9667, Adjusted R-squared:  0.966
## F-statistic:  1522 on 2 and 105 DF,  p-value: < 2.2e-16
```

```
plot(WLS$fitted.values/sqrt(w), rstandard(WLS), xlab = "Fitted Values", ylab = "Standardized Residuals",
    pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



The most accurate way of running weighted least squares estimation would be by using the weights argument in R's built-in lm function to specify the weights we desire. Now, all of the obtained results agree with our previous calculations. Note that lm actually returns the unweighted residuals in this case, so we have to use summary.lm to obtain the correctly weighted residuals. When making predictions according to this fitted regression model, we have to specify the appropriate weights which correspond to our new predictor values. We observe that the resulting prediction region does a better job of capturing the trend in the response variable than the one obtained using ordinary least squares and is very similar to the prediction region obtained by applying a log-transformation on both the predictor and the response variable.

```
WLS = lm(Lab ~ Field, pipeline, weights = w)
summary(WLS$residuals)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -19.2257  -4.0160  -1.5236   0.3931   3.0993  25.2525
```

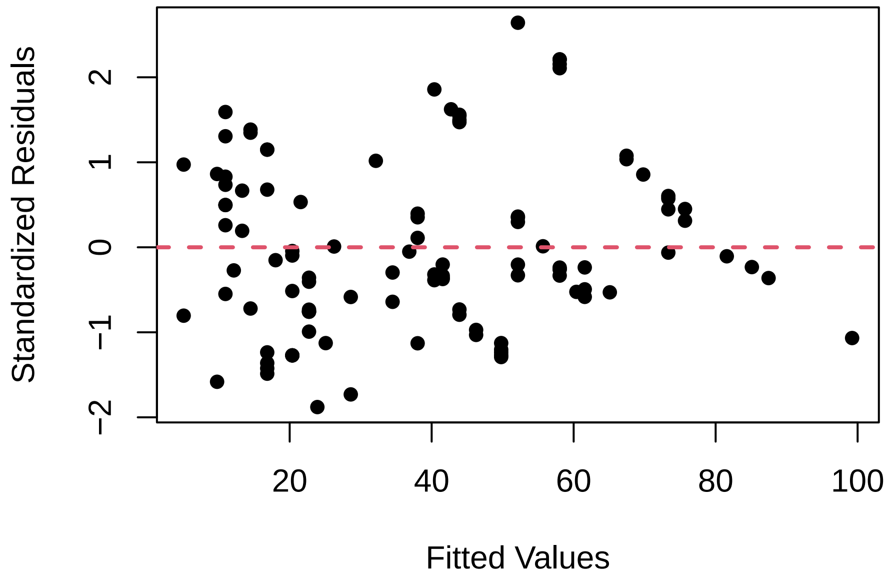
```
summary(summary(WLS)$residuals)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.40003 -0.15467 -0.04935  0.00000  0.13453  0.56117
```

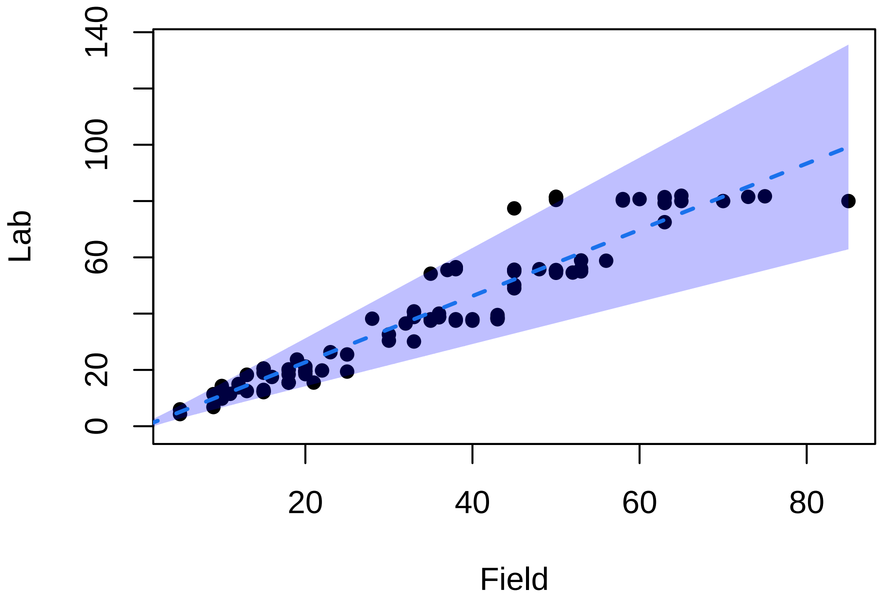
```
summary(WLS)
```

```
##
## Call:
## lm(formula = Lab ~ Field, data = pipeline, weights = w)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -0.40003 -0.15467 -0.04935  0.13453  0.56117
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.81553    0.60177  -1.355    0.178
## Field        1.17696    0.03401  34.607   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2139 on 105 degrees of freedom
## Multiple R-squared:  0.9194, Adjusted R-squared:  0.9186
## F-statistic:  1198 on 1 and 105 DF,  p-value: < 2.2e-16
```

```
plot(WLS$fitted.values, rstandard(WLS), xlab = "Fitted Values", ylab = "Standardized Residuals",
    pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



```r
x = seq(min(X), max(X), 0.1)
predictions = predict(WLS, data.frame(Field = x), interval = "prediction", weights = 1/x^2)
plot(Lab ~ Field, pipeline, ylim = range(predictions), pch = 16)
lines(predictions[, 1] ~ x, gala, "l", col = 4, lty = 2, lwd = 2)
polygon(c(x, rev(x)), c(predictions[, 2], rev(predictions[, 3])), border = NA,
    col = rgb(0, 0, 1, 0.25))
```



More generally, we might believe that $\sigma_i^2 = \alpha_0 \text{Field}_i^{\alpha_1}$ for some unknown parameters $\alpha_0 > 0$ and $\alpha_1 \in \mathbb{R}$. Equivalently, we might believe that $\log \sigma_i^2 = \log \alpha_0 + \alpha_1 \log \text{Field}_i$. We can estimate the unknown parameters α_0, α_1 using a variation of the iteratively reweighted least squares (IRLS) method.

Algorithm 2.9 Iteratively Reweighted Least Squares

```
Input: Random sample $(Y, X)$.
1: We initialize $w_i = 1$ for $i = 1, 2, \dots, n$, regress $Y$ on $X$ and calculate the residual vector $\widehat{\varepsilon}$.
2: We iterate the following steps until convergence of the weights:
    i: We define the auxiliary variable $Y_i^{\text{aux}} = \log \widehat{\varepsilon}_i^2$;
    ii: We regress $Y^{\text{aux}}$ on $\log X_1, \log X_2, \dots, \log X_p$ and calculate the fitted values vector $\widehat{Y}^{\text{aux}}$;
    iii: We redefine the weights as $w_i = e^{-\widehat{Y}_i^{\text{aux}}}$;
    iv: We use the new weights to regress $Y$ on $X$ and calculate a new unweighted residual vector $\widehat{\varepsilon}$.
Output: Weighted least squares estimate.
```

We apply the iteratively reweighted least squares method until the relative difference between 2 consecutive weight vector estimates is less than 10^{-5}. We observe that $\sigma_i^2 \approx 0.05 \cdot \text{Field}_i^{1.67}$ according to the final auxiliary regression model. The final weighted least squares model is very similar to the one we estimated previously. In order to make predictions based on the final weighted least squares model, we first need to predict the weights corresponding to the new predictor values based on the final auxiliary model. The resulting prediction region is slightly narrower than the one calculated previously.

```
weights = 1
err = Inf
while (err > 1e-05) {
    aux = lm(log(residuals^2) ~ log(Field), pipeline)
    err = sum(abs(weights - exp(-aux$fitted.values))/weights)
    weights = exp(-aux$fitted.values)
    WLS = lm(Lab ~ Field, pipeline, w = weights)
    residuals = WLS$residuals
}
aux = lm(log(residuals^2) ~ log(Field), pipeline)
alpha0 = exp(aux$coefficients[1])
print(alpha0)
```

```
## (Intercept)
##  0.04607651
```

```
alpha1 = aux$coefficients[2]
print(alpha1)
```

```
## log(Field)
##   1.668438
```

```
summary(aux)
```

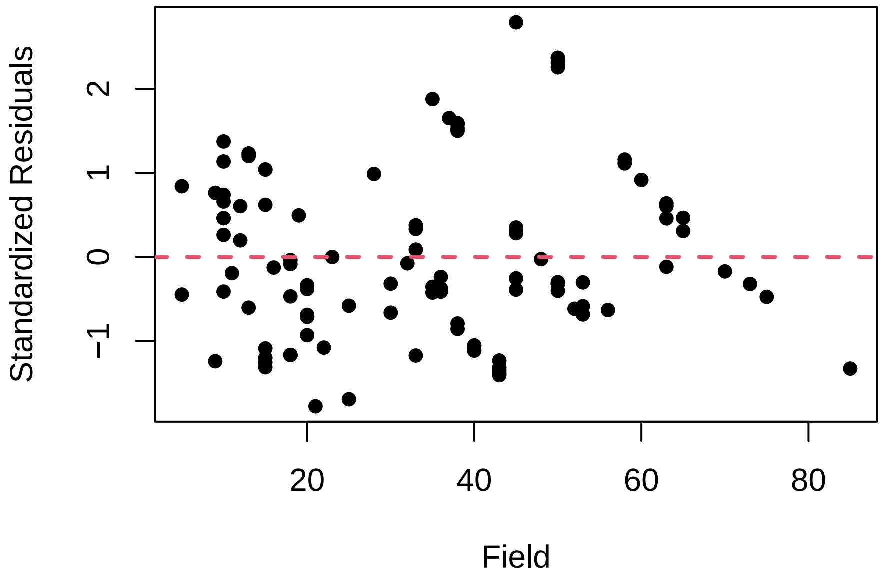
```
##
## Call:
## lm(formula = log(residuals^2) ~ log(Field), data = pipeline)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.2763  -0.9092   0.2493   1.4223   3.1576
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.0775     1.0718  -2.871  0.00495 **
## log(Field)    1.6684     0.3159   5.281 6.99e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.116 on 105 degrees of freedom
## Multiple R-squared:  0.2099, Adjusted R-squared:  0.2023
## F-statistic: 27.89 on 1 and 105 DF,  p-value: 6.987e-07
```

```
summary(WLS)
```

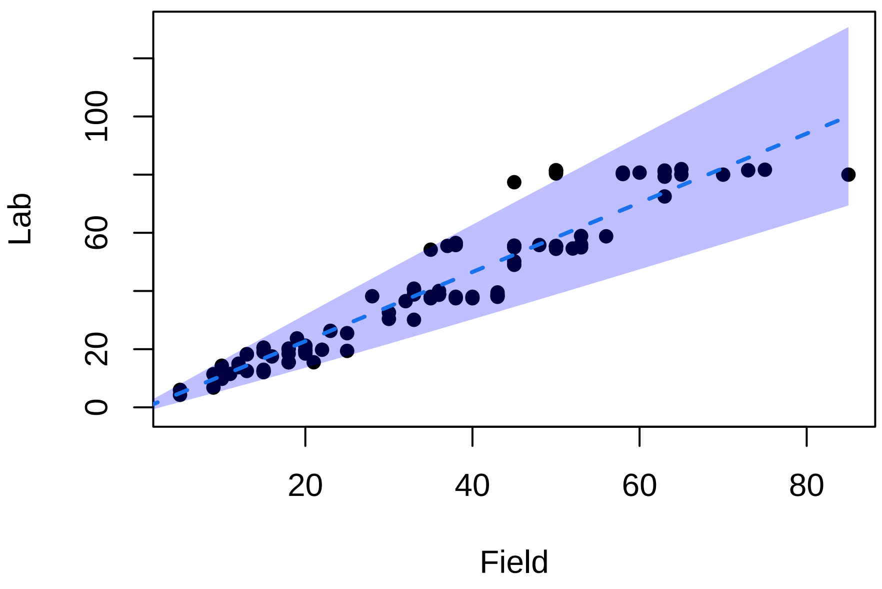
```
##
```

```
## Call:
## lm(formula = Lab ~ Field, data = pipeline, weights = weights)
##
## Weighted Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0968 -1.1944 -0.4437  1.0601  4.8492
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.05707    0.69839  -1.514    0.133
## Field        1.18971    0.03401  34.985   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.75 on 105 degrees of freedom
## Multiple R-squared:  0.921,  Adjusted R-squared:  0.9202
## F-statistic:  1224 on 1 and 105 DF,  p-value: < 2.2e-16
```

```
plot(rstandard(WLS) ~ Field, pipeline, ylab = "Standardized Residuals", pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



```
predictions = predict(WLS, data.frame(Field = x), interval = "prediction", weights = exp(-predict(aux,
    data.frame(Field = x))))
plot(Lab ~ Field, pipeline, ylim = range(predictions), pch = 16)
lines(predictions[, 1] ~ x, gala, "l", col = 4, lty = 2, lwd = 2)
polygon(c(x, rev(x)), c(predictions[, 2], rev(predictions[, 3])), border = NA,
    col = rgb(0, 0, 1, 0.25))
```



Autocorrelation Tests

Inference on linear models also heavily relies on the assumption of uncorrelatedness of the error terms. Hence, it is important to be able to test for departures from this assumption when it is in doubt. One possible form of autocorrelation appears on time series data, where observations on consecutive time points are naturally correlated with each other, and that correlation fades the further away 2 time points lie from each other. There are a number of different tests which can be used to check for autocorrelation.

An example of this form of autocorrelation appears in the longley data set. We can see that the residuals corresponding to consecutive years tend to have the same sign instead of being randomly scattered around the x-axis, which is a clear sign of autocorrelation. Additionally, the residuals appear to be positively correlated with their lagged counterparts from the previous year, which is also a sign of autocorrelation. We also define the sample autocorrelation of the residuals at lag k as follows:

$$r_k = \frac{\sum_{i=k+1}^{n} \widehat{\varepsilon}_i \widehat{\varepsilon}_{i-k}}{\sum_{i=1}^{n} \widehat{\varepsilon}_i^2}.$$

Since the sample average of the residuals is equal to 0 by design, the numerator provides with an estimate of the total covariation between the residuals and their lagged counterpart, while the denominator estimates the total variation in the residuals, providing us with an estimate of the correlation between the residuals and their lagged counterpart. We can verify our calculation using R's built-in acf function, which can also output a plot of the autocorrelation function for increasing lag with appropriate confidence bounds. Any sample autocorrelation r_k outside the bounds implies that the the corresponding true autocorrelation ρ_k is statistically significant at the chosen significance level. The first autocorrelation at lag $k = 0$ is always equal to 1 by definition so it's of not interest. The second autocorrelation at lag $k = 1$ appears to be significant at the 10% significance level. Further autocorrelations which spuriously appear to be statistically significant down the line may be attributed to random chance due to multiple testing or may be a sign of seasonality in the time series.

```
library(dplyr)
n = dim(longley)[1]
p = 3
```

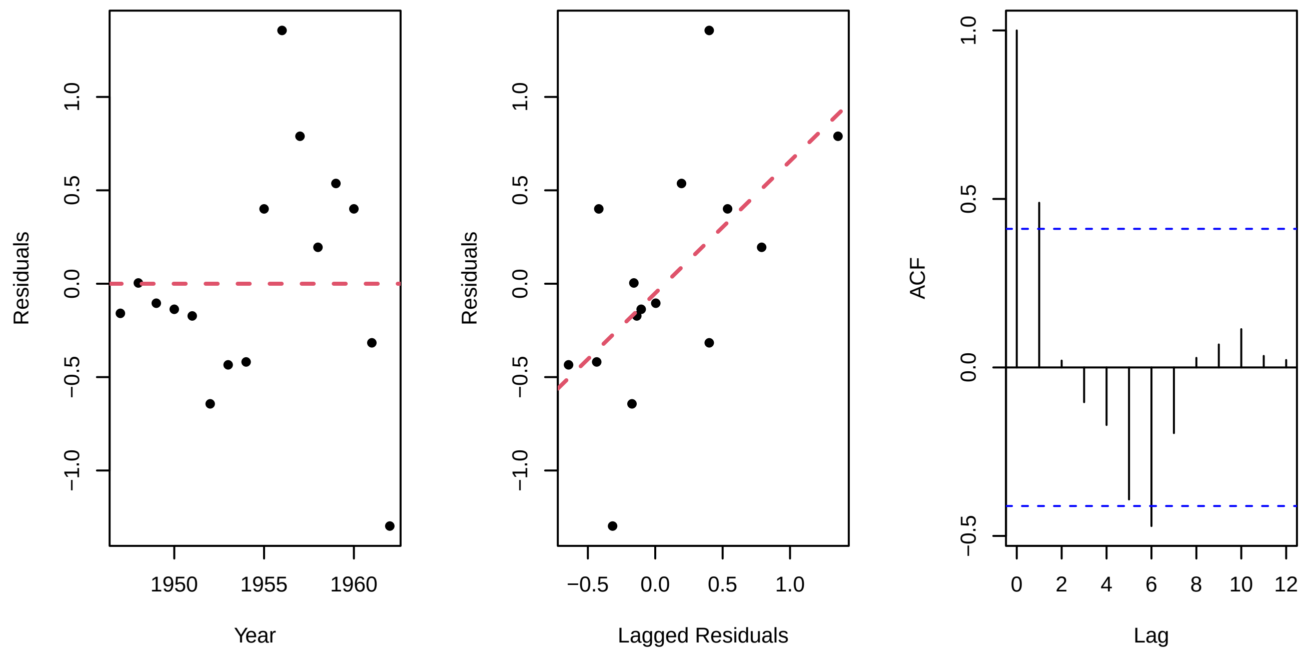
```
k = 1
fit = lm(Employed ~ Unemployed + Population, longley)
Residuals = fit$residuals
ACF = sum(Residuals * lag(Residuals, k), na.rm = TRUE)/sum(Residuals^2)
print(ACF)
```

```
## [1] 0.4886877
```

```
acf(Residuals, plot = FALSE)$acf[k + 1]
```

```
## [1] 0.4886877
```

```
par(mfrow = c(1, 3))
plot(Residuals ~ Year, longley, pch = 16)
abline(h = 0, col = 2, lty = 2, lwd = 2)
plot(lag(Residuals, k), Residuals, xlab = "Lagged Residuals", pch = 16)
abline(lm(Residuals ~ lag(Residuals, k)), col = 2, lty = 2, lwd = 2)
acf(Residuals, ci = 0.9, main = NA)
```



The Durbin-Watson test statistic is defined as:

$$D = \frac{\sum_{i=2}^{n}\left(\widehat{\varepsilon}_i - \widehat{\varepsilon}_{i-1}\right)^2}{\sum_{i=1}^{n}\widehat{\varepsilon}_i^2}.$$

We can observe that:

$$D = \frac{\sum_{i=2}^{n}\left(\widehat{\varepsilon}_i^2 + \widehat{\varepsilon}_{i-1}^2\right)}{\sum_{i=1}^{n}\widehat{\varepsilon}_i^2} - 2\frac{\sum_{i=2}^{n}\widehat{\varepsilon}_i\widehat{\varepsilon}_{i-1}}{\sum_{i=1}^{n}\widehat{\varepsilon}_i^2} \approx 2 - 2r_1 = 2(1 - r_1).$$

Since $r_1 \in [-1, 1]$, we infer that the Durbin-Watson test statistic takes values on $[0, 4]$. When $r_1 \approx 0$, we notice that $D \approx 2$. Hence, observed values of the test statistic away from 2 are a sign of presence of autocorrelation at lag $k = 1$. Unfortunately, the test statistic doesn't follow some known distribution under H_0, so the p-value associated with the observed statistic needs to be estimated via Pan's iterative procedure, a normal approximation or Monte

Carlo simulation.

The dwtest function from the lmtest package uses Pan's algorithm for smaller sample sizes $n < 100$ and a normal approximation for larger sample sizes. The observed value of the test statistic is $0.73 \ll 2$ and the corresponding estimated p-value is $5 \cdot 10^{-4}$, which implies the rejection of the null hypothesis of no autocorrelation at lag $k = 1$ against the two-sided alternative.

```r
library(lmtest)
DW = sum(diff(Residuals)^2)/sum(Residuals^2)
print(DW)
```

```
## [1] 0.7257321
```

```r
dwtest(fit, alternative = "two.sided")
```

```
##
##  Durbin-Watson test
##
## data:  fit
## DW = 0.72573, p-value = 0.0004777
## alternative hypothesis: true autocorrelation is not 0
```

The Ljung-Box test statistic is defined as:

$$\mathrm{LB} = n(n+2)\sum_{h=1}^{k}\frac{r_h^2}{n-h}.$$

Under the null hypothesis of no autocorrelation up to lag k, we know that $\mathrm{LB} \xrightarrow{d} \chi_k^2$. If LB_0 is the observed value of the test statistic, then we calculate the p-value of the test as $P\left(\mathrm{LB} \geqslant \mathrm{LB}_0\right)$.

The observed value of the test statistic is 4.59 and the corresponding p-value is 0.03, which implies the rejection of the null hypothesis of no autocorrelation up to lag $k = 1$. We verify our calculations by using R's built-in Box.test function and specifying the number k of tested lags and type = "Ljung-Box".

```r
LB = n * (n + 2) * sum(acf(Residuals, plot = FALSE)$acf[2:(k + 1)]^2/(n - (1:k)))
print(LB)
```

```
## [1] 4.585261
```

```r
pchisq(LB, k, lower.tail = FALSE)
```

```
## [1] 0.03224807
```

```r
Box.test(Residuals, k, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  Residuals
## X-squared = 4.5853, df = 1, p-value = 0.03225
```

The Box-Pierce test statistic is defined as:

$$\mathrm{BP} = n \sum_{h=1}^{k} r_h^2.$$

Under the null hypothesis of no autocorrelation up to lag k, we know that $\mathrm{BP} \xrightarrow{d} \chi_k^2$. If BP_0 is the observed value of the test statistic, then we calculate the p-value of the test as $P\left(\mathrm{BP} \geqslant \mathrm{BP}_0\right)$. It should be noted that the Ljung-Box test statistic more closely approximates the null χ_k^2 distribution thatn the Box-Pierce test statistic, even for smaller sample sizes.

The observed value of the test statistic is 3.82 and the corresponding p-value is 0.0506, which is borderline above the usual significance level. We verify our calculations by again using R's built-in Box.test function and specifying the number k of tested lags.

```
BP = n * sum(acf(Residuals, plot = FALSE)$acf[2:(k + 1)]^2)
print(BP)
```

```
## [1] 3.82105
```

```
pchisq(BP, k, lower.tail = FALSE)
```

```
## [1] 0.0506125
```

```
Box.test(Residuals, k)
```

```
##
##  Box-Pierce test
##
## data:  Residuals
## X-squared = 3.8211, df = 1, p-value = 0.05061
```

Algorithm 2.10 Breusch-Godfrey χ^2 Test

```
    Input: Random sample $(Y, X)$ and lag $k$.
1:  We regress $Y$ on $X$ and calculate the residual vector $\widehat{\varepsilon}$.
2:  We regress $\widehat{\varepsilon}$ on $X$ and the lagged residual vectors of up to order $k$. We calculate the coefficient of determination:
        $R^2_{\mathrm{aux}} = \frac{\mathrm{SSR}_{\mathrm{aux}}}{\mathrm{SST}_{\mathrm{aux}}}$.
3:  We calculate the observed value $\mathrm{BG}_0$ of the Breusch-Godfrey test statistic BG as $\mathrm{BG}_0 = (n-k)R^2_{\mathrm{aux}}$. We know that $\mathrm{BG} \xrightarrow{d} \chi_k^2$ under the null hypothesis of no autocorrelation up to lag $k$.
4:  We calculate the p-value of the test as $P(\mathrm{BG} \geqslant \mathrm{BG}_0)$.
    Output: Observed test statistic $\mathrm{BG}_0$ and p-value.
```

The observed value of the test statistic is 6.55 and the corresponding p-value is 0.01, which implies the rejection of the null hypothesis of no autocorrelation up to lag $k = 1$.

```
Lagged = sapply(1:k, function(x) {
    lag(Residuals, x)
```

```r
})
aux = lm(Residuals ~ Unemployed + Population + Lagged, longley)
BG = (n - k) * summary(aux)$r.squared
print(BG)
```

```
## [1] 6.551443
```

```r
pchisq(BG, k, lower.tail = FALSE)
```

```
## [1] 0.01047991
```

Algorithm 2.11 Breusch-Godfrey F Test

```
    Input: Random sample $(Y, X)$ and lag $k$.
1:  We regress $Y$ on $X \in \mathbb{R}^{n \times p}$ and calculate the residual vector $\widehat{\varepsilon}$.
2:  We regress $\widehat{\varepsilon}$ on $X \in \mathbb{R}^{n \times p}$ and the lagged residual vectors of up to order $k$ to obtain a full model.
3:  We regress $\widehat{\varepsilon}$ on just $X \in \mathbb{R}^{n \times p}$ to obtain a reduced model. We perform an $F$ test of significance to compare the reduced against the full model.
4:  We know that $F \sim F_{k,n-p-2k}$ under the null hypothesis of no autocorrelation up to lag $k$. We calculate the p-value of the test as $P(F \geqslant f)$.
    Output: Observed test statistic $f$ and p-value.
```

The F test obviously has better small sample properties than the χ^2 test, since it doesn't rely on an asymptotic approximation of the null distribution of the test statistic. The observed value of the test statistic is 8.51 and the corresponding p-value is 0.01, which implies the rejection of the null hypothesis of no autocorrelation up to lag $k = 1$.

```r
reduced = lm(Residuals ~ Unemployed + Population, longley, (k + 1):n)
anova(reduced, aux)
```

```
## Analysis of Variance Table
##
## Model 1: Residuals ~ Unemployed + Population
## Model 2: Residuals ~ Unemployed + Population + Lagged
##   Res.Df    RSS Df Sum of Sq      F  Pr(>F)
## 1     12 5.7237
## 2     11 3.2263  1    2.4974 8.5148 0.01399 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Cochrane–Orcutt Estimation

Suppose that we are interested in the linear model $Y_i = X_i^{\mathrm{T}}\beta + \varepsilon_i$, where $\varepsilon_i = \rho\varepsilon_{i-1} + \omega_i$, $\rho \in (-1, 1)$ and $\omega_i \sim \mathcal{N}\left(0, \sigma^2\right)$ are independent. We observe that:

$$Y_i = X_i^{\mathrm{T}}\beta + \rho\varepsilon_{i-1} + \omega_i \Rightarrow$$

$$Y_i = X_i^{\mathrm{T}}\beta + \rho\left(Y_{i-1} - X_{i-1}^{\mathrm{T}}\beta\right) + \omega_i \Rightarrow$$

$$\underbrace{Y_i - \rho Y_{i-1}}_{\widetilde{Y}_i} = \underbrace{\left(X_i - \rho X_{i-1}\right)^{\mathrm{T}}}_{\widetilde{X}_i^{\mathrm{T}}}\beta + \omega_i.$$

Hence, we arrive at a transformed linear model $\widetilde{Y}_i = \widetilde{X}_i^{\mathrm{T}}\beta + \omega_i$ with uncorrelated errors ω_i. The true coefficient ρ is obviously unknown, so we can try estimating it by applying the Cochrane-Orcutt iterative procedure.

Algorithm 2.12 Cochrane–Orcutt Estimation

```
    Input: Random sample $(Y, X)$.
1:  We regress $Y$ on $X \in \mathbb{R}^{n\times p}$, including an intercept, and calculate the residual vector $\widehat{\varepsilon}$.
2:  We regress the residual vector on the lagged residual vector without an intercept and extract the estimated slope coefficient $\rho$.
3:  We iterate the following steps until convergence of $\rho$:
      i: We define the auxiliary variables $Y_i^{\mathrm{CO}} = Y_i - \rho Y_{i-1}$ and $X_{i,j}^{\mathrm{CO}} = X_{i,j} - \rho X_{i-1,j}$;
      ii: We regress $Y^{\mathrm{CO}} \in \mathbb{R}^{n-1}$ on $X^{\mathrm{CO}} \in \mathbb{R}^{(n-1)\times p}$ without an intercept, since a transformed intercept is included in $X^{\mathrm{CO}}$, and extract the least squares estimator $\widehat{\beta}_{\mathrm{CO}}$;
      iii: We calculate a new residual vector $\widehat{\varepsilon} = Y - X\widehat{\beta}_{\mathrm{CO}}$;
      iv: We regress the new residual vector on the new lagged residual vector without an intercept and extract the estimated slope coefficient $\rho$.
    Output: Fitted regression model with uncorrelated residuals.
```

Now, we illustrate the Cochrane-Orcutt estimation procedure on the longley data set. We know that there's statistically significant lag-1 autocorrelation in the residuals from the longley data set according to the Durbin-Watson test. The effect of both the unemployed and the population predictors on the employed variable is estimated to be statistically significant.

```r
library(lmtest)
n = dim(longley)[1]
fit = lm(Employed ~ Unemployed + Population, longley)
residuals = fit$residuals
Y = fit$model[, 1]
X = model.matrix(fit)
dwtest(fit, alternative = "two.sided")
```

```
##
##  Durbin-Watson test
##
## data:  fit
## DW = 0.72573, p-value = 0.0004777
## alternative hypothesis: true autocorrelation is not 0
```

```r
summary(fit)
```

```
##
```

```
## Call:
## lm(formula = Employed ~ Unemployed + Population, data = longley)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2975 -0.3420 -0.1206  0.4006  1.3560
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.135325   3.487991  -0.039 0.969641
## Unemployed  -0.011151   0.002528  -4.410 0.000704 ***
## Population   0.587728   0.033967  17.303 2.34e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6654 on 13 degrees of freedom
## Multiple R-squared:  0.9689, Adjusted R-squared:  0.9641
## F-statistic: 202.5 on 2 and 13 DF,  p-value: 1.598e-10
```

After applying the Cochrane-Orcutt estimation procedure, we can see that the estimate of the coefficient ρ starts off at 0.69 and converges to 0.99, which implies very strong lag-1 autocorrelation in the error terms of the linear model of interest. The Durbin-Watson test statistic on the residuals of the final linear model is much closer to 2 and implies a failure to reject the null hypothesis of no autocorrelation. We observe that the standard error of the estimated coefficient of the unemployed predictor has become slightly smaller, while the standard error of the estimated coefficient of the population variable has grown much larger, leading to it not being a significant predictor of the employed variable anymore.

```r
library(dplyr)
aux = lm(residuals ~ 0 + lag(residuals))
rho = aux$coefficients
summary(aux)
```

```
##
## Call:
## lm(formula = residuals ~ 0 + lag(residuals))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.07891 -0.24878 -0.07803  0.07175  1.07934
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## lag(residuals)   0.6907     0.2578    2.68    0.018 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.5201 on 14 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.339,  Adjusted R-squared:  0.2918
## F-statistic: 7.181 on 1 and 14 DF,  p-value: 0.01795
```

```
err = Inf
while (err > 1e-05) {
    YCO = Y - rho * lag(Y)
    XCO = X - rho * lag(X)
    CO = lm(YCO ~ 0 + XCO)
    residuals = Y - X %*% CO$coefficients
    aux = lm(residuals ~ 0 + lag(residuals))
    err = abs((rho - aux$coefficients)/rho)
    rho = aux$coefficients
}
summary(aux)
```

```
##
## Call:
## lm(formula = residuals ~ 0 + lag(residuals))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.63148 -0.30440 -0.09813  0.26493  0.92324
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## lag(residuals) 0.992208   0.001088     912   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4363 on 14 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 8.318e+05 on 1 and 14 DF,  p-value: < 2.2e-16
```

```
YCO = Y - rho * lag(Y)
XCO = X - rho * lag(X)
CO = lm(YCO ~ 0 + XCO)
dwtest(CO, alternative = "two.sided")
```

```
##
##  Durbin-Watson test
##
```

```
## data:  CO
## DW = 1.5172, p-value = 0.2528
## alternative hypothesis: true autocorrelation is not 0
```

```
summary(CO)
```

```
##
## Call:
## lm(formula = YCO ~ 0 + XCO)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.6325 -0.3054 -0.0992  0.2639  0.9222
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## XCO(Intercept) 171.546867  80.272558   2.137   0.0539 .
## XCOUnemployed   -0.012550   0.001367  -9.178 8.98e-07 ***
## XCOPopulation    0.008389   0.255047   0.033   0.9743
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4712 on 12 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.9374, Adjusted R-squared:  0.9217
## F-statistic: 59.88 on 3 and 12 DF,  p-value: 1.72e-07
```

Alternatively, we can utilize the cochrane.orcutt function from the orcutt package to perform a similar iterative procedure.

```
library(orcutt)
CO = cochrane.orcutt(fit, 5, 1000)
print(CO)
```

```
## Cochrane-orcutt estimation for first order autocorrelation
##
## Call:
## lm(formula = Employed ~ Unemployed + Population, data = longley)
##
##  number of interaction: 559
##  rho 0.992305
##
## Durbin-Watson statistic
## (original):    0.72573 , p-value: 2.389e-04
## (transformed): 1.51787 , p-value: 1.267e-01
##
```

```
##  coefficients:
## (Intercept)  Unemployed  Population
##   173.09296    -0.01255     0.00764
```

```
summary(CO)
```

```
## Call:
## lm(formula = Employed ~ Unemployed + Population, data = longley)
##
##                Estimate  Std. Error t value  Pr(>|t|)
## (Intercept) 173.0929618  80.9979272   2.137   0.05388 .
## Unemployed   -0.0125499   0.0013673  -9.179 8.967e-07 ***
## Population    0.0076405   0.2553010   0.030   0.97662
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4712 on 12 degrees of freedom
## Multiple R-squared:  0.8754 ,  Adjusted R-squared:  0.8546
## F-statistic: 42.1 on 2 and 12 DF,  p-value: < 3.744e-06
##
## Durbin-Watson statistic
## (original):    0.72573 , p-value: 2.389e-04
## (transformed): 1.51787 , p-value: 1.267e-01
```

Generalized Least Squares

Suppose that $Y = X\beta + \varepsilon$, where $X \in \mathbb{R}^{n\times p}$, $\varepsilon \sim \mathcal{N}_n\left(0, \sigma^2\Omega\right)$ and Ω is a positive definite matrix. Let $\Omega = LL^{\mathrm{T}}$ be the Cholesky decomposition of Ω, where L is a lower diagonal matrix. We define:

$$\widetilde{Y} = L^{-1}Y, \quad \widetilde{X} = L^{-1}X, \quad \widetilde{\varepsilon} = L^{-1}\varepsilon \sim \mathcal{N}_n\left(\mathbf{0}_n, \sigma^2 L^{-1}\Sigma L^{-\mathrm{T}}\right) \equiv \mathcal{N}_n\left(\mathbf{0}_n, \sigma^2 I_n\right).$$

Then, we observe that the transformed linear model $\widetilde{Y} = \widetilde{X}\beta + \widetilde{\varepsilon}$ has uncorrelated errors and can be fitted using ordinary least squares. The ordinary least squares estimator of the transformed linear model is equal to the generalized least squares estimator of the original linear model:

$$\widehat{\beta}_{\mathrm{GLS}} = \left(\widetilde{X}^{\mathrm{T}}\widetilde{X}\right)^{-1}\widetilde{X}^{\mathrm{T}}\widetilde{Y} = \left(X^{\mathrm{T}}L^{-\mathrm{T}}L^{-1}X\right)^{-1}X^{\mathrm{T}}L^{-\mathrm{T}}L^{-1}Y = \left(X^{\mathrm{T}}\Omega^{-1}X\right)^{-1}X^{\mathrm{T}}\Omega^{-1}Y.$$

We calculate that:

$$E\left(\widehat{\beta}_{\mathrm{GLS}}\right) = \cancel{\left(X^{\mathrm{T}}\Omega^{-1}X\right)^{-1}}\,\cancel{X^{\mathrm{T}}\Omega^{-1}X}\beta = \beta,$$

$$\mathrm{Var}\left(\widehat{\beta}_{\mathrm{GLS}}\right) = \sigma^2\left(X^{\mathrm{T}}\Omega^{-1}X\right)^{-1}X^{\mathrm{T}}\cancel{\Omega^{-1}}\cancel{\Omega}\Omega^{-1}X\left(X^{\mathrm{T}}\Omega^{-1}X\right)^{-1} = \sigma^2\left(X^{\mathrm{T}}\Omega^{-1}X\right)^{-1}.$$

Let $\widehat{Y} = X\widehat{\beta}_{\mathrm{GLS}}$ and $\widehat{\varepsilon} = Y - \widehat{Y}$. Then, the residual variance may be estimated as:

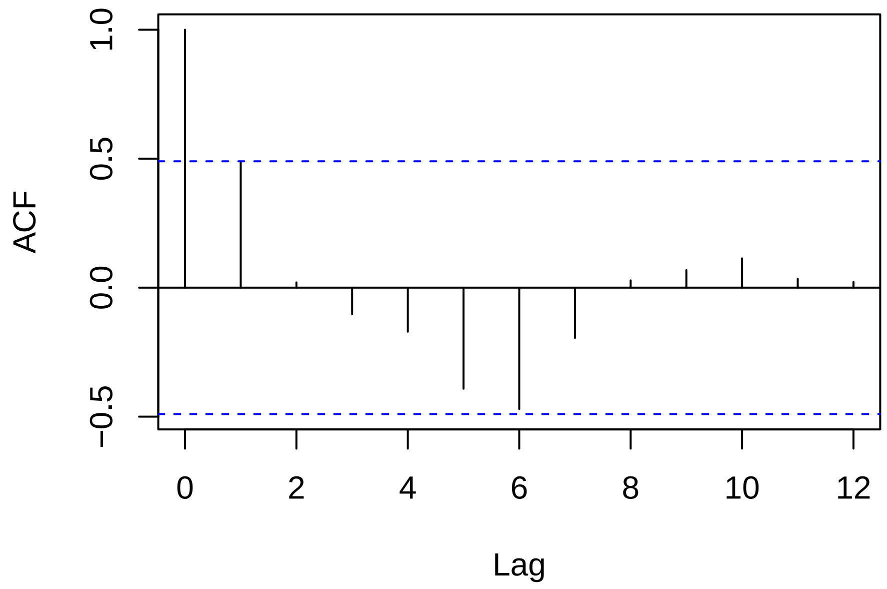
$$S^2 = \frac{1}{n-p}\widehat{\varepsilon}^{\mathrm{T}}\Omega^{-1}\widehat{\varepsilon}.$$

Now, we illustrate the generalized least squares method on the longley data set. We know that there's statistically significant lag-1 autocorrelation in the residuals from the longley data set. The effect of both the unemployed and the population predictors on the employed variable is estimated to be statistically significant.

```r
fit = lm(Employed ~ Unemployed + Population, longley)
residuals = fit$residuals
Y = fit$model[, 1]
X = model.matrix(fit)
n = dim(X)[1]
p = dim(X)[2]
summary(fit)
```

```
##
## Call:
## lm(formula = Employed ~ Unemployed + Population, data = longley)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2975 -0.3420 -0.1206  0.4006  1.3560
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.135325   3.487991  -0.039 0.969641
## Unemployed  -0.011151   0.002528  -4.410 0.000704 ***
## Population   0.587728   0.033967  17.303 2.34e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6654 on 13 degrees of freedom
## Multiple R-squared:  0.9689, Adjusted R-squared:  0.9641
## F-statistic: 202.5 on 2 and 13 DF,  p-value: 1.598e-10
```

```r
acf(residuals, main = NA)
```



The true covariance matrix Ω is obviously unknown and consists of too many elements to be estimated consistently, so it's usually parametrized in a specific manner which fits the data. In the presence of lag-1 autocorrelation in the response variable, the usual parametrization of $\Omega = [\Omega_{ij}]$ is $\Omega_{ij} = \rho^{|i-j|}$, where $\rho \in (-1, 1)$ is a parameter to be estimated from the data. One possible way to estimate it is to follow an iterative estimation procedure similar to iteratively reweighted least squares.

Algorithm 2.13 Feasible Generalized Least Squares

```
**Input**: Random sample $(Y, X)$.
1: We regress $Y$ on $X \in \mathbb{R}^{n\times p}$, including an intercept, and calculate the residual vector $\widehat{\varepsilon}$.
2: We regress the residual vector on the lagged residual vector without an intercept and extract the estimated
   slope coefficient $\rho$.
3: We iterate the following steps until convergence of $\rho$:
   i: We calculate $\Omega_{ij} = \rho^{|i-j|}$ for $i, j = 1, 2, \ldots, n$ and compute the Cholesky decomposition $\Omega = LL^{\mathrm{T}}$;
   ii: We define the auxiliary variables $Y_{\mathrm{GLS}} = L^{-1}Y$ and $X_{\mathrm{GLS}} = L^{-1}X$;
   iii: We regress $Y_{\mathrm{GLS}}$ on $X_{\mathrm{GLS}}$ without an intercept, since a transformed intercept is included in $X_{\mathrm{GLS}}$, and
   extract the least squares estimator $\widehat{\beta}_{\mathrm{GLS}}$;
   iv: We calculate a new residual vector $\widehat{\varepsilon} = Y - X\widehat{\beta}_{\mathrm{GLS}}$;
   v: We regress the new residual vector on the new lagged residual vector without an intercept and extract the
   estimated slope coefficient $\rho$.
**Output**: Generalized least squares estimate.
```

We apply the feasible generalized least squares method until the relative difference between 2 consecutive ρ estimates is less than 10^{-5}. We can see that the estimate of the coefficient ρ starts off at 0.69 and converges to 0.77, which implies strong lag-1 autocorrelation in the error terms of the linear model of interest. We observe that the estimated residual standard error of the final model is slightly higher than its ordinary least squares estimate. We observe that the autocorrelation plot of the left-multiplied residuals of the final model by L^{-1} displays no significant signs of autocorrelation.

```r
library(nlme)
aux = lm(residuals ~ 0 + lag(residuals))
rho = aux$coefficients
summary(aux)
```

```
##
## Call:
## lm(formula = residuals ~ 0 + lag(residuals))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.07891 -0.24878 -0.07803  0.07175  1.07934
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## lag(residuals)    0.6907     0.2578    2.68    0.018 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5201 on 14 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.339,  Adjusted R-squared:  0.2918
## F-statistic: 7.181 on 1 and 14 DF,  p-value: 0.01795
```

```r
err = Inf
while (err > 1e-05) {
    Omega = rho^abs(outer(1:n, 1:n, "-"))
    L = t(chol(Omega))
    YGLS = solve(L, Y)
    XGLS = solve(L, X)
    GLS = lm(YGLS ~ 0 + XGLS)
    residuals = Y - X %*% GLS$coefficients
    aux = lm(residuals ~ 0 + lag(residuals))
    err = abs((rho - aux$coefficients)/rho)
    rho = aux$coefficients
}
summary(aux)
```

```
##
## Call:
## lm(formula = residuals ~ 0 + lag(residuals))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.88763 -0.16066  0.05272  0.17683  1.12371
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## lag(residuals)   0.7748     0.1709   4.534 0.000468 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4848 on 14 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.5949, Adjusted R-squared:  0.5659
## F-statistic: 20.56 on 1 and 14 DF,  p-value: 0.000468
```

```
Omega = rho^abs(outer(1:n, 1:n, "-"))
L = t(chol(Omega))
betaGLS = drop(solve(crossprod(X, solve(Omega, X)), crossprod(X, solve(Omega,
    Y))))
residuals = drop(solve(L, Y - X %*% betaGLS))
summary(residuals)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1.403954 -0.329055 -0.005465  0.048974  0.269370  1.777351
```

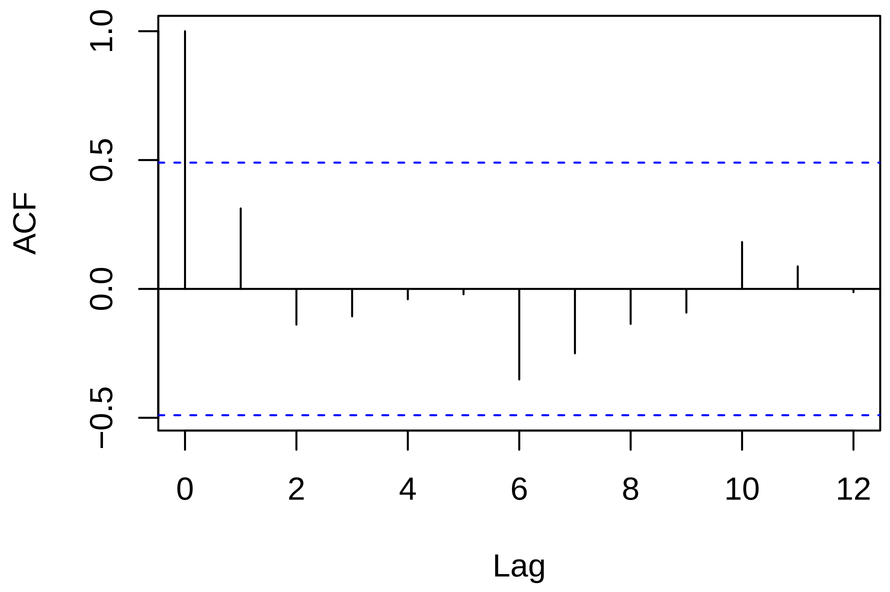
```
S = sqrt(sum(residuals^2)/(n - p))
varGLS = S^2 * solve(crossprod(X, solve(Omega, X)))
GLS = matrix(0, p, 4)
rownames(GLS) = names(fit$coefficients)
colnames(GLS) = colnames(summary(fit)$coef)
GLS[, 1] = betaGLS
GLS[, 2] = sqrt(diag(varGLS))
GLS[, 3] = GLS[, 1]/GLS[, 2]
GLS[, 4] = 2 * pt(abs(GLS[, 3]), n - p, lower.tail = FALSE)
print(GLS)
```

```
##                Estimate  Std. Error    t value     Pr(>|t|)
## (Intercept)  4.52230942 5.831980487  0.7754329 4.519598e-01
## Unemployed  -0.01241088 0.001638973 -7.5723514 4.059106e-06
## Population   0.54975749 0.050279433 10.9340431 6.324584e-08
```

```
print(S)
```

```
## [1] 0.8047888
```

```
acf(residuals, main = NA)
```



We can verify our calculations by using the gls function from the nlme package with the corAR1 correlation option and specifying a fixed value for the estimated ρ coefficient.

```
YGLS = solve(L, Y)
XGLS = solve(L, X)
GLS = lm(YGLS ~ 0 + XGLS)
summary(GLS)
```

```
##
## Call:
## lm(formula = YGLS ~ 0 + XGLS)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.40395 -0.32906 -0.00547  0.26937  1.77735
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## XGLS(Intercept)  4.522309   5.831980   0.775    0.452
## XGLSUnemployed  -0.012411   0.001639  -7.572 4.06e-06 ***
## XGLSPopulation   0.549757   0.050279  10.934 6.32e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8048 on 13 degrees of freedom
## Multiple R-squared:  0.9993, Adjusted R-squared:  0.9992
## F-statistic:  6430 on 3 and 13 DF,  p-value: < 2.2e-16
```

```
GLS = gls(Employed ~ Unemployed + Population, longley, corAR1(rho, ~Year, fixed = TRUE))
summary(GLS)
```

```
## Generalized least squares fit by REML
##   Model: Employed ~ Unemployed + Population
##   Data: longley
##        AIC      BIC    logLik
##   44.56469 46.82449 -18.28234
##
## Correlation Structure: AR(1)
##  Formula: ~Year
##  Parameter estimate(s):
##       Phi
## 0.7747749
##
## Coefficients:
##                  Value Std.Error   t-value p-value
## (Intercept)   4.522309  5.831980  0.775433   0.452
## Unemployed   -0.012411  0.001639 -7.572351   0.000
## Population    0.549757  0.050279 10.934043   0.000
##
##  Correlation:
##            (Intr) Unmply
## Unemployed  0.172
## Population -0.993 -0.258
##
## Standardized residuals:
##        Min         Q1        Med         Q3        Max
## -0.9400699 -0.4239052 -0.1805582  1.0158902  1.9410506
##
## Residual standard error: 0.8047888
## Degrees of freedom: 16 total; 13 residual
```

Otherwise, we could let the gls function estimate the autocorrelation parameter by itself, leading to an estimated value of 0.73, which is close to the one we have estimated ourselves with the previous algorithm.

```
GLS = gls(Employed ~ Unemployed + Population, longley, corAR1(form = ~Year),
    method = "ML")
summary(GLS)
```

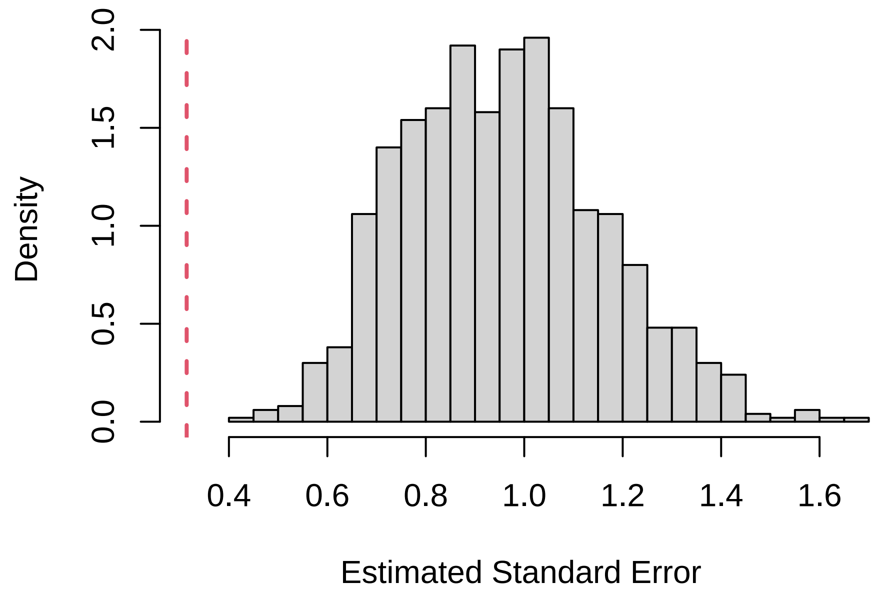
```
## Generalized least squares fit by maximum likelihood
##   Model: Employed ~ Unemployed + Population
##   Data: longley
##        AIC      BIC    logLik
##   31.32745 35.19039 -10.66372
##
## Correlation Structure: AR(1)
##  Formula: ~Year
##  Parameter estimate(s):
##       Phi
## 0.7318004
##
## Coefficients:
##                 Value Std.Error   t-value p-value
## (Intercept)  3.869311  5.409927  0.715224  0.4871
## Unemployed  -0.012371  0.001676 -7.381542  0.0000
## Population   0.555460  0.046895 11.844796  0.0000
##
##  Correlation:
##            (Intr) Unmply
## Unemployed  0.194
## Population -0.993 -0.287
##
## Standardized residuals:
##        Min         Q1        Med         Q3        Max
## -1.1216586 -0.4881036 -0.2007797  1.1147117  2.2618841
##
## Residual standard error: 0.6750361
## Degrees of freedom: 16 total; 13 residual
```

3 Experimental Design

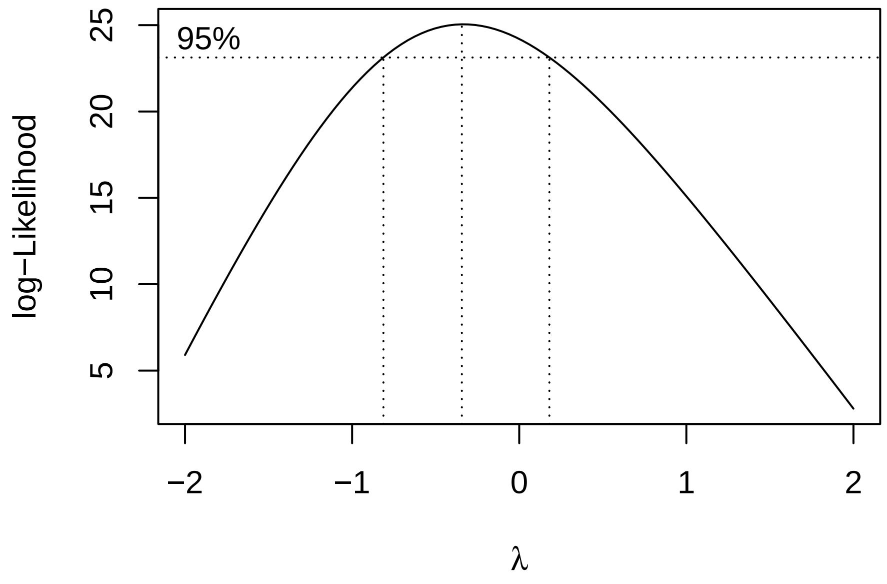
```
library(xtable)
n = 40
nblock = 10
nsim = 1000
block = factor(rep(1:nblock, each = n/nblock))
betasim = matrix(0, nsim, 4)
SE = matrix(0, nsim, 4)
for (i in 1:nsim) {
    epsilon = rnorm(n)
    u = rep(rnorm(nblock, 0, 3), each = n/nblock)
    treatment = is.element(1:n, sample(n, n/2))
    Y = 2 * treatment + u + epsilon
    fit = lm(Y ~ treatment)
    betasim[i, 1] = fit$coefficients[2]
    SE[i, 1] = summary(fit)$coefficients[2, 2]
    fit = lm(Y ~ treatment + block)
    betasim[i, 2] = fit$coefficients[2]
    SE[i, 2] = summary(fit)$coefficients[2, 2]
    treatment = numeric(n)
    for (b in 1:nblock) {
        treatment[block == b][sample(n/nblock, n/(2 * nblock))] = 1
    }
    Y = 2 * treatment + u + epsilon
    fit = lm(Y ~ treatment)
    betasim[i, 3] = fit$coefficients[2]
    SE[i, 3] = summary(fit)$coefficients[2, 2]
    fit = lm(Y ~ treatment + block)
    betasim[i, 4] = fit$coefficients[2]
    SE[i, 4] = summary(fit)$coefficients[2, 2]
}
sim = matrix(0, 4, 3)
sim[, 1] = apply(betasim, 2, median)
sim[, 2] = apply(SE, 2, median)
sim[, 3] = apply(betasim, 2, sd)
rownames(sim) = c("Complete Randomization Excluding Block", "Complete Randomization Including Block",
    "Block Randomization Excluding Block", "Block Randomization Including Block")
colnames(sim) = c("Median Coefficient", "Median S.E.", "Estimated S.E.")
print(xtable(sim), comment = FALSE)
```

	Median Coefficient	Median S.E.	Estimated S.E.
Complete Randomization Excluding Block	1.97	0.95	0.96
Complete Randomization Including Block	2.00	0.36	0.37
Block Randomization Excluding Block	1.99	0.95	0.31
Block Randomization Including Block	1.99	0.32	0.31

```
hist(SE[, 3], "FD", freq = FALSE, main = NA, xlim = c(sim[3, 3], max(SE[, 3])),
    xlab = "Estimated Standard Error")
abline(v = sim[3, 3], col = 2, lty = 2, lwd = 2)
```



```
time = c(6.4, 10.9, 9.8, 7.5, 4.6, 4.9, 6.8, 6.2, 7.9, 6, 4, 4.2, 12.7, 13.4,
    12.5, 7.3, 6.1, 7.4, 8.8, 10.2, 12.5, 8.6, 6.1, 5.6, 7.4, 10, 8.3, 6.4,
    4.3, 5.6, 13.1, 12, 12, 11.3, 6.1, 9.7)
saw = factor(c("F", "E", "D", "B", "A", "C", "B", "C", "E", "A", "D", "F", "E",
    "A", "B", "C", "F", "D", "C", "D", "A", "F", "E", "B", "D", "B", "F", "E",
    "C", "A", "A", "F", "C", "D", "B", "E"))
brand = factor(saw, labels = rep(1:3, 2))
species = factor(rep(c(rep("spruce", 6), rep("pine", 6), rep("larch", 6)), 2))
bark = factor(c(rep("no", 18), rep("yes", 18)))
team = factor(rep(c("I", "II", "III", "IV", "V", "VI"), 6))

library(MASS)
boxcox(lm(time ~ species + bark + brand + team))
```



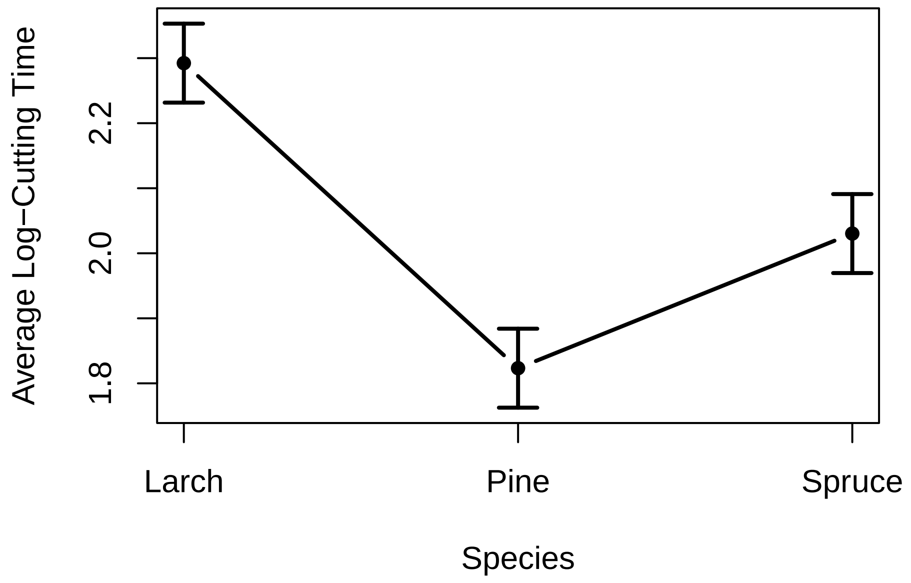
```r
fixed = lm(log(time) ~ species + bark + brand + team)
sigmahat = summary(fixed)$sigma
anova(fixed)
```

```
## Analysis of Variance Table
##
## Response: log(time)
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## species    2 1.32602 0.66301 63.5820 1.568e-10 ***
## bark       1 0.19832 0.19832 19.0188 0.0001952 ***
## brand      2 0.18354 0.09177  8.8007 0.0012778 **
## team       5 2.46624 0.49325 47.3021 6.010e-12 ***
## Residuals 25 0.26069 0.01043
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

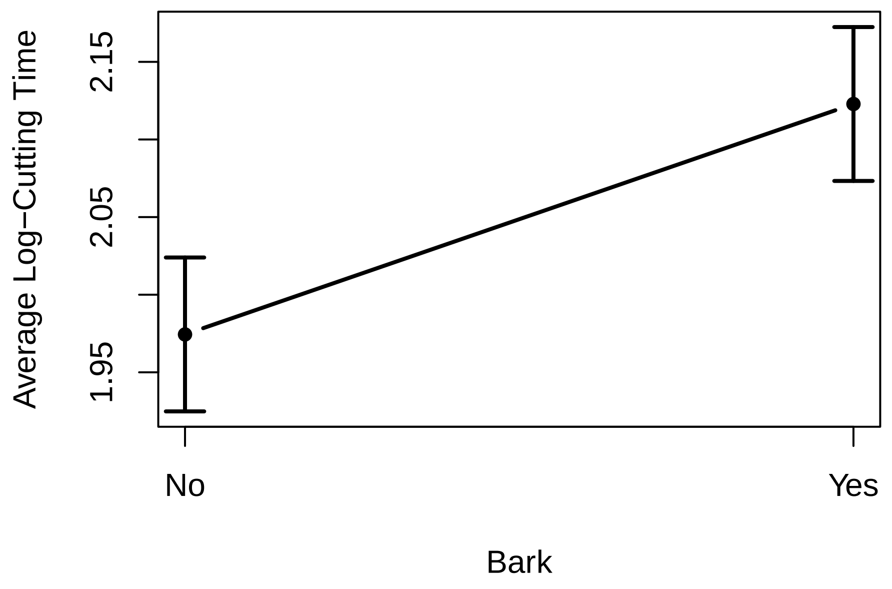
```r
summary(fixed)
```

```
##
## Call:
## lm(formula = log(time) ~ species + bark + brand + team)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.195730 -0.062530  0.005866  0.060520  0.158912
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    2.394203   0.056447  42.415  < 2e-16 ***
## speciespine   -0.469033   0.041689 -11.251 2.82e-11 ***
## speciesspruce -0.262046   0.041689  -6.286 1.41e-06 ***
## barkyes        0.148444   0.034039   4.361 0.000195 ***
## brand2         0.006011   0.041689   0.144 0.886511
## brand3        -0.148373   0.041689  -3.559 0.001522 **
## teamII         0.142463   0.058956   2.416 0.023305 *
## teamIII        0.156366   0.058956   2.652 0.013686 *
## teamIV        -0.139378   0.058956  -2.364 0.026153 *
## teamV         -0.544639   0.058956  -9.238 1.55e-09 ***
## teamVI        -0.386701   0.058956  -6.559 7.17e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1021 on 25 degrees of freedom
## Multiple R-squared:  0.9412, Adjusted R-squared:  0.9177
## F-statistic: 40.03 on 10 and 25 DF,  p-value: 6.883e-13
```

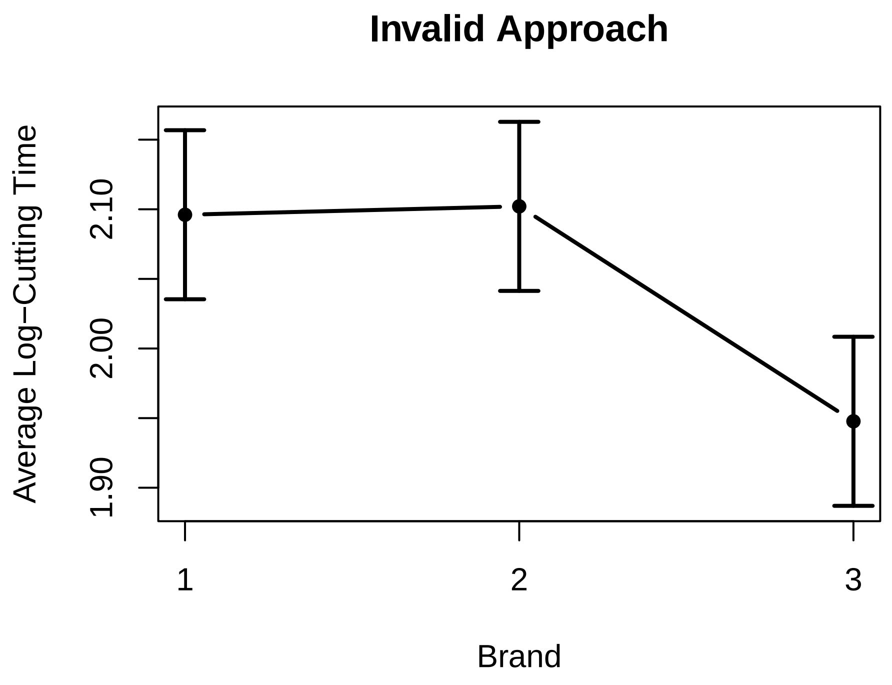
```
meanspecies = aggregate(log(time) ~ species, FUN = mean)[, 2]
CIspecies = cbind(meanspecies - qt(0.975, fixed$df.residual) * sigmahat/sqrt(12),
    meanspecies + qt(0.975, fixed$df.residual) * sigmahat/sqrt(12))
plot(meanspecies, type = "b", ylim = range(CIspecies), xlab = "Species", ylab = "Average Log-Cutting Time'
    xaxt = "n", pch = 16, lwd = 2)
arrows(1:3, CIspecies[, 1], 1:3, CIspecies[, 2], 0.1, 90, 3, lwd = 2)
axis(1, 1:3, c("Larch", "Pine", "Spruce"))
```



```
meanbark = aggregate(log(time) ~ bark, FUN = mean)[, 2]
CIbark = cbind(meanbark - qt(0.975, fixed$df.residual) * sigmahat/sqrt(18),
    meanbark + qt(0.975, fixed$df.residual) * sigmahat/sqrt(18))
plot(meanbark, type = "b", ylim = range(CIbark), xlab = "Bark", ylab = "Average Log-Cutting Time",
    xaxt = "n", pch = 16, lwd = 2)
arrows(1:2, CIbark[, 1], 1:2, CIbark[, 2], 0.1, 90, 3, lwd = 2)
axis(1, 1:2, c("No", "Yes"))
```



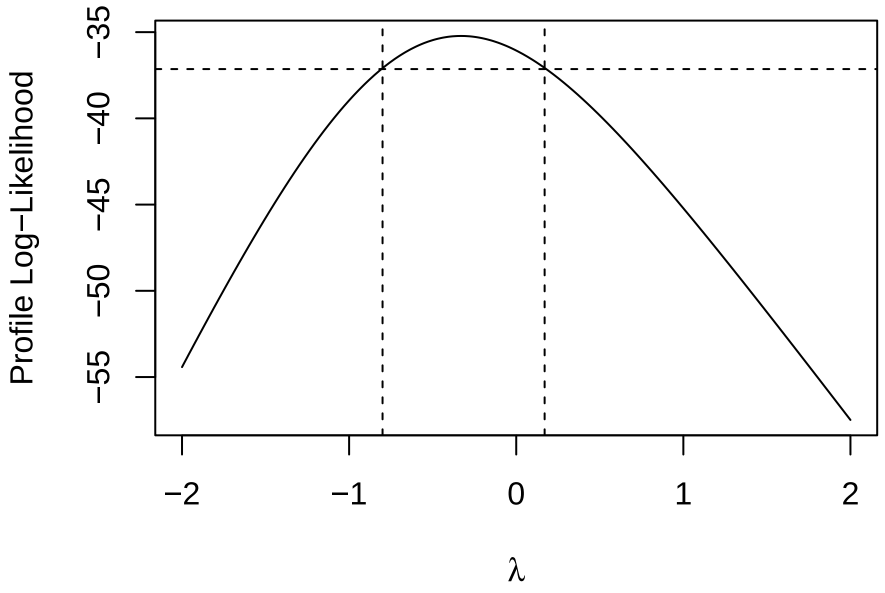
```r
meanbrand = aggregate(log(time) ~ brand, FUN = mean)[, 2]
CIbrand = cbind(meanbrand - qt(0.975, fixed$df.residual) * sigmahat/sqrt(12),
    meanbrand + qt(0.975, fixed$df.residual) * sigmahat/sqrt(12))
plot(meanbrand, type = "b", ylim = range(CIbrand), main = "Invalid Approach",
    xlab = "Brand", ylab = "Average Log-Cutting Time", xaxt = "n", pch = 16,
    lwd = 2)
arrows(1:3, CIbrand[, 1], 1:3, CIbrand[, 2], 0.1, 90, 3, lwd = 2)
axis(1, 1:3, c("1", "2", "3"))
```



```r
library(lme4)
lambda = seq(-2, 2, 0.01)
loglik = numeric(401)
for (i in 1:401) {
    if (lambda[i] == 0) {
        Ypower = log(time)
    } else {
        Ypower = (time^lambda[i] - 1)/lambda[i]
    }
    power = lmer(Ypower ~ species + bark + brand + team + (1 | saw), REML = FALSE)
    loglik[i] = (lambda[i] - 1) * sum(log(time)) + logLik(power)[1]
}
CI = range(lambda[loglik > max(loglik) - qchisq(0.95, 1)/2])
print(CI)
```

```
## [1] -0.80  0.17
```

```
plot(lambda, loglik, "l", xlab = expression(lambda), ylab = "Profile Log-Likelihood")
abline(h = max(loglik) - qchisq(0.95, 1)/2, lty = 2)
abline(v = CI, lty = 2)
```



```
mixed = lmer(log(time) ~ species + bark + brand + team + (1 | saw), REML = FALSE)
LR = -2 * (logLik(fixed)[1] - summary(mixed)$logLik[1])
pchisq(LR, 1, lower.tail = FALSE)
```

```
## [1] 0.7236955
```

```
mixed = lmer(log(time) ~ species + bark + brand + team + (1 | saw))
sigmahat = as.data.frame(summary(mixed)$varcor)$sdcor
names(sigmahat) = as.data.frame(summary(mixed)$varcor)$grp
print(sigmahat^2/sum(sigmahat^2))
```

```
##       saw  Residual
## 0.1208102 0.8791898
```

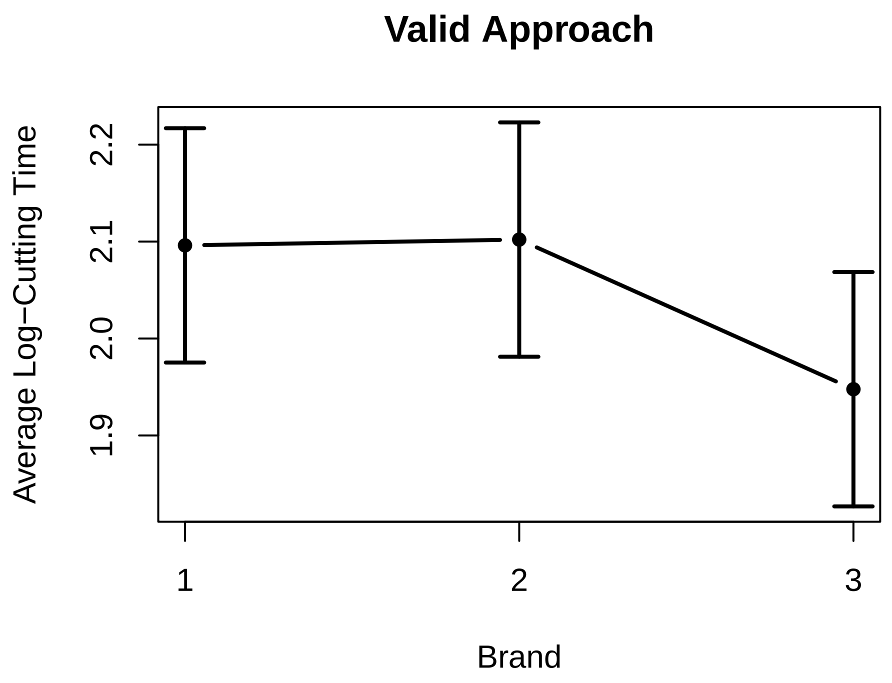
```r
anova(mixed)
```

```
## Analysis of Variance Table
##         npar  Sum Sq Mean Sq F value
## species    2 1.32602 0.66301  69.873
## bark       1 0.19832 0.19832  20.900
## brand      2 0.10060 0.05030   5.301
## team       5 2.46624 0.49325  51.982
```

```r
summary(mixed, correlation = FALSE)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: log(time) ~ species + bark + brand + team + (1 | saw)
##
## REML criterion at convergence: -23
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.7609 -0.6291  0.1416  0.5010  1.5028
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  saw      (Intercept) 0.001304 0.03611
##  Residual             0.009489 0.09741
## Number of obs: 36, groups:  saw, 6
##
## Fixed effects:
##                 Estimate Std. Error t value
## (Intercept)     2.394203   0.059593  40.176
## speciespine    -0.469033   0.039768 -11.794
## speciesspruce -0.262046   0.039768  -6.589
## barkyes         0.148444   0.032470   4.572
## brand2          0.006011   0.053715   0.112
## brand3         -0.148373   0.053715  -2.762
## teamII          0.142463   0.056240   2.533
## teamIII         0.156366   0.056240   2.780
## teamIV         -0.139378   0.056240  -2.478
## teamV          -0.544639   0.056240  -9.684
## teamVI         -0.386701   0.056240  -6.876
```

```r
SEbrand = sqrt(sigmahat[1]^2/2 + sigmahat[2]^2/12)
CIbrand = cbind(meanbrand - qt(0.975, 3) * SEbrand, meanbrand + qt(0.975, 3) *
    SEbrand)
plot(meanbrand, type = "b", ylim = range(CIbrand), main = "Valid Approach",
    xlab = "Brand", ylab = "Average Log-Cutting Time", xaxt = "n", pch = 16,
    lwd = 2)
arrows(1:3, CIbrand[, 1], 1:3, CIbrand[, 2], 0.1, 90, 3, lwd = 2)
axis(1, 1:3, c("1", "2", "3"))
```



```
mixed = lmer(log(time) ~ species + bark + brand + (1 | team) + (1 | saw), REML = FALSE)
interact = lmer(log(time) ~ species * bark + brand + (1 | team) + (1 | saw),
    REML = FALSE)
LR = -2 * (summary(mixed)$logLik[1] - summary(interact)$logLik[1])
pchisq(LR, 2, lower.tail = FALSE)
```

```
## [1] 0.4927374
```

```
interact = lmer(log(time) ~ species + bark * brand + (1 | team) + (1 | saw),
    REML = FALSE)
LR = -2 * (summary(mixed)$logLik[1] - summary(interact)$logLik[1])
pchisq(LR, 2, lower.tail = FALSE)
```

```
## [1] 0.05753004
```

```
interact = lmer(log(time) ~ bark + species * brand + (1 | team) + (1 | saw),
    REML = FALSE)
LR = -2 * (summary(mixed)$logLik[1] - summary(interact)$logLik[1])
pchisq(LR, 4, lower.tail = FALSE)
```

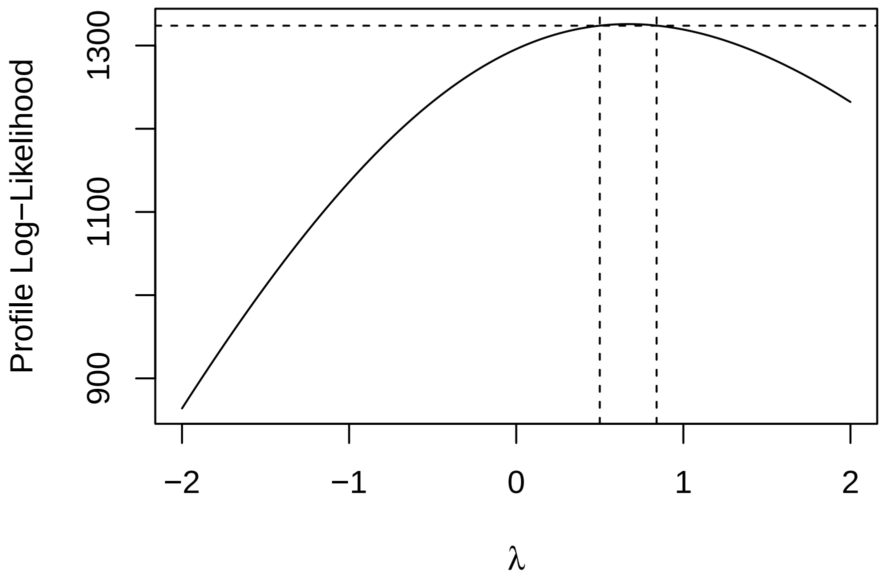
```
## [1] 0.8050065
```

```
boreal = read.csv("Reich2018NaturePaperDataAug2018.csv")[, c(1:5, 8:10, 13)]
boreal = boreal[!(boreal$plot_id %in% c("d1", "l1(2)")) & boreal$Asat > 0, ]
boreal$year = as.character(boreal$year)
boreal$doy = as.character(boreal$doy)
```

```r
boreal$day = paste0(boreal$year, boreal$doy)

library(lme4)
soil = boreal[match(data.frame(t(unique(boreal[, c(4, 10)]))), data.frame(t(boreal[,
    c(4, 10)]))), ]
lambda = seq(-2, 2, 0.01)
loglik = numeric(401)
for (i in 1:401) {
    if (lambda[i] == 0) {
        Ypower = log(soil$soil_water_VWC)
    } else {
        Ypower = (soil$soil_water_VWC^lambda[i] - 1)/lambda[i]
    }
    power = lmer(Ypower ~ warming_treatment + site + year + (1 | plot_id) +
        (1 | day), soil, REML = FALSE, lmerControl(optimizer = "Nelder_Mead"))
    loglik[i] = (lambda[i] - 1) * sum(log(soil$soil_water_VWC), na.rm = TRUE) +
        logLik(power)[1]
}
CI = range(lambda[loglik > max(loglik) - qchisq(0.95, 1)/2])
print(CI)
```

```
## [1] 0.50 0.84
```

```r
plot(lambda, loglik, "l", xlab = expression(lambda), ylab = "Profile Log-Likelihood")
abline(h = max(loglik) - qchisq(0.95, 1)/2, lty = 2)
abline(v = CI, lty = 2)
```



```r
fit = lmer(sqrt(soil_water_VWC) ~ warming_treatment + site + year + (1 | plot_id) +
    (1 | day), soil)
sigmahat = as.data.frame(summary(fit)$varcor)$sdcor
```

```r
names(sigmahat) = as.data.frame(summary(fit)$varcor)$grp
print(sigmahat^2/sum(sigmahat^2))
```

```
##       day   plot_id  Residual
## 0.6861437 0.1894802 0.1243760
```

```r
anova(fit)
```

```
## Analysis of Variance Table
##                   npar    Sum Sq   Mean Sq F value
## warming_treatment    1 0.0067888 0.0067888 22.1850
## site                 1 0.0104774 0.0104774 34.2393
## year                 2 0.0035804 0.0017902  5.8502
```

```r
summary(fit, correlation = FALSE)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: sqrt(soil_water_VWC) ~ warming_treatment + site + year + (1 |
##     plot_id) + (1 | day)
##    Data: soil
##
## REML criterion at convergence: -2328.9
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.71575 -0.60711 -0.05245  0.62072  3.05704
##
## Random effects:
##  Groups   Name        Variance  Std.Dev.
##  day      (Intercept) 0.0016881 0.04109
##  plot_id  (Intercept) 0.0004662 0.02159
##  Residual             0.0003060 0.01749
## Number of obs: 501, groups:  day, 49; plot_id, 24
##
## Fixed effects:
##                         Estimate Std. Error t value
## (Intercept)              0.43526    0.01368  31.815
## warming_treatmentwarmed -0.03834    0.00811  -4.727
## sitehwrc                -0.08486    0.01480  -5.734
## year2010                 0.03947    0.01473   2.680
## year2011                -0.00820    0.01423  -0.576
```

```r
fit = lmer(sqrt(soil_water_VWC) ~ warming_treatment + site + year + (1 | plot_id) +
    (1 | day), soil, REML = FALSE)
interact = lmer(sqrt(soil_water_VWC) ~ warming_treatment * site + year + (1 |
    plot_id) + (1 | day), soil, REML = FALSE)
```

```r
LR = -2 * (summary(fit)$logLik[1] - summary(interact)$logLik[1])
pchisq(LR, 1, lower.tail = FALSE)
```

```
## [1] 0.003921968
```

```r
interact = lmer(sqrt(soil_water_VWC) ~ warming_treatment * year + site + (1 |
    plot_id) + (1 | day), soil, REML = FALSE)
LR = -2 * (summary(fit)$logLik[1] - summary(interact)$logLik[1])
pchisq(LR, 2, lower.tail = FALSE)
```

```
## [1] 1.992776e-07
```

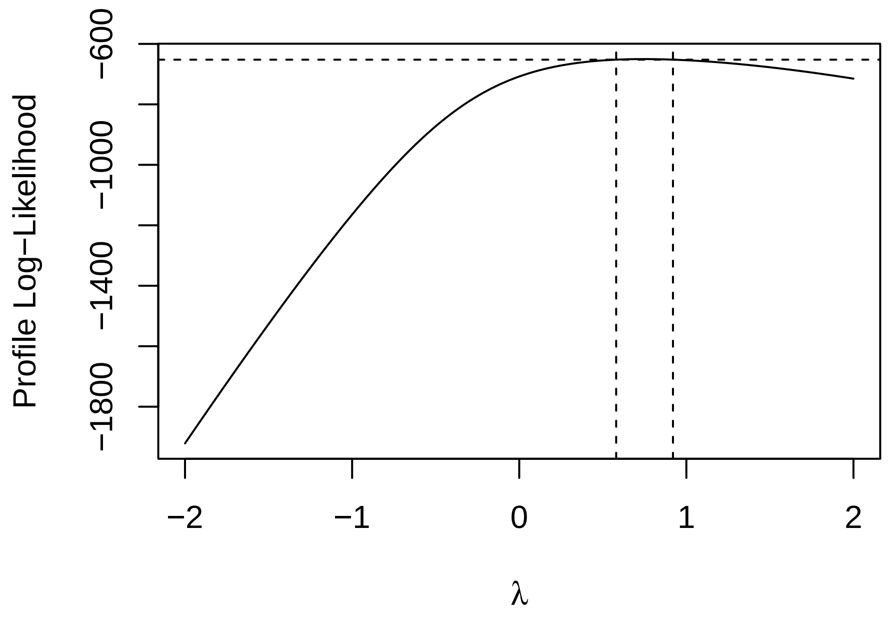
```r
interact = lmer(sqrt(soil_water_VWC) ~ warming_treatment + site * year + (1 |
    plot_id) + (1 | day), soil, REML = FALSE)
LR = -2 * (summary(fit)$logLik[1] - summary(interact)$logLik[1])
pchisq(LR, 4, lower.tail = FALSE)
```

```
## [1] 0.0009699407
```

```r
queru = boreal[boreal$species == "queru", ]
lambda = seq(-2, 2, 0.01)
loglik = numeric(401)
for (i in 1:401) {
    if (lambda[i] == 0) {
        Ypower = log(queru$Asat)
    } else {
        Ypower = (queru$Asat^lambda[i] - 1)/lambda[i]
    }
    power = lmer(Ypower ~ sqrt(soil_water_VWC) + warming_treatment + year +
        (1 | plot_id) + (1 | day), queru, REML = FALSE)
    loglik[i] = (lambda[i] - 1) * sum(log(queru$Asat)) + logLik(power)[1]
}
CI = range(lambda[loglik > max(loglik) - qchisq(0.95, 1)/2])
print(CI)
```

```
## [1] 0.58 0.92
```

```r
plot(lambda, loglik, "l", xlab = expression(lambda), ylab = "Profile Log-Likelihood")
abline(h = max(loglik) - qchisq(0.95, 1)/2, lty = 2)
abline(v = CI, lty = 2)
```



```
fit = lmer(Asat ~ soil_water_VWC + warming_treatment + year + (1 | plot_id) +
    (1 | day), queru)
sigmahat = as.data.frame(summary(fit)$varcor)$sdcor
names(sigmahat) = as.data.frame(summary(fit)$varcor)$grp
print(sigmahat^2/sum(sigmahat^2))
```

```
##       day   plot_id  Residual
## 0.3909222 0.1426298 0.4664480
```

```
anova(fit)
```

```
## Analysis of Variance Table
##                   npar Sum Sq Mean Sq F value
## soil_water_VWC       1 27.952  27.952  2.9695
## warming_treatment    1 49.690  49.690  5.2787
## year                 2 64.830  32.415  3.4436
```

```
summary(fit, correlation = FALSE)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: Asat ~ soil_water_VWC + warming_treatment + year + (1 | plot_id) +
##     (1 | day)
##    Data: queru
##
## REML criterion at convergence: 1295.9
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.3384 -0.6761 -0.0618  0.6042  2.8008
##
## Random effects:
```

```
##  Groups   Name        Variance Std.Dev.
##  day      (Intercept) 7.889    2.809
##  plot_id  (Intercept) 2.878    1.697
##  Residual             9.413    3.068
## Number of obs: 241, groups:  day, 31; plot_id, 24
##
## Fixed effects:
##                           Estimate Std. Error t value
## (Intercept)                 6.1059     2.0649   2.957
## soil_water_VWC             17.0339    10.0635   1.693
## warming_treatmentwarmed     1.7488     0.8869   1.972
## year2010                    3.4977     1.5805   2.213
## year2011                    0.4586     1.4596   0.314
```

```
fit = lmer(Asat ~ sqrt(soil_water_VWC) + warming_treatment + year + (1 | plot_id) +
    (1 | day), queru, REML = FALSE)
interact = lmer(Asat ~ sqrt(soil_water_VWC) * warming_treatment + year + (1 |
    plot_id) + (1 | day), queru, REML = FALSE)
LR = -2 * (summary(fit)$logLik[1] - summary(interact)$logLik[1])
pchisq(LR, 1, lower.tail = FALSE)
```

```
## [1] 6.398453e-05
```

```
interact = lmer(Asat ~ sqrt(soil_water_VWC) + warming_treatment * year + (1 |
    plot_id) + (1 | day), queru, REML = FALSE)
LR = -2 * (summary(fit)$logLik[1] - summary(interact)$logLik[1])
pchisq(LR, 2, lower.tail = FALSE)
```

```
## [1] 0.0470728
```

```
interact = lmer(Asat ~ sqrt(soil_water_VWC) * year + warming_treatment + (1 |
    plot_id) + (1 | day), queru, REML = FALSE)
LR = -2 * (summary(fit)$logLik[1] - summary(interact)$logLik[1])
pchisq(LR, 4, lower.tail = FALSE)
```

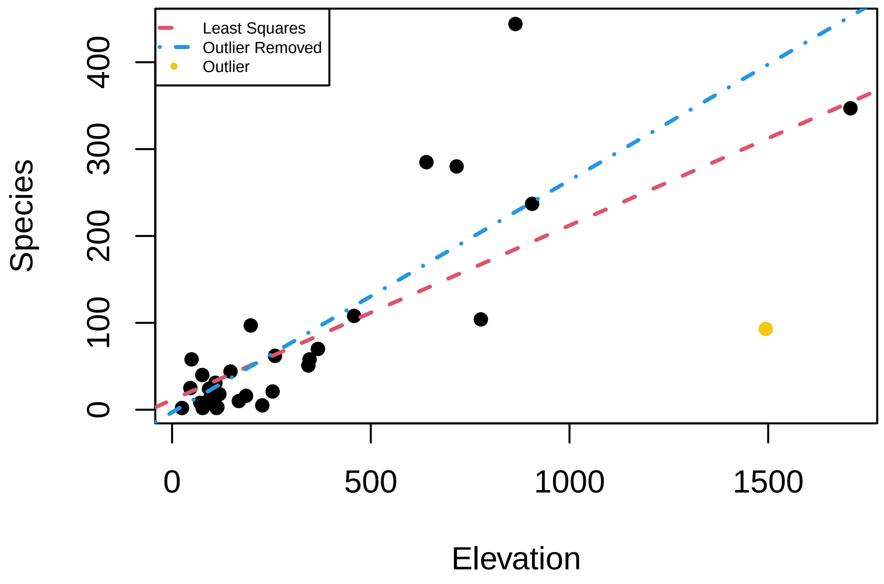
```
## [1] 0.09448143
```

4 Robust Regression

```
library(faraway)
library(xtable)
fit = lm(Species ~ Elevation, gala)
print(xtable(summary(fit)), comment = FALSE)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	11.3351	19.2053	0.59	0.5598
Elevation	0.2008	0.0346	5.80	0.0000

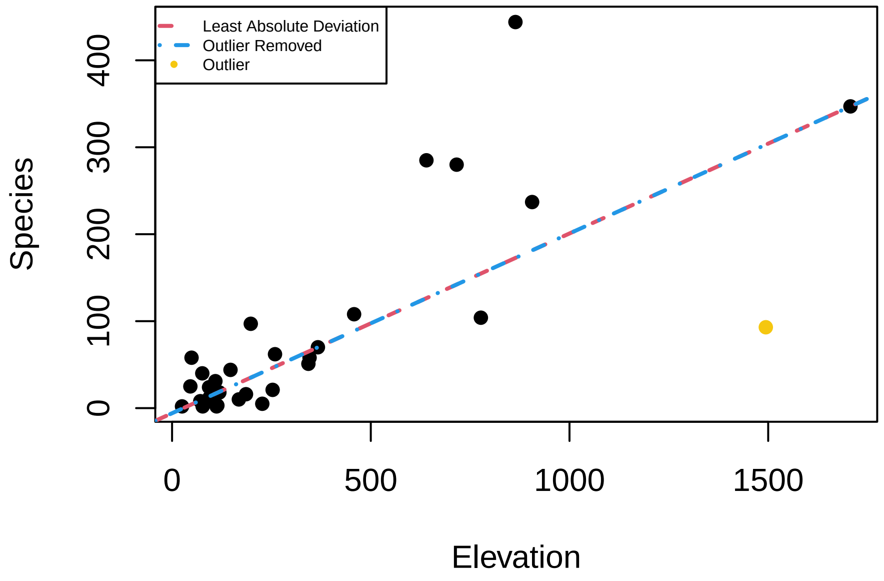
```r
outlier = cooks.distance(fit) == max(cooks.distance(fit))
removed = lm(Species ~ Elevation, gala, !outlier)
plot(Species ~ Elevation, gala, col = 6 * outlier + 1, pch = 16)
abline(fit, col = 2, lty = 2, lwd = 2)
abline(removed, col = 4, lty = 4, lwd = 2)
legend("topleft", c("Least Squares", "Outlier Removed", "Outlier"), col = c(2,
    4, 7), lty = c(2, 4, NA), lwd = c(2, 2, NA), pch = c(NA, NA, 16), cex = 0.5)
```



```r
library(quantreg)
fit = rq(Species ~ Elevation, gala, tau = 0.5)
print(xtable(summary(fit)$coefficients), comment = FALSE)
```

	coefficients	lower bd	upper bd
(Intercept)	-5.86	-9.78	9.75
Elevation	0.21	0.11	0.28

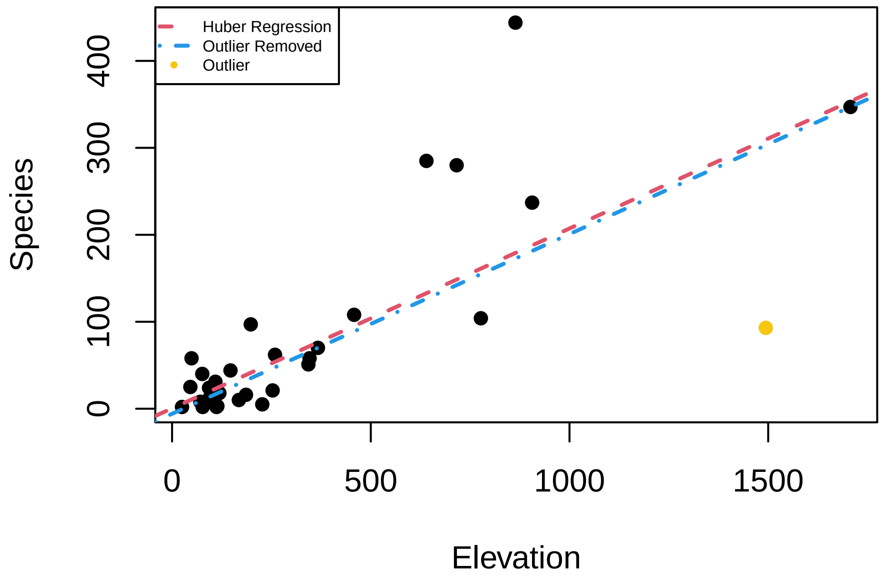
```r
removed = rq(Species ~ Elevation, gala, !outlier, tau = 0.5)
plot(Species ~ Elevation, gala, col = 6 * outlier + 1, pch = 16)
abline(fit, col = 2, lty = 2, lwd = 2)
abline(removed, col = 4, lty = 4, lwd = 2)
legend("topleft", c("Least Absolute Deviation", "Outlier Removed", "Outlier"),
    col = c(2, 4, 7), lty = c(2, 4, NA), lwd = c(2, 2, NA), pch = c(NA, NA,
        16), cex = 0.5)
```



```
library(MASS)
fit = rlm(Species ~ Elevation, data = gala)
print(xtable(summary(fit)$coefficients), comment = FALSE)
```

	Value	Std. Error	t value
(Intercept)	0.39	9.76	0.04
Elevation	0.21	0.02	11.75

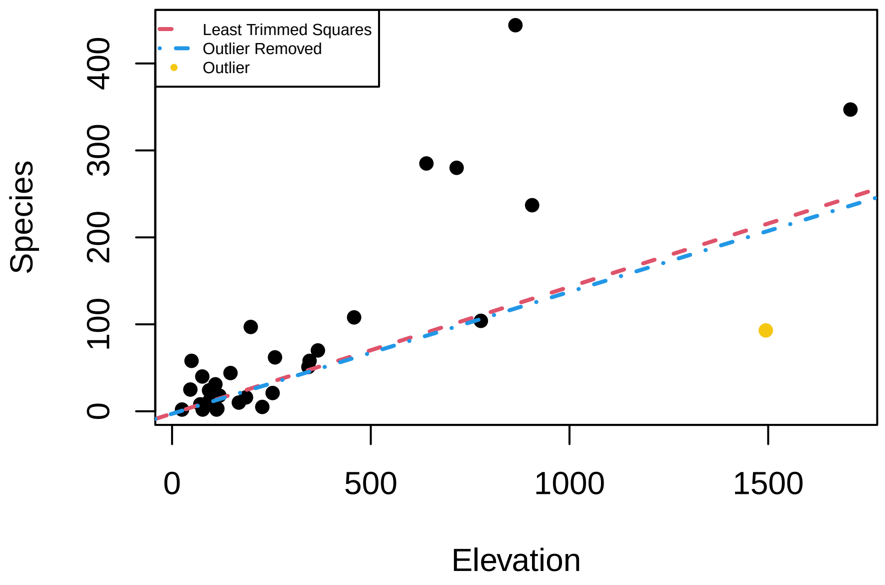
```
removed = rq(Species ~ Elevation, data = gala, subset = !outlier)
plot(Species ~ Elevation, gala, col = 6 * outlier + 1, pch = 16)
abline(fit, col = 2, lty = 2, lwd = 2)
abline(removed, col = 4, lty = 4, lwd = 2)
legend("topleft", c("Huber Regression", "Outlier Removed", "Outlier"), col = c(2,
    4, 7), lty = c(2, 4, NA), lwd = c(2, 2, NA), pch = c(NA, NA, 16), cex = 0.5)
```



```r
fit = lqs(Species ~ Elevation, gala)
print(xtable(data.frame(fit$coefficients)), comment = FALSE)
```

	fit.coefficients
(Intercept)	-2.63
Elevation	0.15

```r
removed = lqs(Species ~ Elevation, gala, subset = !outlier)
plot(Species ~ Elevation, gala, col = 6 * outlier + 1, pch = 16)
abline(fit, col = 2, lty = 2, lwd = 2)
abline(removed, col = 4, lty = 4, lwd = 2)
legend("topleft", c("Least Trimmed Squares", "Outlier Removed", "Outlier"),
    col = c(2, 4, 7), lty = c(2, 4, NA), lwd = c(2, 2, NA), pch = c(NA, NA,
        16), cex = 0.5)
```



```r
library(faraway)
n = dim(gala)[1]
fit = lm(Species ~ Elevation, gala)
outliers = cooks.distance(fit) > 1
fit = lm(Species ~ Elevation, gala, !outliers)
while (sum(cooks.distance(fit) > 1) > 0) {
    outliers[!outliers] = cooks.distance(fit) > 1
    fit = lm(Species ~ Elevation, gala, !outliers)
}

library(quantreg)
library(MASS)
library(xtable)
nboot = 10000
```

```r
betaboot = matrix(0, nboot, 5)
for (i in 1:nboot) {
    fit = lm(Species ~ Elevation, gala, sample(n, replace = TRUE))
    betaboot[i, 1] = fit$coefficients[1]
    fit = lm(Species ~ Elevation, gala, sample((1:n)[!outliers], replace = TRUE))
    betaboot[i, 2] = fit$coefficients[1]
    fit = rq(Species ~ Elevation, gala, sample(n, replace = TRUE), tau = 0.5)
    betaboot[i, 3] = fit$coefficients[1]
    fit = rlm(Species ~ Elevation, data = gala, subset = sample(n, replace = TRUE),
        maxit = 1000)
    betaboot[i, 4] = fit$coefficients[1]
    fit = lqs(Species ~ Elevation, gala, subset = sample(n, replace = TRUE))
    betaboot[i, 5] = fit$coefficients[1]
}
boot = matrix(0, 5, 2)
boot[, 1] = apply(betaboot, 2, median)
boot[, 2] = apply(betaboot, 2, sd)
rownames(boot) = c("Least Squares", "Outliers Removed", "Least Absolute Deviation",
    "Huber Regression", "Least Trimmed Squares")
colnames(boot) = c("Bootstrap Coefficient", "Bootstrap Standard Error")
print(xtable(boot), comment = FALSE)
```

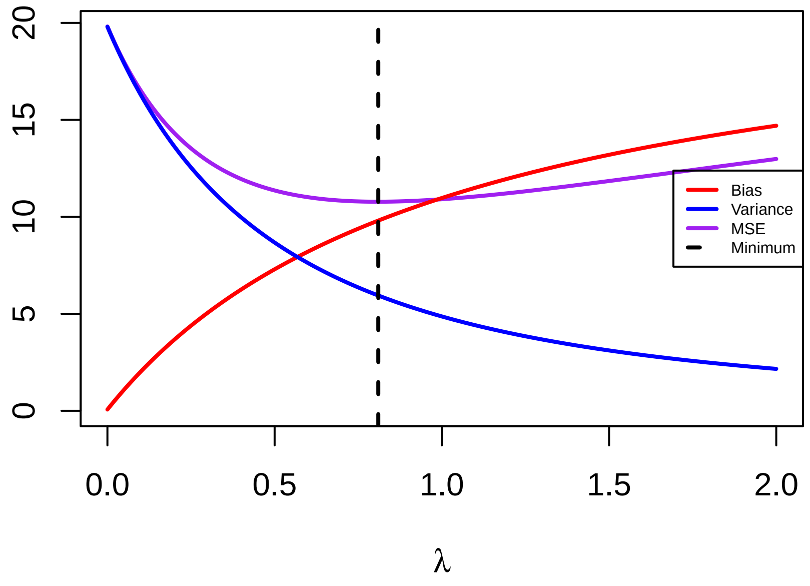
	Bootstrap Coefficient	Bootstrap Standard Error
Least Squares	9.64	15.93
Outliers Removed	-10.63	10.93
Least Absolute Deviation	-5.16	12.29
Huber Regression	-0.07	13.29
Least Trimmed Squares	-2.24	14.40

5 Penalized Regression

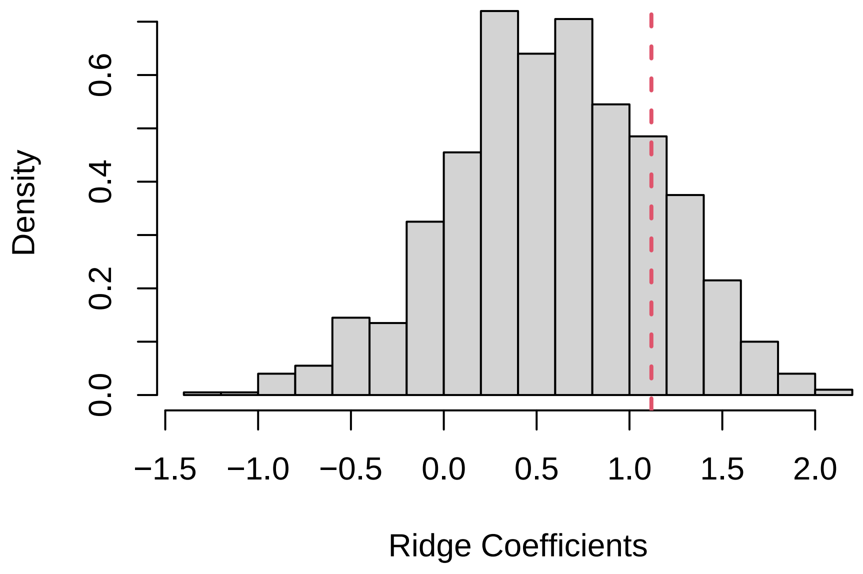
Ridge Regression

```r
n = 1000
p = 20
beta = rep(5/sqrt(p), p)
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
nsim = 1000
Y = drop(X %*% beta) + matrix(rnorm(n * nsim), n)
Y = t(t(Y)/apply(Y, 2, sd))/sqrt(1 - 1/n)
lambda = seq(0, 2, 0.01)
Bias = numeric(201)
```

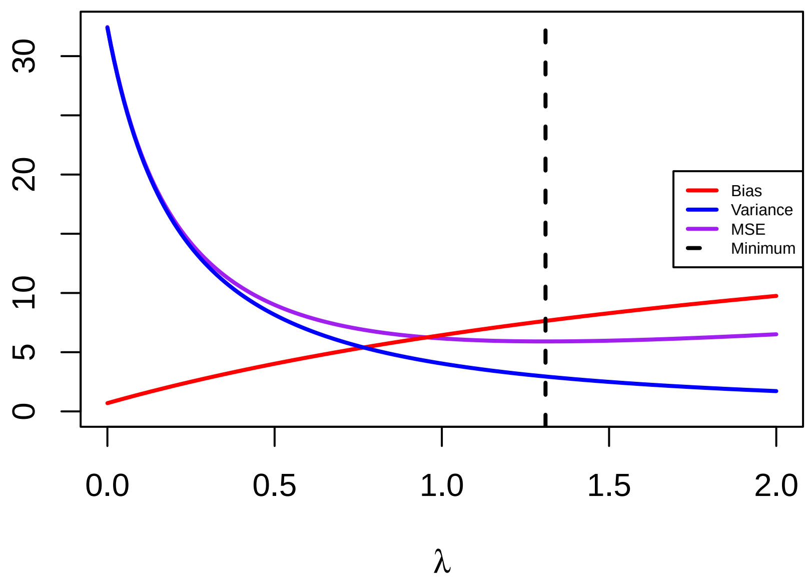
```
Var = numeric(201)
MSE = numeric(201)
for (k in 1:201) {
    betaridge = solve(crossprod(X) + lambda[k] * diag(p), crossprod(X, Y))
    Bias[k] = sum(rowMeans(betaridge) - beta)
    Var[k] = sum(rowMeans((betaridge - rowMeans(betaridge))^2))
    MSE[k] = sum(rowMeans((betaridge - beta)^2))
}
plot(lambda, MSE, "l", ylim = c(0, max(MSE)), xlab = expression(lambda), ylab = NA,
    col = "purple", lwd = 2)
lines(lambda, abs(Bias), col = "red", lwd = 2)
lines(lambda, Var, col = "blue", lwd = 2)
abline(v = lambda[which.min(MSE)], lty = 2, lwd = 2)
legend("right", c("Bias", "Variance", "MSE", "Minimum"), col = c("red", "blue",
    "purple", "black"), lty = c(rep(1, 3), 2), lwd = rep(2, 4), cex = 0.5)
```



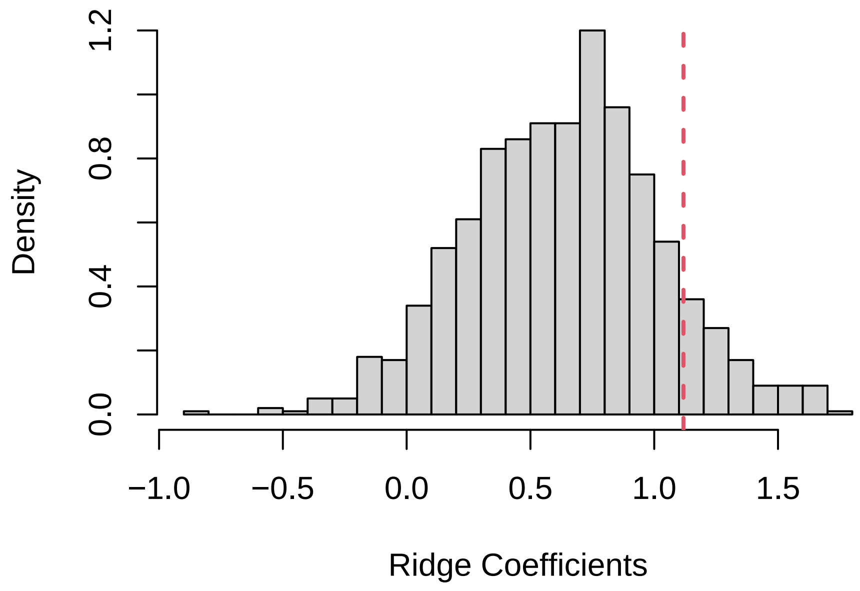
```
betaridge = solve(crossprod(X) + lambda[which.min(MSE)] * diag(p), crossprod(X,
    Y))
hist(betaridge[1, ], "FD", freq = FALSE, main = NA, xlab = "Ridge Coefficients")
abline(v = beta[1], col = 2, lty = 2, lwd = 2)
```



```
library(MASS)
n = 1000
p = 20
beta = rep(5/sqrt(p), p)
Sigma = 0.5^abs(outer(1:p, 1:p, "-"))
X = mvrnorm(n, numeric(p), Sigma)
X = t(t(X)/sqrt(colSums(X^2)))
nsim = 1000
Y = drop(X %*% beta) + matrix(rnorm(n * nsim), n)
Y = t(t(Y)/apply(Y, 2, sd))/sqrt(1 - 1/n)
lambda = seq(0, 2, 0.01)
Bias = numeric(201)
Var = numeric(201)
MSE = numeric(201)
for (k in 1:201) {
    betaridge = solve(crossprod(X) + lambda[k] * diag(p), crossprod(X, Y))
    Bias[k] = sum(rowMeans(betaridge) - beta)
    Var[k] = sum(rowMeans((betaridge - rowMeans(betaridge))^2))
    MSE[k] = sum(rowMeans((betaridge - beta)^2))
}
plot(lambda, MSE, "l", ylim = c(0, max(MSE)), xlab = expression(lambda), ylab = NA,
    col = "purple", lwd = 2)
lines(lambda, abs(Bias), col = "red", lwd = 2)
lines(lambda, Var, col = "blue", lwd = 2)
abline(v = lambda[which.min(MSE)], lty = 2, lwd = 2)
legend("right", c("Bias", "Variance", "MSE", "Minimum"), col = c("red", "blue",
    "purple", "black"), lty = c(rep(1, 3), 2), lwd = rep(2, 4), cex = 0.5)
```

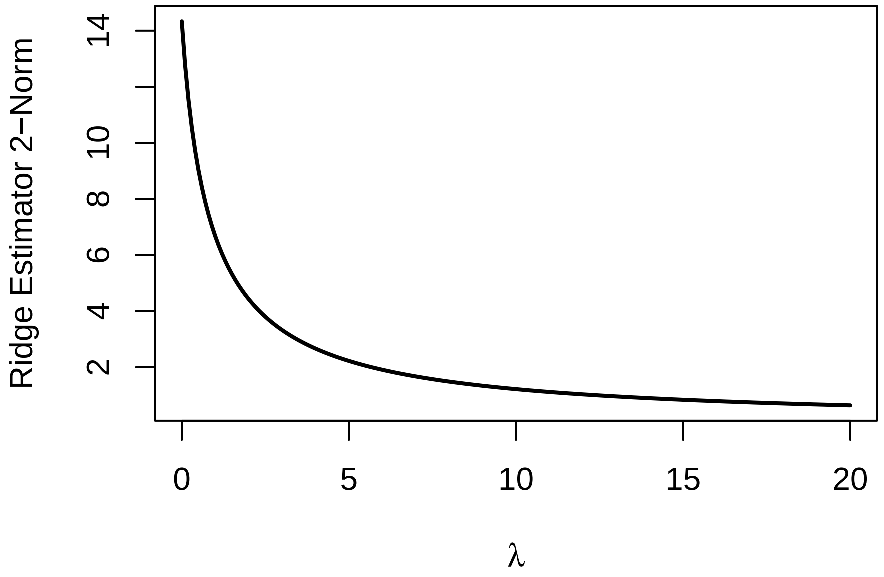


```
betaridge = solve(crossprod(X) + lambda[which.min(MSE)] * diag(p), crossprod(X,
    Y))
hist(betaridge[1, ], "FD", freq = FALSE, main = NA, xlab = "Ridge Coefficients")
abline(v = beta[1], col = 2, lty = 2, lwd = 2)
```



```
n = 1000
p = 100
beta = rnorm(p)
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
Y = X %*% beta + rnorm(n)
Y = Y/sd(Y)/sqrt(1 - 1/n)

lambda = seq(0, 20, 0.1)
betaridge = matrix(0, 201, 100)
```

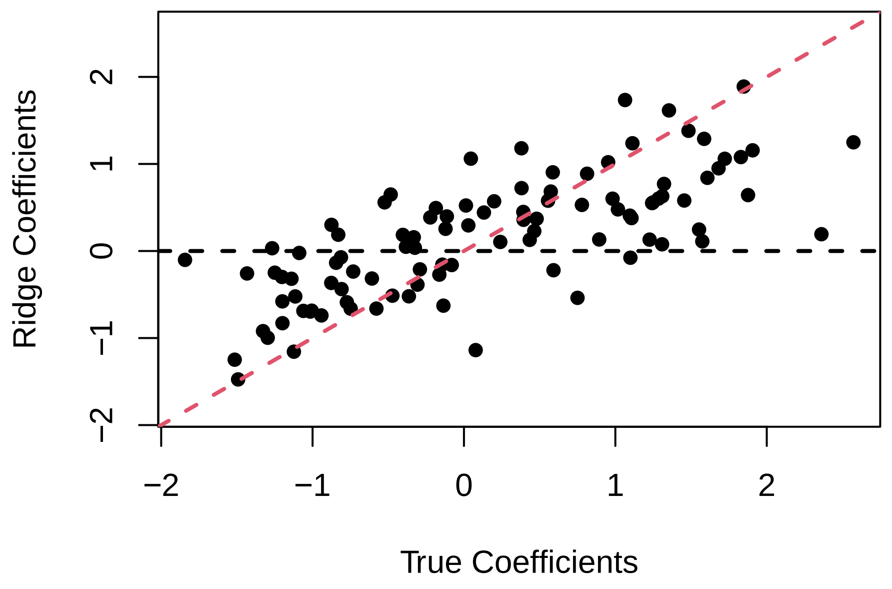
```
for (k in 1:201) {
    betaridge[k, ] = solve(crossprod(X) + lambda[k] * diag(p), crossprod(X,
        Y))
}
plot(lambda, sqrt(rowSums(betaridge^2)), "l", xlab = expression(lambda), ylab = "Ridge Estimator 2-Norm",
    lwd = 2)
```



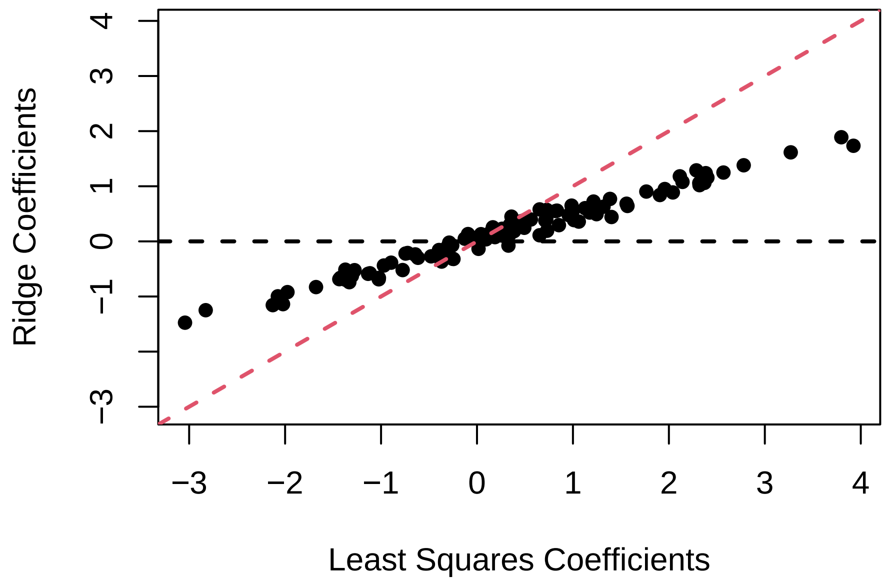
```
library(glmnet)
lambdamin = cv.glmnet(X, Y, alpha = 0)$lambda.min
print(lambdamin)
```

```
## [1] 0.8975042
```

```
betaridge = solve(crossprod(X) + lambdamin * diag(p), crossprod(X, Y))
betaols = solve(crossprod(X), crossprod(X, Y))
plot(beta, betaridge, ylim = range(beta), xlab = "True Coefficients", ylab = "Ridge Coefficients",
    pch = 16)
abline(h = 0, lty = 2, lwd = 2)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```

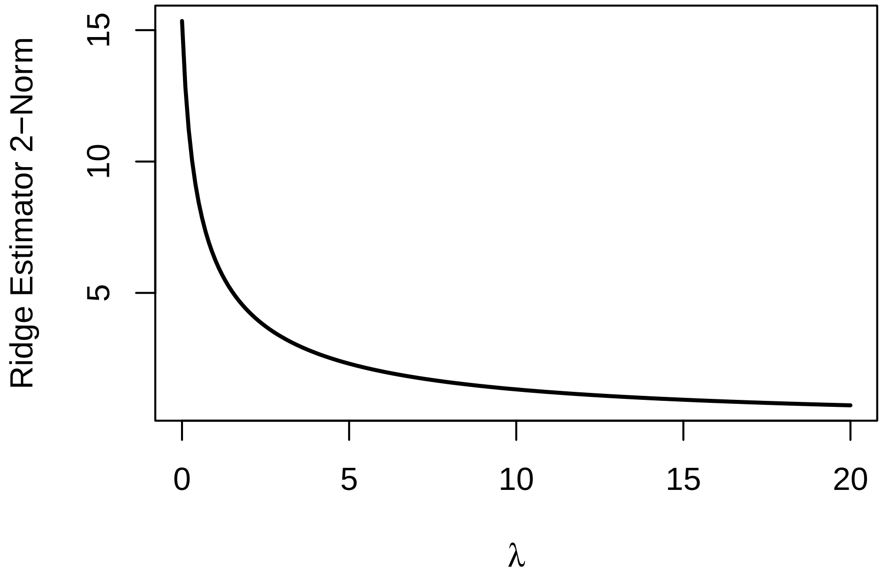


```
plot(betaols, betaridge, ylim = range(betaols), xlab = "Least Squares Coefficients",
    ylab = "Ridge Coefficients", pch = 16)
abline(h = 0, lty = 2, lwd = 2)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```



```
library(MASS)
n = 1000
p = 100
beta = rnorm(p)
Sigma = 0.5^abs(outer(1:p, 1:p, "-"))
X = mvrnorm(n, numeric(p), Sigma)
X = t(t(X)/sqrt(colSums(X^2)))
Y = X %*% beta + rnorm(n)
Y = Y/sd(Y)/sqrt(1 - 1/n)
```

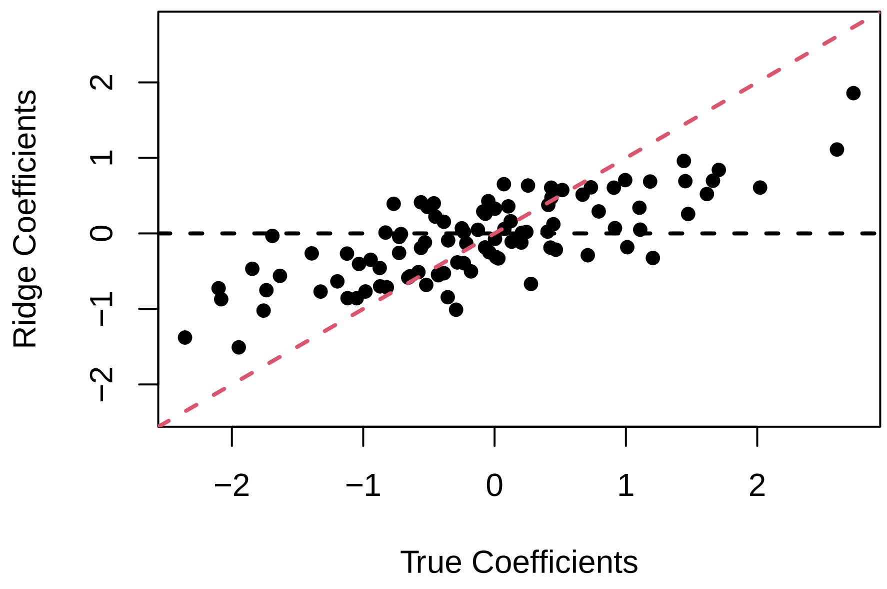
```r
lambda = seq(0, 20, 0.1)
betaridge = matrix(0, 201, 100)
for (k in 1:201) {
    betaridge[k, ] = solve(crossprod(X) + lambda[k] * diag(p), crossprod(X,
        Y))
}
plot(lambda, sqrt(rowSums(betaridge^2)), "l", xlab = expression(lambda), ylab = "Ridge Estimator 2-Norm",
    lwd = 2)
```



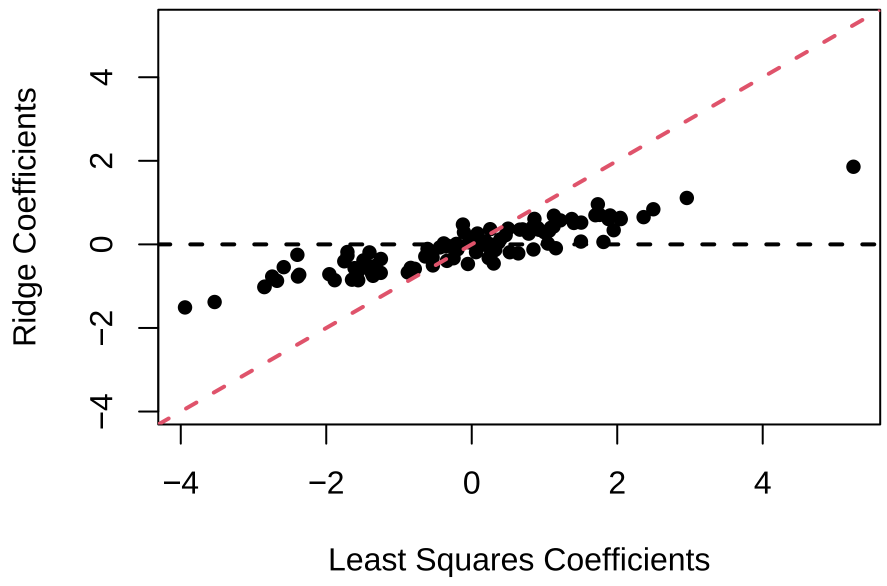
```r
library(glmnet)
lambdamin = cv.glmnet(X, Y, alpha = 0)$lambda.min
print(lambdamin)
```

```
## [1] 1.224639
```

```r
betaridge = solve(crossprod(X) + lambdamin * diag(p), crossprod(X, Y))
betaols = solve(crossprod(X), crossprod(X, Y))
plot(beta, betaridge, ylim = range(beta), xlab = "True Coefficients", ylab = "Ridge Coefficients",
    pch = 16)
abline(h = 0, lty = 2, lwd = 2)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```



```r
plot(betaols, betaridge, ylim = range(betaols), xlab = "Least Squares Coefficients",
    ylab = "Ridge Coefficients", pch = 16)
abline(h = 0, lty = 2, lwd = 2)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```



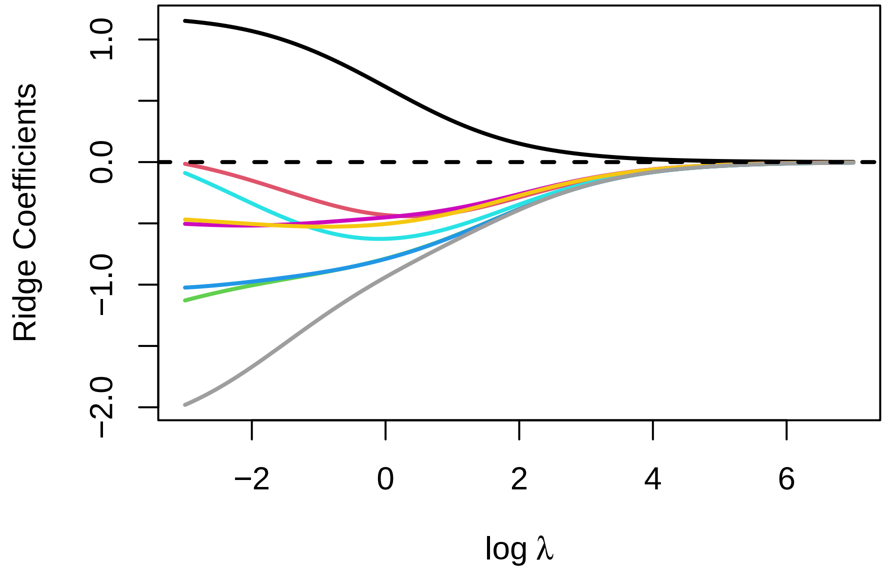
```r
library(faraway)
n = dim(seatpos)[1]
fit = lm(hipcenter ~ ., seatpos)
print(fit$coefficients[-1])
```

```
##          Age       Weight      HtShoes           Ht       Seated          Arm
##   0.77571620   0.02631308  -2.69240774   0.60134458   0.53375170  -1.32806864
##        Thigh          Leg
##  -1.14311888  -6.43904627
```

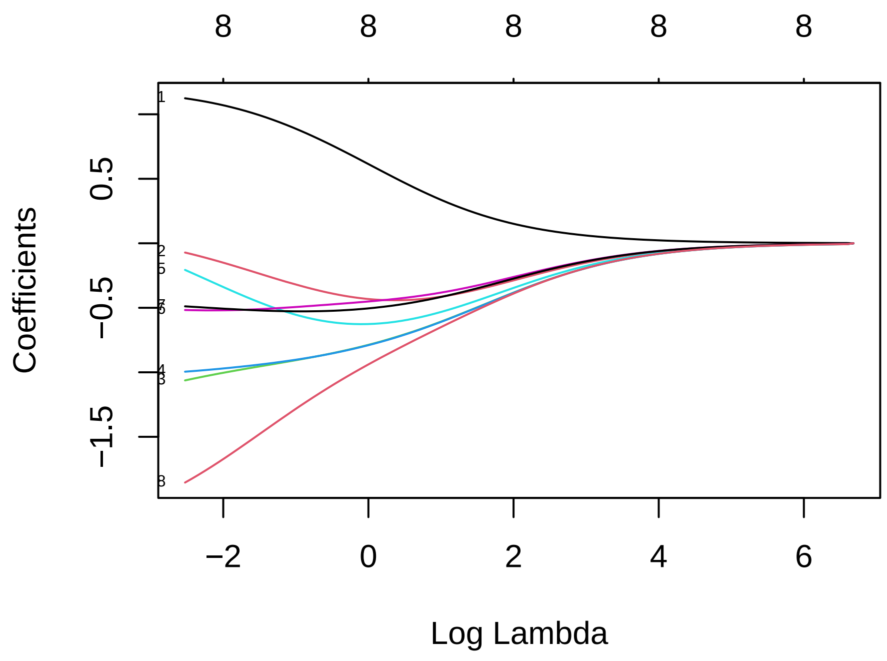
```r
vif(fit)
```

```
##        Age     Weight    HtShoes         Ht     Seated        Arm      Thigh
##   1.997931   3.647030 307.429378 333.137832   8.951054   4.496368   2.762886
##        Leg
##   6.694291
```

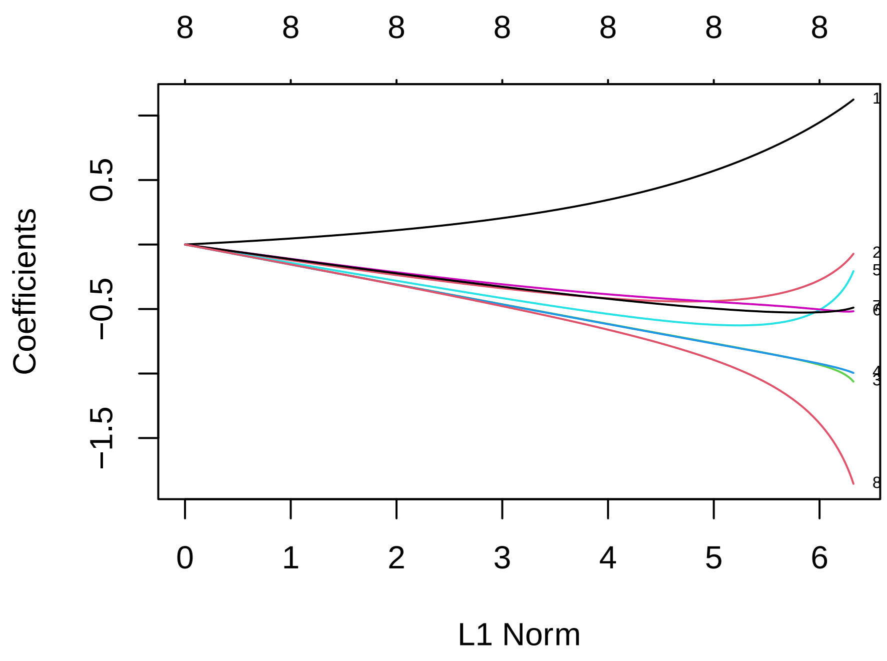
```r
Y = seatpos$hipcenter
Y = Y/sd(Y)/sqrt(1 - 1/n)
X = as.matrix(seatpos[, -9])
X = t(t(X) - colMeans(X))
X = t(t(X)/sqrt(colSums(X^2)))
loglambda = seq(-3, 7, 0.01)

library(glmnet)
betaridge = matrix(0, 1001, 8)
for (k in 1:1001) {
    betaridge[k, ] = solve(crossprod(X) + exp(loglambda[k]) * diag(8), crossprod(X,
        Y))
}
plot(loglambda, betaridge[, 1], "l", ylim = range(betaridge), xlab = expression("log" ~
    lambda), ylab = "Ridge Coefficients", lwd = 2)
for (j in 2:8) {
    lines(loglambda, betaridge[, j], col = j, lwd = 2)
}
abline(h = 0, lty = 2, lwd = 2)
```



```r
ridge = glmnet(X, Y, alpha = 0)
plot(ridge, "lambda", label = TRUE)
```



```
plot(ridge, label = TRUE)
```



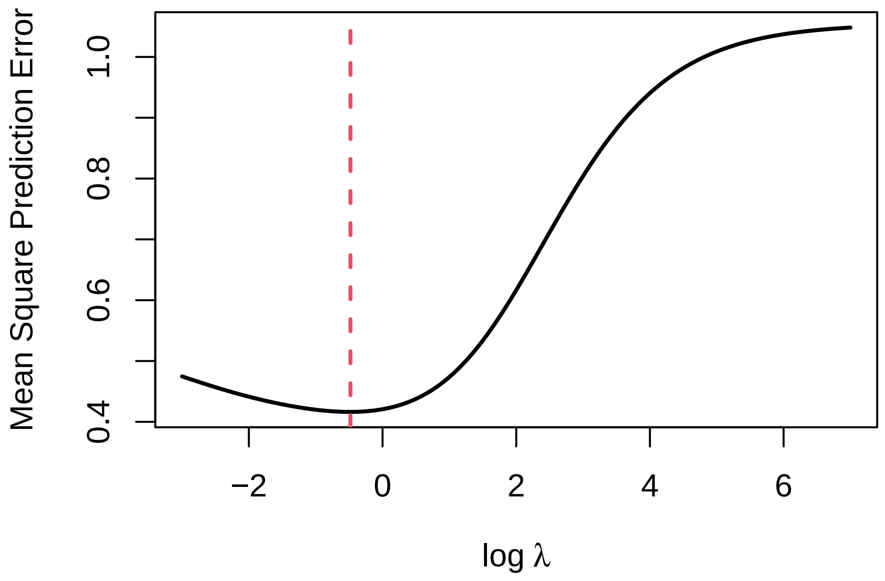
```
error = matrix(0, 1001, n)
for (k in 1:1001) {
    for (i in 1:n) {
        betaridge = drop(coef(glmnet(X[-i, ], Y[-i], alpha = 0, lambda = exp(loglambda[k]))))
        error[k, i] = (Y[i] - crossprod(c(1, X[i, ]), betaridge))^2
    }
}
MSPE = rowMeans(error)
lambdamin = exp(loglambda[which.min(MSPE)])
print(lambdamin)
```

```
## [1] 0.6187834
```

```
min(MSPE)
```

```
## [1] 0.4163351
```

```
plot(loglambda, MSPE, "l", xlab = expression("log" ~ lambda), ylab = "Mean Square Prediction Error",
    lwd = 2)
abline(v = log(lambdamin), col = 2, lty = 2, lwd = 2)
```



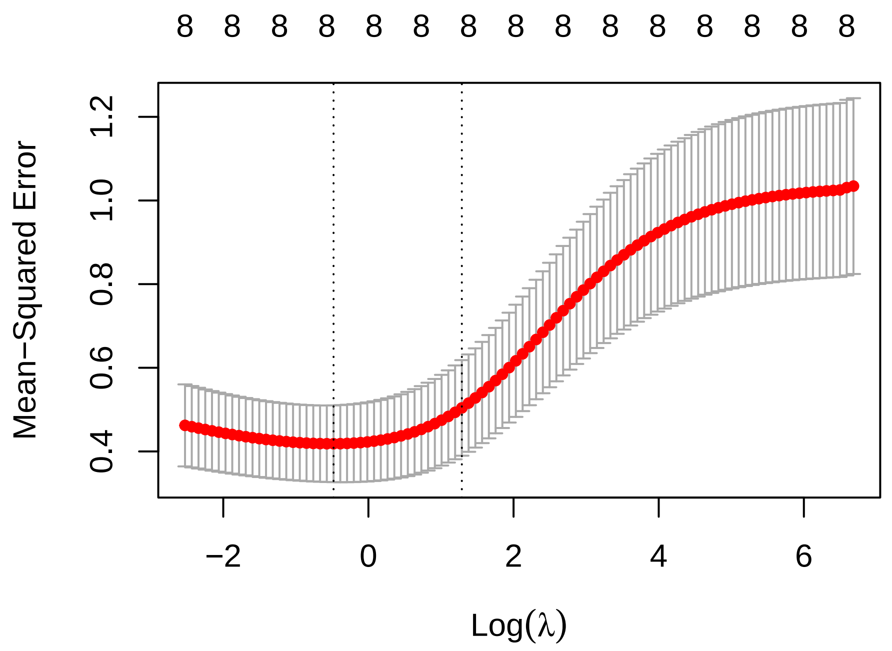
```
cvridge = cv.glmnet(X, Y, alpha = 0)
print(cvridge$lambda.min)
```

```
## [1] 0.6185805
```

```
min(cvridge$cvm)
```

```
## [1] 0.4183451
```

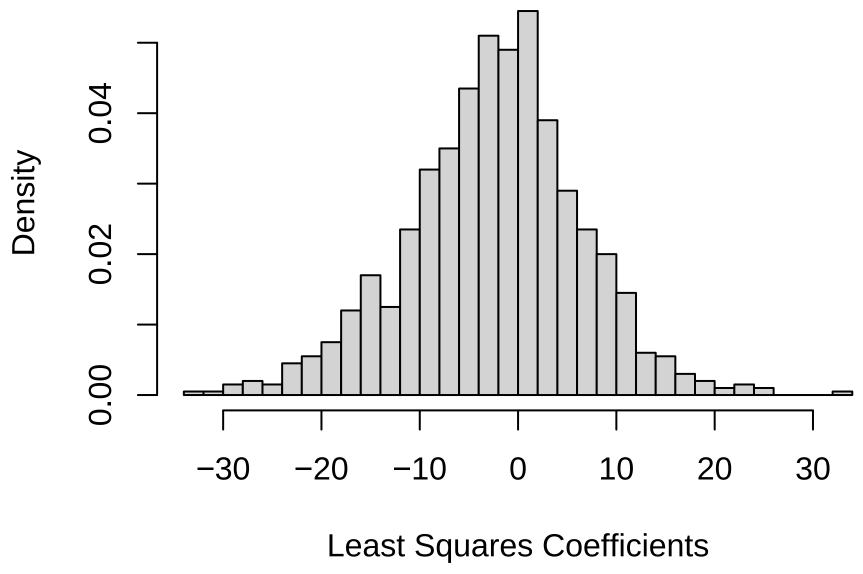
```
plot(cvridge)
```



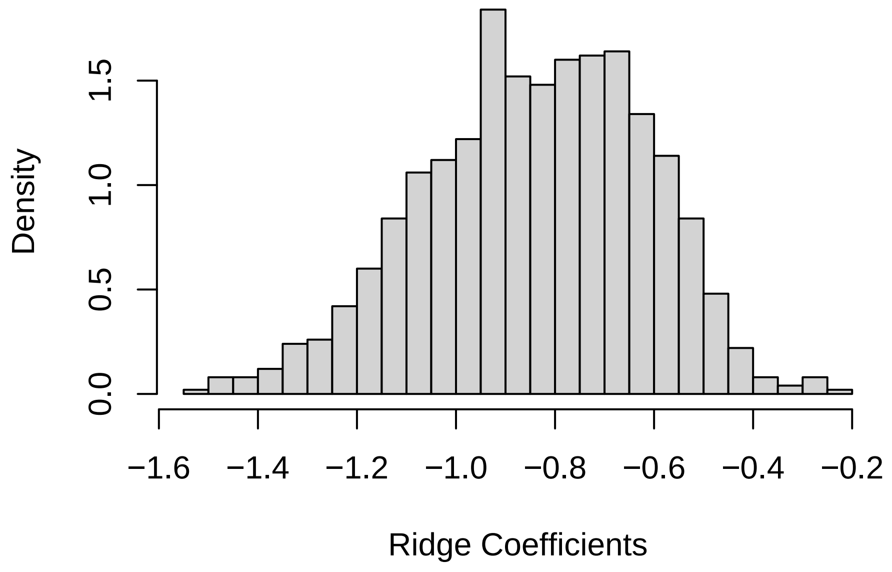
```
betaridge = drop(solve(crossprod(X) + lambdamin * diag(8), crossprod(X, Y)))
print(betaridge)
```

```
##        Age     Weight    HtShoes         Ht     Seated        Arm      Thigh
##  0.7529740 -0.3923536 -0.8512323 -0.8514846 -0.6131235 -0.4721104 -0.5223654
##        Leg
## -1.0946116
```

```
nboot = 1000
betaols = matrix(0, nboot, 8)
betaridge = matrix(0, nboot, 8)
for (i in 1:nboot) {
    ind = sample(n, replace = TRUE)
    betaols[i, ] = lm(Y ~ X, subset = ind)$coefficients[-1]
    betaridge[i, ] = drop(glmnet(X[ind, ], Y[ind], alpha = 0, lambda = lambdamin)$beta)
}
hist(betaols[, 3], "FD", freq = FALSE, main = NA, xlab = "Least Squares Coefficients")
```



```
hist(betaridge[, 3], "FD", freq = FALSE, main = NA, xlab = "Ridge Coefficients")
```



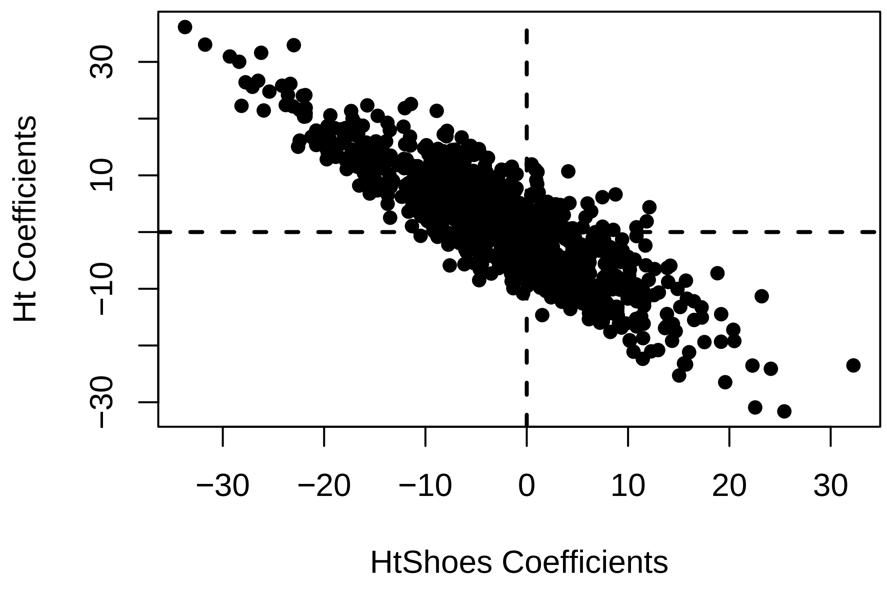
```
cor(betaols[, 3], betaols[, 4])
```

```
## [1] -0.8592313
```

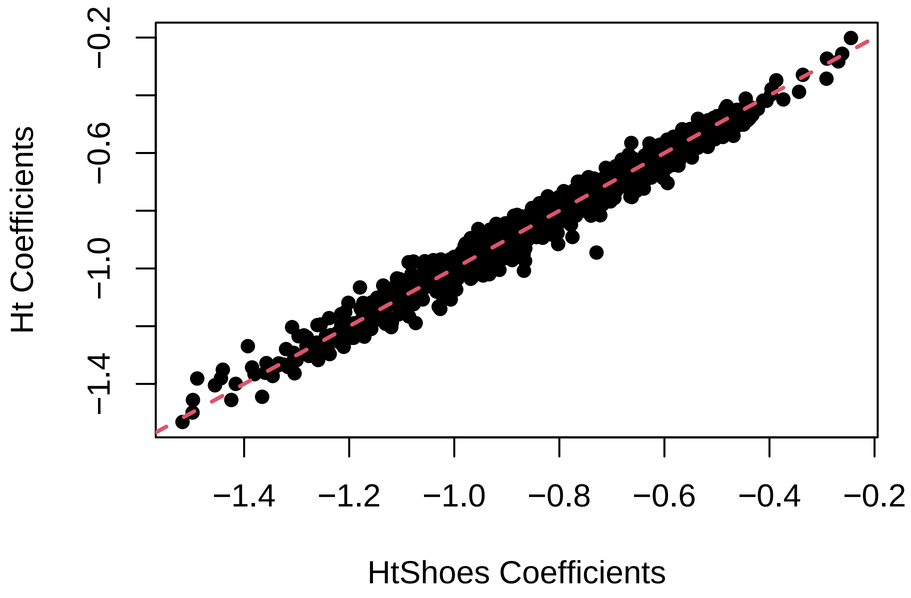
```
cor(betaridge[, 3], betaridge[, 4])
```

```
## [1] 0.9864823
```

```
plot(betaols[, 3:4], xlab = "HtShoes Coefficients", ylab = "Ht Coefficients",
    pch = 16)
abline(h = 0, lty = 2, lwd = 2)
abline(v = 0, lty = 2, lwd = 2)
```



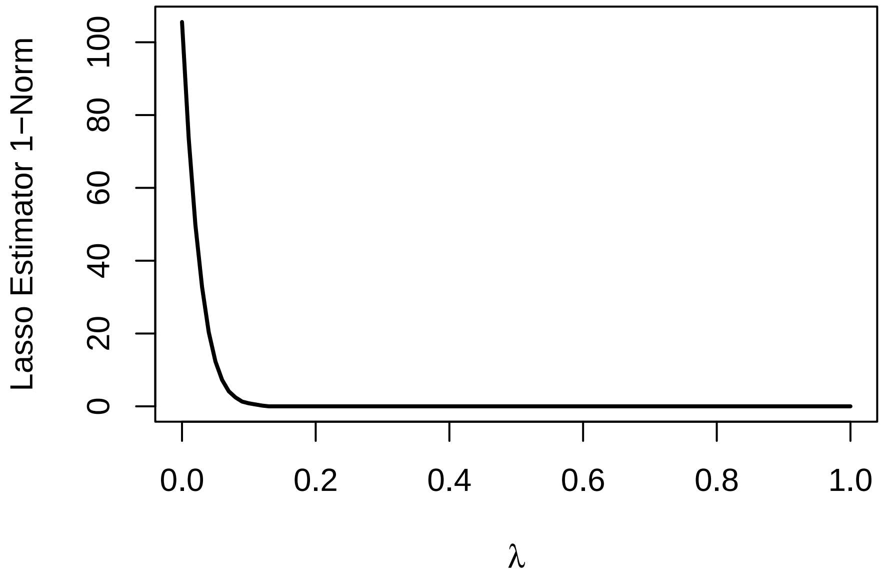
```r
plot(betaridge[, 3:4], xlab = "HtShoes Coefficients", ylab = "Ht Coefficients",
    pch = 16)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```



Lasso Regression

```r
n = 1000
p = 100
beta = rnorm(p)
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
Y = X %*% beta + rnorm(n)
Y = Y/sd(Y)/sqrt(1 - 1/n)
```

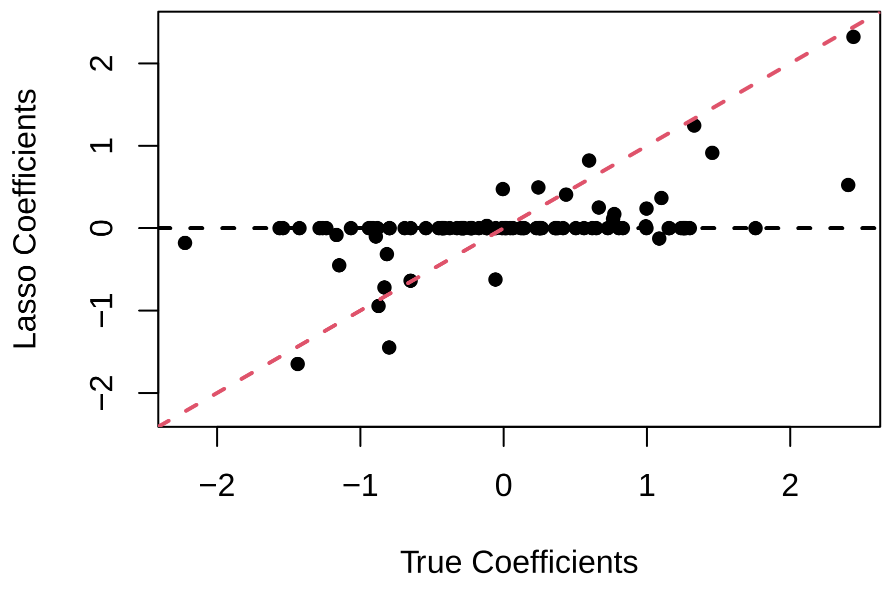
```
library(glmnet)
lambda = seq(0, 1, 0.01)
betalasso = matrix(0, 101, 100)
for (k in 1:101) {
    betalasso[k, ] = drop(glmnet(X, Y, lambda = lambda[k])$beta)
}
plot(lambda, rowSums(abs(betalasso)), "l", xlab = expression(lambda), ylab = "Lasso Estimator 1-Norm",
    lwd = 2)
```



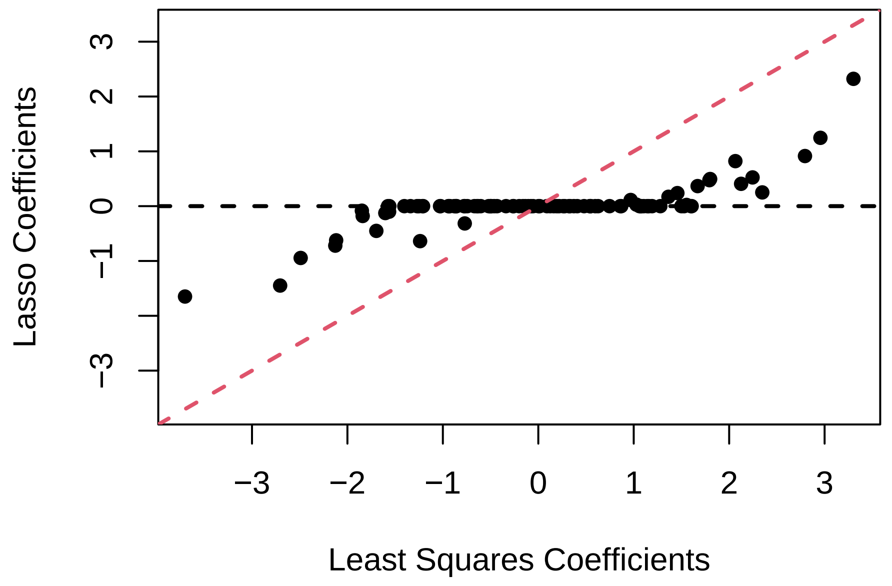
```
lambdamin = cv.glmnet(X, Y)$lambda.min
print(lambdamin)
```

```
## [1] 0.04537194
```

```
betalasso = drop(glmnet(X, Y, lambda = lambdamin)$beta)
betaols = lm(Y ~ X)$coefficients[-1]
plot(beta, betalasso, ylim = range(beta), xlab = "True Coefficients", ylab = "Lasso Coefficients",
    pch = 16)
abline(h = 0, lty = 2, lwd = 2)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```

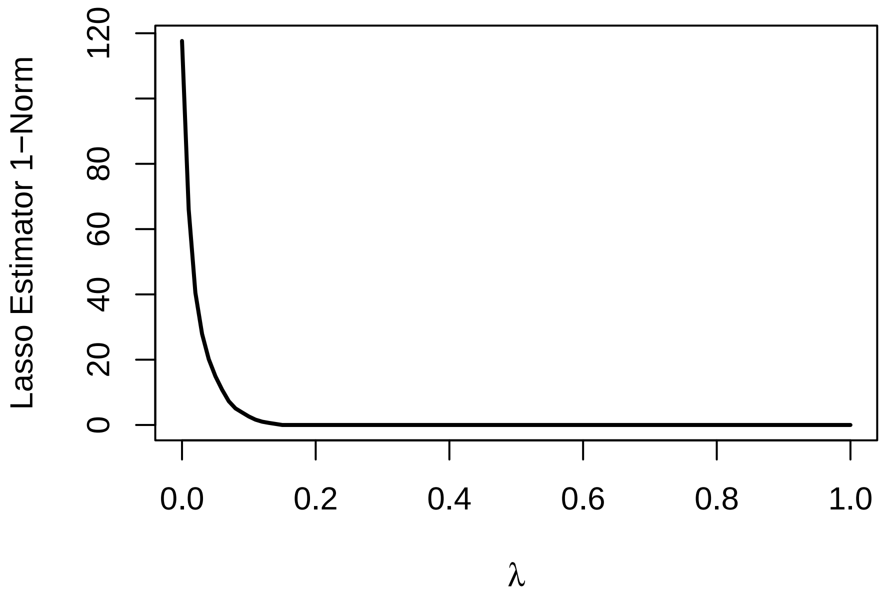


```
plot(betaols, betalasso, ylim = range(betaols), xlab = "Least Squares Coefficients",
    ylab = "Lasso Coefficients", pch = 16)
abline(h = 0, lty = 2, lwd = 2)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```



```
library(MASS)
n = 1000
p = 100
beta = rnorm(p)
Sigma = 0.5^abs(outer(1:p, 1:p, "-"))
X = mvrnorm(n, numeric(p), Sigma)
X = t(t(X)/sqrt(colSums(X^2)))
Y = X %*% beta + rnorm(n)
Y = Y/sd(Y)/sqrt(1 - 1/n)
```

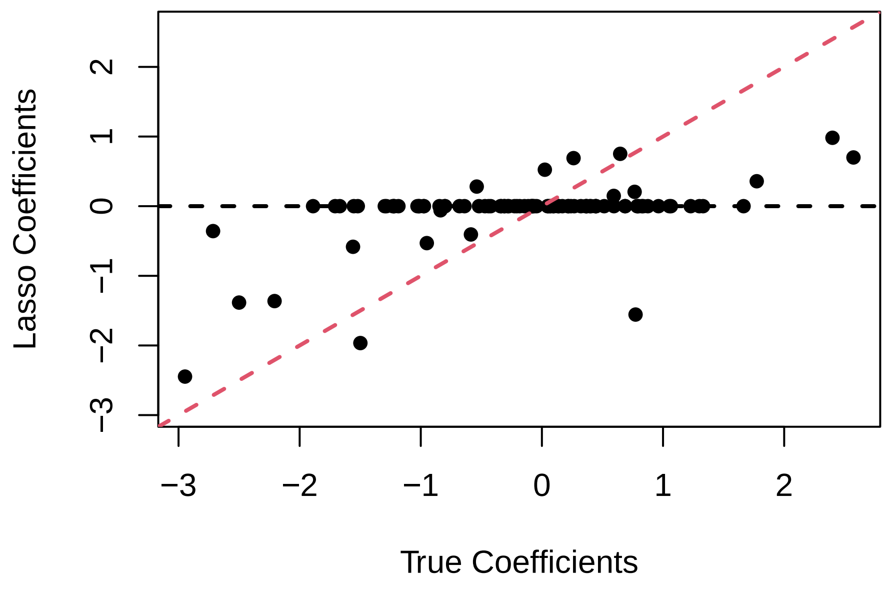
```r
library(glmnet)
lambda = seq(0, 1, 0.01)
betalasso = matrix(0, 101, 100)
for (k in 1:101) {
    betalasso[k, ] = drop(glmnet(X, Y, lambda = lambda[k])$beta)
}
plot(lambda, rowSums(abs(betalasso)), "l", xlab = expression(lambda), ylab = "Lasso Estimator 1-Norm",
    lwd = 2)
```



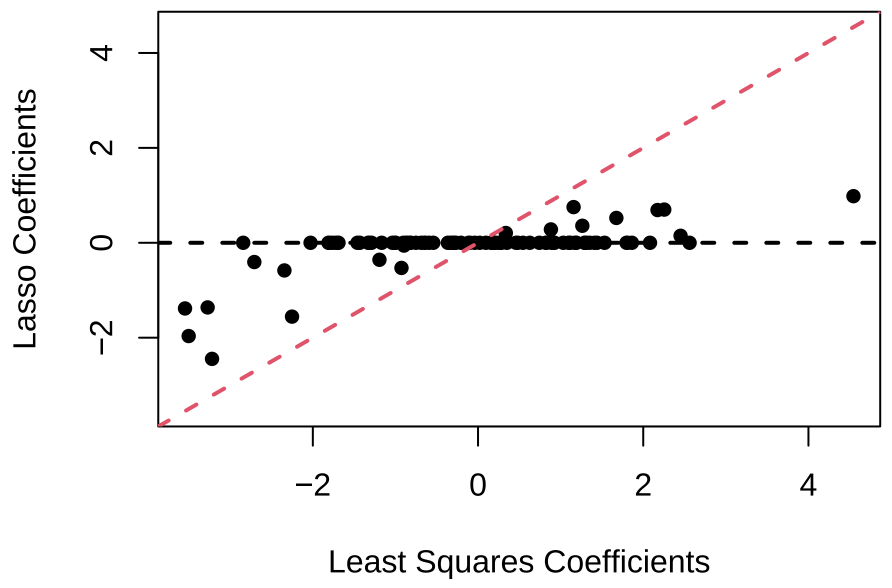
```r
lambdamin = cv.glmnet(X, Y)$lambda.min
print(lambdamin)
```

```
## [1] 0.04913336
```

```r
betalasso = drop(glmnet(X, Y, lambda = lambdamin)$beta)
betaols = lm(Y ~ X)$coefficients[-1]
plot(beta, betalasso, ylim = range(beta), xlab = "True Coefficients", ylab = "Lasso Coefficients",
    pch = 16)
abline(h = 0, lty = 2, lwd = 2)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```



```
plot(betaols, betalasso, ylim = range(betaols), xlab = "Least Squares Coefficients",
    ylab = "Lasso Coefficients", pch = 16)
abline(h = 0, lty = 2, lwd = 2)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```



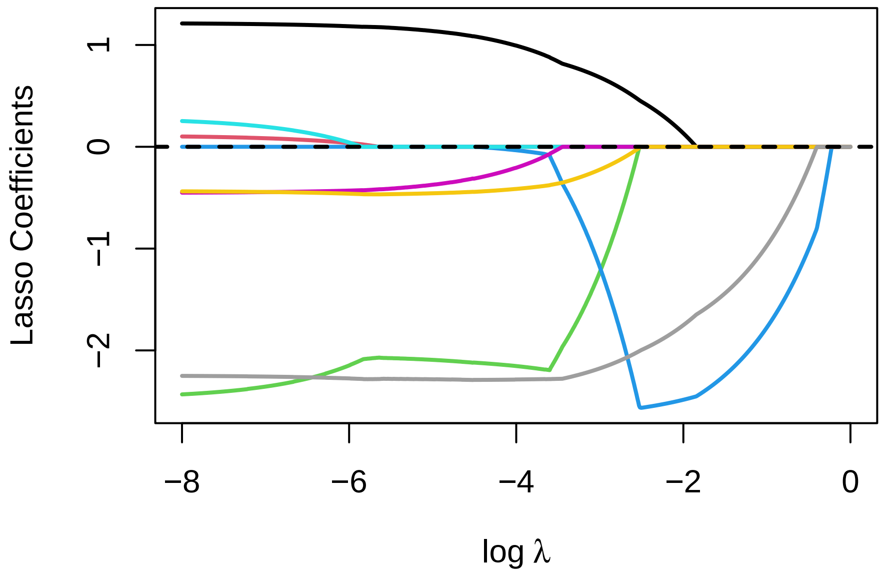
```
library(faraway)
n = dim(seatpos)[1]
fit = lm(hipcenter ~ ., seatpos)
print(fit$coefficients[-1])
```

```
##         Age      Weight     HtShoes          Ht      Seated         Arm
##  0.77571620  0.02631308 -2.69240774  0.60134458  0.53375170 -1.32806864
##       Thigh         Leg
## -1.14311888 -6.43904627
```

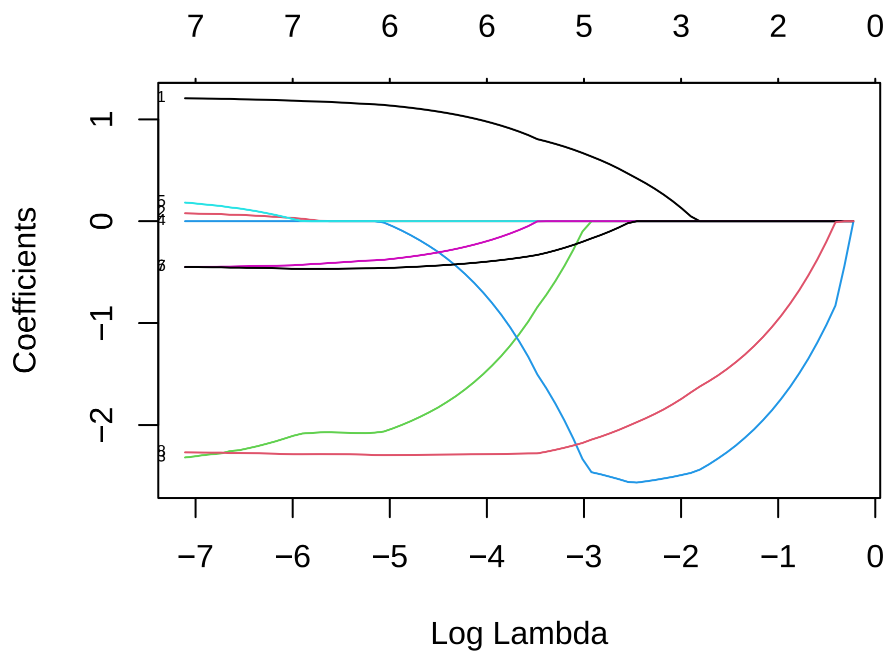
```
vif(fit)
```

```
##        Age     Weight    HtShoes         Ht     Seated        Arm      Thigh
##   1.997931   3.647030 307.429378 333.137832   8.951054   4.496368   2.762886
##        Leg
##   6.694291
```

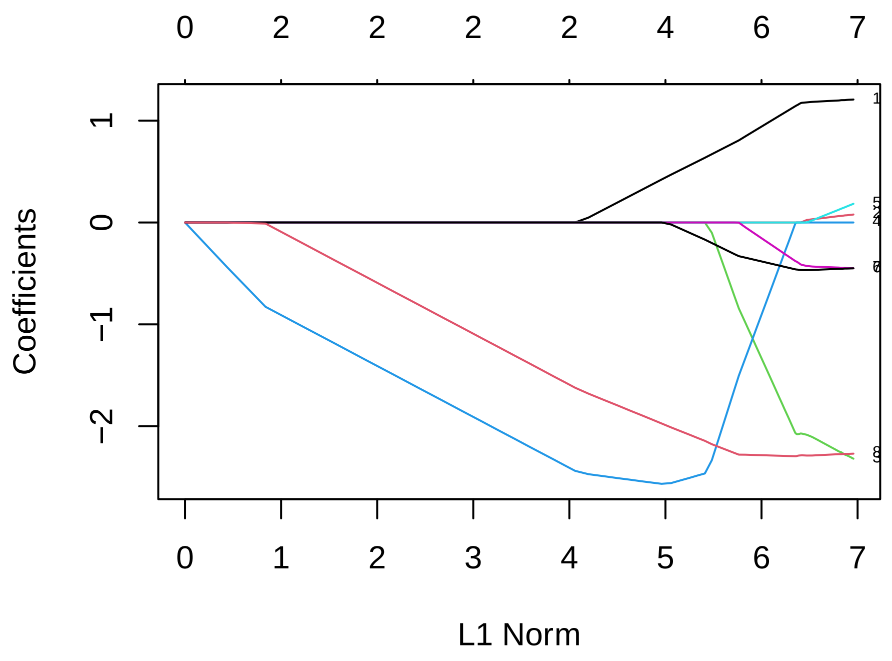
```
Y = seatpos$hipcenter
Y = Y/sd(Y)/sqrt(1 - 1/n)
X = as.matrix(seatpos[, -9])
X = t(t(X) - colMeans(X))
X = t(t(X)/sqrt(colSums(X^2)))
loglambda = seq(-8, 0, 0.01)

library(glmnet)
betalasso = matrix(0, 801, 8)
for (k in 1:801) {
    betalasso[k, ] = drop(glmnet(X, Y, lambda = exp(loglambda[k]))$beta)
}
plot(loglambda, betalasso[, 1], "l", ylim = range(betalasso), xlab = expression("log" ~
    lambda), ylab = "Lasso Coefficients", lwd = 2)
for (j in 2:8) {
    lines(loglambda, betalasso[, j], col = j, lwd = 2)
}
abline(h = 0, lty = 2, lwd = 2)
```



```
lasso = glmnet(X, Y)
plot(lasso, "lambda", label = TRUE)
```



```
plot(lasso, label = TRUE)
```



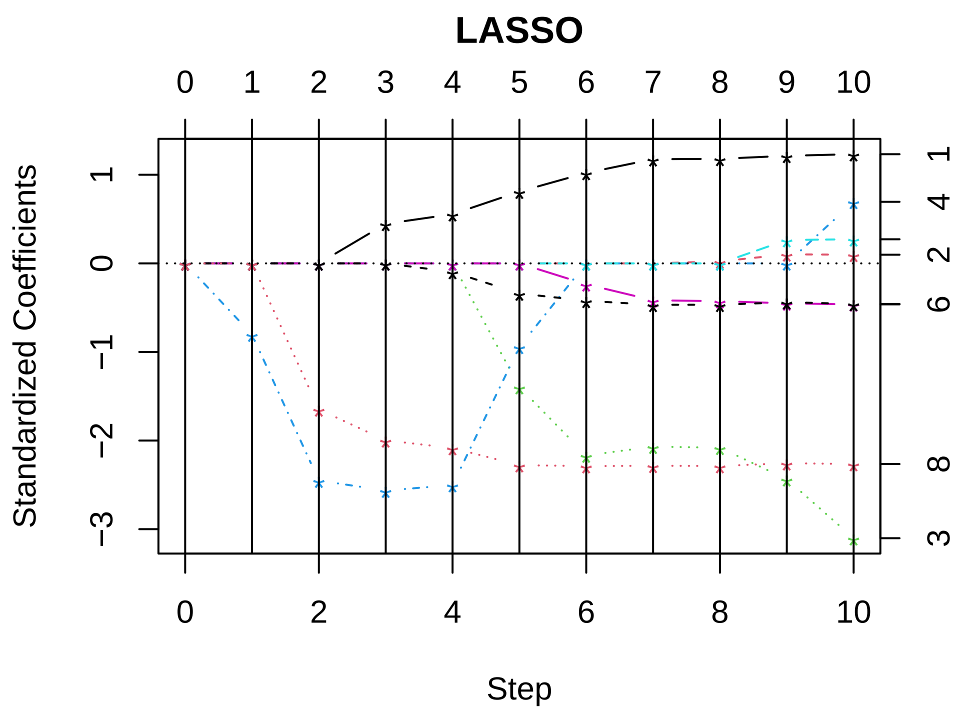
```
library(lars)
lasso = lars(X, Y)
lambda = lasso$lambda
nsteps = length(lambda)
print(lambda)
```

```
##  [1] 4.924919395 4.121719628 0.974055116 0.502716844 0.405267767 0.194168434
##  [7] 0.098477117 0.021131559 0.017520553 0.001050538
```

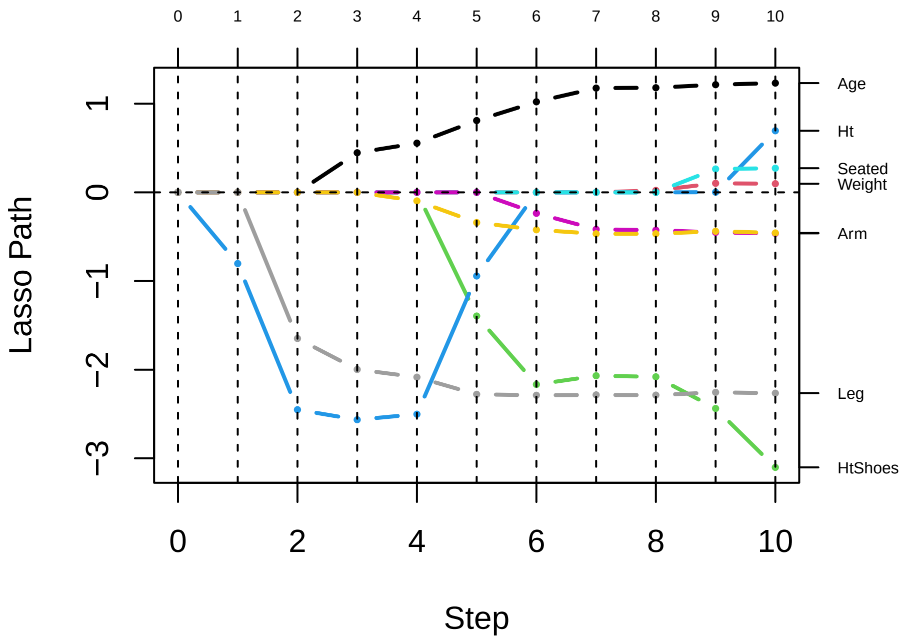
```
print(data.frame(lasso$beta))
```

```
##          Age     Weight   HtShoes         Ht    Seated        Arm       Thigh
## 0  0.0000000 0.00000000  0.000000  0.0000000 0.0000000  0.0000000  0.00000000
## 1  0.0000000 0.00000000  0.000000 -0.8031998 0.0000000  0.0000000  0.00000000
## 2  0.0000000 0.00000000  0.000000 -2.4514053 0.0000000  0.0000000  0.00000000
## 3  0.4463463 0.00000000  0.000000 -2.5645352 0.0000000  0.0000000  0.00000000
## 4  0.5545848 0.00000000  0.000000 -2.5022093 0.0000000  0.0000000 -0.09684545
## 5  0.8096615 0.00000000 -1.396380 -0.9416681 0.0000000  0.0000000 -0.34045568
## 6  1.0210143 0.00000000 -2.168355  0.0000000 0.0000000 -0.2364043 -0.42230734
## 7  1.1755669 0.00000000 -2.068416  0.0000000 0.0000000 -0.4171627 -0.46751531
## 8  1.1797593 0.02162282 -2.080253  0.0000000 0.0000000 -0.4266430 -0.46719410
## 9  1.2140630 0.10200028 -2.436091  0.0000000 0.2633237 -0.4516702 -0.43849895
## 10 1.2320899 0.09730313 -3.102046  0.6943939 0.2720472 -0.4627426 -0.45778532
##          Leg
## 0   0.000000
## 1   0.000000
## 2  -1.648206
## 3  -1.997729
## 4  -2.084393
## 5  -2.277647
## 6  -2.288418
## 7  -2.284386
## 8  -2.287087
## 9  -2.255455
## 10 -2.265018
```

```
plot(lasso, "step")
```



```r
betalasso = matrix(0, nsteps + 1, 8)
for (i in 2:nsteps) {
    betalasso[i, ] = betalasso[i - 1, ]
    err = Inf
    while (err > 1e-05) {
        betacurr = betalasso[i, ]
        for (j in 1:8) {
            r = Y - X[, -j] %*% betalasso[i, -j]
            z = sum(X[, j] * r)
            betalasso[i, j] = sign(z) * max(abs(z) - lambda[i], 0)
        }
        err = sum(abs((betalasso[i, ] - betacurr)))
    }
}
betalasso[nsteps + 1, ] = solve(crossprod(X), crossprod(X, Y))
par(mar = c(5.1, 4.1, 4.1, 4.1))
plot(0:nsteps, betalasso[, 1], "b", ylim = range(betalasso), xlab = "Step",
    ylab = "Lasso Path", pch = 16, cex = 0.5, lwd = 2)
for (j in 2:8) {
    lines(0:nsteps, betalasso[, j], "b", col = j, pch = 16, cex = 0.5, lwd = 2)
}
abline(h = 0, lty = 2)
abline(v = 0:10, lty = 2)
axis(4, betalasso[nsteps + 1, ], colnames(X), las = 2, cex.axis = 0.5)
axis(3, 0:nsteps, 0:nsteps, cex.axis = 0.5)
```



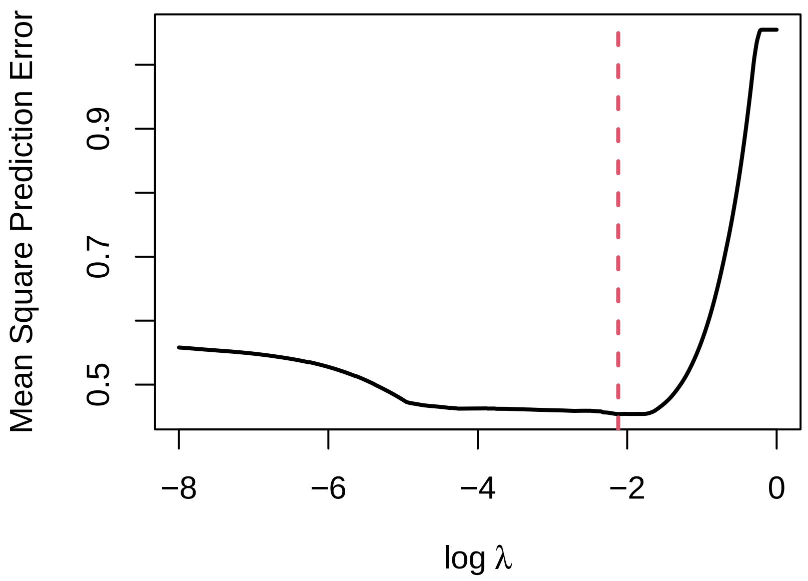
```
error = matrix(0, 801, n)
for (k in 1:801) {
    for (i in 1:n) {
        betalasso = drop(coef(glmnet(X[-i, ], Y[-i], lambda = exp(loglambda[k]))))
        error[k, i] = (Y[i] - crossprod(c(1, X[i, ]), betalasso))^2
    }
}
MSPE = rowMeans(error)
lambdamin = exp(loglambda[which.min(MSPE)])
print(lambdamin)
```

```
## [1] 0.1200316
```

```
min(MSPE)
```

```
## [1] 0.4539938
```

```
plot(loglambda, MSPE, "l", xlab = expression("log" ~ lambda), ylab = "Mean Square Prediction Error",
    lwd = 2)
abline(v = log(lambdamin), col = 2, lty = 2, lwd = 2)
```



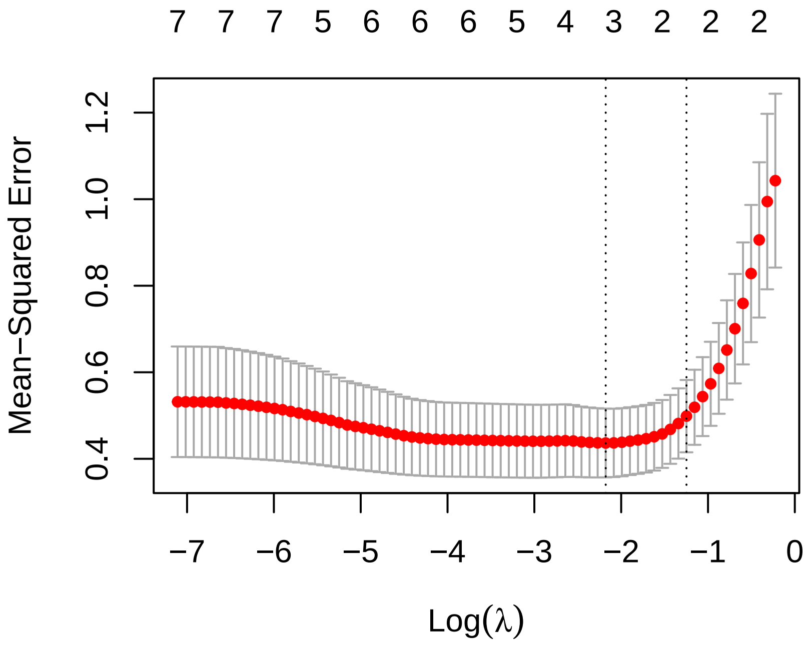
```
cvlasso = cv.glmnet(X, Y)
print(cvlasso$lambda.min)
```

```
## [1] 0.1132459
```

```
min(cvlasso$cvm)
```

```
## [1] 0.4365229
```

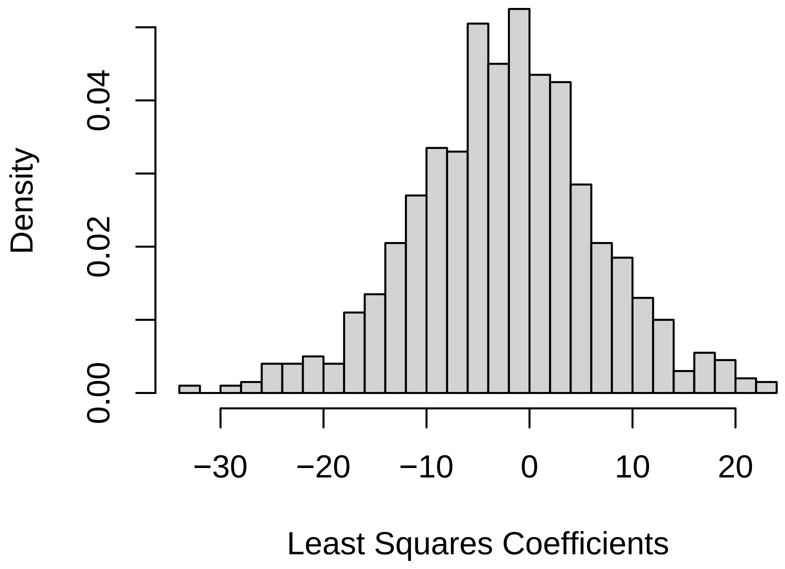
```
plot(cvlasso)
```



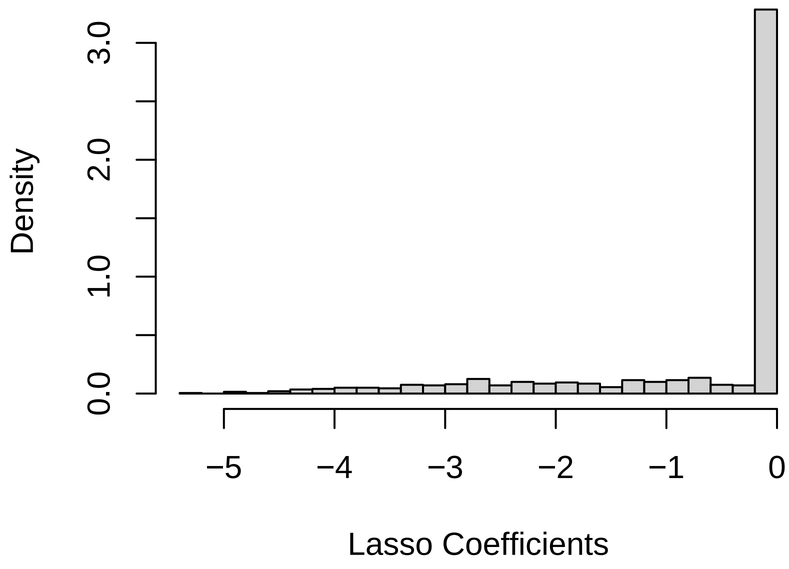
```
betalasso = drop(glmnet(X, Y, lambda = lambdamin)$beta)
print(betalasso)
```

```
##        Age     Weight    HtShoes          Ht     Seated        Arm      Thigh
##  0.2217171  0.0000000  0.0000000 -2.5075853  0.0000000  0.0000000  0.0000000
##        Leg
## -1.8218406
```

```
nboot = 1000
betaols = matrix(0, nboot, 8)
betalasso = matrix(0, nboot, 8)
for (i in 1:nboot) {
    ind = sample(n, replace = TRUE)
    betaols[i, ] = lm(Y ~ X, subset = ind)$coefficients[-1]
    betalasso[i, ] = drop(glmnet(X[ind, ], Y[ind], lambda = lambdamin)$beta)
}
hist(betaols[, 3], "FD", freq = FALSE, main = NA, xlab = "Least Squares Coefficients")
```

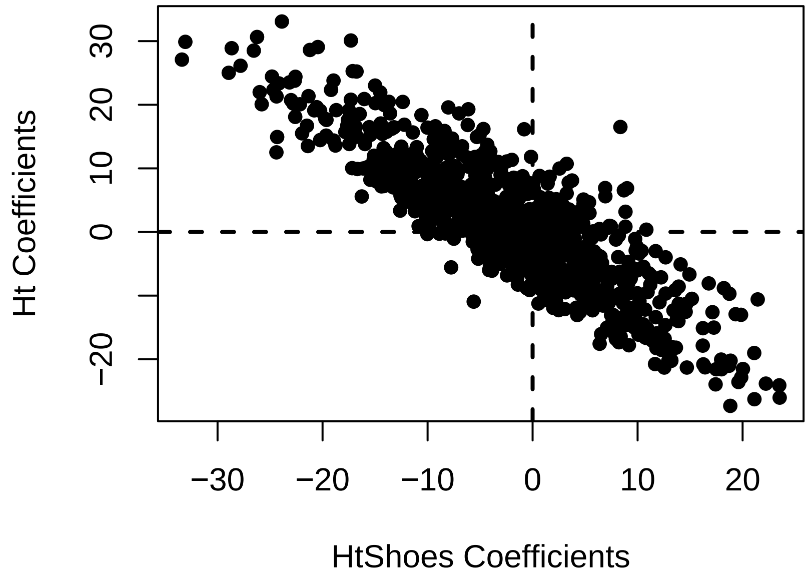


```
hist(betalasso[, 3], "FD", freq = FALSE, main = NA, xlab = "Lasso Coefficients")
```

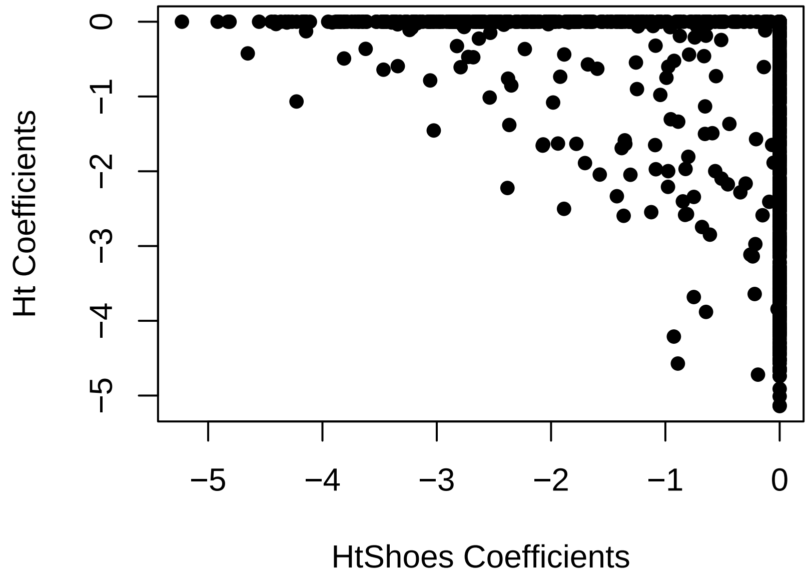


```
plot(betaols[, 3:4], xlab = "HtShoes Coefficients", ylab = "Ht Coefficients",
    pch = 16)
```

```
abline(h = 0, lty = 2, lwd = 2)
abline(v = 0, lty = 2, lwd = 2)
```



```
plot(betalasso[, 3:4], xlab = "HtShoes Coefficients", ylab = "Ht Coefficients",
    pch = 16)
```



6 Multiple Testing

Testing the Global Null

```
library(qqconf)
n = 100
p = 10
beta = numeric(p)
alpha = 0.05
```

```r
nsim = 1000
t = matrix(0, p, nsim)
for (i in 1:nsim) {
    Y = matrix(rnorm(p * n, beta, 1), p)
    t[, i] = sqrt(n) * rowMeans(Y)/apply(Y, 1, sd)
}
pval = 2 * pt(abs(t), n - 1, lower.tail = FALSE)
bonferroni = apply(pval, 2, min)
fisher = -2 * colSums(log(pval))
pval = apply(pval, 2, sort)
simes = colSums(pval <= (1:p) * alpha/p)
mean(bonferroni < alpha/p)
```

```
## [1] 0.054
```

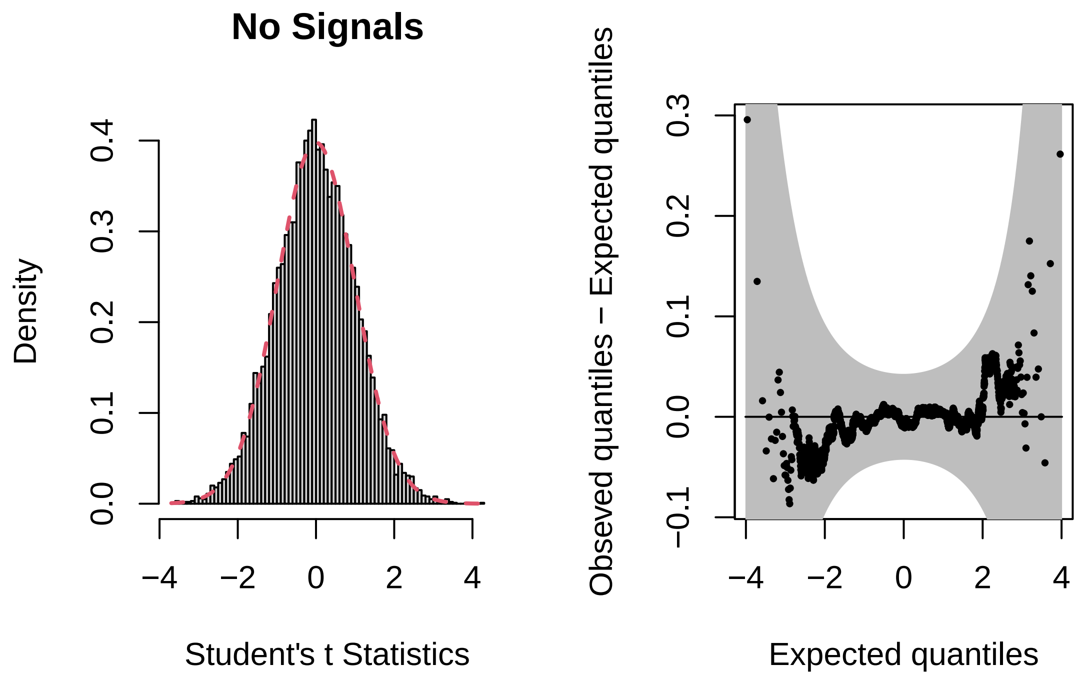
```r
mean(fisher > qchisq(1 - alpha, 2 * p))
```

```
## [1] 0.063
```

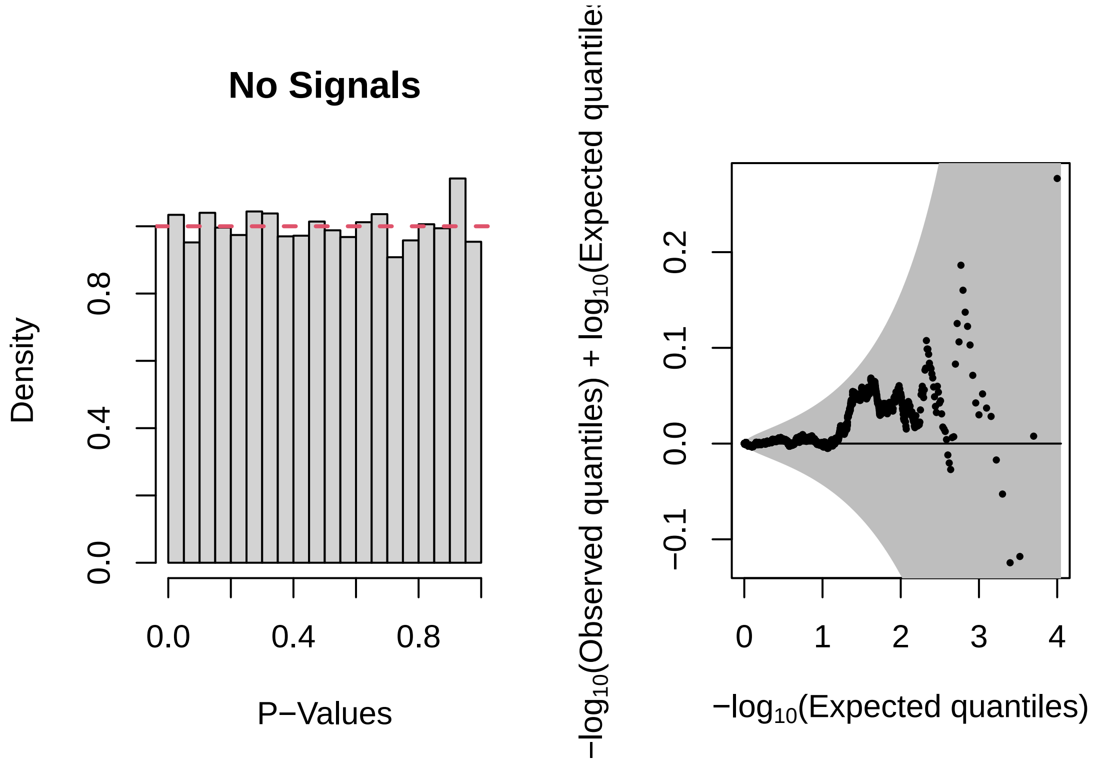
```r
mean(simes > 0)
```

```
## [1] 0.058
```

```r
par(mfrow = c(1, 2))
hist(t, "FD", freq = FALSE, main = "No Signals", xlab = "Student's t Statistics")
curve(dt(x, n - 1), add = TRUE, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(t), qt, difference = TRUE, dparams = list(df = n - 1),
    points_params = list(pch = 16, cex = 0.5))
```



```
hist(pval, "FD", freq = FALSE, main = "No Signals", xlab = "P-Values")
abline(h = 1, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(pval), qunif, difference = TRUE, log10 = TRUE, points_params = list(pch = 16,
    cex = 0.5))
```



```
library(qqconf)
n = 100
p = 10
beta = c(0.25, numeric(p - 1))
alpha = 0.05
nsim = 1000
t = matrix(0, p, nsim)
for (i in 1:nsim) {
    Y = matrix(rnorm(p * n, beta, 1), p)
    t[, i] = sqrt(n) * rowMeans(Y)/apply(Y, 1, sd)
}
pval = 2 * pt(abs(t), n - 1, lower.tail = FALSE)
bonferroni = apply(pval, 2, min)
fisher = -2 * colSums(log(pval))
pval = apply(pval, 2, sort)
simes = colSums(pval <= (1:p) * alpha/p)
mean(bonferroni < alpha/p)
```

```
## [1] 0.384
```

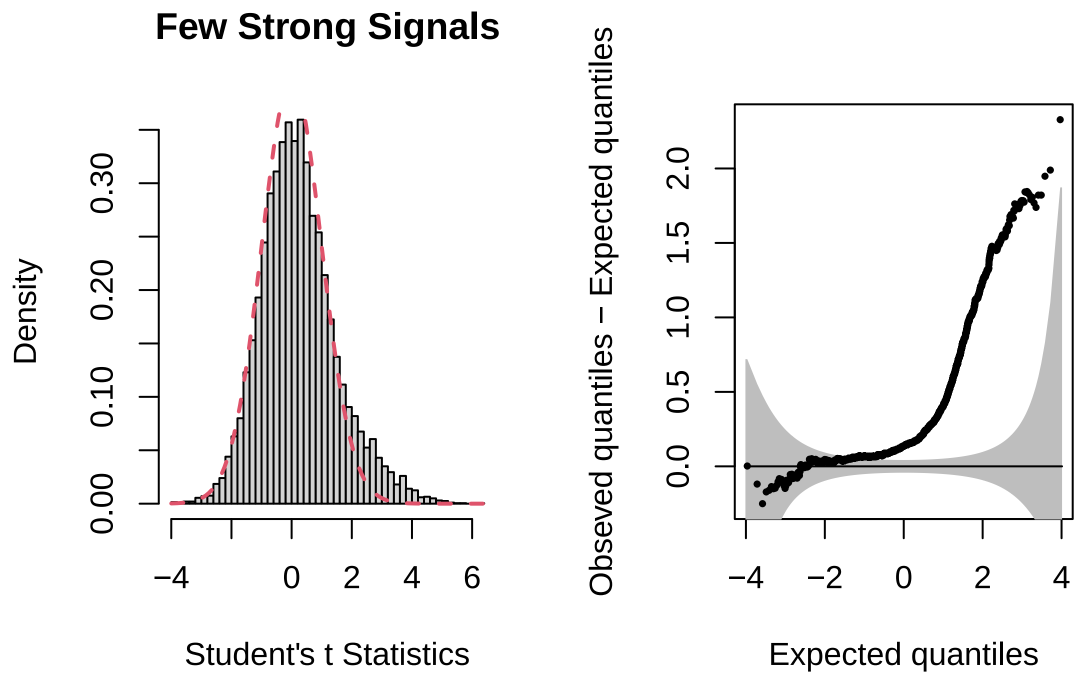
```
mean(fisher > qchisq(1 - alpha, 2 * p))
```

```
## [1] 0.307
```

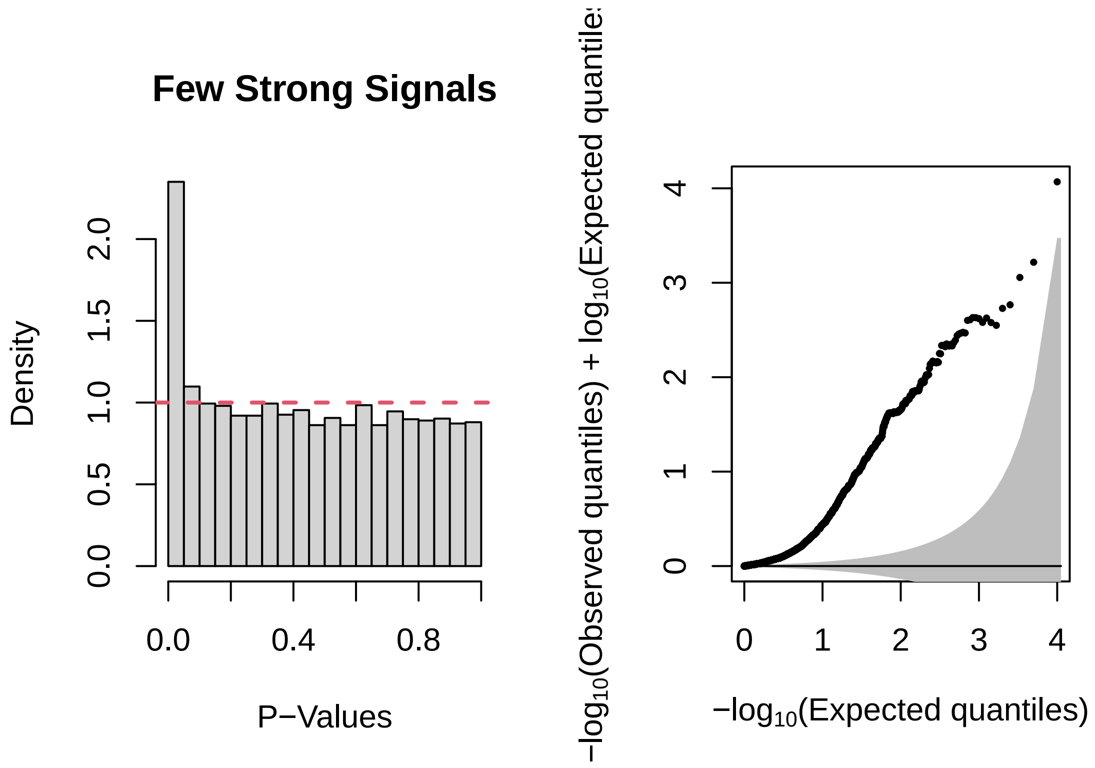
```
mean(simes > 0)
```

```
## [1] 0.394
```

```
par(mfrow = c(1, 2))
hist(t, "FD", freq = FALSE, main = "Few Strong Signals", xlab = "Student's t Statistics")
curve(dt(x, n - 1), add = TRUE, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(t), qt, difference = TRUE, dparams = list(df = n - 1),
    points_params = list(pch = 16, cex = 0.5))
```



```
hist(pval, "FD", freq = FALSE, main = "Few Strong Signals", xlab = "P-Values")
abline(h = 1, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(pval), qunif, difference = TRUE, log10 = TRUE, points_params = list(pch = 16,
    cex = 0.5))
```



```
library(qqconf)
n = 100
p = 10
beta = rep(0.1, p)
alpha = 0.05
nsim = 1000
t = matrix(0, p, nsim)
for (i in 1:nsim) {
    Y = matrix(rnorm(p * n, beta, 1), p)
    t[, i] = sqrt(n) * rowMeans(Y)/apply(Y, 1, sd)
}
pval = 2 * pt(abs(t), n - 1, lower.tail = FALSE)
bonferroni = apply(pval, 2, min)
fisher = -2 * colSums(log(pval))
pval = apply(pval, 2, sort)
simes = colSums(pval <= (1:p) * alpha/p)
mean(bonferroni < alpha/p)
```

```
## [1] 0.296
```

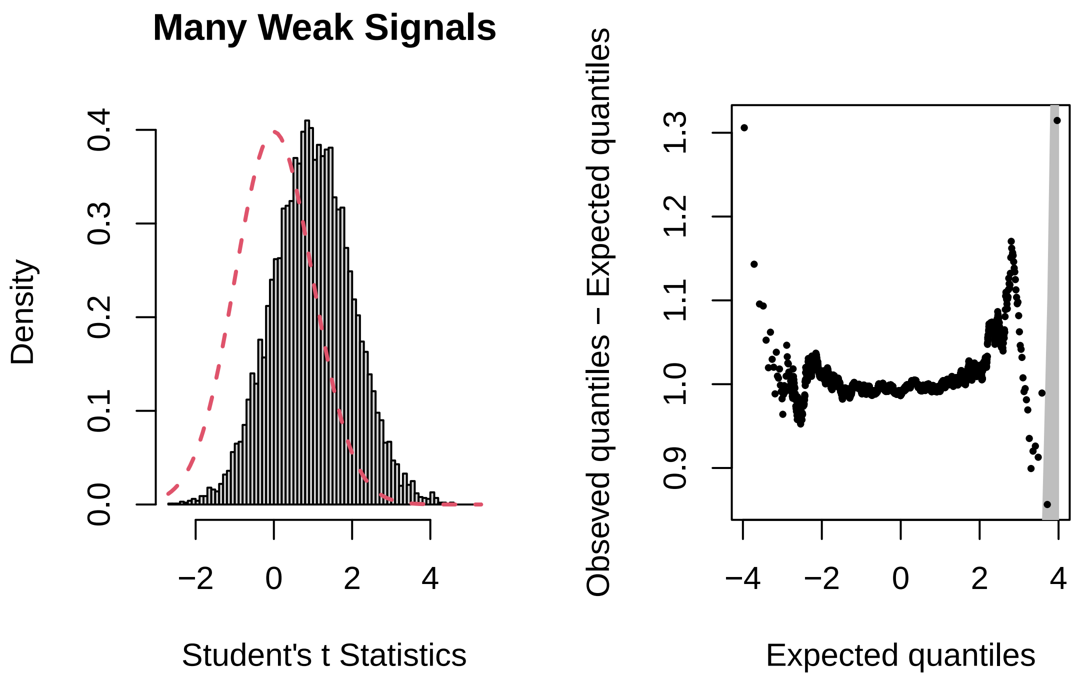
```
mean(fisher > qchisq(1 - alpha, 2 * p))
```

```
## [1] 0.53
```

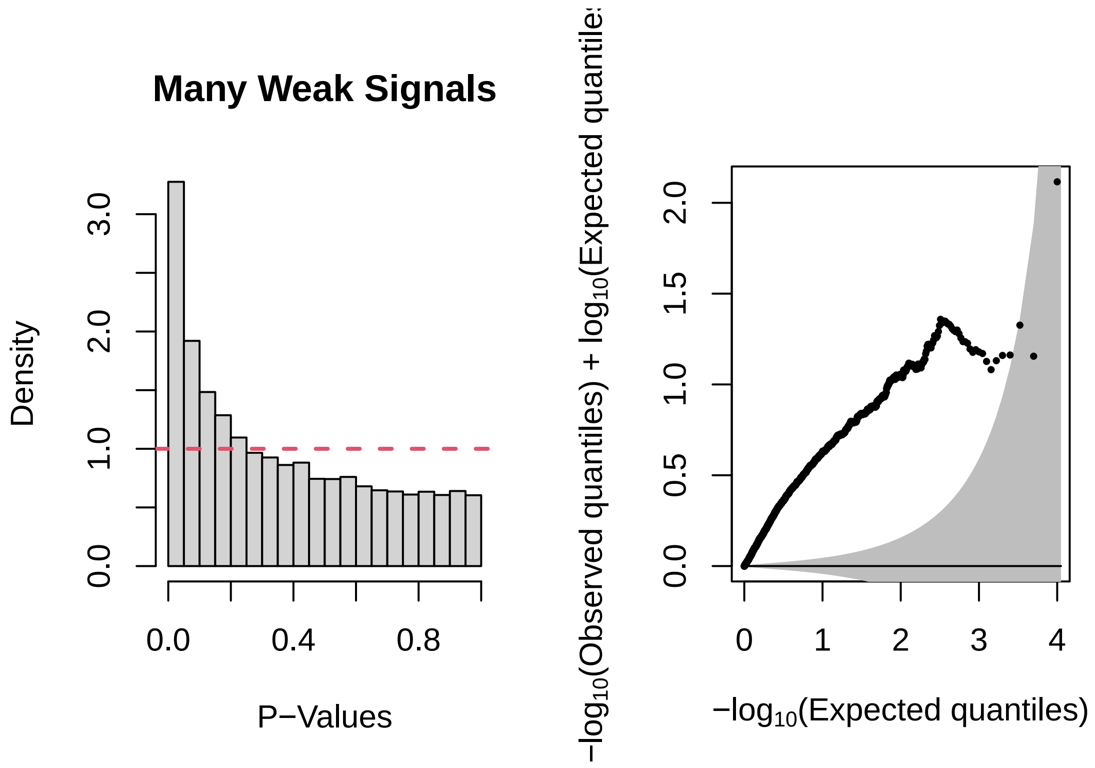
```
mean(simes > 0)
```

```
## [1] 0.308
```

```
par(mfrow = c(1, 2))
hist(t, "FD", freq = FALSE, main = "Many Weak Signals", xlab = "Student's t Statistics")
curve(dt(x, n - 1), add = TRUE, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(t), qt, difference = TRUE, dparams = list(df = n - 1),
    points_params = list(pch = 16, cex = 0.5))
```



```
hist(pval, "FD", freq = FALSE, main = "Many Weak Signals", xlab = "P-Values")
abline(h = 1, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(pval), qunif, difference = TRUE, log10 = TRUE, points_params = list(pch = 16,
    cex = 0.5))
```



```
library(qqconf)
n = 1000
p = 10
beta = numeric(p)
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
alpha = 0.05
nsim = 1000
t = matrix(0, p, nsim)
pval = matrix(0, p, nsim)
for (i in 1:nsim) {
    Y = X %*% beta + rnorm(n)
    fit = lm(Y ~ X)
    t[, i] = summary(fit)$coefficients[-1, 3]
    pval[, i] = summary(fit)$coefficients[-1, 4]
}
bonferroni = apply(pval, 2, min)
fisher = -2 * colSums(log(pval))
pval = apply(pval, 2, sort)
simes = colSums(pval <= (1:p) * alpha/p)
mean(bonferroni < alpha/p)
```

```
## [1] 0.049
```

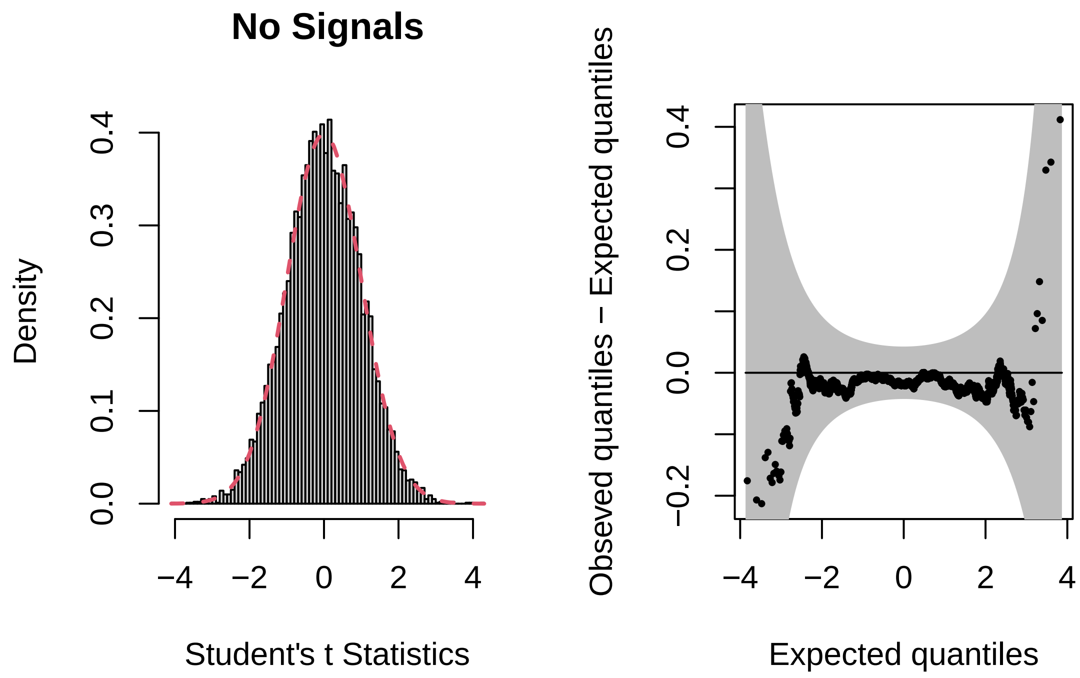
```r
mean(fisher > qchisq(1 - alpha, 2 * p))
```

```
## [1] 0.06
```

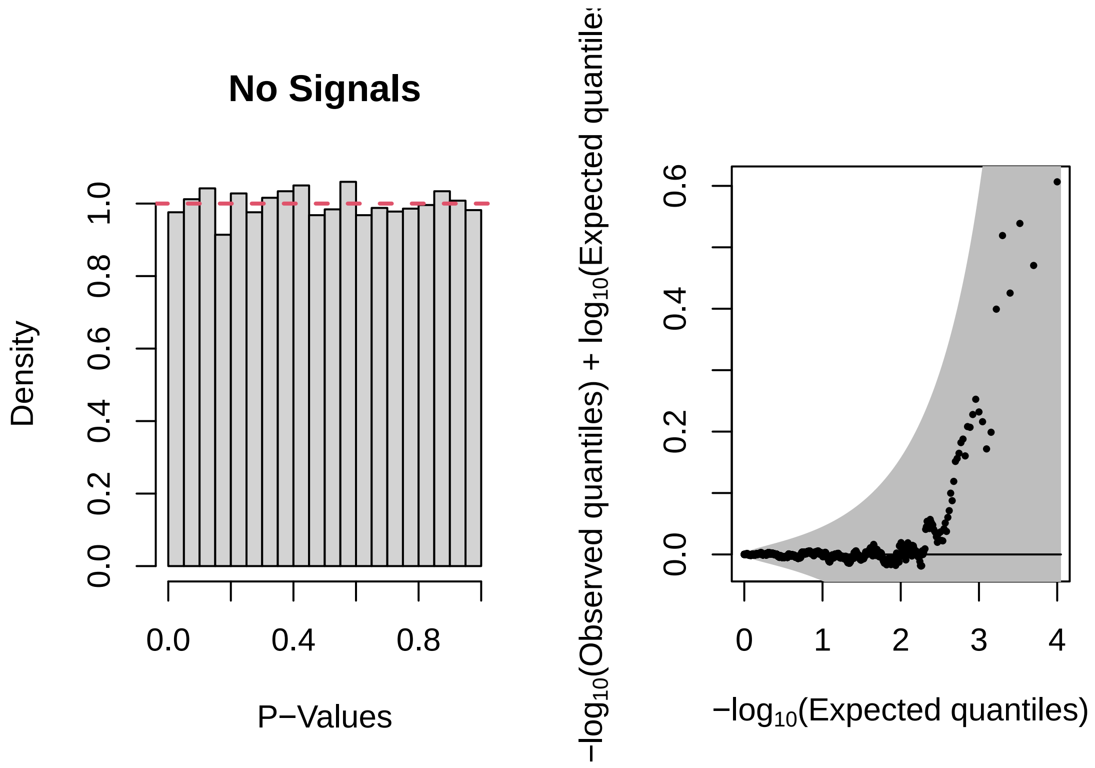
```r
mean(simes > 0)
```

```
## [1] 0.05
```

```r
par(mfrow = c(1, 2))
hist(t, "FD", freq = FALSE, main = "No Signals", xlab = "Student's t Statistics")
curve(dt(x, n - 1), add = TRUE, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(t), qt, difference = TRUE, dparams = list(df = n - p -
    1), points_params = list(pch = 16, cex = 0.5))
```



```r
hist(pval, "FD", freq = FALSE, main = "No Signals", xlab = "P-Values")
abline(h = 1, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(pval), qunif, difference = TRUE, log10 = TRUE, points_params = list(pch = 16,
    cex = 0.5))
```



```
library(qqconf)
n = 1000
p = 10
beta = c(2.5, numeric(p - 1))
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
alpha = 0.05
nsim = 1000
t = matrix(0, p, nsim)
pval = matrix(0, p, nsim)
for (i in 1:nsim) {
    Y = X %*% beta + rnorm(n)
    fit = lm(Y ~ X)
    t[, i] = summary(fit)$coefficients[-1, 3]
    pval[, i] = summary(fit)$coefficients[-1, 4]
}
bonferroni = apply(pval, 2, min)
fisher = -2 * colSums(log(pval))
pval = apply(pval, 2, sort)
simes = colSums(pval <= (1:p) * alpha/p)
mean(bonferroni < alpha/p)
```

```
## [1] 0.391
```

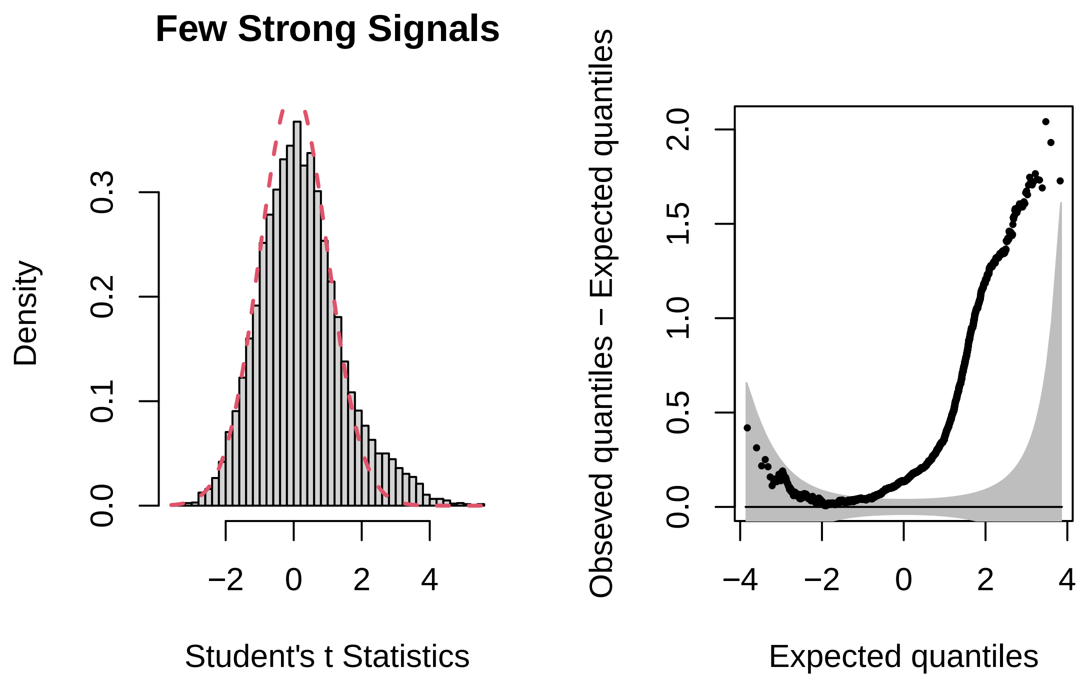
```
mean(fisher > qchisq(1 - alpha, 2 * p))
```

```
## [1] 0.302
```

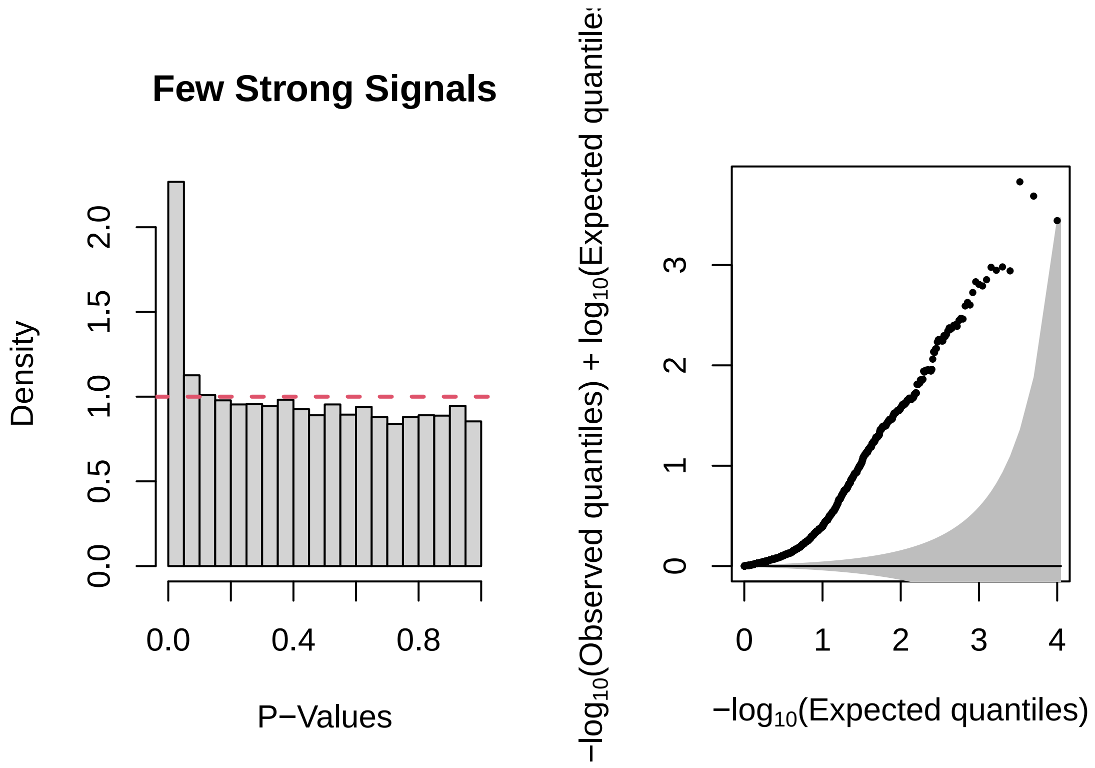
```
mean(simes > 0)
```

```
## [1] 0.395
```

```
par(mfrow = c(1, 2))
hist(t, "FD", freq = FALSE, main = "Few Strong Signals", xlab = "Student's t Statistics")
curve(dt(x, n - 1), add = TRUE, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(t), qt, difference = TRUE, dparams = list(df = n - p -
    1), points_params = list(pch = 16, cex = 0.5))
```



```
hist(pval, "FD", freq = FALSE, main = "Few Strong Signals", xlab = "P-Values")
abline(h = 1, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(pval), qunif, difference = TRUE, log10 = TRUE, points_params = list(pch = 16,
    cex = 0.5))
```



```
library(qqconf)
n = 1000
p = 10
beta = rep(1, p)
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
alpha = 0.05
nsim = 1000
t = matrix(0, p, nsim)
pval = matrix(0, p, nsim)
for (i in 1:nsim) {
    Y = X %*% beta + rnorm(n)
    fit = lm(Y ~ X)
    t[, i] = summary(fit)$coefficients[-1, 3]
    pval[, i] = summary(fit)$coefficients[-1, 4]
}
bonferroni = apply(pval, 2, min)
fisher = -2 * colSums(log(pval))
pval = apply(pval, 2, sort)
simes = colSums(pval <= (1:p) * alpha/p)
mean(bonferroni < alpha/p)
```

```
## [1] 0.298
```

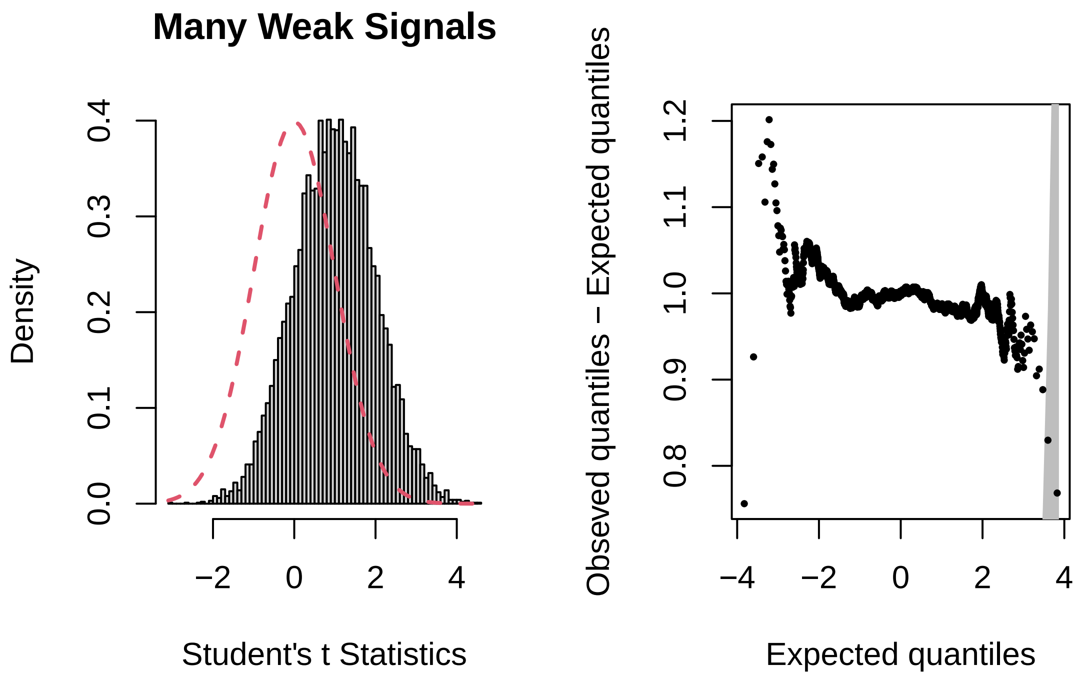
```
mean(fisher > qchisq(1 - alpha, 2 * p))
```

```
## [1] 0.553
```

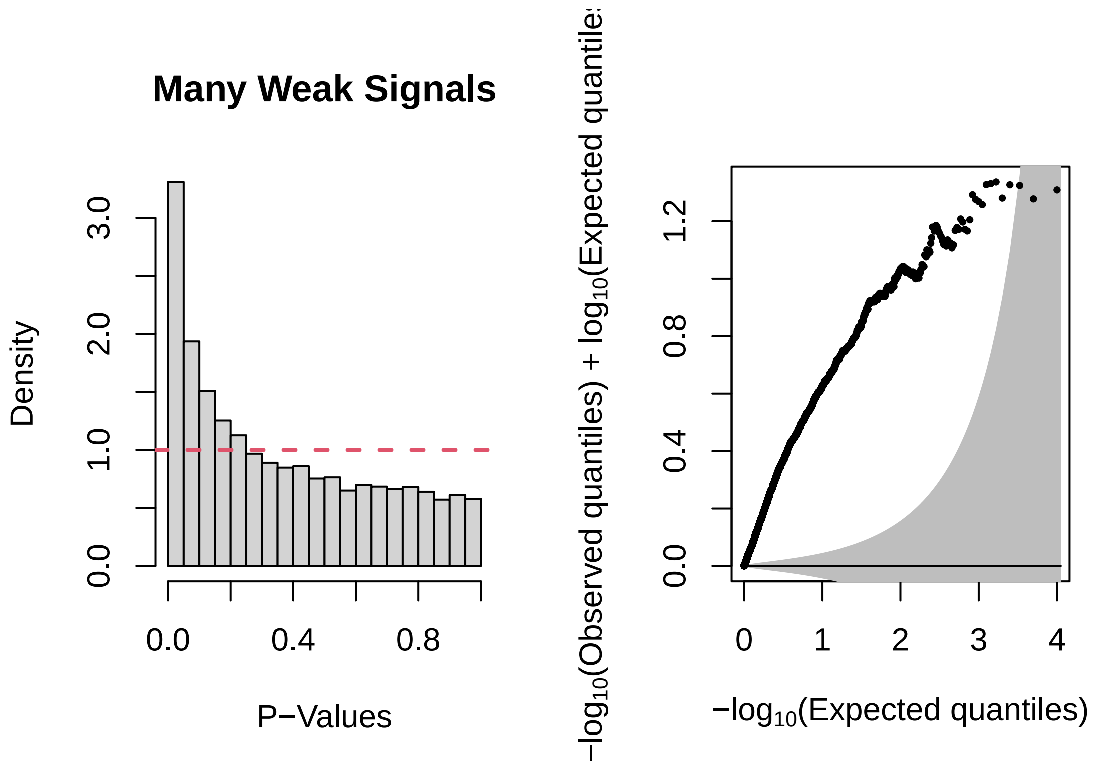
```
mean(simes > 0)
```

```
## [1] 0.313
```

```
par(mfrow = c(1, 2))
hist(t, "FD", freq = FALSE, main = "Many Weak Signals", xlab = "Student's t Statistics")
curve(dt(x, n - 1), add = TRUE, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(t), qt, difference = TRUE, dparams = list(df = n - p -
    1), points_params = list(pch = 16, cex = 0.5))
```



```
hist(pval, "FD", freq = FALSE, main = "Many Weak Signals", xlab = "P-Values")
abline(h = 1, col = 2, lty = 2, lwd = 2)
qq_conf_plot(as.vector(pval), qunif, difference = TRUE, log10 = TRUE, points_params = list(pch = 16,
    cex = 0.5))
```



Multiple Testing

```
library(xtable)
n = 100
p = 1000
beta = numeric(p)
alpha = 0.05
gamma = 0.5
nsim = 1000
uncorrected = matrix(0, p, nsim)
HB = matrix(0, p, nsim)
BH = matrix(0, p, nsim)
storey = matrix(0, p, nsim)
R = matrix(0, nsim, 4)
V = matrix(0, nsim, 4)
for (i in 1:nsim) {
    Y = matrix(rnorm(p * n, beta, 1), p)
    t = sqrt(n) * rowMeans(Y)/apply(Y, 1, sd)
    pval = 2 * pt(abs(t), n - 1, lower.tail = FALSE)
    uncorrected[, i] = pval < alpha
    R[i, 1] = sum(uncorrected[, i])
    V[i, 1] = sum(beta == 0 & uncorrected[, i])
    ord = order(order(pval))
    pval = sort(pval)
    ind = which(pval > alpha/(p - (1:p) + 1))
    R[i, 2] = ifelse(length(ind) > 0, min(ind) - 1, p)
    HB[, i] = c(rep(TRUE, R[i, 2]), logical(p - R[i, 2]))[ord]
    V[i, 2] = sum(beta == 0 & HB[, i])
    ind = which(pval <= (1:p) * alpha/p)
    R[i, 3] = ifelse(length(ind) > 0, max(ind), 0)
    BH[, i] = c(rep(TRUE, R[i, 3]), logical(p - R[i, 3]))[ord]
    V[i, 3] = sum(beta == 0 & BH[, i])
    pi0 = (1 + sum(pval > gamma))/(p * (1 - gamma))
    ind = which(pval <= pmin((1:p) * alpha/(pi0 * p), gamma))
    R[i, 4] = ifelse(length(ind) > 0, max(ind), 0)
    storey[, i] = c(rep(TRUE, R[i, 4]), logical(p - R[i, 4]))[ord]
    V[i, 4] = sum(beta == 0 & storey[, i])
}
rates = matrix(0, 4, 3)
rownames(rates) = c("Uncorrected", "Holm-Bonferroni", "Benjamini-Hochberg",
    "Storey")
colnames(rates) = c("FWER", "FDR", "pFDR")
rates[, 1] = colMeans(V > 0)
Q = ifelse(R > 0, V/R, 0)
rates[, 2] = colMeans(Q)
rates[, 3] = rates[, 2]/colMeans(R > 0)
print(xtable(rates, digits = c(0, 4, 4, 4)), comment = FALSE)
```

	FWER	FDR	pFDR
Uncorrected	1.0000	1.0000	1.0000
Holm-Bonferroni	0.0480	0.0480	1.0000
Benjamini-Hochberg	0.0480	0.0480	1.0000
Storey	0.0480	0.0480	1.0000

```
print(xtable(table(rep(beta, nsim), as.vector(uncorrected))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.949852	0.050148

```
print(xtable(table(rep(beta, nsim), as.vector(HB))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.999952	0.000048

```r
print(xtable(table(rep(beta, nsim), as.vector(BH))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.999947	0.000053

```r
print(xtable(table(rep(beta, nsim), as.vector(storey))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.999947	0.000053

```r
library(xtable)
n = 100
p = 1000
beta = c(rep(0.4, p/2), numeric(p/2))
alpha = 0.05
gamma = 0.5
nsim = 1000
uncorrected = matrix(0, p, nsim)
HB = matrix(0, p, nsim)
BH = matrix(0, p, nsim)
storey = matrix(0, p, nsim)
R = matrix(0, nsim, 4)
V = matrix(0, nsim, 4)
for (i in 1:nsim) {
    Y = matrix(rnorm(p * n, beta, 1), p)
    t = sqrt(n) * rowMeans(Y)/apply(Y, 1, sd)
    pval = 2 * pt(abs(t), n - 1, lower.tail = FALSE)
    uncorrected[, i] = pval < alpha
    R[i, 1] = sum(uncorrected[, i])
    V[i, 1] = sum(beta == 0 & uncorrected[, i])
    ord = order(order(pval))
    pval = sort(pval)
    ind = which(pval > alpha/(p - (1:p) + 1))
    R[i, 2] = ifelse(length(ind) > 0, min(ind) - 1, p)
    HB[, i] = c(rep(TRUE, R[i, 2]), logical(p - R[i, 2]))[ord]
    V[i, 2] = sum(beta == 0 & HB[, i])
    ind = which(pval <= (1:p) * alpha/p)
    R[i, 3] = ifelse(length(ind) > 0, max(ind), 0)
    BH[, i] = c(rep(TRUE, R[i, 3]), logical(p - R[i, 3]))[ord]
    V[i, 3] = sum(beta == 0 & BH[, i])
    pi0 = (1 + sum(pval > gamma))/(p * (1 - gamma))
    ind = which(pval <= pmin((1:p) * alpha/(pi0 * p), gamma))
    R[i, 4] = ifelse(length(ind) > 0, max(ind), 0)
    storey[, i] = c(rep(TRUE, R[i, 4]), logical(p - R[i, 4]))[ord]
    V[i, 4] = sum(beta == 0 & storey[, i])
}
rates = matrix(0, 4, 3)
rownames(rates) = c("Uncorrected", "Holm-Bonferroni", "Benjamini-Hochberg",
    "Storey")
colnames(rates) = c("FWER", "FDR", "pFDR")
rates[, 1] = colMeans(V > 0)
Q = ifelse(R > 0, V/R, 0)
rates[, 2] = colMeans(Q)
rates[, 3] = rates[, 2]/colMeans(R > 0)
print(xtable(rates, digits = c(0, 4, 4, 4)), comment = FALSE)
```

	FWER	FDR	pFDR
Uncorrected	1.0000	0.0489	0.0489
Holm-Bonferroni	0.0290	0.0001	0.0001
Benjamini-Hochberg	1.0000	0.0252	0.0252
Storey	1.0000	0.0503	0.0503

```
print(xtable(table(rep(beta, nsim), as.vector(uncorrected))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.474817	0.025183
0.4	0.011286	0.488714

```
print(xtable(table(rep(beta, nsim), as.vector(HB))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.499971	0.000029
0.4	0.280810	0.219190

```
print(xtable(table(rep(beta, nsim), as.vector(BH))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.487623	0.012377
0.4	0.022262	0.477738

```
print(xtable(table(rep(beta, nsim), as.vector(storey))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.474034	0.025966
0.4	0.010989	0.489011

```
library(xtable)
n = 100
p = 1000
beta = rep(0.3, p)
alpha = 0.05
gamma = 0.5
nsim = 1000
uncorrected = matrix(0, p, nsim)
HB = matrix(0, p, nsim)
BH = matrix(0, p, nsim)
storey = matrix(0, p, nsim)
R = matrix(0, nsim, 4)
V = matrix(0, nsim, 4)
for (i in 1:nsim) {
    Y = matrix(rnorm(p * n, beta, 1), p)
    t = sqrt(n) * rowMeans(Y)/apply(Y, 1, sd)
    pval = 2 * pt(abs(t), n - 1, lower.tail = FALSE)
    uncorrected[, i] = pval < alpha
    R[i, 1] = sum(uncorrected[, i])
    V[i, 1] = sum(beta == 0 & uncorrected[, i])
    ord = order(order(pval))
    pval = sort(pval)
    ind = which(pval > alpha/(p - (1:p) + 1))
    R[i, 2] = ifelse(length(ind) > 0, min(ind) - 1, p)
    HB[, i] = c(rep(TRUE, R[i, 2]), logical(p - R[i, 2]))[ord]
    V[i, 2] = sum(beta == 0 & HB[, i])
    ind = which(pval <= (1:p) * alpha/p)
    R[i, 3] = ifelse(length(ind) > 0, max(ind), 0)
    BH[, i] = c(rep(TRUE, R[i, 3]), logical(p - R[i, 3]))[ord]
    V[i, 3] = sum(beta == 0 & BH[, i])
    pi0 = (1 + sum(pval > gamma))/(p * (1 - gamma))
    ind = which(pval <= pmin((1:p) * alpha/(pi0 * p), gamma))
    R[i, 4] = ifelse(length(ind) > 0, max(ind), 0)
    storey[, i] = c(rep(TRUE, R[i, 4]), logical(p - R[i, 4]))[ord]
    V[i, 4] = sum(beta == 0 & storey[, i])
}
```

```r
rates = matrix(0, 4, 3)
rownames(rates) = c("Uncorrected", "Holm-Bonferroni", "Benjamini-Hochberg",
    "Storey")
colnames(rates) = c("FWER", "FDR", "pFDR")
rates[, 1] = colMeans(V > 0)
Q = ifelse(R > 0, V/R, 0)
rates[, 2] = colMeans(Q)
rates[, 3] = rates[, 2]/colMeans(R > 0)
print(xtable(rates, digits = c(0, 4, 4, 4)), comment = FALSE)
```

	FWER	FDR	pFDR
Uncorrected	0.0000	0.0000	0.0000
Holm-Bonferroni	0.0000	0.0000	0.0000
Benjamini-Hochberg	0.0000	0.0000	0.0000
Storey	0.0000	0.0000	0.0000

```r
print(xtable(table(rep(beta, nsim), as.vector(uncorrected))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0.3	0.155939	0.844061

```r
print(xtable(table(rep(beta, nsim), as.vector(HB))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0.3	0.873567	0.126433

```r
print(xtable(table(rep(beta, nsim), as.vector(BH))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0.3	0.177024	0.822976

```r
print(xtable(table(rep(beta, nsim), as.vector(storey))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0.3	0.010090	0.989910

```r
library(xtable)
n = 1000
p = 100
beta = numeric(p)
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
alpha = 0.05
gamma = 0.5
nsim = 1000
uncorrected = matrix(0, p, nsim)
HB = matrix(0, p, nsim)
BH = matrix(0, p, nsim)
storey = matrix(0, p, nsim)
R = matrix(0, nsim, 4)
V = matrix(0, nsim, 4)
for (i in 1:nsim) {
    Y = X %*% beta + rnorm(n)
    fit = lm(Y ~ X)
    pval = summary(fit)$coefficients[-1, 4]
    uncorrected[, i] = pval < alpha
    R[i, 1] = sum(uncorrected[, i])
    V[i, 1] = sum(beta == 0 & uncorrected[, i])
    ord = order(order(pval))
    pval = sort(pval)
    ind = which(pval > alpha/(p - (1:p) + 1))
    R[i, 2] = ifelse(length(ind) > 0, min(ind) - 1, p)
    HB[, i] = c(rep(TRUE, R[i, 2]), logical(p - R[i, 2]))[ord]
    V[i, 2] = sum(beta == 0 & HB[, i])
    ind = which(pval <= (1:p) * alpha/p)
    R[i, 3] = ifelse(length(ind) > 0, max(ind), 0)
    BH[, i] = c(rep(TRUE, R[i, 3]), logical(p - R[i, 3]))[ord]
    V[i, 3] = sum(beta == 0 & BH[, i])
    pi0 = (1 + sum(pval > gamma))/(p * (1 - gamma))
    ind = which(pval <= pmin((1:p) * alpha/(pi0 * p), gamma))
    R[i, 4] = ifelse(length(ind) > 0, max(ind), 0)
    storey[, i] = c(rep(TRUE, R[i, 4]), logical(p - R[i, 4]))[ord]
    V[i, 4] = sum(beta == 0 & storey[, i])
}
rates = matrix(0, 4, 3)
rownames(rates) = c("Uncorrected", "Holm-Bonferroni", "Benjamini-Hochberg",
    "Storey")
colnames(rates) = c("FWER", "FDR", "pFDR")
rates[, 1] = colMeans(V > 0)
```

```
Q = ifelse(R > 0, V/R, 0)
rates[, 2] = colMeans(Q)
rates[, 3] = rates[, 2]/colMeans(R > 0)
print(xtable(rates, digits = c(0, 4, 4, 4)), comment = FALSE)
```

	FWER	FDR	pFDR
Uncorrected	0.9910	0.9910	1.0000
Holm-Bonferroni	0.0450	0.0450	1.0000
Benjamini-Hochberg	0.0460	0.0460	1.0000
Storey	0.0470	0.0470	1.0000

```
print(xtable(table(rep(beta, nsim), as.vector(uncorrected))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.950520	0.049480

```
print(xtable(table(rep(beta, nsim), as.vector(HB))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.999550	0.000450

```
print(xtable(table(rep(beta, nsim), as.vector(BH))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.999490	0.000510

```
print(xtable(table(rep(beta, nsim), as.vector(storey))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.999470	0.000530

```
library(xtable)
n = 1000
p = 100
beta = c(rep(4, p/2), numeric(p/2))
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
alpha = 0.05
```

```r
gamma = 0.5
nsim = 1000
uncorrected = matrix(0, p, nsim)
HB = matrix(0, p, nsim)
BH = matrix(0, p, nsim)
storey = matrix(0, p, nsim)
R = matrix(0, nsim, 4)
V = matrix(0, nsim, 4)
for (i in 1:nsim) {
    Y = X %*% beta + rnorm(n)
    fit = lm(Y ~ X)
    pval = summary(fit)$coefficients[-1, 4]
    uncorrected[, i] = pval < alpha
    R[i, 1] = sum(uncorrected[, i])
    V[i, 1] = sum(beta == 0 & uncorrected[, i])
    ord = order(order(pval))
    pval = sort(pval)
    ind = which(pval > alpha/(p - (1:p) + 1))
    R[i, 2] = ifelse(length(ind) > 0, min(ind) - 1, p)
    HB[, i] = c(rep(TRUE, R[i, 2]), logical(p - R[i, 2]))[ord]
    V[i, 2] = sum(beta == 0 & HB[, i])
    ind = which(pval <= (1:p) * alpha/p)
    R[i, 3] = ifelse(length(ind) > 0, max(ind), 0)
    BH[, i] = c(rep(TRUE, R[i, 3]), logical(p - R[i, 3]))[ord]
    V[i, 3] = sum(beta == 0 & BH[, i])
    pi0 = (1 + sum(pval > gamma))/(p * (1 - gamma))
    ind = which(pval <= pmin((1:p) * alpha/(pi0 * p), gamma))
    R[i, 4] = ifelse(length(ind) > 0, max(ind), 0)
    storey[, i] = c(rep(TRUE, R[i, 4]), logical(p - R[i, 4]))[ord]
    V[i, 4] = sum(beta == 0 & storey[, i])
}
rates = matrix(0, 4, 3)
rownames(rates) = c("Uncorrected", "Holm-Bonferroni", "Benjamini-Hochberg",
    "Storey")
colnames(rates) = c("FWER", "FDR", "pFDR")
rates[, 1] = colMeans(V > 0)
Q = ifelse(R > 0, V/R, 0)
rates[, 2] = colMeans(Q)
rates[, 3] = rates[, 2]/colMeans(R > 0)
print(xtable(rates, digits = c(0, 4, 4, 4)), comment = FALSE)
```

	FWER	FDR	pFDR
Uncorrected	0.9100	0.0482	0.0482
Holm-Bonferroni	0.0340	0.0010	0.0010
Benjamini-Hochberg	0.6740	0.0241	0.0241
Storey	0.8930	0.0494	0.0494

```
print(xtable(table(rep(beta, nsim), as.vector(uncorrected))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.475080	0.024920
4	0.016760	0.483240

```
print(xtable(table(rep(beta, nsim), as.vector(HB))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.499640	0.000360
4	0.171730	0.328270

```
print(xtable(table(rep(beta, nsim), as.vector(BH))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.488130	0.011870
4	0.031680	0.468320

```
print(xtable(table(rep(beta, nsim), as.vector(storey))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
0	0.474200	0.025800
4	0.017310	0.482690

```
library(xtable)
n = 1000
p = 100
beta = rep(3, p)
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
alpha = 0.05
gamma = 0.5
```

```r
nsim = 1000
uncorrected = matrix(0, p, nsim)
HB = matrix(0, p, nsim)
BH = matrix(0, p, nsim)
storey = matrix(0, p, nsim)
R = matrix(0, nsim, 4)
V = matrix(0, nsim, 4)
for (i in 1:nsim) {
    Y = X %*% beta + rnorm(n)
    fit = lm(Y ~ X)
    pval = summary(fit)$coefficients[-1, 4]
    uncorrected[, i] = pval < alpha
    R[i, 1] = sum(uncorrected[, i])
    V[i, 1] = sum(beta == 0 & uncorrected[, i])
    ord = order(order(pval))
    pval = sort(pval)
    ind = which(pval > alpha/(p - (1:p) + 1))
    R[i, 2] = ifelse(length(ind) > 0, min(ind) - 1, p)
    HB[, i] = c(rep(TRUE, R[i, 2]), logical(p - R[i, 2]))[ord]
    V[i, 2] = sum(beta == 0 & HB[, i])
    ind = which(pval <= (1:p) * alpha/p)
    R[i, 3] = ifelse(length(ind) > 0, max(ind), 0)
    BH[, i] = c(rep(TRUE, R[i, 3]), logical(p - R[i, 3]))[ord]
    V[i, 3] = sum(beta == 0 & BH[, i])
    pi0 = (1 + sum(pval > gamma))/(p * (1 - gamma))
    ind = which(pval <= pmin((1:p) * alpha/(pi0 * p), gamma))
    R[i, 4] = ifelse(length(ind) > 0, max(ind), 0)
    storey[, i] = c(rep(TRUE, R[i, 4]), logical(p - R[i, 4]))[ord]
    V[i, 4] = sum(beta == 0 & storey[, i])
}
rates = matrix(0, 4, 3)
rownames(rates) = c("Uncorrected", "Holm-Bonferroni", "Benjamini-Hochberg",
    "Storey")
colnames(rates) = c("FWER", "FDR", "pFDR")
rates[, 1] = colMeans(V > 0)
Q = ifelse(R > 0, V/R, 0)
rates[, 2] = colMeans(Q)
rates[, 3] = rates[, 2]/colMeans(R > 0)
print(xtable(rates, digits = c(0, 4, 4, 4)), comment = FALSE)
```

	FWER	FDR	pFDR
Uncorrected	0.0000	0.0000	0.0000
Holm-Bonferroni	0.0000	0.0000	0.0000
Benjamini-Hochberg	0.0000	0.0000	0.0000
Storey	0.0000	0.0000	0.0000

```
print(xtable(table(rep(beta, nsim), as.vector(uncorrected))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
3	0.190360	0.809640

```
print(xtable(table(rep(beta, nsim), as.vector(HB))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
3	0.711010	0.288990

```
print(xtable(table(rep(beta, nsim), as.vector(BH))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
3	0.219740	0.780260

```
print(xtable(table(rep(beta, nsim), as.vector(storey))/(p * nsim), digits = c(0,
    6, 6)), comment = FALSE)
```

	0	1
3	0.015630	0.984370

```
covid = read.csv("COVID-19_Cases_US.csv")
covid = covid[covid$Confirmed > 0, ]
n = dim(covid)[1]
p0 = sum(covid$Deaths)/sum(covid$Confirmed)
print(p0)
```

```
[1] 0.05820638
```

```
alpha = 0.05
gamma = 0.5

library(xtable)
pvalless = pbinom(covid$Deaths, covid$Confirmed, p0)
```

```
unless = pvalless < alpha
print(xtable(t(table(unless))), comment = FALSE)
```

	FALSE	TRUE
1	2311	658

```
ord = order(order(pvalless))
pval = sort(pvalless)
ind = which(pval > alpha/(n - (1:n) + 1))
R = ifelse(length(ind) > 0, min(ind) - 1, n)
HBless = c(rep(TRUE, R), logical(n - R))[ord]
HBadj = numeric(n)
for (i in 1:n) {
    HBadj[i] = min(max((n - (1:i) + 1) * pval[1:i]), 1)
}
HBadj = HBadj[ord]
print(xtable(table(HBless, HBadj < alpha)), comment = FALSE)
```

	FALSE	TRUE
FALSE	2736	0
TRUE	0	233

```
HBadj = p.adjust(pvalless, "holm")
print(xtable(table(HBless, HBadj < alpha)), comment = FALSE)
```

	FALSE	TRUE
FALSE	2736	0
TRUE	0	233

```
ind = which(pval <= (1:n) * alpha/n)
R = ifelse(length(ind) > 0, max(ind), 0)
BHless = c(rep(TRUE, R), logical(n - R))[ord]
BHadj = numeric(n)
for (i in n:1) {
    BHadj[i] = min(min(n * pval[i:n]/(i:n)), 1)
}
BHadj = BHadj[ord]
print(xtable(table(BHless, BHadj < alpha)), comment = FALSE)
```

	FALSE	TRUE
FALSE	2523	0
TRUE	0	446

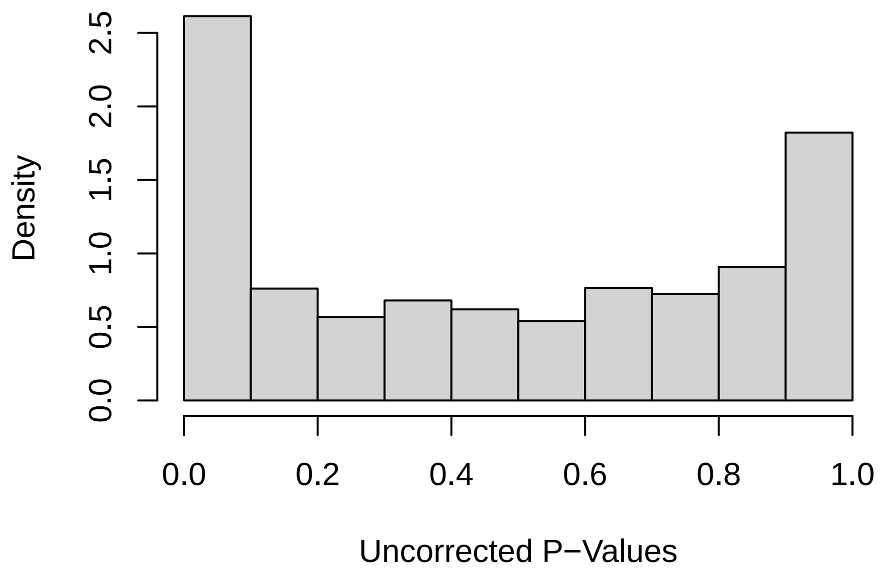
```
BHadj = p.adjust(pvalless, "BH")
print(xtable(table(BHless, BHadj < alpha)), comment = FALSE)
```

	FALSE	TRUE
FALSE	2523	0
TRUE	0	446

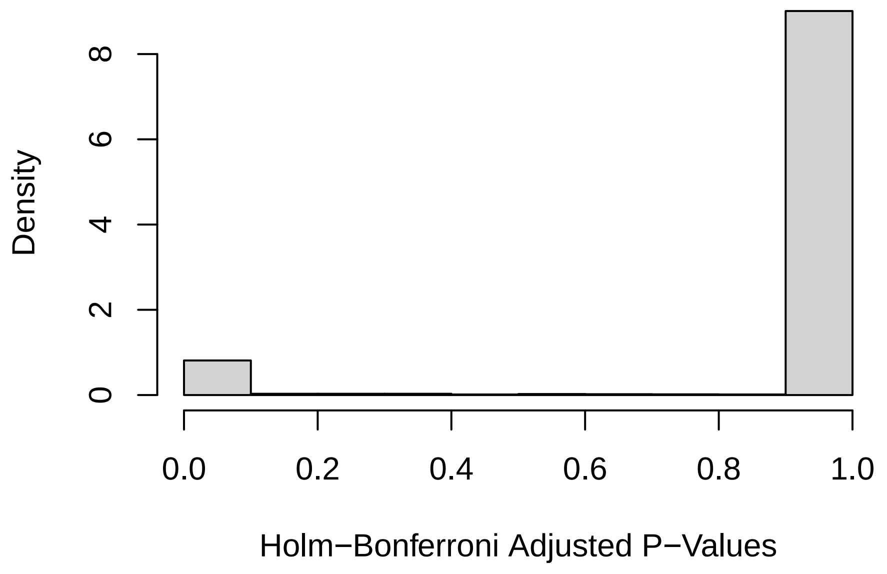
```
pi0 = (1 + sum(pval > gamma))/(n * (1 - gamma))
ind = which(pval <= pmin((1:n) * alpha/(pi0 * n), gamma))
R = ifelse(length(ind) > 0, max(ind), 0)
Storeyless = c(rep(TRUE, R), logical(n - R))[ord]
Storeyadj = numeric(n)
for (i in n:1) {
    Storeyadj[i] = min(min(pmax(pi0 * n/(i:n), gamma) * pval[i:n]), 1)
}
Storeyadj = Storeyadj[ord]
print(xtable(table(Storeyless, Storeyadj < alpha)), comment = FALSE)
```

	FALSE	TRUE
FALSE	2515	0
TRUE	0	454

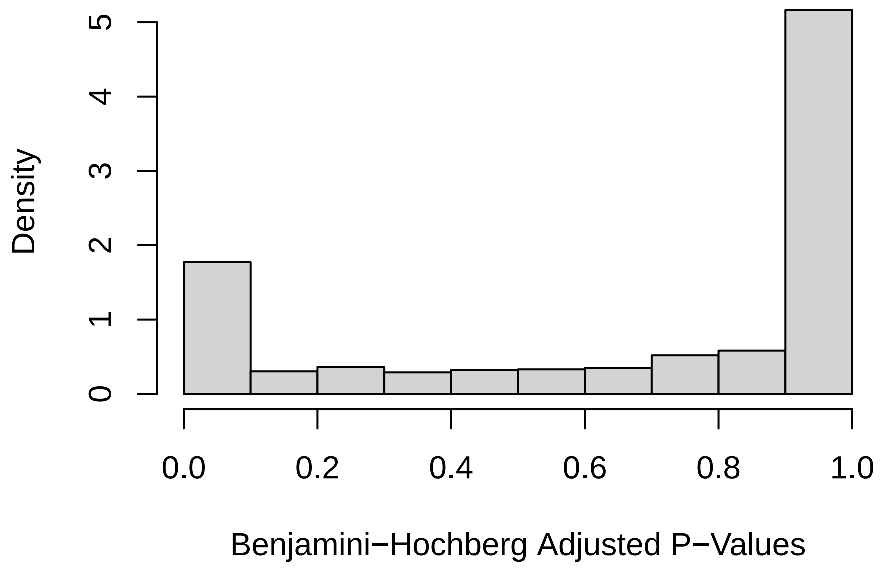
```
hist(pvalless, "FD", freq = FALSE, main = NA, xlab = "Uncorrected P-Values")
```



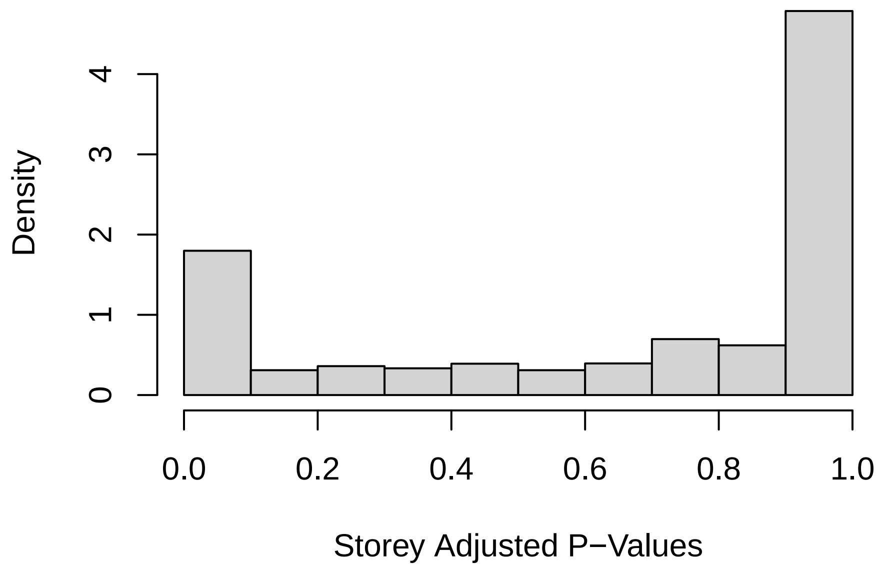
```
hist(HBadj, "FD", freq = FALSE, main = NA, xlab = "Holm-Bonferroni Adjusted P-Values")
```



```
hist(BHadj, "FD", freq = FALSE, main = NA, xlab = "Benjamini-Hochberg Adjusted P-Values")
```



```
hist(Storeyadj, "FD", freq = FALSE, main = NA, xlab = "Storey Adjusted P-Values")
```



```r
pvalgreater = 1 - pvalless + dbinom(covid$Deaths, covid$Confirmed, p0)
ungreater = pvalgreater < alpha
print(xtable(t(table(ungreater))), comment = FALSE)
```

	FALSE	TRUE
1	2724	245

```r
ord = order(order(pvalgreater))
pval = sort(pvalgreater)
ind = which(pval > alpha/(n - (1:n) + 1))
R = ifelse(length(ind) > 0, min(ind) - 1, n)
HBgreater = c(rep(TRUE, R), logical(n - R))[ord]
HBadj = numeric(n)
for (i in 1:n) {
    HBadj[i] = min(max((n - (1:i) + 1) * pval[1:i]), 1)
}
HBadj = HBadj[ord]
print(xtable(table(HBgreater, HBadj < alpha)), comment = FALSE)
```

	FALSE	TRUE
FALSE	2888	0
TRUE	0	81

```r
HBadj = p.adjust(pvalgreater, "holm")
print(xtable(table(HBgreater, HBadj < alpha)), comment = FALSE)
```

	FALSE	TRUE
FALSE	2888	0
TRUE	0	81

```r
ind = which(pval <= (1:n) * alpha/n)
R = ifelse(length(ind) > 0, max(ind), 0)
BHgreater = c(rep(TRUE, R), logical(n - R))[ord]
BHadj = numeric(n)
for (i in n:1) {
    BHadj[i] = min(min(n * pval[i:n]/(i:n)), 1)
}
BHadj = BHadj[ord]
print(xtable(table(BHgreater, BHadj < alpha)), comment = FALSE)
```

	FALSE	TRUE
FALSE	2827	0
TRUE	0	142

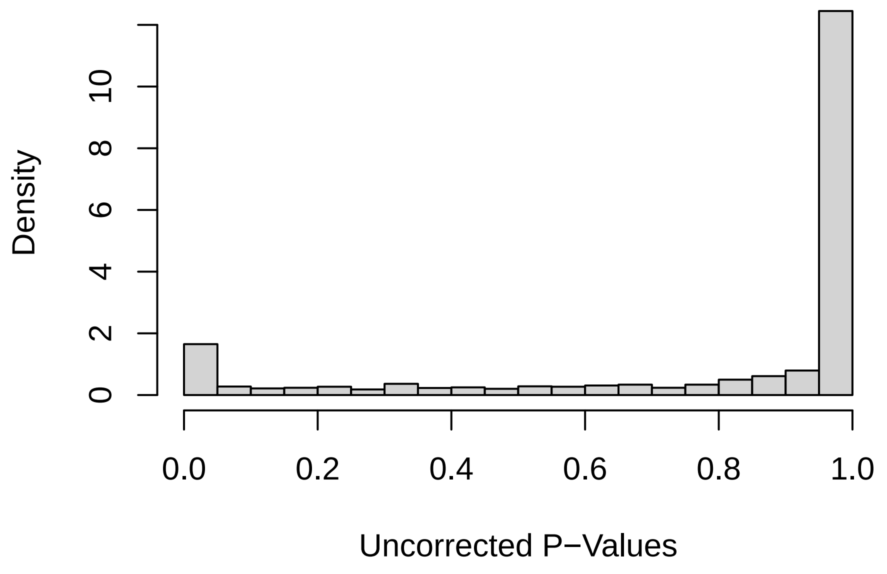
```r
BHadj = p.adjust(pvalgreater, "BH")
print(xtable(table(BHgreater, BHadj < alpha)), comment = FALSE)
```

	FALSE	TRUE
FALSE	2827	0
TRUE	0	142

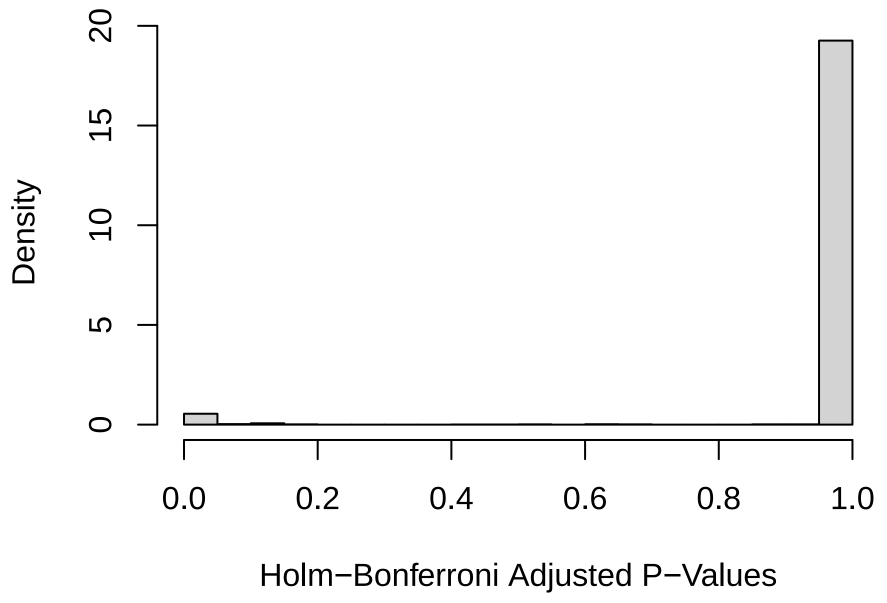
```r
pi0 = (1 + sum(pval > gamma))/(n * (1 - gamma))
ind = which(pval <= pmin((1:n) * alpha/(pi0 * n), gamma))
R = ifelse(length(ind) > 0, max(ind), 0)
Storeygreater = c(rep(TRUE, R), logical(n - R))[ord]
Storeyadj = numeric(n)
for (i in n:1) {
    Storeyadj[i] = min(min(pmax(pi0 * n/(i:n), gamma) * pval[i:n]), 1)
}
Storeyadj = Storeyadj[ord]
print(xtable(table(Storeygreater, Storeyadj < alpha)), comment = FALSE)
```

	FALSE	TRUE
FALSE	2839	0
TRUE	0	130

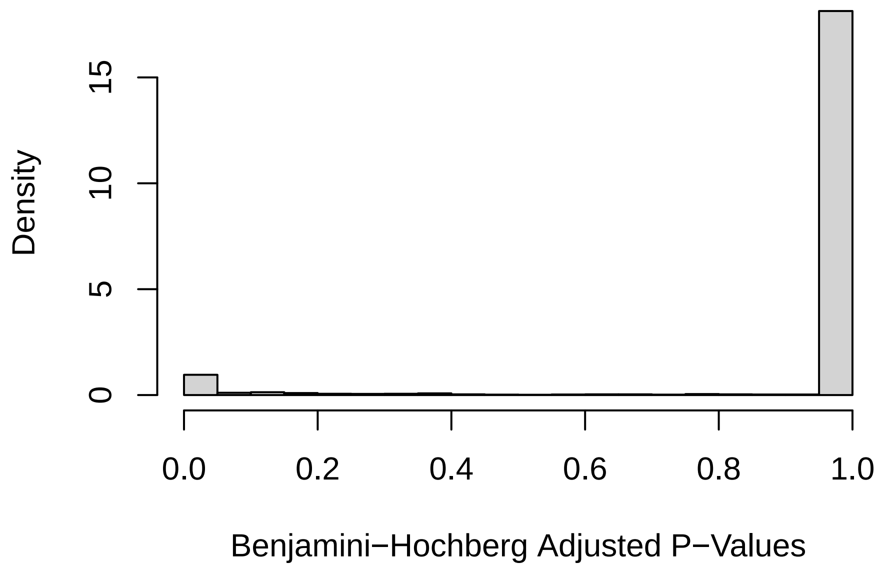
```r
hist(pvalgreater, "FD", freq = FALSE, main = NA, xlab = "Uncorrected P-Values")
```



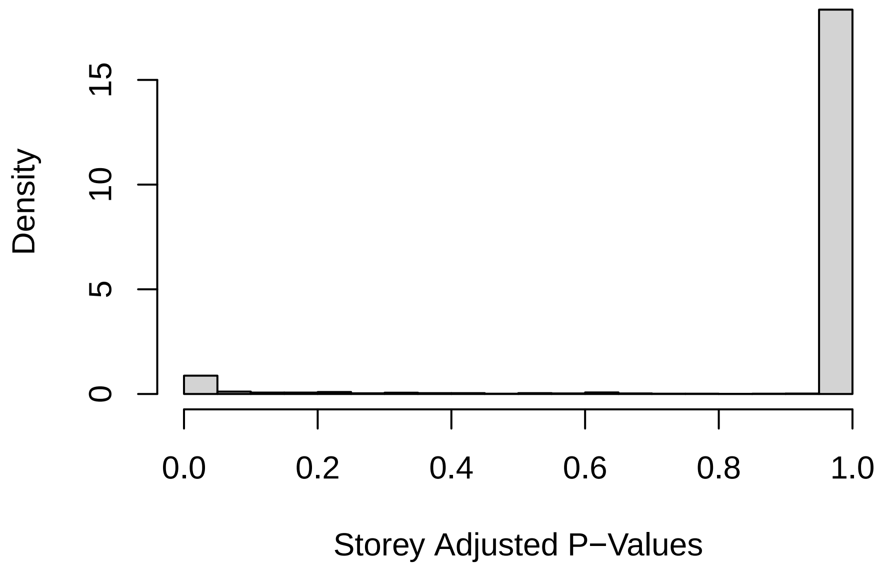
```r
hist(HBadj, "FD", freq = FALSE, main = NA, xlab = "Holm-Bonferroni Adjusted P-Values")
```



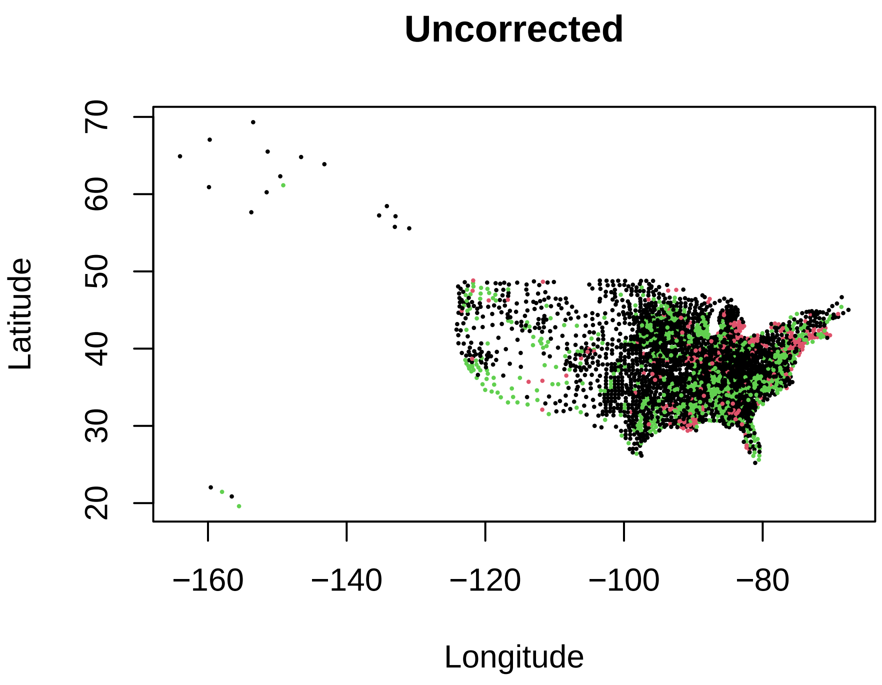
```r
hist(BHadj, "FD", freq = FALSE, main = NA, xlab = "Benjamini-Hochberg Adjusted P-Values")
```



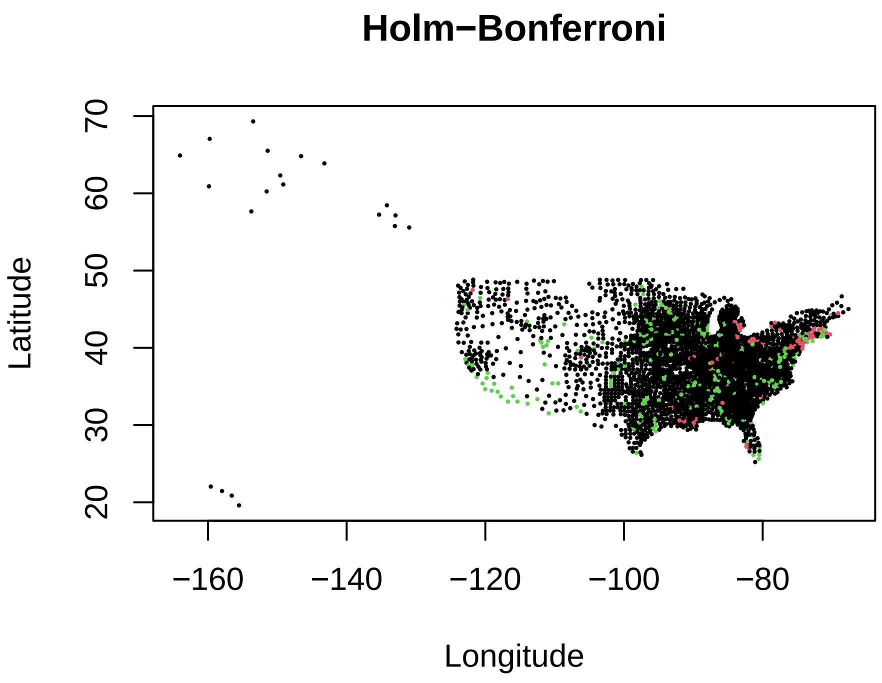
```
hist(Storeyadj, "FD", freq = FALSE, main = NA, xlab = "Storey Adjusted P-Values")
```



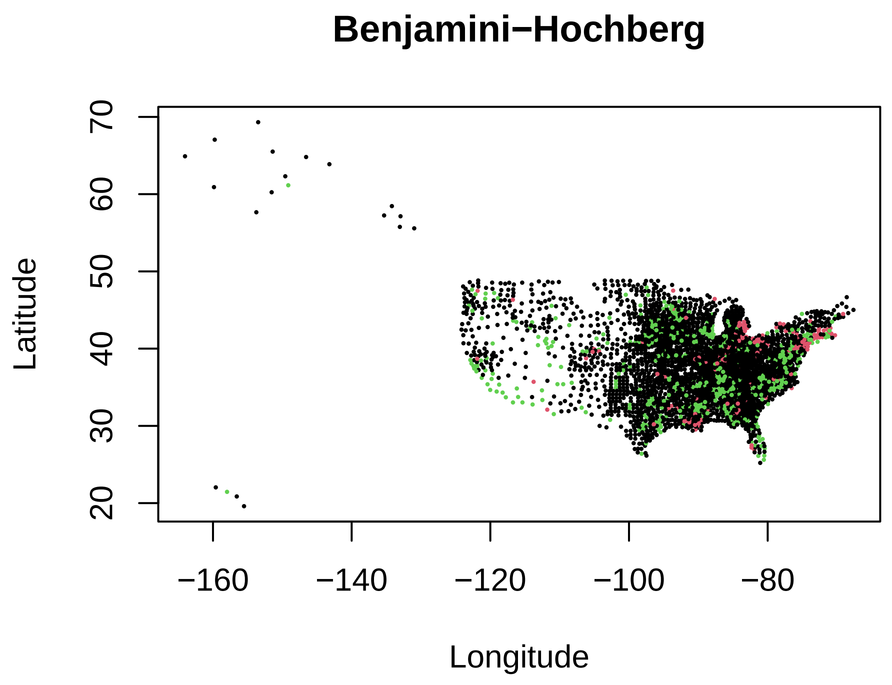
```
plot(Lat ~ Long_, covid, main = "Uncorrected", xlab = "Longitude", ylab = "Latitude",
    col = 1 + ungreater + 2 * unless, pch = 16, cex = 0.3)
```



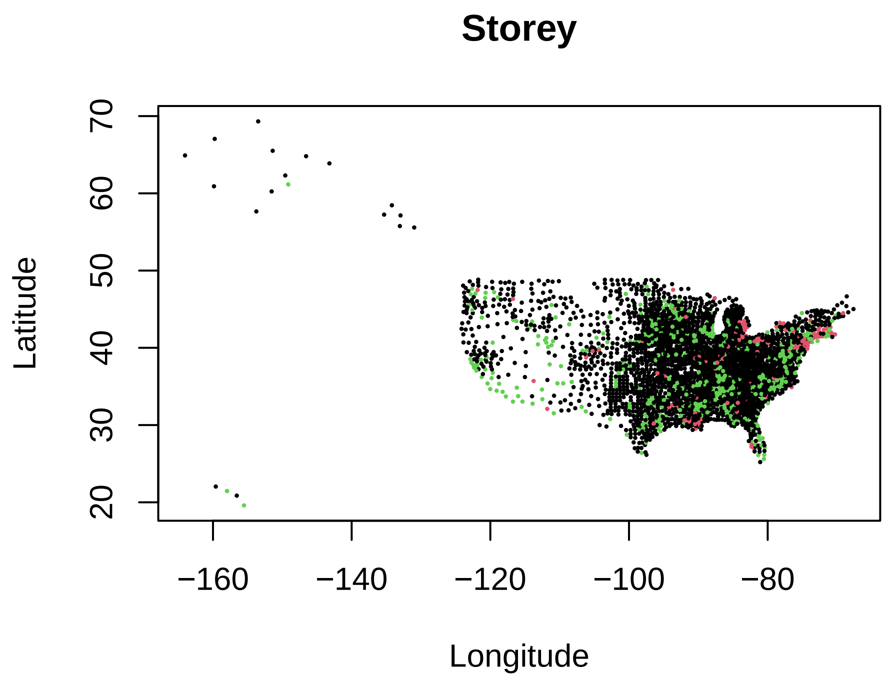
```r
plot(Lat ~ Long_, covid, main = "Holm-Bonferroni", xlab = "Longitude", ylab = "Latitude",
    col = 1 + HBgreater + 2 * HBless, pch = 16, cex = 0.3)
```



```r
plot(Lat ~ Long_, covid, main = "Benjamini-Hochberg", xlab = "Longitude", ylab = "Latitude",
    col = 1 + BHgreater + 2 * BHless, pch = 16, cex = 0.3)
```



```
plot(Lat ~ Long_, covid, main = "Storey", xlab = "Longitude", ylab = "Latitude",
    col = 1 + Storeygreater + 2 * Storeyless, pch = 16, cex = 0.3)
```



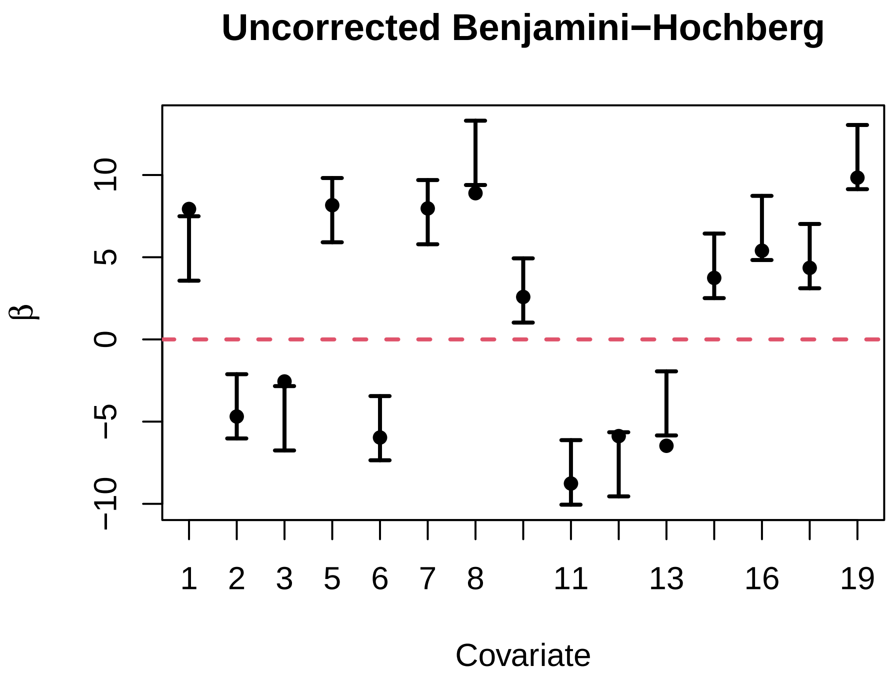
7 Selective Inference

```
n = 1000
p = 200
beta = c(runif(20, -10, 10), numeric(p - 20))
X = matrix(rnorm(n * p), n)
```

```
X = t(t(X)/sqrt(colSums(X^2)))
Y = X %*% beta + rnorm(n)
alpha = 0.05

fit = lm(Y ~ X)
pval = summary(fit)$coefficients[-1, 4]
ord = order(order(pval))
pval = sort(pval)
ind = which(pval <= (1:p) * alpha/p)
R = ifelse(length(ind) > 0, max(ind), 0)
BH = c(rep(TRUE, R), logical(p - R))[ord]
ind = which(BH)
fit = lm(Y ~ X[, ind])
CI = confint(fit)[-1, ]
mean(CI[, 1] <= beta[ind] & beta[ind] <= CI[, 2])
```

```
## [1] 0.7333333
```

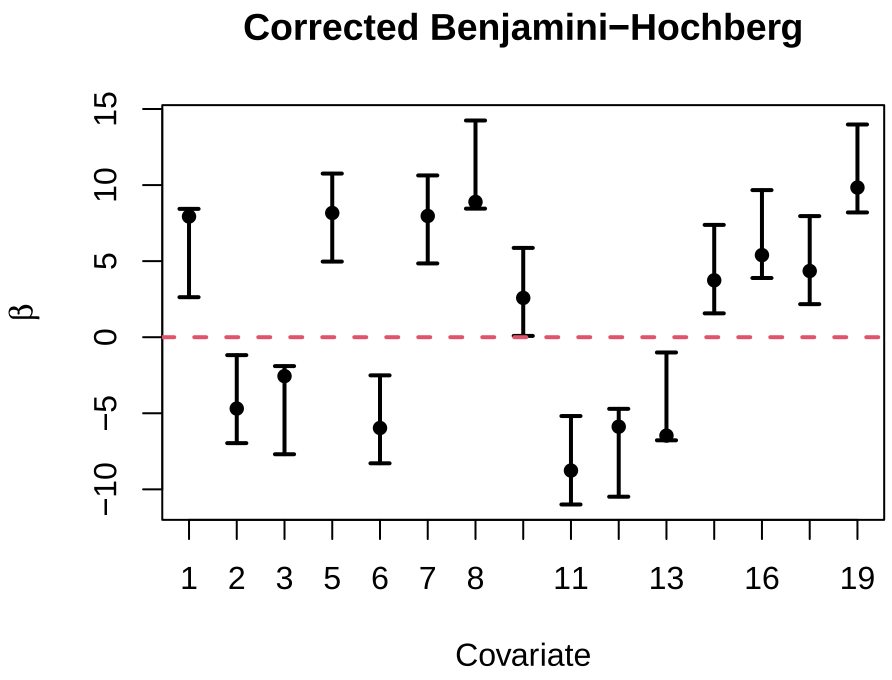
```
plot(beta[ind], ylim = range(CI), main = "Uncorrected Benjamini-Hochberg", xlab = "Covariate",
    ylab = expression(beta), xaxt = "n", pch = 16)
arrows(1:R, CI[, 1], 1:R, CI[, 2], 0.05, 90, 3, lwd = 2)
axis(1, 1:R, ind)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



```
CI = confint(fit, level = 1 - R * alpha/p)[-1, ]
mean(CI[, 1] <= beta[ind] & beta[ind] <= CI[, 2])
```

```
## [1] 1
```

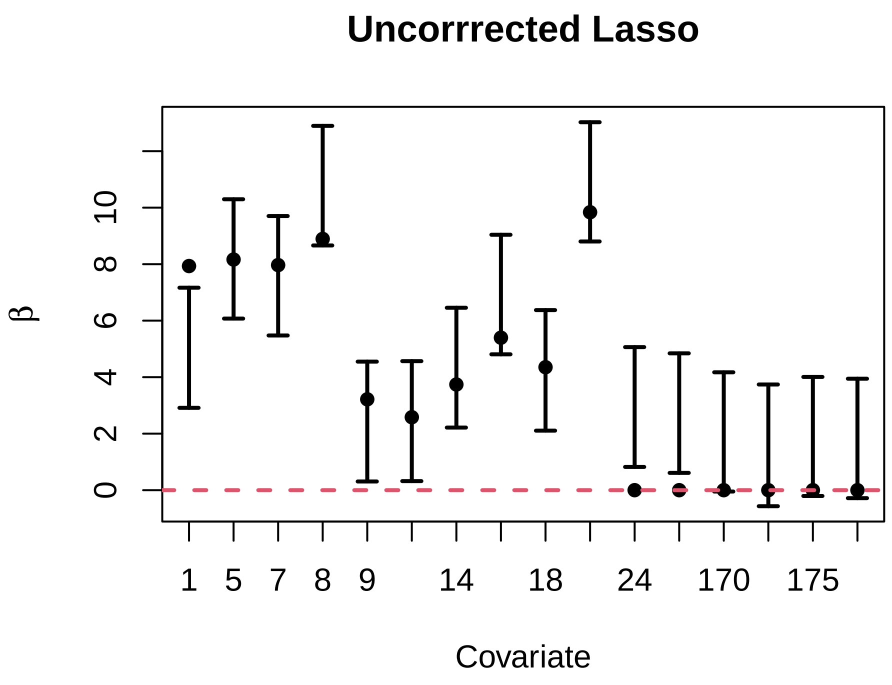
```r
plot(beta[ind], ylim = range(CI), main = "Corrected Benjamini-Hochberg", xlab = "Covariate",
    ylab = expression(beta), xaxt = "n", pch = 16)
arrows(1:R, CI[, 1], 1:R, CI[, 2], 0.05, 90, 3, lwd = 2)
axis(1, 1:R, ind)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



```r
library(glmnet)
cvlasso = cv.glmnet(X, Y)
lambdamin = cvlasso$lambda.min
betalasso = drop(glmnet(X, Y, lambda = lambdamin)$beta)
ind = which(betalasso > 0)
R = length(ind)
fit = lm(Y ~ X[, ind])
CI = confint(fit)[-1, ]
mean(CI[, 1] <= beta[ind] & beta[ind] <= CI[, 2])
```

```
## [1] 0.8125
```

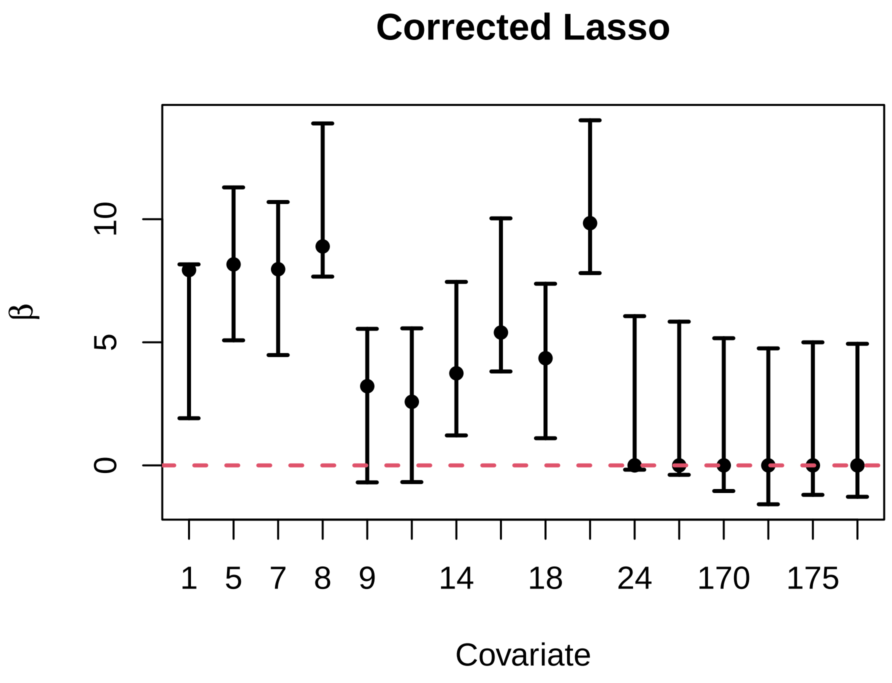
```r
plot(beta[ind], ylim = range(CI), main = "Uncorrrected Lasso", xlab = "Covariate",
    ylab = expression(beta), xaxt = "n", pch = 16)
arrows(1:R, CI[, 1], 1:R, CI[, 2], 0.05, 90, 3, lwd = 2)
axis(1, 1:R, ind)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



```r
CI = confint(fit, level = 1 - R * alpha/p)[-1, ]
mean(CI[, 1] <= beta[ind] & beta[ind] <= CI[, 2])
```

```
## [1] 1
```

```r
plot(beta[ind], ylim = range(CI), main = "Corrected Lasso", xlab = "Covariate",
    ylab = expression(beta), xaxt = "n", pch = 16)
arrows(1:R, CI[, 1], 1:R, CI[, 2], 0.05, 90, 3, lwd = 2)
axis(1, 1:R, ind)
abline(h = 0, col = 2, lty = 2, lwd = 2)
```



```r
library(leaps)
n = 1000
p = 100
beta = numeric(p)
X = matrix(rnorm(n * p), n)
X = t(t(X)/sqrt(colSums(X^2)))
train = sample(n, n/2)
nsim = 1000
covforward = numeric(nsim)
covcorrect = numeric(nsim)
tforward = matrix(0, nsim, 10)
Fforward = numeric(nsim)
covrand = numeric(nsim)
trand = matrix(0, nsim, 10)
Frand = numeric(nsim)
covtrain = numeric(nsim)
ttrain = matrix(0, nsim, 10)
Ftrain = numeric(nsim)
for (i in 1:nsim) {
    Y = X %*% beta + rnorm(n)
    ind = summary(regsubsets(X, Y, nvmax = 10, method = "forward"))$which[10,
        -1]
    fit = lm(Y ~ X[, ind])
    CI = confint(fit)[-1, ]
    covforward[i] = mean(CI[, 1] <= beta[ind] & beta[ind] <= CI[, 2])
    CI = confint(fit, level = 1 - 10 * alpha/p)[-1, ]
    covcorrect[i] = mean(CI[, 1] <= beta[ind] & beta[ind] <= CI[, 2])
    tforward[i, ] = summary(fit)$coefficients[-1, 3]
    Fforward[i] = summary(fit)$fstatistic[1]
    ind = sample(p, 10)
    fit = lm(Y ~ X[, ind])
    CI = confint(fit)[-1, ]
    covrand[i] = mean(CI[, 1] <= beta[ind] & beta[ind] <= CI[, 2])
    trand[i, ] = summary(fit)$coefficients[-1, 3]
    Frand[i] = summary(fit)$fstatistic[1]
    ind = summary(regsubsets(X[train, ], Y[train], nvmax = 10, method = "forward"))$which[10,
        -1]
    fit = lm(Y ~ X[, ind], subset = setdiff(1:n, train))
    CI = confint(fit)[-1, ]
    covtrain[i] = mean(CI[, 1] <= beta[ind] & beta[ind] <= CI[, 2])
    ttrain[i, ] = summary(fit)$coefficients[-1, 3]
    Ftrain[i] = summary(fit)$fstatistic[1]
}
```

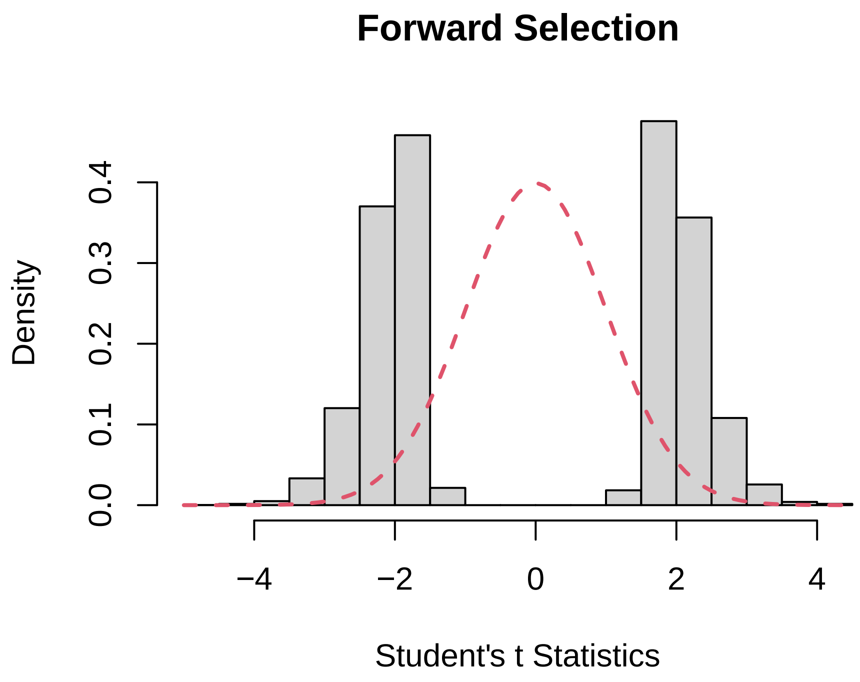
```
mean(covforward)
```

```
## [1] 0.4429
```

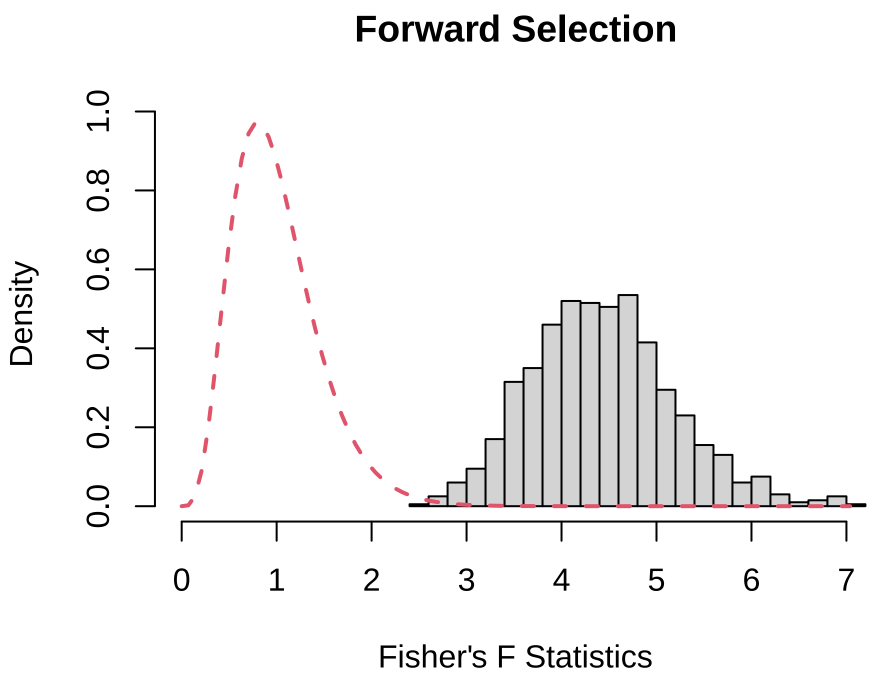
```
mean(covcorrect)
```

```
## [1] 0.9356
```

```
hist(as.vector(tforward), "FD", freq = FALSE, main = "Forward Selection", xlab = "Student's t Statistics")
curve(dt(x, n - 11), add = TRUE, col = 2, lty = 2, lwd = 2)
```



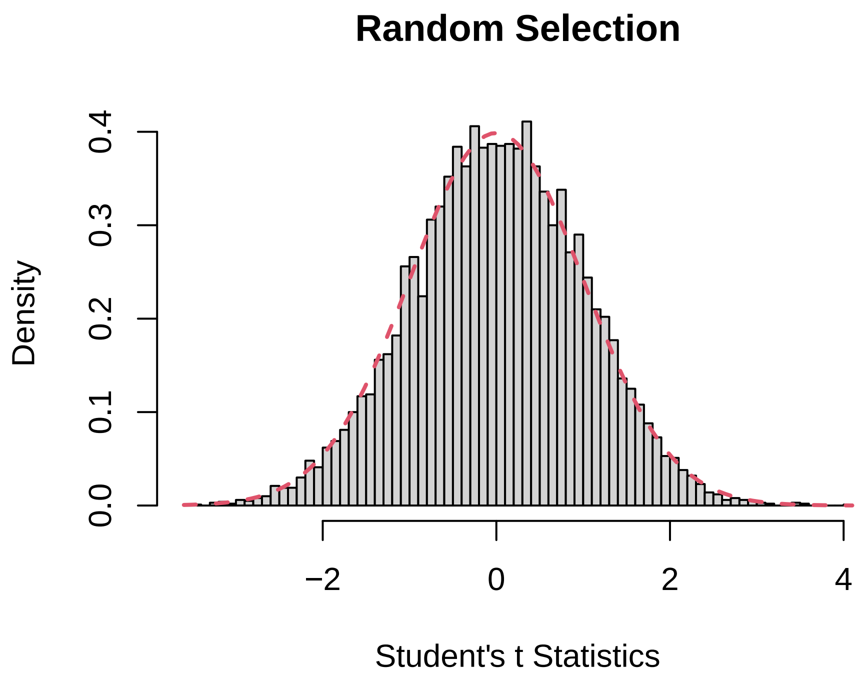
```
hist(Fforward, "FD", freq = FALSE, main = "Forward Selection", xlim = c(0, max(Fforward)),
    ylim = c(0, df(4 * (n - 11)/(5 * (n - 9)), 10, n - 11)), xlab = "Fisher's F Statistics")
curve(df(x, 10, n - 11), add = TRUE, col = 2, lty = 2, lwd = 2)
```



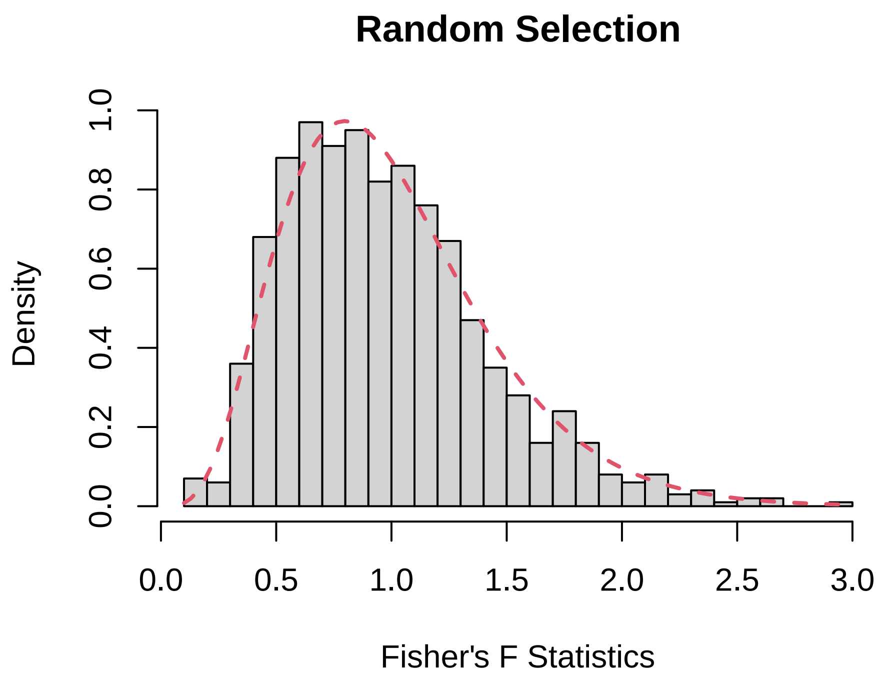
```
mean(covrand)
```

```
## [1] 0.9535
```

```
hist(as.vector(trand), "FD", freq = FALSE, main = "Random Selection", xlab = "Student's t Statistics")
curve(dt(x, n - 11), add = TRUE, col = 2, lty = 2, lwd = 2)
```



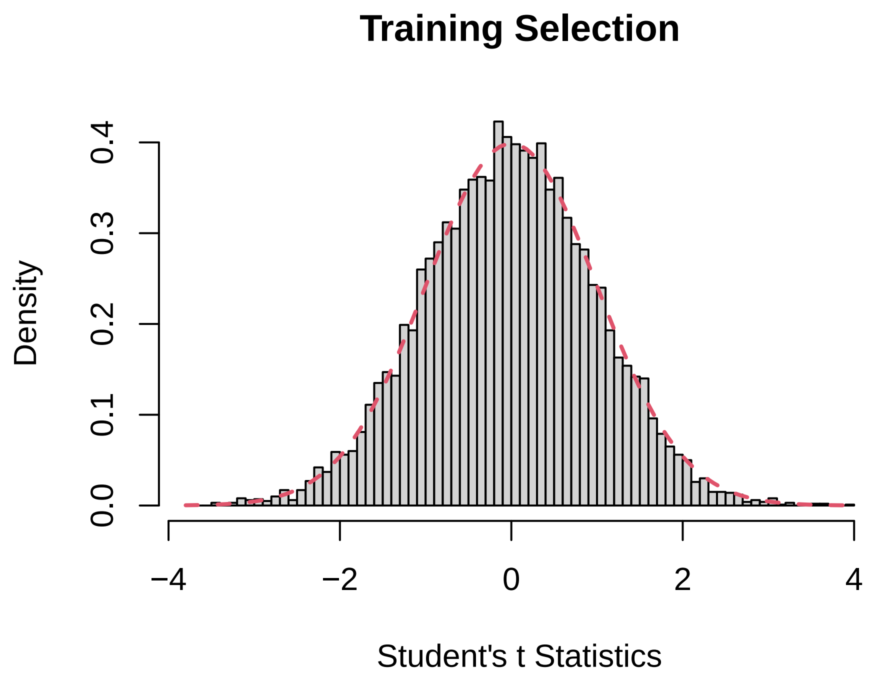
```
hist(Frand, "FD", freq = FALSE, main = "Random Selection", xlab = "Fisher's F Statistics")
curve(df(x, 10, n - 11), add = TRUE, col = 2, lty = 2, lwd = 2)
```



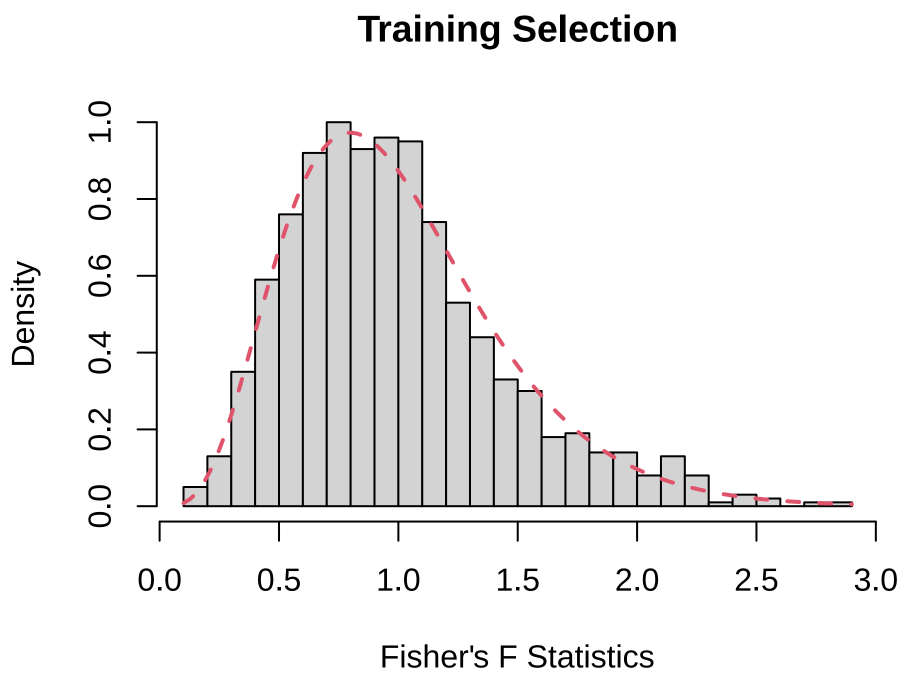
```r
mean(covtrain)
```

```
## [1] 0.9526
```

```r
hist(as.vector(ttrain), "FD", freq = FALSE, main = "Training Selection", xlab = "Student's t Statistics")
curve(dt(x, n - 11), add = TRUE, col = 2, lty = 2, lwd = 2)
```



```r
hist(Ftrain, "FD", freq = FALSE, main = "Training Selection", xlab = "Fisher's F Statistics")
curve(df(x, 10, n - 11), add = TRUE, col = 2, lty = 2, lwd = 2)
```



8 Conformal Inference

```
n = 10000
X = rnorm(n)
X = X/sqrt(sum(X^2))
Y = 2 * X + rnorm(n)
train = sample(n, n/4)
valid = sample(setdiff(1:n, train), n/4)
test = setdiff(1:n, c(train, valid))
alpha = 0.05

fit = lm(Y ~ X, subset = c(train, valid))
CI = predict(fit, data.frame(X = X[test]), interval = "prediction")[, 2:3]
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.947
```

```
fit = lm(Y ~ X, subset = train)
R = abs(Y[valid] - predict(fit, data.frame(X = X[valid])))
pred = predict(fit, data.frame(X = X[test]))
Q = quantile(R, 1 - alpha)
CI = cbind(pred - Q, pred + Q)
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9464
```

```r
R = numeric(n/2)
predplus = matrix(0, n/2, n/2)
k = 1
for (i in c(train, valid)) {
    fit = lm(Y ~ X, subset = setdiff(c(train, valid), i))
    R[k] = abs(Y[i] - predict(fit, data.frame(X = X[i])))
    predplus[k, ] = predict(fit, data.frame(X = X[test]))
    k = k + 1
}
pred = predict(fit, data.frame(X = X[test]))
Q = quantile(R, 1 - alpha)
CI = cbind(pred - Q, pred + Q)
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9452
```

```r
CI = cbind(apply(predplus - R, 2, quantile, probs = alpha), apply(predplus +
    R, 2, quantile, probs = 1 - alpha))
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9452
```

```r
n = 10000
X = rnorm(n)
X = X/sqrt(sum(X^2))
Y = 2 * X + rcauchy(n)
train = sample(n, n/4)
valid = sample(setdiff(1:n, train), n/4)
test = setdiff(1:n, c(train, valid))
alpha = 0.05

fit = lm(Y ~ X, subset = c(train, valid))
CI = predict(fit, data.frame(X = X[test]), interval = "prediction")[, 2:3]
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.998
```

```r
fit = lm(Y ~ X, subset = train)
R = abs(Y[valid] - predict(fit, data.frame(X = X[valid])))
pred = predict(fit, data.frame(X = X[test]))
Q = quantile(R, 1 - alpha)
CI = cbind(pred - Q, pred + Q)
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9424
```

```r
R = numeric(n/2)
predplus = matrix(0, n/2, n/2)
k = 1
for (i in c(train, valid)) {
    fit = lm(Y ~ X, subset = setdiff(c(train, valid), i))
    R[k] = abs(Y[i] - predict(fit, data.frame(X = X[i])))
    predplus[k, ] = predict(fit, data.frame(X = X[test]))
    k = k + 1
}
pred = predict(fit, data.frame(X = X[test]))
Q = quantile(R, 1 - alpha)
CI = cbind(pred - Q, pred + Q)
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9464
```

```r
CI = cbind(apply(predplus - R, 2, quantile, probs = alpha), apply(predplus +
    R, 2, quantile, probs = 1 - alpha))
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9464
```

```r
library(VGAM)
n = 10000
X = rnorm(n)
X = X/sqrt(sum(X^2))
Y = 2 * X + rlaplace(n)
train = sample(n, n/4)
valid = sample(setdiff(1:n, train), n/4)
test = setdiff(1:n, c(train, valid))
alpha = 0.05

fit = lm(Y ~ X, subset = c(train, valid))
CI = predict(fit, data.frame(X = X[test]), interval = "prediction")[, 2:3]
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9388
```

```r
fit = lm(Y ~ X, subset = train)
R = abs(Y[valid] - predict(fit, data.frame(X = X[valid])))
pred = predict(fit, data.frame(X = X[test]))
Q = quantile(R, 1 - alpha)
CI = cbind(pred - Q, pred + Q)
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9446
```

```r
R = numeric(n/2)
predplus = matrix(0, n/2, n/2)
k = 1
for (i in c(train, valid)) {
    fit = lm(Y ~ X, subset = setdiff(c(train, valid), i))
    R[k] = abs(Y[i] - predict(fit, data.frame(X = X[i])))
    predplus[k, ] = predict(fit, data.frame(X = X[test]))
    k = k + 1
}
pred = predict(fit, data.frame(X = X[test]))
Q = quantile(R, 1 - alpha)
CI = cbind(pred - Q, pred + Q)
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9492
```

```r
CI = cbind(apply(predplus - R, 2, quantile, probs = alpha), apply(predplus +
    R, 2, quantile, probs = 1 - alpha))
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9492
```

```r
library(sn)
n = 10000
X = rnorm(n)
X = X/sqrt(sum(X^2))
Y = 2 * X + rsn(n, alpha = 10)
train = sample(n, n/4)
valid = sample(setdiff(1:n, train), n/4)
test = setdiff(1:n, c(train, valid))
alpha = 0.05

fit = lm(Y ~ X, subset = c(train, valid))
CI = predict(fit, data.frame(X = X[test]), interval = "prediction")[, 2:3]
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.949
```

```r
fit = lm(Y ~ X, subset = train)
R = Y[valid] - predict(fit, data.frame(X = X[valid]))
pred = predict(fit, data.frame(X = X[test]))
Q = quantile(abs(R), 1 - alpha)
CI = cbind(pred - Q, pred + Q)
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9414
```

```r
CI = cbind(pred + quantile(R, alpha/2), pred + quantile(R, 1 - alpha/2))
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9482
```

```r
R = numeric(n/2)
predplus = matrix(0, n/2, n/2)
k = 1
for (i in c(train, valid)) {
    fit = lm(Y ~ X, subset = setdiff(c(train, valid), i))
    R[k] = Y[i] - predict(fit, data.frame(X = X[i]))
    predplus[k, ] = predict(fit, data.frame(X = X[test]))
    k = k + 1
}
pred = predict(fit, data.frame(X = X[test]))
Q = quantile(abs(R), 1 - alpha)
CI = cbind(pred - Q, pred + Q)
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.9416
```

```r
CI = cbind(pred + quantile(R, alpha/2), pred + quantile(R, 1 - alpha/2))
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

```
## [1] 0.951
```

```r
CI = cbind(apply(predplus - abs(R), 2, quantile, probs = alpha), apply(predplus +
    abs(R), 2, quantile, probs = 1 - alpha))
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

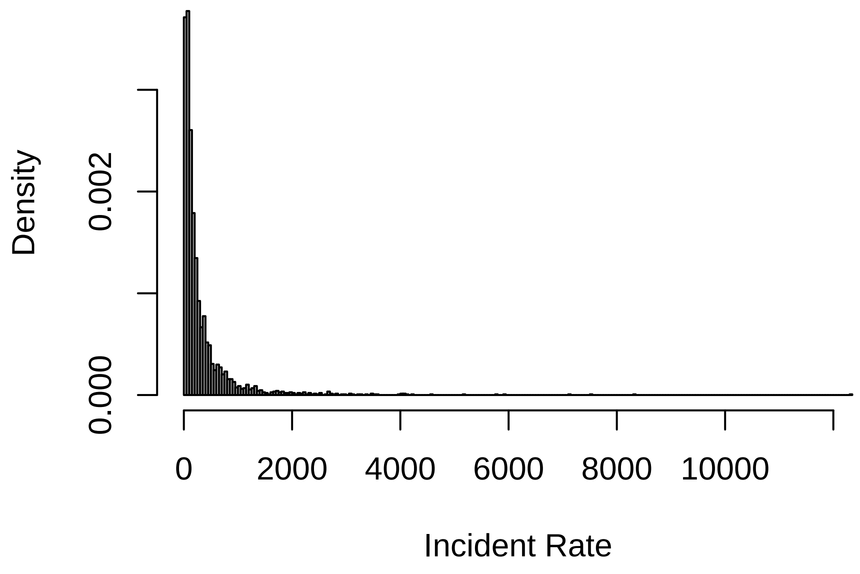
```
## [1] 0.9416
```

```r
CI = cbind(apply(predplus + R, 2, quantile, probs = alpha/2), apply(predplus +
    R, 2, quantile, probs = 1 - alpha/2))
mean(CI[, 1] <= Y[test] & Y[test] <= CI[, 2])
```

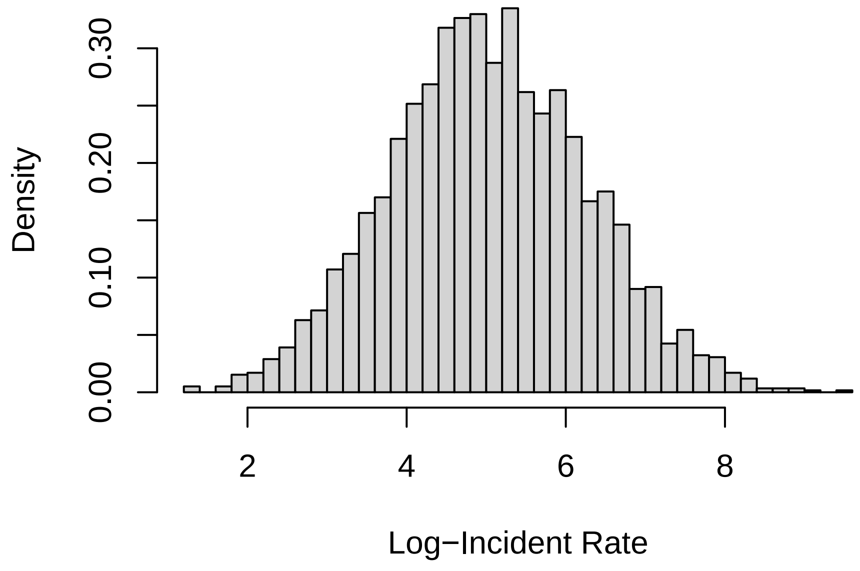
```
## [1] 0.951
```

9 Gaussian Processes

```r
library(geosphere)
covid = read.csv("COVID-19_Cases_US.csv")
covid = covid[!is.na(covid$Incident_Rate) & covid$Incident_Rate > 0, ]
hist(covid$Incident_Rate, "FD", freq = FALSE, main = NA, xlab = "Incident Rate")
```



```
hist(log(covid$Incident_Rate), "FD", freq = FALSE, main = NA, xlab = "Log-Incident Rate")
```



```
n = 500
covid = covid[sample(dim(covid)[1], n), ]
X = covid[, 8:7]
Y = log(covid$Incident_Rate)
dist = distm(X)/1000

mu = mean(Y)
print(mu)
```

```
## [1] 5.044165
```

```
sigma = sqrt(mean((Y - mu)^2))
print(sigma)
```

```
## [1] 1.274528
```

```
loglik0 = sum(dnorm(Y, mu, sigma, log = TRUE))
print(loglik0)
```

```
## [1] -830.7573
```

```
loglik = function(param, Y, dist) {
    library(mvtnorm)
    n = length(Y)
    mu = param[1]
    sigma = exp(param[2])
    lambda = exp(param[3])
    tau = exp(param[4])
    Sigma = lambda^2 * exp(-dist^2/(2 * tau^2))
    dmvnorm(Y, rep(mu, n), Sigma + sigma^2 * diag(n), log = TRUE)
}

opt = optim(c(mean(Y), log(sd(Y)/sqrt(2)), log(sd(Y)/sqrt(2)), 5), loglik, Y = Y,
    dist = dist, control = list(fnscale = -1))
mu = opt$par[1]
print(mu)
```

```
## [1] 4.794305
```

```
sigma = exp(opt$par[2])
print(sigma)
```

```
## [1] 0.8089796
```

```
lambda = exp(opt$par[3])
print(lambda)
```

```
## [1] 0.9323451
```

```
tau = exp(opt$par[4])
print(tau)
```

```
## [1] 126.9078
```

```
loglik1 = opt$value
print(loglik1)
```

```
## [1] -728.077
```

```
LR = -2 * (loglik0 - loglik1)
print(LR)
```

```
## [1] 205.3606
```

```
Sigma = lambda^2 * exp(-dist^2/(2 * tau^2)) + sigma^2 * diag(n)
```

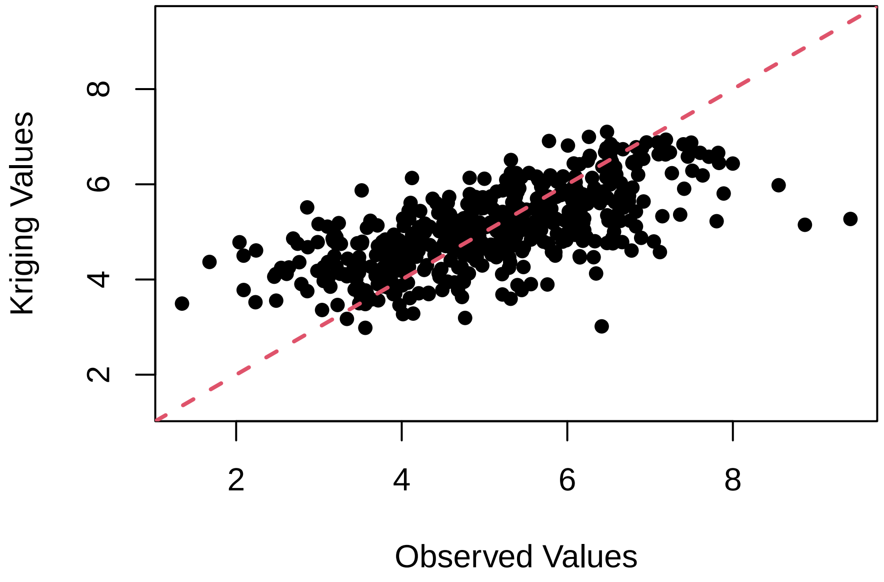
```r
impute = numeric(n)
kriging = numeric(n)
for (i in 1:n) {
    impute[i] = mean(Y[-i])
    kriging[i] = mu + crossprod(Sigma[-i, i], solve(Sigma[-i, -i], Y[-i] - mu))
}
mean((Y - impute)^2)
```

```
## [1] 1.630939
```

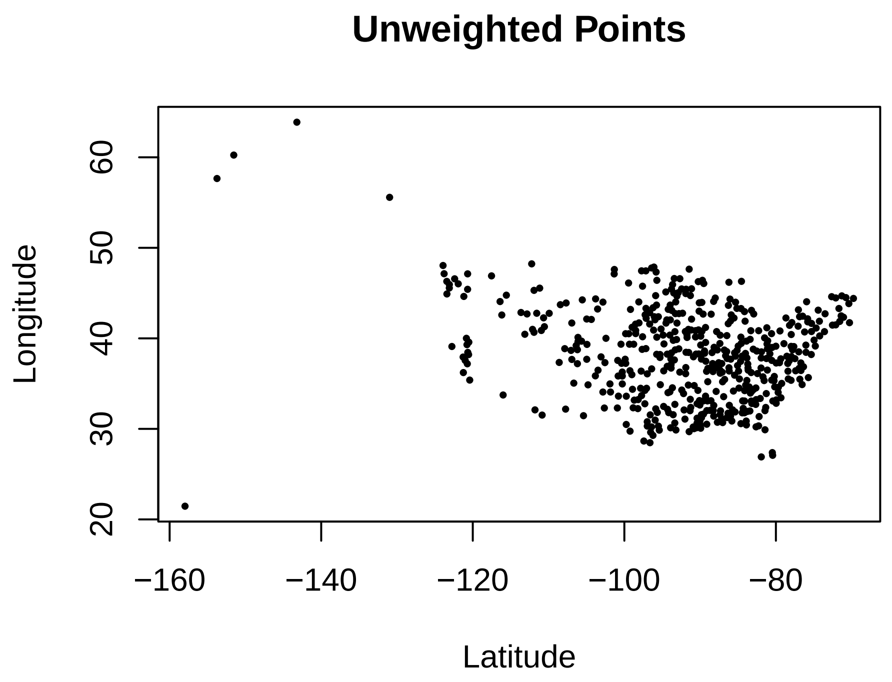
```r
mean((Y - kriging)^2)
```

```
## [1] 0.9100077
```

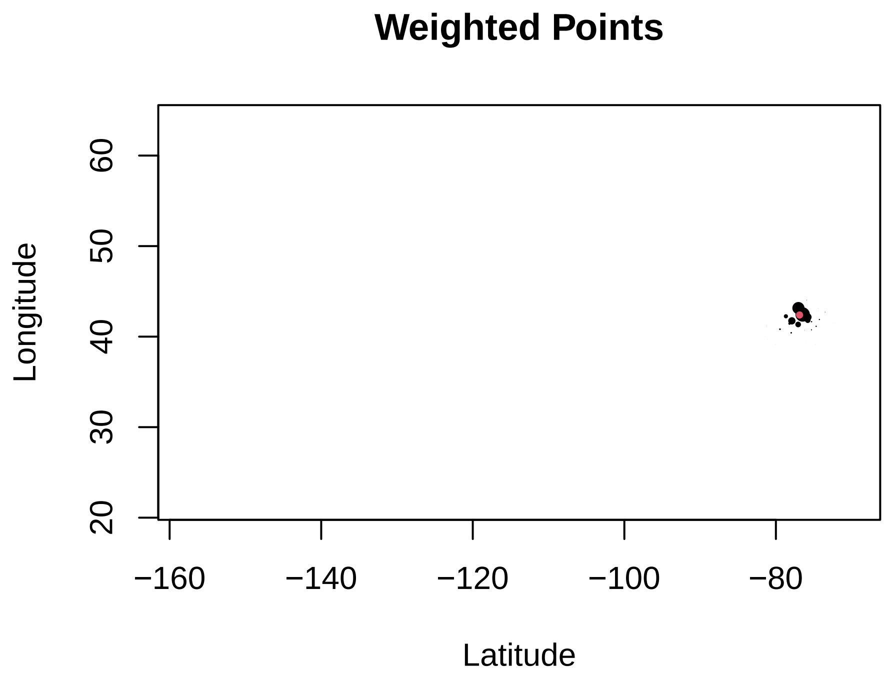
```r
plot(Y, kriging, xlim = range(Y), ylim = range(c(Y, kriging)), xlab = "Observed Values",
    ylab = "Kriging Values", pch = 16)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```



```r
weights = abs(solve(Sigma[-n, -n], Sigma[-n, n]))
weights = weights/max(weights)
plot(X, main = "Unweighted Points", xlab = "Latitude", ylab = "Longitude", pch = 16,
    cex = 0.5)
```



```r
plot(X[-n, ], main = "Weighted Points", xlab = "Latitude", ylab = "Longitude",
    pch = 16, cex = weights)
points(X[n, ], col = 2, pch = 16, cex = 0.5)
```



10 Empirical Bayes

```r
n = 1000
mu = 1
sigma = 2
theta = rnorm(n, mu, sigma)
```

```
S = sort(rexp(n, 1))
Y = rnorm(n, theta, S)

loglik = function(param, Y, S) {
    mu = param[1]
    sigma = exp(param[2])
    sum(dnorm(Y, mu, sqrt(S^2 + sigma^2), log = TRUE))
}

opt = optim(c(0, 0), loglik, Y = Y, S = S, control = list(fnscale = -1))
muhat = opt$par[1]
print(muhat)
```

```
## [1] 0.9943924
```

```
sigmahat = exp(opt$par[2])
print(sigmahat)
```

```
## [1] 1.979245
```

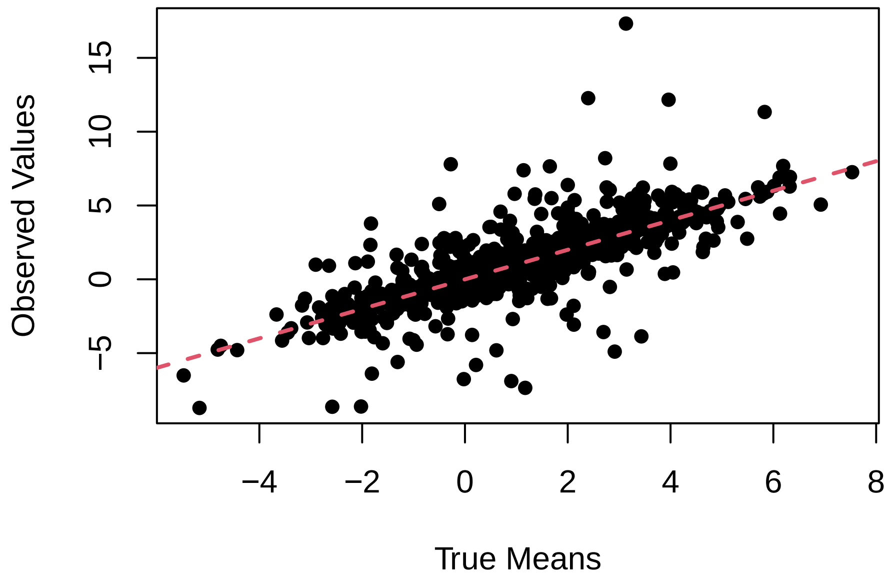
```
thetahat = (sigmahat^2 * Y + S^2 * muhat)/(sigmahat^2 + S^2)
SE = sqrt(sigmahat^2 * S^2/(sigmahat^2 + S^2))
mean((Y - theta)^2)
```

```
## [1] 2.291367
```

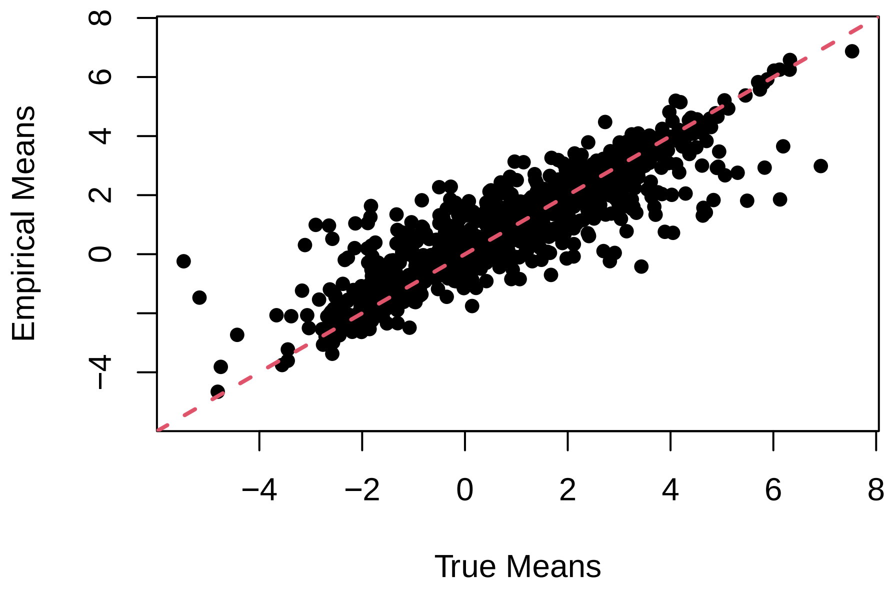
```
mean((thetahat - theta)^2)
```

```
## [1] 0.8582998
```

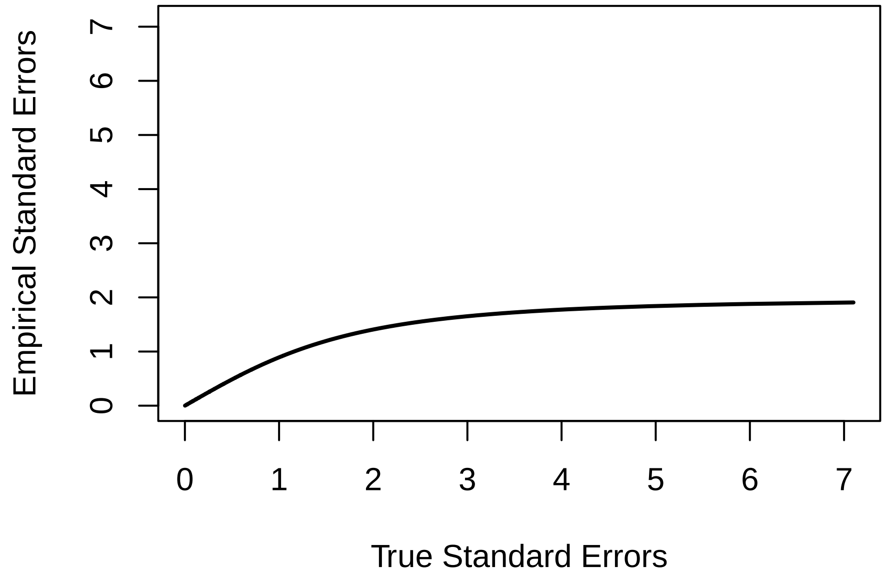
```
plot(theta, Y, xlab = "True Means", ylab = "Observed Values", pch = 16)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```



```r
plot(theta, thetahat, ylim = range(theta), xlab = "True Means", ylab = "Empirical Means",
    pch = 16)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```



```r
plot(S, SE, "l", ylim = range(S), xlab = "True Standard Errors", ylab = "Empirical Standard Errors",
    lwd = 2)
```



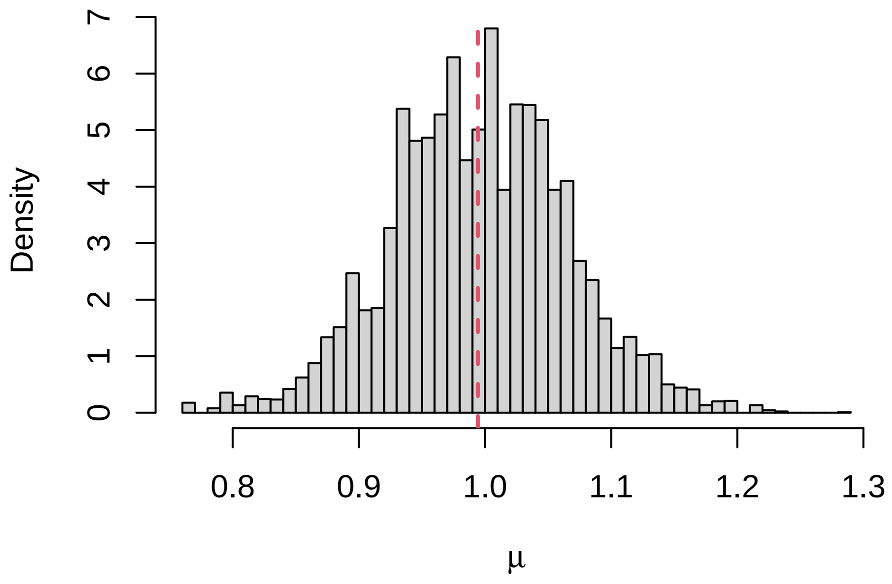
```r
musd = 0.5
etasd = 0.1
niter = 10000
musample = numeric(niter)
etasample = numeric(niter)
musample[1] = 0
etasample[1] = 0
for (i in 2:niter) {
    newmu = rnorm(1, musample[i - 1], musd)
    logA = loglik(c(newmu, etasample[i - 1]), Y, S) - loglik(c(musample[i -
        1], etasample[i - 1]), Y, S)
    musample[i] = ifelse(log(runif(1)) < logA, newmu, musample[i - 1])
    neweta = rnorm(1, etasample[i - 1], etasd)
    logA = loglik(c(musample[i], neweta), Y, S) - loglik(c(musample[i], etasample[i -
        1]), Y, S)
    etasample[i] = ifelse(log(runif(1)) < logA, neweta, etasample[i - 1])
}
nburn = 1000
musample = musample[-(1:nburn)]
mean(musample)
```

```
## [1] 0.9959566
```

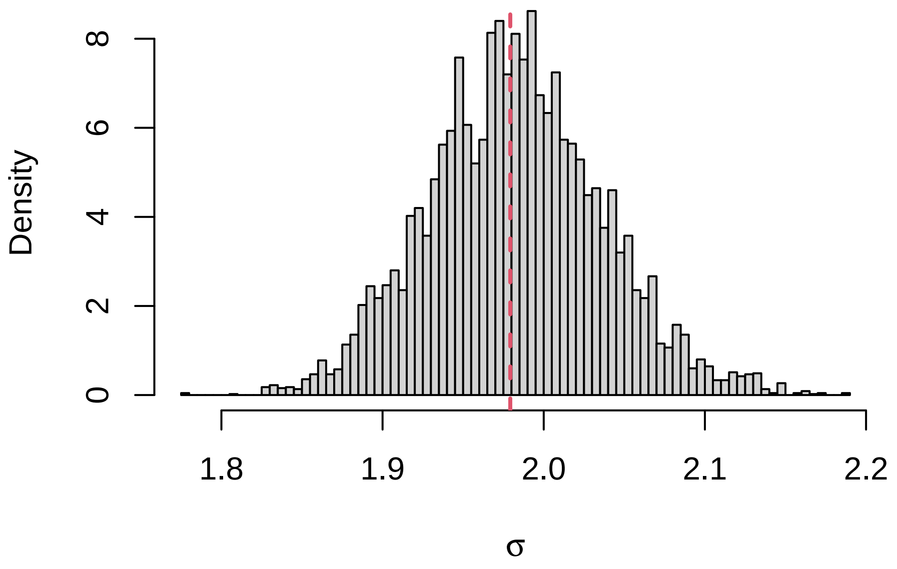
```
sigmasample = exp(etasample[-(1:nburn)])
mean(sigmasample)
```

```
## [1] 1.982614
```

```
hist(musample, "FD", freq = FALSE, main = NA, xlab = expression(mu))
abline(v = muhat, col = 2, lty = 2, lwd = 2)
```



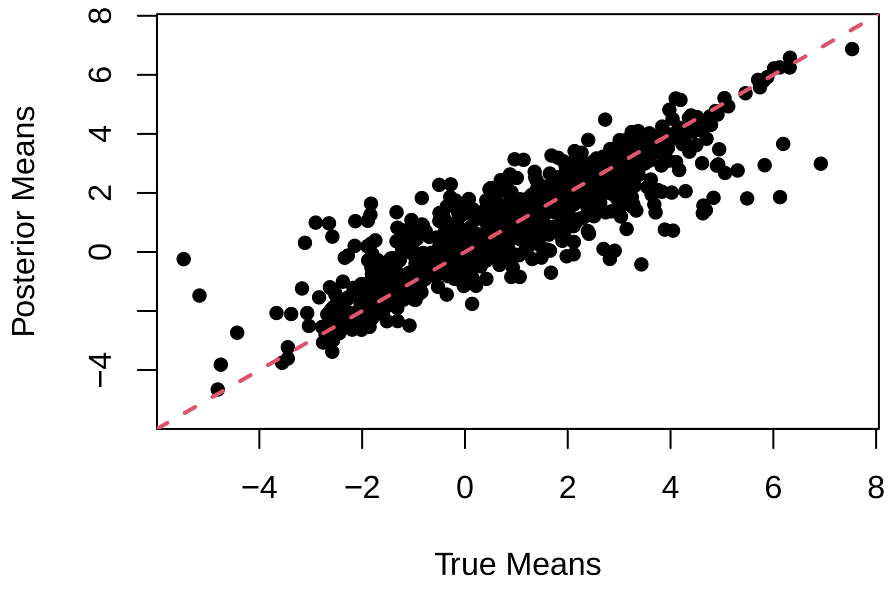
```
hist(sigmasample, "FD", freq = FALSE, main = NA, xlab = expression(sigma))
abline(v = sigmahat, col = 2, lty = 2, lwd = 2)
```



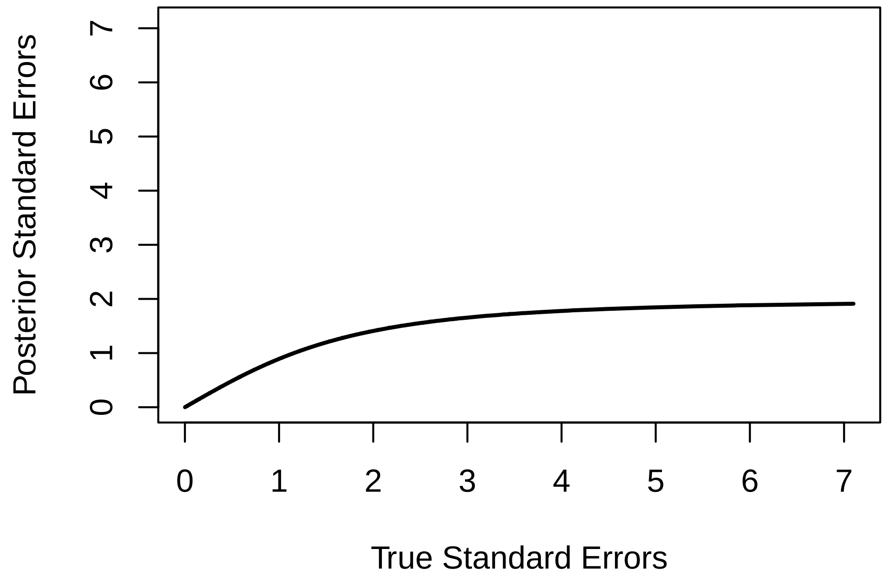
```
thetahat = numeric(n)
SE = numeric(n)
for (i in 1:n) {
    thetahat[i] = mean((sigmasample^2 * Y[i] + S[i]^2 * musample)/(sigmasample^2 +
        S[i]^2))
    SE[i] = sqrt(mean((sigmasample^2 * Y[i] + S[i]^2 * musample)^2/(sigmasample^2 +
        S[i]^2)^2 + sigmasample^2 * S[i]^2/(sigmasample^2 + S[i]^2)) - thetahat[i]^2)
}
mean((thetahat - theta)^2)
```

```
## [1] 0.8583024
```

```
plot(theta, thetahat, ylim = range(theta), xlab = "True Means", ylab = "Posterior Means",
    pch = 16)
abline(0, 1, col = 2, lty = 2, lwd = 2)
```



```
plot(S, SE, "l", ylim = range(S), xlab = "True Standard Errors", ylab = "Posterior Standard Errors",
    lwd = 2)
```



```
covid = read.csv("COVID-19_Cases_US.csv")
covid = covid[covid$Confirmed > 0, ]
p0 = sum(covid$Deaths)/sum(covid$Confirmed)
print(p0)
```

```
## [1] 0.05820638
```

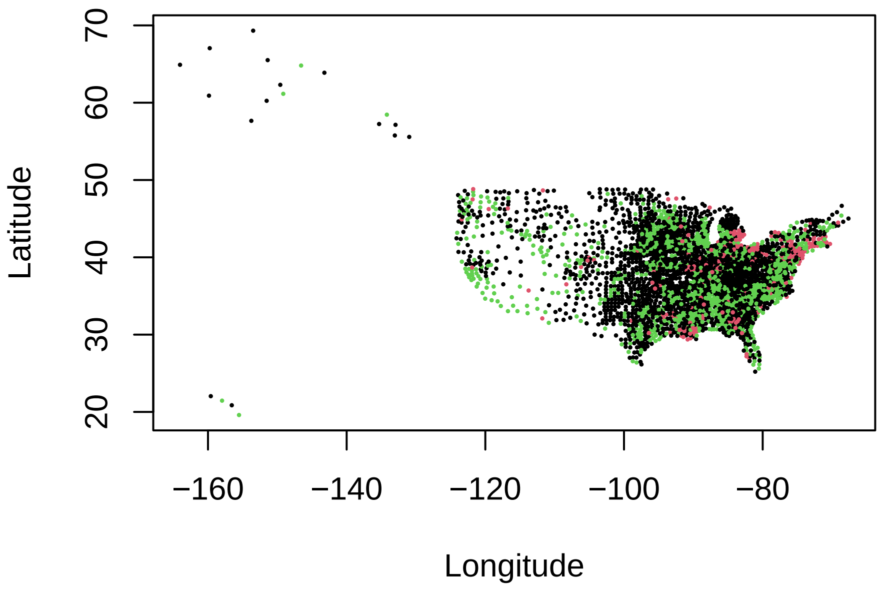
```
alpha = 0.05

loglik = function(param, confirmed, deaths) {
    n = length(confirmed)
    a = exp(param[1])
    b = exp(param[2])
    n * (lgamma(a + b) - lgamma(a) - lgamma(b)) + sum(lgamma(deaths + a) + lgamma(confirmed -
        deaths + b) - lgamma(confirmed + a + b))
}

opt = optim(c(1, 1), loglik, confirmed = covid$Confirmed, deaths = covid$Deaths,
    control = list(fnscale = -1))
a = exp(opt$par[1])
print(a)
```

```
## [1] 1.123923
```

```
b = exp(opt$par[2])
print(b)
```

```
## [1] 25.76441
```

```
posterior = pbeta(p0, covid$Deaths + a, covid$Confirmed - covid$Deaths + b)
empless = posterior > 1 - alpha
empgreater = posterior < alpha
plot(Lat ~ Long_, covid, xlab = "Longitude", ylab = "Latitude", col = 1 + empgreater +
    2 * empless, pch = 16, cex = 0.3)
```


Latitude
Longitude