UNIVERSITETET I OSLO

Department of Physics

STK4051 - Computational Statistics

Mandatory Assignment Part 1/2

Author:
Jonas Semprini Næss

Date: 1st May 2024

Table of Contents

List of Figures

List of Tables

Feedback

The mandatory assignment has been engaging, exciting, challenging, and at times quite time-consuming. I have gained a deeper insight into various numerical methods that can be used in statistics, as well as a new perspective on other nuances of theory and methods that I have already touched upon. However, unfortunately, I did not have the opportunity to complete the entire second part of the assignment, since the assignment as a whole (both part 1 and 2) require its time and diligence complete to a satisfactory level. Nevertheless, I hope that what has been answered in these PDF's provide, and demonstrates the amount of work that has been put in.

Problem 1 - (Lp-regularization)

We will in this exercise consider the problem where the number of parameters we will estimate are of the same size as the number of data. In this case it is common to use regularization to impose additional constraints on the model. In a simplified regression model, we have data on the form:

$$y_i = \beta_i + \varepsilon_i, \quad i = 1, 2, \ldots, n$$

Where y_i is the data, β_i is the parameter and ε_i is an error term, we will assume that the error term has variations according to a normal distribution with mean zero and unit variance, i.e. $\epsilon_i \sim \mathcal{N}(0, 1^2)$

a.)

Derive the maximum likelihood estimator for $\beta_i,\ i = 1, 2, \ldots, n$

Solution:

By definition we observe that when predicting or modelling y_i we are conditioned on the current estimate of β_i. Moreover, it is understood that the according probability of determining y_i is given by looking at the probability of ε_i, namely $y_i - \beta_i$. From this we recall that the probability density function of an arbitrary standard normal distributed variable X is given as

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

where in our case we turn to ε_i

$$f(\varepsilon_i) = \frac{1}{\sqrt{2\pi}} e^{-{\varepsilon_i}^2/2} \tag{1}$$

for $\varepsilon_i \in (-\infty, \infty)$. Hence, can we express the likelihood as

$$L(\beta|y) = \prod_{i=1}^{n} f(\varepsilon_i) = \prod_{i=1}^{n} f(y_i - \beta_i) \tag{2}$$

and consequently the log-likelihood

$$\begin{aligned}\ell(\beta|y) &= \log\left(L(\beta|y)\right) \\ &= \log\left(\prod_{i=1}^{n} f(y_i-\beta_i)\right) \\ &= \sum_{i=1}^{n}\log\left(f(y_i-\beta_i)\right) \\ &= \sum_{i=1}^{n}\log\left(\frac{1}{\sqrt{2\pi}}e^{-(y_i-\beta_i)^2/2}\right) \\ &= \sum_{i=1}^{n}-\log(\sqrt{2\pi})-\frac{1}{2}(y_i-\beta_i)^2\end{aligned}$$

whereby we achieve the maximum likelihood estimator for β_i when differentiating ℓ w.r.t β_i and solvig the resulting equation when equal to zero. That yields

$$\frac{\partial\ell(\beta|y)}{\partial\beta_i} = \frac{2(y_i-\beta_i)}{2} = y_i-\beta_i = 0 \tag{3}$$

meaning the maximum likelihood estimator for β_i is thus

$$\hat{\beta}_i = y_i, \quad i = 1,2,\ldots,n \tag{4}$$

■

This is reasonable considering we are working with a model where the parameter space is of equal size to the data samples (i), and additionally does not demand any further transformations.

b.)

Show that:

$$f_{p,\gamma}(\beta_i) = \beta_i + \gamma\cdot\text{sign}(\beta_i)|\beta_i|^{p-1} \tag{5}$$

Solution:

When now turning to penalized least squares we recall from (a) that we already have an expression for the log-likelihood which by direct insertion into our model expression gives

$$\min_{\beta}\left\{\log(\sqrt{2\pi})+\frac{1}{2}(y_i-\beta_i)^2+\frac{\gamma}{p}\|\beta\|_p^p\right\}. \tag{6}$$

Similarly as in (a.) we differentiate the expression w.r.t β_i

$$\begin{aligned}\frac{\partial}{\partial\beta_i}\left(\log(\sqrt{2\pi})+\frac{1}{2}(y_i-\beta_i)^2+\frac{\gamma}{p}\|\beta\|_p^p\right) &= \left(-(y_i-\beta_i)+\frac{\gamma}{p}p|\beta_i|^{p-1}\frac{\partial|\beta_i|}{\partial\beta_i}\right) \\ &= -(y_i-\beta_i)+\gamma|\beta_i|^{p-1}\frac{\partial|\beta_i|}{\partial\beta_i} \\ &= -(y_i-\beta_i)+\gamma|\beta_i|^{p-1}\text{sign}(\beta_i)\end{aligned}$$

and set it equal to zero

$$-(y_i-\beta_i)+\gamma|\beta_i|^{p-1}\text{sign}(\beta_i) = 0 \tag{7}$$

$$\Updownarrow \tag{8}$$

$$y_i = \beta_i+\gamma\cdot\text{sign}(\beta_i)|\beta_i|^{p-1} \tag{9}$$

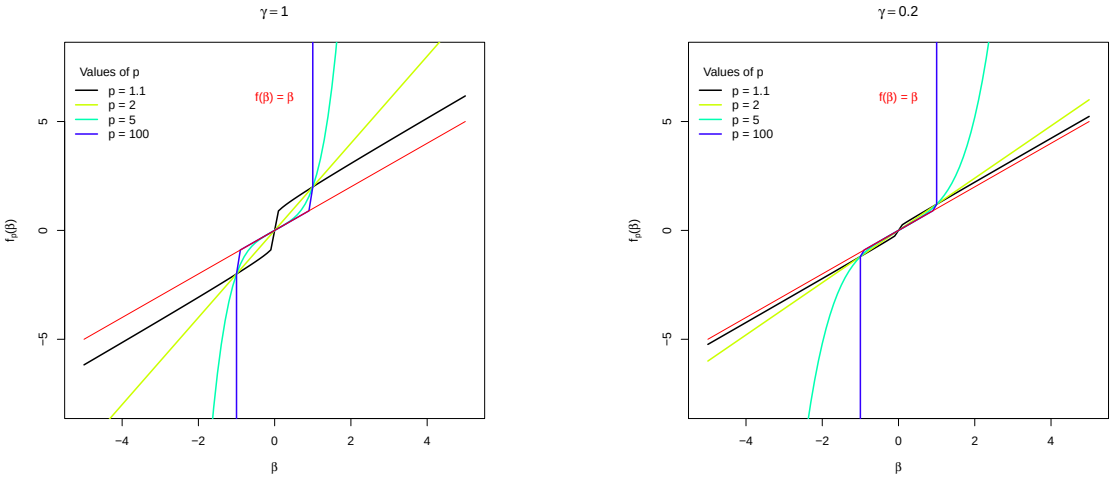
which is what we wanted to show. ■

c.)

For $\gamma = 1$ and $\gamma = 0.2$, plot the function $f_{p,\gamma}(\beta)$ for $p = 1.1, 2, 5$ and 100 on the square $[-5, 5] \times [-5, 5]$, plot also the inverse function by flipping the order of the arguments in the plotting function. Give an interpretation of the results.

Solution:

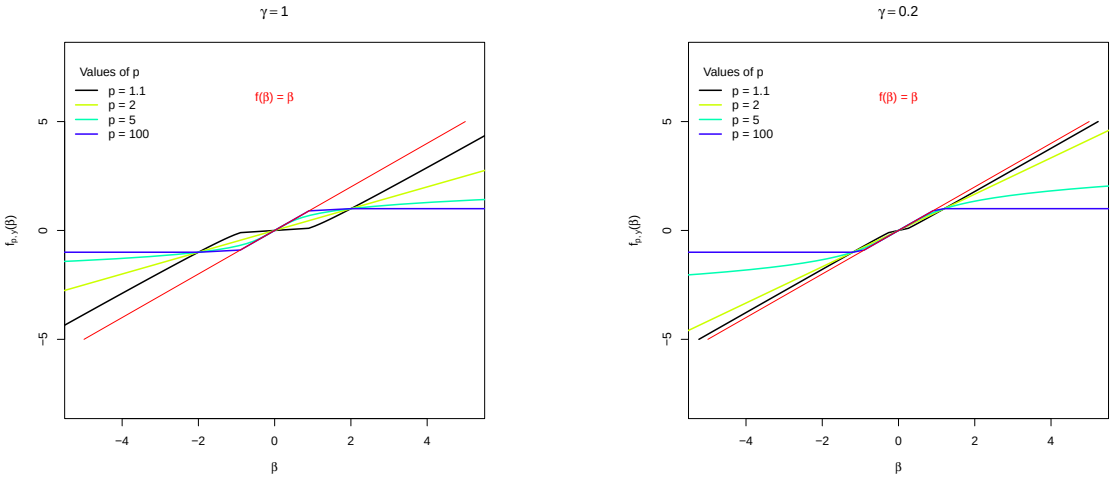
The implementation can be found in Appendix B, from code listing (1). It produces the following plots.



(a) $f_{p,\gamma}(\beta_i)$ for $\gamma = 1$ (b) $f_{p,\gamma}(\beta_i)$ for $\gamma = 0.2$

Figure 1: Plots of $f_{p,\gamma}(\beta_i)$ where $f(\beta) = \beta$ is added for reference

with the respective inverse functions



(a) $f^{-1}_{p,\gamma}(\beta_i)$ for $\gamma = 1$ (b) $f^{-1}_{p,\gamma}(\beta_i)$ for $\gamma = 0.2$

Figure 2: Plots of the inverse of $f_{p.\gamma}(\beta_i)$ where $f(\beta) = \beta$ is added for reference

The constant γ determines the impact of the penalty term on the function f. When γ equals zero, the penalty term is eliminated, resulting in a straight line. As γ increases, the regularization effect becomes more pronounced.

The p-value influences the type of L_p-norm used for regularization. Higher p-values lead to a greater penalty for β values larger than 1 in absolute value, while smaller p-values result in a lesser penalty, as raising β to the pth power tends toward zero for large p-values.

Overall, higher p-values result in a heavier penalty for large β values, and the γ parameter allows control over the overall magnitude of this penalty term.

d.)

Implement a function which finds the root of expression (3). Use the method of bisection (book page 23). The root of (3) will be the estimator $\hat{\beta}_{\gamma,p}(y)$. *What are good starting values for upper and lower bounds (in general)? Test the algorithm for* $\gamma = 1$, *and* $p = 1.1$, $p = 2$, *and* $p = 100$. *Evaluate the results for* y *in the interval* $[-5, 5]$, *and plot the results in the same form as in c.)*

Solution:

The implementation can be found in Appendix B, from code listing (2). It produces the following plot:

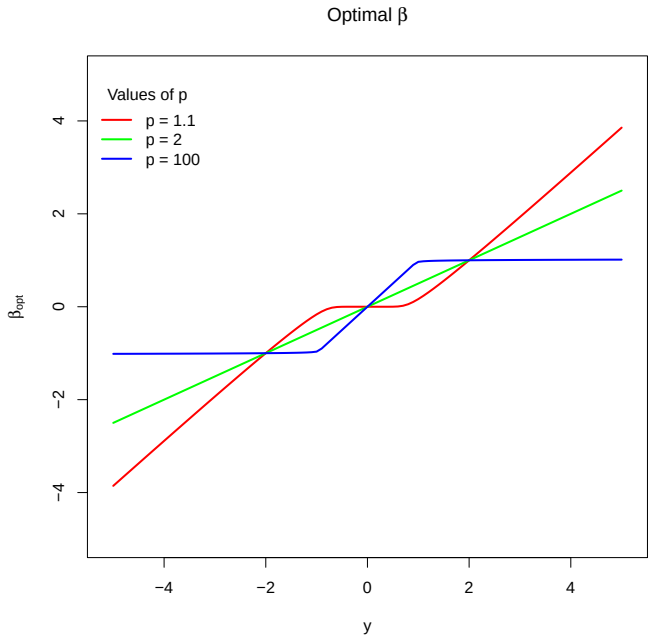


Figure 3: Finding the solution to $f_{p,\gamma}(\beta_i) = 0$ through the bisection method

Sensible starting values for upper and lower bounds for the bisection method would be $\mathcal{O}_\beta = |\beta|$ and $\Omega_\beta = -|\beta|$ respectively. Meaning then for some $\beta_0 \in (\Omega_\beta, \mathcal{O}_\beta)$ we could be more assured to reach a solution.

e.)

Use the data set `sparseDataWithErrors.dat` *and perform estimation of parameters* $\beta_i, i = 1, \ldots, n$, *from the data* $y_i, i = 1, \ldots, n$. *Compare the result to the ground truth in terms of residual sum of squares. Do the estimation for penalized regression, i.e. compute* $\hat{\beta}_{\gamma,p}$ *with* $\gamma = 1, p = 1.1, p = 2$ *and* $p = 100$, *compare also to the MLE estimator. For* $p = 100$, *try also to estimate the parameters* β_i *by using the residuals, i.e.* $\hat{\beta}^{Alt}_{1,100} = y - \hat{\beta}_{1,100}$. *Comment on the results. How does MLE measure up?*

Solution:

The implementation can be found in Appendix B, from code listing (3). The estimation gives:

p	$\hat{\beta}_{1,100}$
1.1	273.9674
2.0	1256.5316
100.0	3786.6181

Table 1: Residuals for penalized regression

p	$\hat{\beta}^{\text{Alt}}_{1,100}$
1.1	3590.8957
2.0	1256.5316
100.0	251.9418

Table 2: Residuals for Alternative Estimation

The residual sum of squares of the MLE, RSS $\left(\hat{\theta}_{\text{MLE}}\right)$, gives 998.4152. Comparing the results, we find that the most accurate estimate for $\hat{\beta}^{\text{Alt}}_{1,100}$ occurs at $p = 100$, yielding the lowest residual error. Following closely, the second-best estimate for $\hat{\beta}_{1,100}$ comes at $p = 1.1$.

f.)

In the linear regression problem for $p = n$, we have data on the form:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \varepsilon$$

Where $\boldsymbol{y}$ is the $(n \times 1)$ data vector, $\boldsymbol{\beta}$ is the $(n \times 1)$ parameter vector, $\boldsymbol{X}$ is a $(n \times n)$ design matrix, and $\boldsymbol{\varepsilon}$ is the $(n \times 1)$ error vector, with $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, I)$. How can you use the ADMM algorithm together with the solution to the problem above to derive a solution to the penalized regression problem?

$$\min_{\beta} \left\{ -\ell(\boldsymbol{\beta}|\boldsymbol{y}, \boldsymbol{X}) + \frac{\gamma}{p}\|\boldsymbol{\beta}\|_p^p \right\}$$

Solution:

To derive a solution through the ADMM algorithm for the objective function

$$\min_{\beta} \left\{ -\ell(\boldsymbol{\beta}|\boldsymbol{y}, \boldsymbol{X}) + \frac{\gamma}{p}\|\boldsymbol{\beta}\|_p^p \right\}$$

where $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \varepsilon$ and ε is a standard normal distributed variable, one can proceed as follows

1. **Objective Function**: Start by defining the objective function, where as we know $-\ell(\boldsymbol{\beta}|\boldsymbol{y}, \boldsymbol{X})$ represents the negative log-likelihood term, but is additionally conditioned on the data $\boldsymbol{X}$:

$$\min_{\beta} \left\{ -\ell(\boldsymbol{\beta}|\boldsymbol{y}, \boldsymbol{X}) + \frac{\gamma}{p}\|\boldsymbol{\beta}\|_p^p \right\}$$

2. **Introduction of Auxiliary Variable**: Introduce an auxiliary variable $\boldsymbol{\lambda}$ such that the problem becomes separable:

$$\min_{\beta} \left\{ -\ell(\boldsymbol{\beta}|\boldsymbol{y}, \boldsymbol{X}) + \frac{\gamma}{p}\|\boldsymbol{\lambda}\|_p^p \right\}$$
$$\text{subject to} \quad \boldsymbol{\beta} = \boldsymbol{\lambda} \iff \boldsymbol{\beta} - \boldsymbol{\lambda} = 0$$

3. **Augmented Lagrangian**: We form the augmented Lagrangian function, combining the objective function and the constraint using a scaled dual variable $\boldsymbol{u}$ (or the Lagrange multiplier):
$$L_\rho(\boldsymbol{\beta}, \boldsymbol{\lambda}, \boldsymbol{u}) = -\ell(\boldsymbol{\beta}|\boldsymbol{y}, \boldsymbol{X}) + \frac{\gamma}{p}\|\boldsymbol{\lambda}\|_p^p + \boldsymbol{u}^T(\boldsymbol{\beta} - \boldsymbol{\lambda}) + \frac{\rho}{2}\|\boldsymbol{\beta} - \boldsymbol{\lambda}\|_2^2$$

4. **ADMM Iterations**: We alternate between updating $\boldsymbol{\beta}$, $\boldsymbol{\lambda}$, and $\boldsymbol{u}$ until convergence:

   - **Update $\boldsymbol{\beta}$**:
   $$\boldsymbol{\beta}^{(k+1)} = \arg\min_{\boldsymbol{\beta}} \left\{ -\ell(\boldsymbol{\beta} \mid \boldsymbol{y}, \boldsymbol{X}) + \frac{\rho}{2}\|\boldsymbol{\beta} - \boldsymbol{\lambda}^{(k)} + \boldsymbol{u}^{(k)}\|^2 \right\}$$

   - **Update $\boldsymbol{\lambda}$**:
   $$\boldsymbol{\lambda}^{(k+1)} = \arg\min_{\boldsymbol{\lambda}} \left\{ \frac{\gamma}{p}\|\boldsymbol{\lambda}\|_p^p + \frac{\rho}{2}\|\boldsymbol{\beta}^{(k+1)} - \boldsymbol{\lambda} + \boldsymbol{u}^{(k)}\|^2 \right\}$$

   - **Update $\boldsymbol{u}$**:
   $$\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \rho(\boldsymbol{\beta}^{(k+1)} - \boldsymbol{\lambda}^{(k+1)})$$

5. **Convergence**: We check for convergence based on the magnitude of the primal and dual residuals.

Problem 2 - (EM-algorithm)

We will now assume that the data $Y_i, i = 1, \ldots n$, are independent and identically distributed according to the mixture distribution:

$$f(y_i) = p \cdot \phi\left(y_i; 0, 1^2\right) + (1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right)$$

where $\phi\left(y_i; \mu, \sigma^2\right)$ is the normal density with mean μ and variance σ^2. We will now consider estimation of the parameters $\theta = \left(p, \tau^2\right)$.

a.)

Give an expression for the likelihood of θ.

Solution:

We recall that the likelihood function of a parameter θ is given by

$$L(\theta|y_i) = \prod_{i=1}^{n} f(y_i) \tag{10}$$

where $f(y_i)$ refers to the probability mass function of y_i. Hence by direct insertion we have that

$$\begin{aligned} L(\theta|y_i) &= \prod_{i=1}^{n} f(y_i) \\ &= \prod_{i=1}^{n} \left[p \cdot \phi\left(y_i; 0, 1^2\right) + (1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right)\right] \end{aligned}$$

and we are done. ■

b.)

Introduce the variable C_i which identify which of the two modes y_i belongs to. Give an expression for the complete log-likelihood using the pairs (C_i, y_i), $i = 1, \dots n$

Solution:

Given the Likelihood function of our mixture distribution (i.e 2a):

$$L(\theta|y_i) = \prod_{i=1}^{n} \left[p \cdot \phi\left(y_i; 0, 1^2\right) + (1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right) \right]$$

We aim to derive the complete log-likelihood function when introducing the variable C_i, which identifies which of the two modes y_i belongs to. The indicator variable C_i is a binary variable, defined accordingly

$$C_i = \begin{cases} 0, & \text{if } y_i \text{ belongs to the first mode,} \\ 1, & \text{if } y_i \text{ belongs to the second mode.} \end{cases}$$

Furthermore, is it understood that C_i takes on the given values with probabilities

$$\mathbb{P}(C_i) = p \iff C_i = 1 \iff y_i \text{ belongs to the second mode}$$

$$\mathbb{P}(C_i) = 1 - p \iff C_i = 0 \iff y_i \text{ belongs to the first mode}$$

From here we start of by converting the Likelihood function to a log-likelihood

$$\ell(\theta|y) = \log\left(L(\theta|y)\right) \tag{11}$$

$$= \log\left(\prod_{i=1}^{n} \left[p \cdot \phi\left(y_i; 0, 1^2\right) + (1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right) \right] \right) \tag{12}$$

$$= \sum_{i=1}^{n} \log\left[p \cdot \phi\left(y_i; 0, 1^2\right) + (1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right) \right] \tag{13}$$

Observing that for each y_i, only one term is chosen from the probability density function, based on the mode for that y_i. Meaning thus that we interpret C_i as an indicator function. By logarithmically transforming both terms, we obtain the complete log-likelihood function for the observed and missing data pairs (C_i, Y_i), $i = 1, \dots, n$, expressed as:

$$\ell(\theta|y_i, C_i) = \sum_{i=1}^{n} \left[C_i \cdot \log\left(p \cdot \phi\left(y_i; 0, 1^2\right)\right) + (1 - C_i) \cdot \log\left((1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right)\right) \right]$$

And we are done. ■

c.)

Give an expression for $Q\left(\theta \mid \theta^{(t)}\right)$, what is the interpretation of $Q\left(\theta \mid \theta^{(t)}\right)$. Derive the estimates for $\theta = \left(p, \tau^2\right)$, using $Q\left(\theta \mid \theta^{(t)}\right)$.

Solution:

To rigidly express $Q\left(\theta|\theta^{(t)}\right)$, we start off by looking at the conditional probabilities of C_i given the observation on y_i.

The probability that $C_i = 0$ given the observation on y_i is:

$$\mathbb{P}(C_i = 0|y_i, \theta^t) = \frac{(1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right)}{p \cdot \phi\left(y_i; 0, 1^2\right) + (1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right)} \tag{14}$$

Analogously is the probability that $C_i = 1$ given the observation on y_i expressed as:

$$\mathbb{P}(C_i = 1|y_i, \theta^t) = \frac{p \cdot \phi\left(y_i; 0, 1^2\right)}{p \cdot \phi\left(y_i; 0, 1^2\right) + (1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right)} \tag{15}$$

Henceforth we proceed to compute the expected value of the joint log-likelihood function of the complete dataset, denoted by $Q\left(\theta|\theta^{(t)}\right)$

$$Q\left(\theta|\theta^{(t)}\right) = \mathbb{E}\left\{\log L(\theta|y) \mid y, \theta^{(t)}\right\} \tag{16}$$

$$\begin{aligned} &= \sum_{i=1}^{n} \mathbb{P}(C_i = 1|y_i, \theta^{(t)}) \log\left(p \cdot \phi\left(y_i; 0, 1^2\right)\right) \\ &\quad + \mathbb{P}(C_i = 0|y_i, \theta^{(t)}) \log\left((1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right)\right) \end{aligned} \tag{17}$$

The Q-function computes the expected complete log-likelihood using both the observed data and the current parameter estimates. By maximizing the Q-function, we adjust the parameters to maximize the expected complete log-likelihood.

To derive the estimates for θ we need to maximize $Q(\theta|\theta^{(t)})$ with respect to θ itself. By ease of notation we define the maximizer to be $\theta^{(t+1)} = \left(p^{(t+1)}, \tau^{2^{(t+1)}}\right)$.

Initially, we look at the derivative of Q with respect to p:

$$\frac{\partial}{\partial p} Q(\theta|\theta^{(t)}) = \frac{\partial}{\partial p} \left[\begin{aligned} &\sum_{i=1}^{n} \mathbb{P}(C_i = 1|y_i, \theta^{(t)}) \log\left(p \cdot \phi\left(y_i; 0, 1^2\right)\right) \\ &+ \mathbb{P}(C_i = 0|y_i, \theta^{(t)}) \log\left((1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right)\right) \end{aligned} \right] \tag{18}$$

$$= \sum_{i=1}^{n} \left[\frac{\mathbb{P}(C_i = 1|y_i, \theta^{(t)})}{p} - \frac{\mathbb{P}(C_i = 0|y_i, \theta^{(t)})}{1-p} \right] \tag{19}$$

then multiply by $p(1-p)$

$$\sum_{i=1}^{n} (1-p) \mathbb{P}\left(C_i = 1|y_i, \theta^{(t)}\right) - p\left(\mathbb{P}\left(C_i = 0|y_i, \theta^{(t)}\right)\right)$$

which by expansion of the first term yields

$$\sum_{i=1}^{n} \mathbb{P}\left(C_i = 1|y_i, \theta^{(t)}\right) - p\left(\mathbb{P}\left(C_i = 1|y_i, \theta^{(t)}\right)\right) - p\left(\mathbb{P}\left(C_i = 0|y_i, \theta^{(t)}\right)\right)$$

and through factoring for p

$$\sum_{i=1}^{n} \mathbb{P}\left(C_i = 1|y_i, \theta^{(t)}\right) - p\left(\mathbb{P}\left(C_i = 1|y_i, \theta^{(t)}\right) + \mathbb{P}\left(C_i = 0|y_i, \theta^{(t)}\right)\right)$$

whereby setting the expression equal to zero conclusively gives

$$\sum_{i=1}^{n} \mathbb{P}\left(C_i = 1|y_i, \theta^{(t)}\right) - p\left(\mathbb{P}\left(C_i = 1|y_i, \theta^{(t)}\right) + \mathbb{P}\left(C_i = 0|y_i, \theta^{(t)}\right)\right) = 0$$

$$\Updownarrow$$

$$p\left(\sum_{i=1}^{n} \mathbb{P}\left(C_i = 1|y_i, \theta^{(t)}\right) + \mathbb{P}\left(C_i = 0|y_i, \theta^{(t)}\right)\right) = \sum_{i=1}^{n} \mathbb{P}\left(C_i = 1|y_i, \theta^{(t)}\right)$$

meaning

$$p = \frac{\sum_{i=1}^{n} \mathbb{P}\left(C_i = 1 | y_i, \theta^{(t)}\right)}{\sum_{i=1}^{n} \mathbb{P}\left(C_i = 1 | y_i, \theta^{(t)}\right) + \mathbb{P}\left(C_i = 0 | y_i, \theta^{(t)}\right)}$$

which says that the updating rule on p is

$$p^{(t+1)} = \frac{1}{n} \sum_{1}^{n} \frac{p^{(t)} \cdot \phi\left(y_i; 0, 1^2\right)}{p^{(t)} \cdot \phi\left(y_i; 0, 1^2\right) + (1 - p^{(t)}) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right)}. \tag{20}$$

Similarly, we proceed by differentiating Q with respect to τ^2:

$$\frac{\partial}{\partial \tau^2} Q(\theta | \theta^{(t)}) = \sum_{i=1}^{n} (1 - p) \left(\frac{y_i^2}{(\tau^2 + 1)^2} - \frac{1}{\tau^2 + 1} \right) \tag{21}$$

[1] and equating to zero

$$\sum_{i=1}^{n} (1 - p) \left(\frac{y_i^2}{(\tau^2 + 1)^2} - \frac{1}{\tau^2 + 1} \right) = 0$$

$$\sum_{i=1}^{n} (1 - p) \left(\frac{y_i^2 - (\tau^2 + 1)}{(\tau^2 + 1)^2} \right) = 0$$

$$\Downarrow$$

$$\sum_{i=1}^{n} (1 - p) \left(y_i^2 - (\tau^2 + 1) \right) = 0$$

$$\Updownarrow$$

$$\sum_{i=1}^{n} (1 - p) y_i^2 = \sum_{i=1}^{n} (1 - p)(\tau^2 + 1)$$

$$\Downarrow$$

$$\tau^2 + 1 = \frac{\sum_{i=1}^{n} (1 - p) y_i^2}{\sum_{i=1}^{n} 1 - p}$$

thus yielding

$$\tau^2 = \frac{\sum_{i=1}^{n} (1 - p) y_i^2}{\sum_{i=1}^{n} 1 - p} - 1$$

Meaning the updating rule on τ^2 is then

$$\tau^{2^{(t+1)}} = \frac{\sum_{i=1}^{n} (1 - p) y_i^2}{\sum_{i=1}^{n} 1 - p} - 1 \tag{22}$$

and we are done. ■

d.)

Implement the solution you derived in c) as a function and apply it to the data: `sparseDataWithErrors.dat`

[1] This result comes from the fact that the Gaussian density function $\phi(\cdot; \mu, \sigma^2)$ has the derivative:

$$\frac{d}{d\sigma^2} \phi(x; \mu, \sigma^2) = \frac{(x - \mu)^2}{\sigma^4} \phi(x; \mu, \sigma^2) - \frac{1}{\sigma^2} \phi(x; \mu, \sigma^2)$$

What is a good initialization?

Solution:

A natural assumption on initialization is to assume that the modes are equal and that the data points are evenly distributed between them. Hence set $p = 0.5$ and $\tau^2 = 0$. More generally we could assume an initialization of $p \in (0, 1)$ (due to it being a probability), and $\tau^2 \neq 1$ (to avoid identical distribution).

The implementation can be found in Appendix B, from code listing (4). The code produces the following result:

$$(p, \tau^2) = (0.946004, 75.005977)$$

where a section of the iteration show the following progression

i	p	τ^2
1	0.5	3
5	0.862 589	30.258 267
10	0.944 501	73.045 733
15	0.945 983	74.977 963
20	0.946 004	75.005 588
30	0.946 004	75.005 977

Table 3: Iterative estimation of p and τ^2 where i denotes the number of iterations

As presented above we see that even for quite low values of p and τ^2, the algorithm converges quickly within around 30 iterations, which of course can be reduced even more given that the starting values are closer to the realised ones.

e.)

Compute a bootstrap estimate of the uncertainty of the two parameters, using the function from d. Sample $B = 1000$ *times and display scatter plot of the values.*

Solution:

The implementation can be found in Appendix B, from code listing (5). The code produces the following plot:

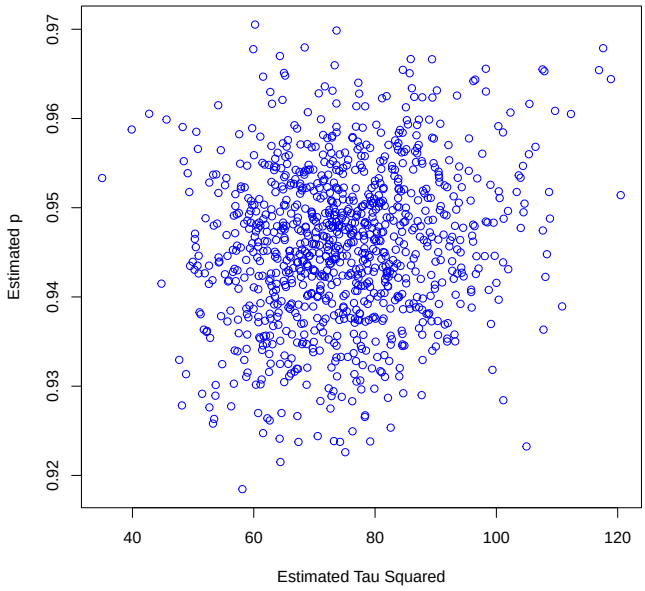


Figure 4: Scatter plot of bootstrap estimations for τ^2 and p

whereby constructing 95 % confidence intervals for p and τ^2 yield

$$\mathbb{P}_p\left(0.9278386 \leq p \leq 0.9631131\right) = 95\% \tag{23}$$

$$\mathbb{P}_{\tau^2}\left(51.4844 \leq \tau^2 \leq 103.3594\right) = 95\% \tag{24}$$

It is observed that there is notably higher uncertainty associated with $\hat{\tau}^2$ compared to $\hat{p}$, which is understandable given that the predicaments of the 95 % confidence interval for p admit values of $p > 0.925$, implying that fewer than 75 samples belong to mode 2. Moreover, the width of $\mathbb{P}_{\tau^2}$ naturally tells us that there are clear indications of instability in the estimate of τ^2.

f.)

Compute the observed information matrix. How can you use the observed information matrix to give an uncertainty estimate for θ? Compare the result to e.

Solution:

To approach this computation it is necessary to define exactly what we want to compute, correctly. We are working with a "true" parameter θ, a consistent estimate $\hat{\theta}$ (from 2c and 2d), the expected (fisher) information $I(\theta)$ at θ and the observed (fisher) information $\mathcal{J}(\theta)$ at θ.

What then needs to be established is that all of these quantities are asymptotically equivalent, respectively. This means that given enough samples (data), the observed fisher information will in probability converge to the expected fisher information and likewise the consistent estimate to the

true parameter. More mathematically this says that

$$\mathcal{J}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial^2}{\partial\theta^2}\ell(\theta|y) \qquad \text{(Observed Fisher information)} \tag{25}$$

$$\lim_{n\to\infty} \mathcal{J}(\theta) = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} \frac{\partial^2}{\partial\theta^2}\ell(\theta|y) \tag{26}$$

$$= \mathbb{E}_\theta\left(\frac{\partial^2}{\partial\theta^2}\ell(\theta|y)\right) \qquad \text{(Expected Fisher information).} \tag{27}$$

The assured convergence follows directly from the law of large numbers.

Since we are primarily looking at using the full dataset in this exercise, there will therefore not be a distinct difference between the two measures and can hence express the observation matrix accordingly

$$\begin{aligned}\mathcal{J}(\theta) &= -\nabla\nabla\ell(\theta) \\ &= -H_\ell\end{aligned}$$

where H_ℓ is the Hessian matrix of the log-likelihood with respect to the parameter θ. In our case the Hessian is simply a 2×2 matrix, with elements

$$H_\ell = \begin{pmatrix} \frac{\partial^2\ell(\theta)}{\partial p^2} & \frac{\partial^2\ell(\theta)}{\partial p\partial\tau^2} \\ \frac{\partial^2\ell(\theta)}{\partial\tau^2\partial p} & \frac{\partial^2\ell(\theta)}{\partial(\tau^2)^2} \end{pmatrix}. \tag{28}$$

By construction of $\ell(\theta|y)$ it is too tedious to compute H_ℓ analytically, and we will numerically approximate it instead. The implementation can be found in Appendix B, from code lisitings (6). Where the resulting observation matrix becomes

$$\mathcal{J}(\theta) = \begin{pmatrix} 13693.156468 & -1.896655 \\ -1.896655 & 0.004318338 \end{pmatrix}$$

The Negative values admitted in $\mathcal{J}(\theta)$ [2] for $\frac{\partial^2\ell(\theta)}{\partial\tau^2\partial p}$ indicate potential anomalies in the data, where p and τ^2 might show significant correlation. This implies that specific care in interpretation of the diagonal elements, which represent mean square error (due to proportionality), is recommended.

The related covariance matrix, or the inverse of the observation matrix, further establishes the aforementioned:

$$\mathcal{J}^{-1}(\theta) = \mathrm{Cov}(\theta) = \begin{pmatrix} 7.775973 \cdot 10^{-5} & 0.03415281 \\ 0.03415281 & 246.57082498 \end{pmatrix}$$

g.)

Compute the likelihood from 2a*, in a dense grid* $\left(p, \tau^2\right) \in [0.8, 1] \times [50, 130]$*. Normalize it by dividing by the maximum value and plot it in a contour plot. Use contours lines [0.01 0.1 0.5 0.95], mark the ML estimator in the plot. Compare the results to e and f.*

Solution:

The implementation can be found in Appendix B, from code lisitings (7). It produces the following plot

[2] I am not completely certain that this is correct, so if not I would fully appreciate a more thorough explanation and result

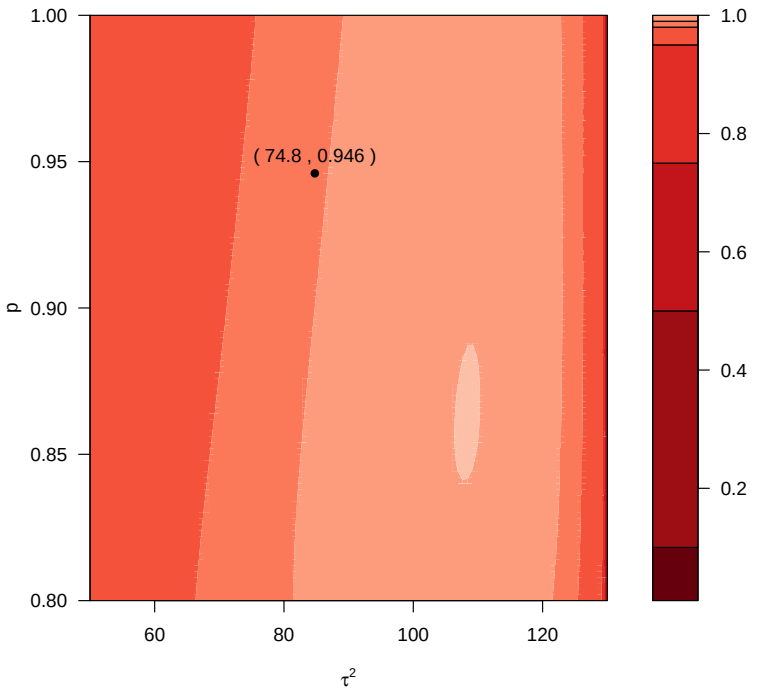


Figure 5: Contour plot of of $L(\theta|y)$ in the grid $\left(p, \tau^2\right) \in [0.8, 1] \times [50, 130]$

which compares well to the estimates from our bootstrap for the MLE.

Problem 3 - (Bayesian Approach)

Assuming that the data Y_i are on the same form as in Exercise 1, i.e.

$$y_i = \beta_i + \varepsilon_i, i = 1, \ldots, n,$$

where the parameters are as described for (1). Assume also that y_i, follows the mixture distribution as in Exercise 2,

$$f\left(y_i\right) = p \cdot \phi\left(y_i; 0, 1^2\right) + (1-p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right)$$

a.)

Use a Bayesian interpretation of β_i, (e.g. assume β_i is random) and argue that $\mathbb{P}\left(\beta_i = 0 \mid C_i = 0\right) = 1$, and that $f\left(\beta_i \mid C_i = 1\right) = \phi\left(\beta_i; 0, \tau^2\right)$. Give an expression for $\mathbb{P}\left(\beta_i = 0 \mid y_i = y\right)$.

Solution:

- **Result 1:** $\mathbb{P}\left(\beta_i = 0 \mid C_i = 0\right) = 1$

By construction of our model we have that

$$Y_i = \beta_i + \varepsilon_i \tag{29}$$

whereby looking at the conditionals with respect to $C_i = 0$ yields

$$Y_i|(C_i = 0) = \beta_i|(C_i = 0) + \varepsilon_i|(C_i = 0) \tag{30}$$

But the ε_i are by assumption independent of the class/mode, and moreover standard normal distributed variables (i.e $\varepsilon_i \sim \mathcal{N}(0,1)$) meaning thus that both the right and left hand sides of Eq.(30) must follow a standard normal distribution with mean $\beta_i|(C_i = 0)$ and variance 1. Since β_i and ε_i are independent it follows that $\mathbb{P}(\beta_i = 0|C_i = 0) = 1$.

- **Result 2:** $f\left(\beta_i \mid C_i = 1\right) = \phi\left(\beta_i; 0, \tau^2\right)$

In like manner we have that

$$Y_i|(C_i = 1) = \beta_i|(C_i = 1) + \varepsilon_i|(C_i = 1) \tag{31}$$

where again ε_i are assumed to be independent of the class/mode. Since we now are conditioning on the class/mode $C_i = 1$, and know that β_i and ε_i are independent, it follows that the only distribution satisfying Eq.(31) is $\beta_i|(C_i = 1) \sim \mathcal{N}(0, \tau^2)$ since

$$\begin{aligned} f(y_i|C_i = 1) = f(\beta_i + \varepsilon_i|C_i = 1) \Rightarrow \boxed{f(y_i|C_i = 1) \sim \mathcal{N}(0, \tau^2 + 1)} \\ &= f(\beta_i|C_i = 1) + f(\varepsilon_i|C_i = 1) \\ &= f(\beta_i|C_i = 1) + \mathcal{N}(0,1) \\ &= \mathcal{N}(0, \tau^2) + \mathcal{N}(0,1) \\ &= \mathcal{N}(0, \tau^2 + 1). \end{aligned}$$

To express $\mathbb{P}\left(\beta_i = 0 \mid y_i = y\right)$ we rewrite it as

$$\mathbb{P}\left(\beta_i = 0 \mid y_i = y\right) = \mathbb{P}(\beta_i = 0|C_i = 0, y_i = y)\mathbb{P}(C_i = 0|y_i = y) \tag{32}$$

$$+ \mathbb{P}(\beta_i = 0|C_i = 1, y_i = y)\mathbb{P}(C_i = 1|y_i = y) \tag{33}$$

where we have used the law of total probability [3]. Further, we apply Bayes´ rule on the complete expression Eq.(33)

$$\begin{aligned} \mathbb{P}\left(\beta_i = 0 \mid y_i = y\right) &= \mathbb{P}(\beta_i = 0|C_i = 0, y_i = y) \cdot \frac{\mathbb{P}(y_i = y|C_i = 0) \cdot \mathbb{P}(C_i = 0)}{\mathbb{P}(y_i = y)} \\ &+ \mathbb{P}(\beta_i = 0|C_i = 1, y_i = y) \cdot \frac{\mathbb{P}(y_i = y|C_i = 1) \cdot \mathbb{P}(C_i = 1)}{\mathbb{P}(y_i = y)} \\ &= \frac{\mathbb{P}(y_i = y|\beta_i = 0, C_i = 0) \cdot \mathbb{P}(\beta_i = 0|C_i = 0) \cdot \mathbb{P}(C_i = 0)}{\mathbb{P}(y_i = y)} \\ &+ \frac{\mathbb{P}(y_i = y|\beta_i = 0, C_i = 1) \cdot \mathbb{P}(\beta_i = 0|C_i = 1) \cdot \mathbb{P}(C_i = 1)}{\mathbb{P}(y_i = y)} \end{aligned}$$

where we then assume $y_i = \beta_i + \varepsilon_i$ and independence on β_i and ε_i, giving

$$\begin{aligned} \mathbb{P}\left(\beta_i = 0 \mid y_i = y\right) &= \mathbb{P}(\varepsilon_i = y) \cdot \frac{1 \cdot \mathbb{P}(C_i = 0)}{\mathbb{P}(y_i = y)} + 0 \cdot \frac{\mathbb{P}(y_i = y|\beta_i = 0, C_i = 1) \cdot \mathbb{P}(C_i = 1)}{\mathbb{P}(y_i = y)} \\ &= \mathbb{P}(\varepsilon_i = y) \cdot \frac{\mathbb{P}(C_i = 0)}{\mathbb{P}(y_i = y)} \\ &= \phi(y; 0, 1) \cdot \frac{p}{p \cdot \phi\left(y_i; 0, 1^2\right) + (1 - p) \cdot \phi\left(y_i; 0, \tau^2 + 1^2\right)}. \end{aligned}$$

Where we have used that, since $\mathbb{P}\left(\beta_i = 0 \mid C_i = 0\right) = 1$ it follows that $\mathbb{P}\left(\beta_i = 0 \mid C_i = 1\right) = 0$.[4] ■

[3] $P(A) = \sum_i P(A|B_i) \cdot P(B_i)$

[4] Also a consequence of the fact that when $C_i = 1$, β is a continuous distribution.

NB!: I want the reader to notice that there has been a natural shift in interpretation of the probabilities for the modes $C_i = 0$ and $C_i = 1$ in comparison to problem 2. There it was stated that we assume $\mathbb{P}(C_i = 1) = p$ and $\mathbb{P}(C_i = 0) = 1 - p$, which in problem 3 now was considered opposite. This does not by any means change the result.

b.)

Argue that the estimator for β_i (i.e. the conditional expectation of the parameter given the data), is $\mathbb{P}(C_i = 1 \mid y_i = y)\,\mathbb{E}(\beta_i \mid y_i = y, C_i = 1)$. Plot this expression as a function of y *in the interval $[-5, 5]$, and compare the result with 1.d. Use values: $p = 0.9$, and $\tau^2 = 80$. Hint: $\mathbb{E}(\beta_i \mid y_i = y, C_i = 1) = \frac{\tau^2}{\tau^2+1} y$.*

Solution:

To proceed, we notice that the problem can be rephrased to showing that

$$\mathbb{E}(\beta_i|y_i) = \mathbb{P}(C_i = 1 \mid y_i = y)\,\mathbb{E}(\beta_i \mid y_i = y, C_i = 1) \tag{34}$$

which is a direct calculation when working with a continuous distribution.

$$\begin{aligned}
\mathbb{E}(\beta_i|y_i) &= \int_{-\infty}^{\infty} \beta_i \left[f(\beta_i|y_i)\right]\ d\beta_i \\
&= \int_{-\infty}^{\infty} \beta_i \left[f(\beta_i|y_i = y, C_i = 0)\mathbb{P}(C_i = 0|y_i = y) + f(\beta_i|y_i = y, C_i = 1)\mathbb{P}(C_i = 1|y_i = y)\right]\ d\beta_i \\
&= \int_{-\infty}^{\infty} \beta_i \left[f(\beta_i|y_i = y, C_i = 0)\mathbb{P}(C_i = 0|y_i = y)\right]\ d\beta_i \\
&+ \int_{-\infty}^{\infty} \beta_i \left[f(\beta_i|y_i = y, C_i = 1)\mathbb{P}(C_i = 1|y_i = y)\right]\ d\beta_i \\
&= \mathbb{P}(C_i = 0|y_i = y) \left[\int_{-\infty}^{\infty} \beta_i f(\beta_i|y_i = y, C_i = 0)\ d\beta_i\right] \\
&+ \mathbb{P}(C_i = 1|y_i = y) \left[\int_{-\infty}^{\infty} \beta_i f(\beta_i|y_i = y, C_i = 1)\ d\beta_i\right] \\
&= \mathbb{P}(C_i = 0|y_i = y) \left[\int_{-\infty}^{\infty} \beta_i \cdot 1\ d\beta_i\right] + \mathbb{P}(C_i = 1|y_i = y)\mathbb{E}(\beta_i \mid y_i = y, C_i = 1) \\
&= 0 + \mathbb{P}(C_i = 1|y_i = y)\mathbb{E}(\beta_i \mid y_i = y, C_i = 1) \\
&= \mathbb{P}(C_i = 1|y_i = y)\mathbb{E}(\beta_i \mid y_i = y, C_i = 1)
\end{aligned}$$

And we are done. ■

The implementation can be found in Appendix B, from code listings (8), and produces the following plot:

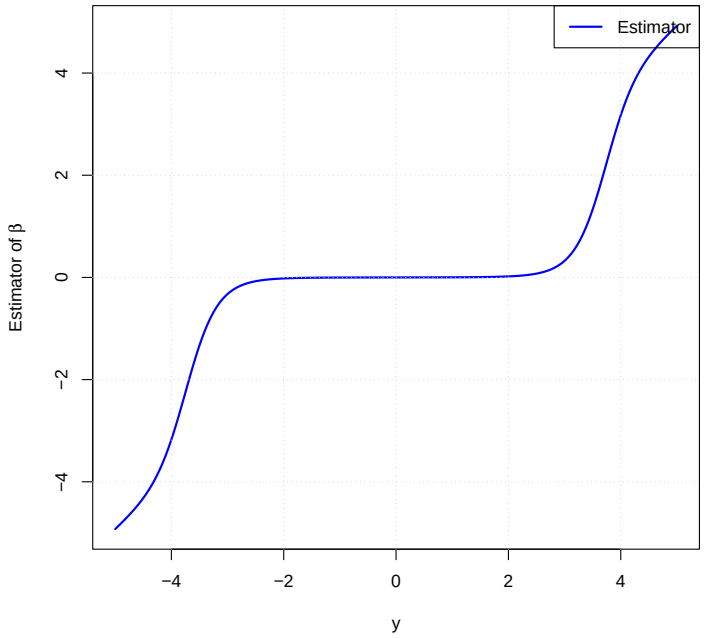


Figure 6: Estimated β for $y \in [-5, 5]$

The estimator for the β-value shares similarities with one of the lines seen in Figure (2a), of Exercise 1d). This resemblance is notable especially for $p = 1.1$, where the optimal β tends to be 0 for $y \in (-1, 1)$, though the central region of Figure (6) is wider. An assumption on the trend of Figure (6) could suggest that increasing γ from 1, in Exercise 1d), would generate an optimal β more similar to the one seen above.

c.)

For the dataset in `sparseDataWithErrors.dat`, *evaluate the estimator using the values* $p = 0.9$, *and* $\tau^2 = 80$. *Compare the result with 1.e, also in terms of residual sum of squares.*

Solution:

The implementation can be found in Appendix B, from code listings (9). It produces the following output:

$$e_{\hat{\beta}_{\text{mix}}} = 108.0825$$

In comparison to the estimates from Exercise 1.e, the $\hat{\beta}_{\text{mix}}$ estimator proves to be the most optimal, by a clear margin.

Problem 4 - (Combinatorial optimization)

In this problem we will consider a logistics problem. Assume that you work in a company which want to distribute their product to 20 cities, starting from its home city. In the file `optimaltransport.dat` you will find the locations of all 21 cities the distances between the cities is the time it takes to travel between them. The first city in the file is the home city. We want to find the loop which reduces the traveling time going from the home city, cycling through all cities and returning to the home city. You can modify the scripts found on the home page of the course to solve the problem.

a.)

Write an optimization using simulated annealing to find the optimal solution. Define your neighborhood using mathematical notation. Define your cooling schedule. Argue that your proposed algorithm can reach all possible states.

Solution:

In this implementation we have defined the neighbourhood

$$\mathcal{N}(\boldsymbol{\theta}) = \left\{\boldsymbol{\theta}^*; \exists\ i, j \text{ such that } \boldsymbol{\theta}_i^* = \boldsymbol{\theta}_j, \boldsymbol{\theta}_j^* = \boldsymbol{\theta}_i\ \wedge\ \boldsymbol{\theta}_k^* = \boldsymbol{\theta}_k, \forall k \neq i, j\right\}$$

which describes the action of swapping or switching to elements/entries in a sequence. For our problem this simply states the action of swapping to towns/cities in our heuristic. More rigidly, we could also define it as follows:

Let $\boldsymbol{\theta}$ be the tour represented as an ordered sequence of cities: $\boldsymbol{\theta} = (t_1, t_2, \ldots, t_n)$, where t_i represents the i-th town in the tour.

The neighborhood $\mathcal{N}(\boldsymbol{\theta})$ of tour $\boldsymbol{\theta}$ obtained by swapping cities i and j is defined as:

$$\mathcal{N}(\boldsymbol{\theta}) = \{\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^* \text{ is obtained by swapping towns } i \text{ and } j \text{ in } \boldsymbol{\theta}\}$$

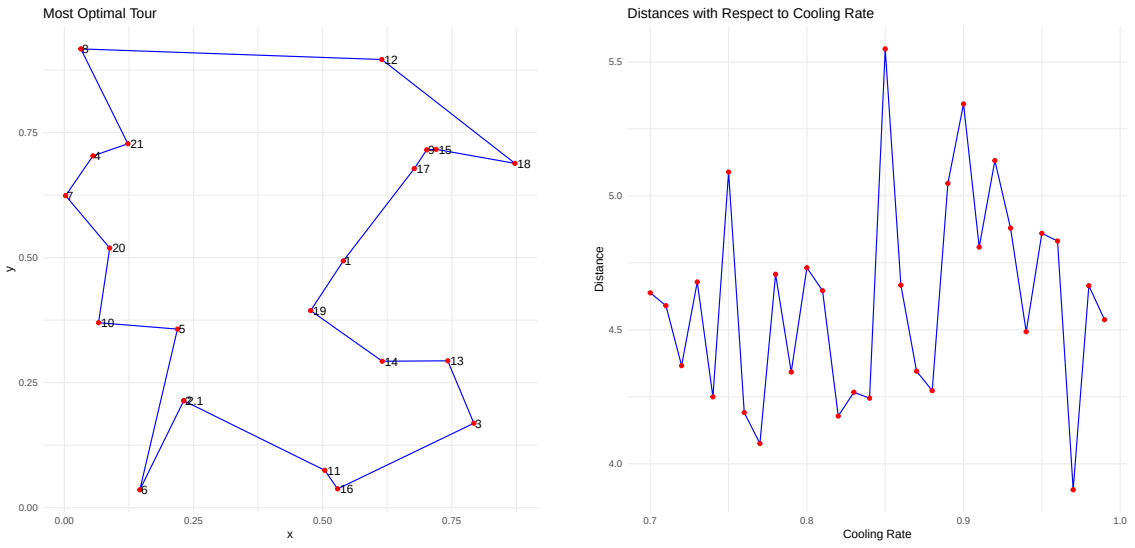
Which in more syntactic language means:

$$\mathcal{N}(\boldsymbol{\theta}) = \{\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^* = (t_1, \ldots, t_{i-1}, t_j, t_{i+1}, \ldots, t_{j-1}, t_i, t_{j+1}, \ldots, t_n)\}$$

where $\boldsymbol{\theta}^*$ is a permutation of the original tour $\boldsymbol{\theta}$ with cities i and j swapped.

The algorithm explores all possible states because it can move between any two tours by swapping adjacent cities. This connected structure ensures that every potential tour configuration can be constructed in a finite number of steps.

The code of the implementation can be found in Appendix B, from code lisitings (10) and produces the following plots



(a) (b)

Figure 7: Plots of the most optimal tour and cooling rate for the latest run.

where the all time best tour $\boldsymbol{\theta}^b$ gave

$$\boldsymbol{\theta}^b = \{17, 9, 15, 18, 12, 8, 21, 4, 7, 20, 10, 5, 2, 6, 11, 16, 3, 13, 14, 19, 1\}$$

$$\text{dist}(\boldsymbol{\theta}^b) = 3.770697$$

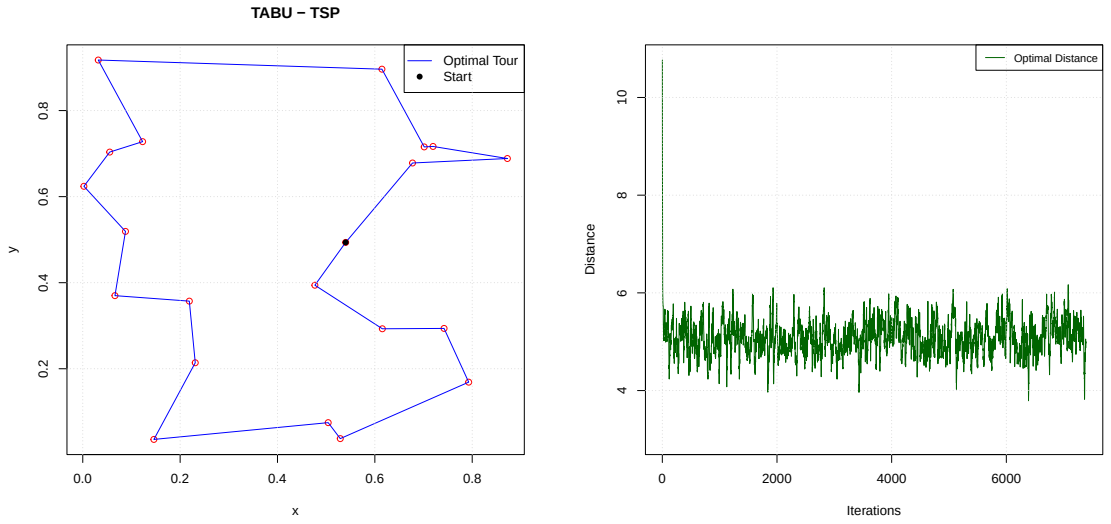
$$\alpha = 0.77 \quad \text{(Cooling rate)}$$

b.)

Implement also a TABU search for the optimal path. In what way does both the simulated annealing and the TABU algorithm differ from a steepest decent algorithm?

Solution:

The code of the implementation can be found in Appendix B, from code lisitings (11) and produces the following plots



(a) (b)

Figure 8: Plots of the most optimal tour and evolution of distance with respect to iterations.

where the all time best tour $\boldsymbol{\theta}^b$ gave

$$\boldsymbol{\theta}^b = \{1, 14, 13, 3, 16, 11, 6, 2, 5, 7, 4, 21, 12, 9, 8, 15, 18, 10, 17, 20, 19\}$$

$$\text{dist}(\boldsymbol{\theta}^b) = 3.581809$$

Simulated Annealing (SA) and Tabu Search (TS) algorithms differ from steepest descent by prioritizing exploration over exploitation. Steepest descent follows a determined search where it solely relies on the local information of the gradient. In contrast, SA and TS adopt more exploratory rules, therefore inclined to observe new territories in the solution space.

Mathematically, steepest descent iterates through solutions following the gradient direction $\nabla f(x_n)$ to minimize the objective function $f(x)$. SA introduces randomness, with the probability of accepting worse solutions given by $\mathbb{P}(\Delta E, T) = e^{-\frac{\Delta E}{T}}$, where ΔE is the change in energy and T is the temperature parameter. TS incorporates a TABU list, preventing revisits to recently explored solutions.

While steepest descent relies on deterministic information, SA and TS take a heuristic approach, utilizing randomness and memory to explore a wider solution space. This allows them to break free from local optima encountered by steepest descent and potentially find more optimal solutions elsewhere.

c.)

If your manager asks you whether the solution you propose is the optimal one, how should you reply? How can you improve confidence in your result?

Solution:

To present our boss with a sensible answer it is initially important to elaborate clearly on how complex the Traveling Salesman problem (TSP) becomes just by minute increases in quantity of data.

The TSP can be mathematically formulated as follows: Given n cities represented by vertices $V = \{1, 2, \ldots, n\}$ and the distances or costs between them represented by a distance matrix $C = [c_{ij}]$, where c_{ij} denotes the cost of traveling from city i to city j, the objective is to find the shortest Hamiltonian cycle that visits each city exactly once and returns to the starting city. A Hamiltonian cycle can be represented as a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of the cities.

The number of possible tours in the TSP is given by the factorial of the number of cities n. This is because there are n choices for the first city, $n-1$ choices for the second city, $n-2$ choices for the third city, and so on, resulting in $n \times (n-1) \times (n-2) \times \ldots \times 1 = n!$ possible tours.

n	$n!$
1	1
5	120
10	3,628,800
15	1,307,674,368,000
20	2,432,902,008,176,640,000

Table 4: $n!$ for n up to 20

However, each tour also has a mirror image (i.e., traversing the tour in reverse), which is considered equivalent. Therefore, the number of unique tours is half of $(n-1)!$, leading to $\frac{(n-1)!}{2}$ unique tours.

n	$\frac{(n-1)!}{2}$
3	1
5	12
10	181,440
15	653,837,184,000
20	1,216,451,004,088,320,000

Table 5: $\frac{(n-1)!}{2}$ for n up to 20

The combinatorial explosion of possible tours makes it computationally infeasible to exhaustively search for the optimal solution, especially for large values of n. This is where heuristic algorithms like Simulated Annealing and TABU Search become influential. They provide efficient methods for exploring the solution space, although without the guarantee of finding the most optimal solution.

To assure our manager, we propose a couple of strategies. Firstly, we could offer a visual representation of the suggested route. By plotting the cities on a map and drawing lines connecting

nearby cities, followed by longer lines connecting these regions, the manager can visually assess the proposed route's reasonability. This method offers a straightforward way to confirm that the suggested route is likely close to the best possible route.

Secondly, we could demonstrate our heuristic approaches using a simpler scenario involving, for instance, 10 cities. In this case, we can compare our results with the exact solution obtained through brute-force evaluation of all possible routes. Additionally, we could rerun our algorithm using different initial conditions and present the outcomes in a table to illustrate the variability across runs.

Furthermore, we could evaluate our heuristic methods against alternative approaches, such as the Lin-Kernighan algorithm. The Lin-Kernighan algorithm shows promise as one of the most efficient algorithms when presented with TSP (especially in cases where it is symmetric) and has a time-complexity of $\mathcal{O}(n^{\lfloor \frac{p}{2} \rfloor})$ where p represents the bactracking depth. This comparison would provide insights into the effectiveness of our methods relative to established techniques.

d.)

The company considers buying an additional lorry and want to find out how much this will reduce the distribution time. If the two lorries start at the same time the distribution time is defined as the time it takes until both lorries have returned to the home city.

Suggest a modification to your simulated annealing algorithm such that you can have two cycles which both must start and end in the home city. Define a neighborhood for this setup and argue that the algorithm can reach all possible states using your proposed neighborhood. You do not need to implement the solution.

Solution:

Let $T = \{1, 2, \ldots, n\}$ denote the number of towns/cities in our database and let H represent the home/starting town.

We then allocate p towns to L_1 (Lorry 1) and the remaining $n - p$ towns to L_2 (Lorry 2). This implies that $|T_1| = p$ and $|T_2| = n - p$ where T_1, T_2 correspond to the cycles of L_1 and L_2 respectively.

For T_1, the neighborhood $\mathcal{N}_1(\boldsymbol{\theta})$ consists of all possible solutions that can be obtained by performing 2-swaps within T_1 (i.e 4a). Mathematically, let $\boldsymbol{\theta} = (t_1, t_2, \ldots, t_p)$ be a solution for T_1, where $t_i \in T_1$ represents the city at position i in the cycle. Then, the neighborhood $\mathcal{N}_1(\boldsymbol{\theta})$ is defined as:

$$\mathcal{N}_1(\boldsymbol{\theta}) = \{\boldsymbol{\theta}^* : \boldsymbol{\theta}^* = (t_1^*, t_2^*, \ldots, t_p^*),\ t_i^* \in T_1,\ \text{and } t_i^* \neq t_j^* \text{ for } i \neq j\}$$

Similarly, for T_2, the neighborhood $\mathcal{N}_2(\boldsymbol{\theta})$ consists of all possible solutions that can be obtained by performing a 2-swaps within T_2:

$$\mathcal{N}_2(\boldsymbol{\theta}) = \{\boldsymbol{\theta}' : \boldsymbol{\theta}' = (t'_{p+1}, t'_{p+2}, \ldots, t'_n),\ t'_i \in T_2,\ \text{and } t'_i \neq t'_j \text{ for } i \neq j\}$$

Additionally, we introduce an inter-cycle 2-swap, where cities are exchanged between T_1 and T_2. Mathematically, let $\boldsymbol{\theta}^s = (t_1^s, t_2^s, \ldots, t_n^s)$ be a solution for the entire problem, where t_i^s represents the city at position i in the combined cycles. Then, the neighborhood $\mathcal{N}_{\text{inter}}(\boldsymbol{\theta}^s)$ is defined as:

$$\mathcal{N}_{\text{inter}}(\boldsymbol{\theta}^s) = \{\boldsymbol{\theta}^p : \boldsymbol{\theta}^p = (t_1^p, t_2^p, \ldots, t_n^p),\ t_i^p \in T,\ \text{and } t_i^p \neq t_j^p \text{ for } i \neq j\}$$

The nested neighborhood, along with the stochastic effect of the individual cycle 2-swaps (all towns have a probability of getting swapped to a different cycle), ensures the algorithm explores all feasible states within the solution space.

Problem 5 - (Stochastic gradient decent, SGD)

In this exercise we will try out the minibatch approach for optimizing neural nets. We will consider a simple 1D situation. The input x and output y are both one dimensional. We will use a model architecture with one hidden layer, and a width of 50. As the activation function, $\sigma(x)$, you should use the ReLu, which is defined as:

$$\sigma(x) = \begin{cases} x \text{ for } x > 0 \\ 0 \text{ for } x \leq 0 \end{cases}$$

Define also the derivative of the activation function to be:

$$\sigma'(x) = \begin{cases} 1 \text{ for } x > 0 \\ 0 \text{ for } x \leq 0 \end{cases}$$

The estimator have the form (also called the architecture):

$$f(x) = \sum_{i=1}^{50} \beta_i \sigma\left(\alpha_i x_i + \alpha_{0,i}\right) + \beta_0$$

We will minimize the squared sum of errors:

$$\text{sse } = \sum_{i=1}^{N} \left(y_i - \hat{f}\left(x_i\right)\right)^2$$

a.)

Adapt the formulas from the lecture slides or the SGD note of Geir Storvik such that these can be used for the specific problem formulation defined above. Present the formulas that are relevant for implementing SGD on the current problem. How many parameters are there in the architecture?

Solution:

When dealing with neural networks, it is crucial to optimize the magnitude of the cost or loss function associated with the architecture. This optimization process involves constructing and examining the backpropagation steps, which are responsible for training our model. Backpropagation essentially estimates the parameters of interest in the given model derived from the information of the gradient. This in turn tells us how we should update the parameters.

By this reasoning we proceed to calculate the gradient of our cost function

$$\mathcal{C}(\theta) = \text{sse} = \sum_{i=1}^{N} \left(y_i - \hat{f}\left(x_i\right)\right)^2$$

with respect to all influential parameters $\beta_0, \beta_i, \alpha_i, \alpha_{0,i}$. This yields

$$\frac{\partial}{\partial \beta_0}\mathcal{C}(\theta) = -2\sum_{i=1}^{N}(y_i - f(x_i)) \tag{35}$$

$$\frac{\partial}{\partial \beta_i}\mathcal{C}(\theta) = -2\sum_{i=1}^{N}(y_i - f(x_i))\,\sigma\left(\alpha_i x + \alpha_{0,i}\right) \tag{36}$$

$$\frac{\partial}{\partial \alpha_i}\mathcal{C}(\theta) = -2\sum_{i=1}^{N}(y_i - f(x_i))\,\beta_i\sigma'\left(\alpha_i x + \alpha_{0,i}\right)x_i \tag{37}$$

$$\frac{\partial}{\partial \alpha_{0,i}}\mathcal{C}(\theta) = -2\sum_{i=1}^{N}(y_i - f(x_i))\,\beta_i\sigma'\left(\alpha_i x + \alpha_{0,i}\right). \tag{38}$$

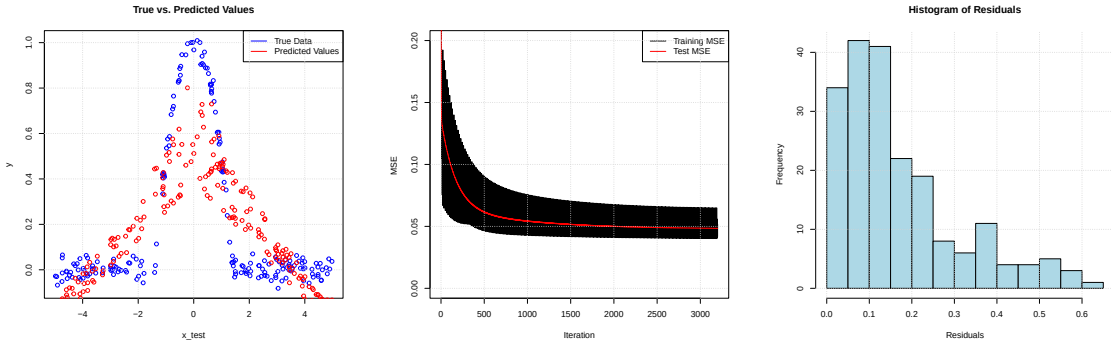
Since we are presented with a network of $N = 50$ nodes there are hence 50 values of β_i, 50 α_i, 50 $\alpha_{0,1}$ and additionally a singular β_0, giving $50 + 50 + 50 + 1 = 151$ parameters.

b.)

Implement the SGD for the architecture above and test it on the data found in `functionEstimationNN.dat` *Test different alternatives for the learning rate and record the test error for each epoch. Use a batch size of 50.*

Solution:

The code of the implementation can be found in Appendix B, from code lisitings (12) and produces the following plots



(a) Point plot, True vs Predicted (b) Evolution of MSE (c) Freq - Historgram, residuals

Figure 9: Various plots of accuracy measurements

c.)

Discuss the choices made and comment on the results from b.

Solution:

The results in (b.) show clear promise of an architecture that yield both feasible and decent estimates. We can be assured that for the given data, and presumably data structured simirlaly in 1D, the network will provide an "accurate" [5] description of its reality. But there are naturally

[5] Accurate is in every sense of its meaning interpreted loosely, especially in statistics, but one might be advised to understand it in this particular instance as a precise regression estimate.

a few catches to elaborate on.

Having a neural network with only one layer of 50 nodes (or any number of nodes > 1) may not be advised for several reasons. Firstly, the complexity of many real-world problems often require more than one layer to capture the intricate and oftentimes stochastic patterns in the data effectively. A single layer might not have enough capacity to learn complex relationships between inputs and outputs. Additionally, a larger number of nodes in a single layer can lead to overfitting, where the network memorizes the training data instead of learning how to generalise patterns, especially if the dataset is not sufficiently large or sparse.

The ReLU activation function, while being simplistic and capable of diminishing the vanishing gradient problem, may not always be the best choice. Its main limitation is the "dying ReLU" problem, where neurons can become inactive during training. Furthermore, ReLU lacks a bounded output range, which can also lead to exploding activations. It is not suitable for models requiring negative activations, such as some generative models. Alternative functions like Leaky ReLU or ELU might be better in such cases. In general, for overall performance in a neural network, functions like ADAM, tanh, Sigmoid and RMSprop might be preffered.

Lastly, a neural network's sensitivity to learning rate comes from its crucial role in controlling the size of parameter updates during training. If the learning rate is too high, the network may fail to converge or oscillate. Conversely, if it is too low, convergence might be slow. Thus, selecting an optimal learning rate is vital for stable and efficient training. In this respective example we observed the prominent sensitivity which the model had towards its parameters during training, notably for higher learning rates. It was found that on many occasions the gradients would simply explode due to the instability of the hyper-parameters which led to more meticulous grid-searches on the optimal balance between inital learning rate and learning rate decay.

Appendix

A. GitHub repository

B. Source Code

Exercise 1 c): Plot the function $f_{\gamma,p}(\beta)$ for $\gamma = 1$ and $\gamma = 0.2$

```r

# Define parameters
beta <- seq(-5, 5, length = 101)
gamma_list <- c(1, 0.2)
p_list <- c(1.1, 2, 5, 100)
colors <- c("black", rainbow(length(p_list) - 1, start = 0.2, end = 0.7))

# Define the function
f <- function(beta, gamma, p) {
  beta + gamma * sign(beta) * abs(beta) ^ (p - 1)
}

# Plotting function
plot_function <- function(gamma, title) {
  plot(beta, rep(0, length(beta)), type = 'n', ylim = c(-8, 8),
       xlim = c(-5.1, 5.1), xlab = expression(beta), lwd = 2, main =
    title, ylab = expression(paste("f"["p, y"], "(", beta, ")")))
  for (i in 1:length(p_list)) {
    lines(beta, f(beta, gamma, p_list[i]), col = colors[i], lwd = 2)
  }
  legend('topleft', legend = paste('p =', p_list), col = colors, lwd = 2,
    bty = 'n', title = "Values of p", xjust = 0, yjust = 1, inset = c(0,
    0.05))
  lines(beta, beta, col = 'red')
  text(mean(range(beta)), max(beta) * 1.1, labels = expression(paste("f",
    "(", beta, ")", " = ", beta)), pos = 3, col = 'red')
}

# Set the size of the plot
options(repr.plot.width = 12, repr.plot.height = 8)

# Plotting for each value of gamma
eq1 <- bquote(bold(gamma == .(gamma_list[1])))
eq2 <- bquote(bold(gamma == .(gamma_list[2])))

plot_function(gamma_list[1], eq1)
plot_function(gamma_list[2], eq2)

# Save figures as PDF
# pdf(file = "plots.pdf")
# plot_function(gamma_list[1], eq1)
# plot_function(gamma_list[2], eq2)
# dev.off()
```

Listing 1: Exercise 1c

Exercise 1 d): Root of bisection method for $f_{\gamma,p}(\beta)$

```r
# Define the function
f <- function(beta, gamma, p) {
  beta + gamma * sign(beta) * abs(beta) ^ (p - 1)
}

# Bisection method for finding the root of an expression
bisection_method <- function(y, gamma, p, init_beta1, init_beta2, eps = 1e-10, max_iter = 1000) {
  func <- function(beta) f(beta, gamma, p) - y
  check_roots <- func(init_beta1) * func(init_beta2)
  if (check_roots == 0) return(ifelse(func(init_beta1) == 0, init_beta1, init_beta2))
  if (check_roots > 0) stop('The initial guesses have the same sign of the function (not acceptable)')
  iter <- 0
  while (iter < max_iter) {
    mid_beta <- (init_beta1 + init_beta2) / 2
    if (func(mid_beta) * func(init_beta1) < 0) init_beta2 <- mid_beta
    else if (func(mid_beta) * func(init_beta2) < 0) init_beta1 <- mid_beta
    else return(mid_beta)
    iter <- iter + 1
    if (abs(func(mid_beta)) < eps) return(mid_beta)
  }
  return(mid_beta)
}
# Test bisection method
gamma_val <- 1
p_values <- c(1.1, 2, 100)
y_values <- seq(-5, 5, length = 101)
init_beta1_val <- -5
init_beta2_val <- 4.9

# Plotting function
plot_function <- function(p) sapply(y_values, function(y_i) bisection_method(y_i, gamma_val, p, init_beta1_val, init_beta2_val))

# Plotting
plot_colors <- rainbow(length(p_values))
plot_legend <- paste('p =', p_values)

pdf("bisection_plots.pdf")
plot(y_values, plot_function(p_values[1]), type='l', col = plot_colors[1], ylim = c(-5, 5),
     xlim = c(-5.1, 5.1), lwd = 2, main = expression(paste("Optimal"," ", beta)), xlab = 'y', ylab = expression(beta["opt"]))
for (i in 2:length(p_values)) lines(y_values, plot_function(p_values[i]), col = plot_colors[i], lwd = 2)
legend('topleft', legend = plot_legend, col = plot_colors, lwd = 2, bty = 'n', title = "Values of p", xjust = 0, yjust = 1, inset = c(0, 0.05))
dev.off()
```

Listing 2: Exercise 1d

Exercise 1 e): Estimation of parameters β_i, `sparseDataWithErrors.dat`

```r
# Define the function
f <- function(beta, gamma_val, p) {
  beta + gamma_val * sign(beta) * abs(beta) ^ (p - 1)
}

# Bisection method for finding the root of an expression
bisection_method <- function(y, gamma, p, init_beta1, init_beta2, eps =
    1e-10, max_iter = 1000) {
  func <- function(beta) f(beta, gamma, p) - y
  check_roots <- func(init_beta1) * func(init_beta2)
  if (check_roots == 0) return(ifelse(func(init_beta1) == 0, init_beta1,
    init_beta2))
  if (check_roots > 0) stop('The initial guesses have the same sign of
    the function (not acceptable)')
  iter <- 0
  while (iter < max_iter) {
    mid_beta <- (init_beta1 + init_beta2) / 2
    if (func(mid_beta) * func(init_beta1) < 0) init_beta2 <- mid_beta
    else if (func(mid_beta) * func(init_beta2) < 0) init_beta1 <- mid_beta
    else return(mid_beta)
    iter <- iter + 1
    if (abs(func(mid_beta)) < eps) return(mid_beta)
  }
  return(mid_beta)
}

# Load data
sparseData <- read.table("data/sparseDataWithErrors.ascii", header =
    FALSE)
colnames(sparseData) <- c('betaGT', 'y')

# Collecting the data from the dataframe
y <- sparseData$y
betaGT <- sparseData$betaGT

# We want to estimate beta_optimal for penalized regression
gamma_val <- 1
p_values <- c(1.1, 2, 100)
init_beta1_val <- -100
init_beta2_val <- 101
residuals <- c()
beta_p <- list()
# Finding the residuals for different p-values
for (p in p_values){
  estimation <- c()
  for (y_i in y){
    estimation <- c(estimation, bisection_method(y_i, gamma_val, p,
    init_beta1_val, init_beta2_val))
  }

  # comparing the estimation vs the ground truth
  beta_p <- c(beta_p, list(estimation))
  residuals <- c(residuals, sum((estimation - betaGT)^2))
}

residuals <- data.frame(p_values, residuals)

# Comparison to the MLE estimator, just equal to y
beta_MLE <- y
residual_MLE <- sum((beta_MLE - betaGT)^2)

# Estimate beta by using the residuals
y_list3 <- rep(list(y), 3)

beta_alternative <- list()
residuals_alternative <- rep(NA, 3)
for (i in 1:3) {
```

```r
  beta_alternative <- c(beta_alternative, list(unlist(y_list3[i]) -
    unlist(beta_p[i])))
  residuals_alternative[i] <- sum((unlist(beta_alternative[i]) -
    betaGT)^2)
}

residuals_alternative_table <- data.frame(p_values, residuals_alternative)

# Print residuals
print(residuals)
print(residual_MLE)
print(residuals_alternative_table)
```

Listing 3: Exercise 1e

Exercise 2 d): EM-algorithm, $Q(\theta|\theta^(t)$, `sparseDataWithErrors.dat`

```r
# Define the function to calculate p_i
calculate_pi <- function(y_i, squared_tau, p) {
  num <- dnorm(y_i, mean = 0, sd = 1) * p
  denom <- num + dnorm(y_i, mean = 0, sd = sqrt(squared_tau + 1)) * (1 -
    p)
  num / denom
}

# Define the EM algorithm function
EM_algorithm <- function(y, p_init, squared_tau_init, printing = FALSE,
    eps = 1e-10, max_iter = 1000) {
  iter <- 1
  converged <- FALSE
  p_prev <- p_init
  squared_tau_prev <- squared_tau_init

  while (!(converged) && iter < max_iter) {
    p_new <- mean(sapply(y, function(y_i) calculate_pi(y_i,
    squared_tau_prev, p_prev)))
    squared_tau_new <- sum(sapply(y, function(y_i) {
      pi_val <- calculate_pi(y_i, squared_tau_prev, p_prev)
      (1 - pi_val) * y_i^2
    })) / sum(sapply(y, function(y_i) 1 - calculate_pi(y_i,
    squared_tau_prev, p_prev))) - 1

    # Check for convergence
    if (abs(p_new - p_prev) + abs(squared_tau_new - squared_tau_prev) <
    eps) {
      converged <- TRUE
    }

    if (printing) {
      print(sprintf('iter: %2.0f, %f %f', iter, p_prev, squared_tau_prev))
    }

    # Update parameters
    p_prev <- p_new
    squared_tau_prev <- squared_tau_new

    iter <- iter + 1
  }

  c(p_new, squared_tau_new)
}
```

```r

# Read the sparse data
sparseData <- read.table("data/sparseDataWithErrors.ascii", header = FALSE)
colnames(sparseData) <- c('betaGT', 'y')

# Extract data from the dataframe
y <- sparseData$y

# Initialize parameters
squared_tau_init <- 3
p_init <- 0.5

# Run the EM algorithm and collect results
values <- EM_algorithm(y, p_init, squared_tau_init, printing = TRUE)
p_opt <- values[1]
squared_tau_opt <- values[2]
```

Listing 4: Exercise 2d

Exercise 2 e): Bootstrap estimate

```r
# Set seed for reproducibility
set.seed(123)

# Read sparse data
sparse_data <- read.table("data/sparseDataWithErrors.ascii", header = FALSE)
colnames(sparse_data) <- c('betaGT', 'y')

# Extract data
y <- sparse_data$y

# Define function to calculate p_i
calculate_pi <- function(y_i, tau_squared, p) {
  numerator <- dnorm(y_i, mean = 0, sd = 1) * p
  denominator <- numerator + dnorm(y_i, mean = 0, sd = sqrt(tau_squared + 1)) * (1 - p)
  numerator / denominator
}

# Define EM algorithm function
EM_algorithm <- function(y, p_init, tau_squared_init, eps = 1e-10, max_iter = 1000) {
  iter <- 1
  converged <- FALSE
  p_prev <- p_init
  tau_squared_prev <- tau_squared_init

  while (!(converged) && iter < max_iter) {
    p_new <- mean(sapply(y, function(y_i) calculate_pi(y_i, tau_squared_prev, p_prev)))
    tau_squared_new <- sum(sapply(y, function(y_i) {
      pi_val <- calculate_pi(y_i, tau_squared_prev, p_prev)
      (1 - pi_val) * y_i^2
    })) / sum(sapply(y, function(y_i) 1 - calculate_pi(y_i, tau_squared_prev, p_prev))) - 1

    # Check for convergence
    if (abs(p_new - p_prev) + abs(tau_squared_new - tau_squared_prev) < eps) {
```

```r
      converged <- TRUE
    }

    # Update parameters
    p_prev <- p_new
    tau_squared_prev <- tau_squared_new

    iter <- iter + 1
  }

  c(p_new, tau_squared_new)
}

# Initialization
tau_squared_init <- mean(y)
p_init <- 0.5

# Run EM algorithm
values <- EM_algorithm(y, p_init, tau_squared_init)
p_opt <- values[1]
tau_squared_opt <- values[2]

# Bootstrap method
B <- 1000  # Increase to 1000
n <- length(y)
p_simulated <- numeric(B)
tau_squared_simulated <- numeric(B)

for (b in 1:B) {
  y_sample <- sample(y, n, replace = TRUE)
  values <- EM_algorithm(y_sample, p_init, tau_squared_init)
  p_simulated[b] <- values[1]
  tau_squared_simulated[b] <- values[2]
  if(b %% 100 == 0){
    print(b)
  }
}

# Plot results and save to PDF with blue color
pdf("bootstrap_results.pdf")
plot(tau_squared_simulated, p_simulated, xlab = "Estimated Tau Squared",
     ylab = "Estimated p", col = "blue")
dev.off()

# Bias
cat("Bias:\n")
print(mean(p_simulated) - p_opt)
print(mean(tau_squared_simulated) - tau_squared_opt)

# Standard error
cat("Standard error:\n")
print(sd(p_simulated))
print(sd(tau_squared_simulated))

# 95% confidence interval for p_simulated
p_conf_int <- quantile(p_simulated, c(0.025, 0.975))
cat("95% Confidence interval for p_simulated:\n")
print(p_conf_int)

# 95% confidence interval for tau_squared_simulated
tau_squared_conf_int <- quantile(tau_squared_simulated, c(0.025, 0.975))
cat("95% Confidence interval for tau_squared_simulated:\n")
print(tau_squared_conf_int)
```

Listing 5: Exercise 2e

Exercise 2 f): Observed information: -H_ℓ

```r
set.seed(123)

# Read sparse data
sparse_data <- read.table("data/sparseDataWithErrors.ascii", header =
    FALSE)
colnames(sparse_data) <- c('betaGT', 'y')

# Extract data
y <- sparse_data$y

# Define the log-likelihood function
log_likelihood <- function(params, y) {
  p <- params[1]
  tau_squared <- params[2]

  log_term <- sum(log(p * dnorm(y, mean = 0, sd = 1) + (1 - p) * dnorm(y,
    mean = 0, sd = sqrt(tau_squared + 1))))
  return(-log_term)  # Note: We negate the log-likelihood for minimization
}

# Set the parameters and data
params <- c(p = 0.6, tau_squared = 3)

# Define a function to compute the negative log-likelihood
neg_log_likelihood <- function(params) {
  return(log_likelihood(params, y))
}

# Use the `optim()` function to estimate the observed Fisher information
observed_fisher_information <- function(params, y) {
  hessian <- optim(params, neg_log_likelihood, method = "BFGS", hessian =
    TRUE, control = list(trace = 0))$hessian
  fisher_info <- hessian
  return(fisher_info)
}

# Compute the observed Fisher information
fisher_info <- observed_fisher_information(params, y)
inv_fisher <- solve(fisher_info)
print(fisher_info)
print(inv_fisher)
```

Listing 6: Exercise 2f

Exercise 2 g): Contour of likelihood: $\ell(\theta|y)$

```r
# Read sparse data and rename columns
data_df <- read.table("data/sparseDataWithErrors.ascii", header = FALSE)
colnames(data_df) <- c('beta_GT', 'y')

# Extract data
y <- data_df$y

# Define the log-likelihood function
likelihood <- function(y, p, tau_sq) {
  fy <- p * dnorm(y, mean = 0, sd = 1) + (1 - p) * dnorm(y, mean = 0, sd
    = sqrt(tau_sq + 1))
  return(sum(log(fy)))
}

# Define the grid for p and tau_sq
p_grid <- seq(0.8, 1, length.out = 101)
tau_sq_grid <- seq(50, 130, length.out = 101)

# Calculate the log-likelihood values on the grid
z <- outer(p_grid, tau_sq_grid, Vectorize(function(p, tau_sq)
    likelihood(y, p, tau_sq)))

# Normalize the log-likelihood values
z <- max(z) / z

# Define contour levels
levels_ <- c(0.01, 0.1, 0.5, 0.75, 0.95, 0.98, 0.99, 0.9999, 1)

red_palette <- colorRampPalette(c("#67000D", "#A50F15", "#CB181D",
    "#EF3B2C", "#FB6A4A", "#FC9272", "#FCBBA1", "#FEE0D2"))

# Plot the contour plot with the reversed custom red color palette
pdf("contour_plot.pdf")
#contour(tau_sq_grid, p_grid, z, levels = levels_, xlab = "squared_tau",
    ylab = "p", col = red_palette(length(levels_)))
filled.contour(tau_sq_grid, p_grid, z, ylab = 'p', xlab =
    expression(tau^2), levels = levels_, col =
    red_palette(length(levels_)))

# Find the ML estimator and mark it in the plot
ML_idx <- which(z == max(z), arr.ind = TRUE)
points(tau_sq_grid[ML_idx[2]], p_grid[ML_idx[1]], col = "black", pch = 16)

# Add label for ML estimator coordinates
text(tau_sq_grid[ML_idx[2]], p_grid[ML_idx[1]], labels = paste("(",
    tau_sq_grid[ML_idx[2]], ",", p_grid[ML_idx[1]], ")"), pos = 3)
dev.off()
```

Listing 7: Exercise 2g

Exercise 3 b): Plot of conditional expectation:
$\mathbb{E}(\beta_i|y_i) = \mathbb{P}(C_i = 1 \mid y_i = y)\,\mathbb{E}(\beta_i \mid y_i = y, C_i = 1)$

```r
# Function to calculate P(C_i=1 | y_i=y)
calc_cond_prob <- function(y, p, tau) {
  num <- p * dnorm(y, 0, 1)
  den <- num + (1 - p) * dnorm(y, 0, tau + 1)
  p_0 <- num / den
  return(1 - p_0)
}

# Function to calculate E(beta | y_i=y)
calc_expectation <- function(y, tau) {
  e <- y * tau / (tau + 1)
  return(e)
}

# Function to calculate beta estimator given y_i=y
calc_beta_hat <- function(y, p, tau) {
  p_c1 <- calc_cond_prob(y, p, tau)
  e <- calc_expectation(y, tau)
  beta_hat <- e * p_c1
  return(beta_hat)
}

# Generate sequence of y values
y_values <- seq(-5, 5, length.out = 501)
p <- 0.90
tau <- 80

# Calculate beta estimator for each y value
beta_hat_y <- calc_beta_hat(y_values, p, tau)

# Plotting
pdf("estimator_plot.pdf")  # Open PDF device
plot(y_values, beta_hat_y, type = "l", col = "blue", lwd = 2, xlab = "y",
     ylab = expression(paste("Estimator of"," ", beta)), ylim =
     range(beta_hat_y))
legend("topright", legend = "Estimator", col = "blue", lty = 1, lwd = 2)
grid()
dev.off()  # Close PDF device
```

Listing 8: Exercise 3b

Exercise 3 c) Evaluation of estimator: `sparseDataWithErrors.dat`, $p = 0.9$ **and** $\tau^2 = 80$

```r
# Read the sparse data
sparseData <- read.table("data/sparseDataWithErrors.ascii", header =
    FALSE)
colnames(sparseData) <- c('betaGT', 'y')
betaGT <- sparseData$betaGT
# Extract data from the dataframe
y <- sparseData$y

# Function to calculate P(C_i=1 | y_i=y)
calc_cond_prob <- function(y, p, tau) {
  num <- p * dnorm(y, 0, 1)
  den <- num + (1 - p) * dnorm(y, 0, tau + 1)
  p_0 <- num / den
  return(1 - p_0)
```

```r
}

# Function to calculate E(beta | y_i=y)
calc_expectation <- function(y, tau) {
  e <- y * tau / (tau + 1)
  return(e)
}

# Function to calculate beta estimator given y_i=y
calc_beta_hat <- function(y, p, tau) {
  p_c1 <- calc_cond_prob(y, p, tau)
  e <- calc_expectation(y, tau)
  beta_hat <- e * p_c1
  return(beta_hat)
}

# Generate sequence of y values
p <- 0.90
tau <- 80

# Calculate beta estimator for each y value
beta_hat_y <- calc_beta_hat(y, p, tau)

residuals = sum((beta_hat_y-betaGT)^2)

print(residuals)
```

Listing 9: Exercise 3c

Exercise 4 a): Simulated annealing for Travelling salesman problem (TSP)

```r
library(ggplot2)

# Read data
optimalTransport <- read.table("data/optimalTransport.ascii", header =
    FALSE)
colnames(optimalTransport) <- c('x', 'y')

set.seed(124)
# Define neighborhood function (swap two cities)
swap_cities <- function(tour) {
  n <- length(tour)
  i <- sample(2:(n - 1), 1)
  j <- sample(setdiff(1:n, i), 1)
  new_tour <- tour
  new_tour[c(i, j)] <- new_tour[c(j, i)]
  return(new_tour)
}

# Simulated annealing algorithm
simulated_annealing <- function(distances, initial_temp, cooling_rate,
    max_iter) {
  # Define objective function (total traveling time)
  total_time <- function(tour, distances) {
    total <- 0
    n <- length(tour)
    for (i in 1:(n - 1)) {
      total <- total + distances[tour[i], tour[i + 1]]
    }
    total <- total + distances[tour[n], tour[1]]  # Return to starting
    city
    return(total)
```

```r
  }

  n <- nrow(distances)
  current_tour <- sample(1:n)
  best_tour <- current_tour
  best_time <- total_time(current_tour, distances)
  current_temp <- initial_temp

  for (iter in 1:max_iter) {
    new_tour <- swap_cities(current_tour)
    # Calculate total time
    delta <- total_time(new_tour, distances) - total_time(current_tour,
    distances)

    if (delta < 0 || runif(1) < exp(-delta / current_temp)) {
      current_tour <- new_tour
      current_time <- total_time(new_tour, distances)
      if (current_time < best_time) {
        best_tour <- new_tour
        best_time <- current_time
      }
    }

    current_temp <- current_temp * cooling_rate
  }

  return(list(best_tour = best_tour, best_time = best_time))
}

# Calculate distances matrix (Euclidean distance between cities)
distances <- as.matrix(dist(cbind(optimalTransport$x,
    optimalTransport$y)))

# Set parameters
initial_temp <- 10
cooling_rates <- seq(0.7, 0.99, by = 0.01)  # Range of cooling rates
max_iter <- 10000

# Store results
results <- list()

# Run simulated annealing for each cooling rate
for (cooling_rate in cooling_rates) {
  result <- simulated_annealing(distances, initial_temp, cooling_rate,
    max_iter)
  best_tour <- result$best_tour
  best_time <- result$best_time
  results[[as.character(cooling_rate)]] <- list(tour = best_tour,
    total_time = best_time)
}

# Find the best cooling rate
best_result <- which.min(sapply(results, function(x) x$total_time))
best_cooling_rate <- as.numeric(names(results)[best_result])

# Output best tour and total time for the best cooling rate
cat("Best Cooling Rate:", best_cooling_rate, "\n")
cat("Best Tour:", results[[as.character(best_cooling_rate)]]$tour, "\n")
cat("Best Total Time:",
    results[[as.character(best_cooling_rate)]]$total_time, "\n")

# Plot the most optimal trip
optimal_path <- c(results[[as.character(best_cooling_rate)]]$tour,
    results[[as.character(best_cooling_rate)]]$tour[1])
cities <- optimalTransport[optimal_path, ]

pdf("opt_annealing.pdf")
ggplot() +
  geom_path(data = cities, aes(x = x, y = y), color = "blue") +
  geom_point(data = cities, aes(x = x, y = y), color = "red") +
```

```r
  geom_text(data = cities, aes(x = x, y = y, label = rownames(cities)),
    hjust = -0.2, vjust = 0.5) +
  ggtitle("Most Optimal Tour") +
  theme_minimal()
dev.off()
# Plot distances with respect to cooling rates
data <- data.frame(cooling_rates = as.numeric(names(results)),
                   total_time = sapply(results, function(x) x$total_time))

pdf("opt_cooling.pdf")
ggplot(data, aes(x = cooling_rates, y = total_time)) +
  geom_line(color = "blue") +
  geom_point(color = "red") +
  labs(x = "Cooling Rate", y = "Distance") +
  ggtitle("Distances with Respect to Cooling Rate") +
  theme_minimal()

dev.off()
#Random seed
# Best Cooling Rate: 0.77
# > cat("Best Tour:", results[[as.character(best_cooling_rate)]]$tour,
    "\n")
# Best Tour: 17 9 15 18 12 8 21 4 7 20 10 5 2 6 11 16 3 13 14 19 1
# > cat("Best Total Time:",
    results[[as.character(best_cooling_rate)]]$total_time, "\n")
# Best Total Time: 3.770697


#Seed(124)
# Best Cooling Rate: 0.97
# > cat("Best Tour:", results[[as.character(best_cooling_rate)]]$tour,
    "\n")
# Best Tour: 2 11 16 3 13 14 19 1 17 9 15 18 12 8 21 4 7 20 10 5 6
# > cat("Best Total Time:",
    results[[as.character(best_cooling_rate)]]$total_time, "\n")
# Best Total Time: 3.902836
```

Listing 10: Exercise 4a

Exercise 4 b): TABU - Search for Travelling salesman problem (TSP)

```r

set.seed(2000)
# Read city data
city_data <- read.table("data/optimalTransport.ascii", header = FALSE)
colnames(city_data) <- c("x", "y")

# Function to calculate total journey distance
calculate_journey_distance <- function(route, distances) {
  total <- sum(distances[cbind(route[-length(route)], route[-1])])
  total <- total + distances[route[length(route)], route[1]]  # Return to
    starting point
  return(total)
}

# Function to perform TABU search
find_optimal_route <- function(distances, max_iterations,
    tabu_list_length) {
  num_cities <- nrow(distances)  # Number of cities
  theta <- c(1, sample(2:num_cities), 1)# Random order and returns to the
    starting city
  d <- as.vector(as.matrix(distances))
```

```r
  initial_distance <- calculate_journey_distance(theta, distances)  # Total distance of the initial route
  optimal_distance <- initial_distance  # Initial optimal distance
  distance_sequence <- initial_distance  # Sequence of distances
  num_swaps <- num_cities * (num_cities - 1) / 2 - (num_cities - 1)  # Number of iterations for neighbor search
  search_table <- combn(2:num_cities, 2)  # Table of neighbor search
  tabu_table <- numeric(0)  # Tabu table
  for (iteration in 1:max_iterations) {
    optimal_distance2 <- optimal_distance + 10
    for (i in 1:num_swaps) {
      if (!(i %in% tabu_table)) {
        swap_index1 <- search_table[1, i]
        swap_index2 <- search_table[2, i]
        new_route <- theta
        new_route[c(swap_index1, swap_index2)] <- theta[c(swap_index2, swap_index1)]
        new_distance <- calculate_journey_distance(new_route, distances)
        if (new_distance < optimal_distance2) {
          optimal_distance2 <- new_distance
          swap_index <- i
        }
      }
    }
    theta[c(search_table[1, swap_index], search_table[2, swap_index])] <- theta[c(search_table[2, swap_index], search_table[1, swap_index])]
    optimal_distance <- optimal_distance2
    distance_sequence <- c(distance_sequence, optimal_distance)
    tabu_table <- c(tabu_table, swap_index)
    if (length(tabu_table) > tabu_list_length) {
      tabu_table <- tabu_table[-1]
    }
    if (optimal_distance < initial_distance) {
      optimal_route <- theta
      initial_distance <- optimal_distance
    }
  }
  return(list(optimal_route = optimal_route, shortest_distance = initial_distance, distance_sequence = distance_sequence))
}


# Calculate distances between cities
city_distances <- as.matrix(dist(city_data))

# Set parameters for TABU search
max_iterations <- 1000
tabu_list_length <- 40

# Run TABU search
result <- find_optimal_route(city_distances, max_iterations, tabu_list_length)
optimal_route <- result$optimal_route
shortest_distance <- result$shortest_distance
distance_sequence <- result$distance_sequence

# Print the results
cat("Shortest distance:", shortest_distance, "\n")
cat("Optimal route:\n")
print(optimal_route)
cat("Number of iterations to find optimal route:", which.min(distance_sequence), "\n")

pdf("opt_tabu.pdf")
plot(city_data, main = "TABU - TSP", col = "red")
lines(city_data[optimal_route, 1], city_data[optimal_route, 2], col = "blue")
points(city_data[1, ], col = "black", pch = 16)
legend("topright", legend = c("Optimal Tour", "Start"), col = c("blue", "black"), pch = c(NA, 16),  lty=c(1, NA))
grid()
```

```r
dev.off()

# Plot time series of the shortest distance found over iterations
optimal_route_iteration <- which.min(distance_sequence)
reasonable_plot_range <- min(optimal_route_iteration + 1000,
    max_iterations)
pdf("distance_iter.pdf")
plot.ts(distance_sequence[1:reasonable_plot_range], xlab = "Iterations",
    ylab = "Distance", col = "darkgreen", ylim = c(3,
    max(distance_sequence)))
legend("topright", legend = c("Optimal Distance"), col = c("darkgreen"),
    lty = 1, cex = 0.8)
grid()
dev.off()




# Seed 2000

# > cat("Shortest distance:", shortest_distance, "\n")
# Shortest distance: 3.776257
# > cat("Optimal route:\n")
# Optimal route:
#   > print(optimal_route)
# [1]  1 19 13  3 16 11  6  2  5 10 20  7  4 21  8 12  9 15 18 17 17  1
# > cat("Number of iterations to find optimal route:",
    which.min(distance_sequence), "\n")
# Number of iterations to find optimal route: 927

#Random seed

# > cat("Shortest distance:", shortest_distance, "\n")
# Shortest distance: 3.732432
# > cat("Optimal route:\n")
# Optimal route:
#   > print(optimal_route)
# [1]  1 14  3 18 18 17 17  9  9 15 12  8 21  4  7 20 10  6  2  5 19  1
# > cat("Number of iterations to find optimal route:",
    which.min(distance_sequence), "\n")
# Number of iterations to find optimal route: 900

#All time best random seed

# > cat("Shortest distance:", shortest_distance, "\n")
# Shortest distance: 3.581809
# > cat("Optimal route:\n")
# Optimal route:
#   > print(optimal_route)
# [1]  1 14 13  3 16 11  6  2  5  7  4 21 12  9  8 15 18 10 17 20
    19
# > cat("Number of iterations to find optimal route:",
    which.min(distance_sequence), "\n")
# Number of iterations to find optimal route: 292
```

Listing 11: Exercise 4b

Exercise 5 b): Stochastic Gradient Descent

```r
# Load data
dataset <- read.table("data/functionEstimationNN.ascii", header = FALSE,
    col.names = c("x", "y"))
```

```r

# Sample a subset of the data
set.seed(123) # For reproducibility
subset_size <- 1000
subset <- dataset[sample(nrow(dataset), subset_size), ]

# Train-test split
train_ratio <- 0.8
train_indices <- sample(1:nrow(subset), train_ratio * nrow(subset))
train_set <- subset[train_indices, ]
test_set <- subset[-train_indices, ]

# Extract train and test data
x_train <- train_set$x
y_train <- train_set$y
x_test <- test_set$x
y_test <- test_set$y

# Network parameters
n_inputs <- 1
n_hidden <- 50
n_outputs <- 1

# Initialize weights and biases
weights_hidden <- matrix(rnorm(n_inputs * n_hidden, sd =
    sqrt(2/n_inputs)), nrow = n_inputs, ncol = n_hidden)
bias_hidden <- runif(n_hidden, min = 0, max = 1)
weights_output <- matrix(rnorm(n_hidden * n_outputs, sd =
    sqrt(2/n_hidden)), nrow = n_hidden, ncol = n_outputs)
bias_output <- runif(n_outputs, min = 0, max = 1)

# Hyperparameters
learning_rate <- 0.001
learning_rate_decay <- 1e-4
batch_size <- 50
n_epochs <- 200

# Train network with SGD
SSE_seq <- MSE_seq <- MSE_test_seq <- NULL
iter <- 0
for (epoch in 1:n_epochs) {
  learning_rate <- learning_rate / (1 + epoch * learning_rate_decay)
  x_shuffled <- x_train
  y_shuffled <- y_train
  n_batches <- ceiling(length(x_train) / batch_size)
  batches <- split(1:length(x_train), rep(1:n_batches, each = batch_size,
    length.out = length(x_train)))

  for (batch in batches) {
    iter <- iter + 1
    x_batch <- x_shuffled[batch]
    y_batch <- y_shuffled[batch]

    # Forward propagation
    hidden_output <- pmax(x_batch %*% weights_hidden + bias_hidden, 0) #
    ReLU activation
    y_pred_batch <- hidden_output %*% weights_output + bias_output

    # Loss calculation
    sse_batch <- sum((y_pred_batch - y_batch)^2)
    mse_batch <- sse_batch / length(batch)
    SSE_seq <- c(SSE_seq, sse_batch)
    MSE_seq <- c(MSE_seq, mse_batch)

    # Backward propagation
    d_y_sse_batch <- - 2 * (y_batch - y_pred_batch) / length(batch)
    d_weights_output <- t(hidden_output) %*% d_y_sse_batch
    d_bias_output <- colSums(d_y_sse_batch)
    d_hidden <- d_y_sse_batch %*% t(weights_output) * (hidden_output > 0)
    d_weights_hidden <- t(x_batch) %*% d_hidden
    d_bias_hidden <- colSums(d_hidden)
```

```r

    # Update weights and biases
    weights_hidden <- weights_hidden - learning_rate * d_weights_hidden
    bias_hidden <- bias_hidden - learning_rate * d_bias_hidden
    weights_output <- weights_output - learning_rate * d_weights_output
    bias_output <- bias_output - learning_rate * d_bias_output

    # Forward propagation for test data
    hidden_output_test <- pmax(x_test %*% weights_hidden + bias_hidden, 0)
    y_pred_test <- hidden_output_test %*% weights_output + bias_output
    mse_test <- sum((y_pred_test - y_test)^2) / length(y_test)
    MSE_test_seq <- c(MSE_test_seq, mse_test)
  }

  # Print MSE for current epoch
  cat("Epoch:", epoch, "| MSE:", MSE_seq[length(MSE_seq)], "| Learning
    Rate:", learning_rate, "\n")
}

# Plot MSE for training and test data
# Plot MSE for training and test data with y-axis scaled from 0 to 2
pdf("MSE_train_test.pdf")
plot(MSE_seq, type = "l", xlab = "Iteration", ylab = "MSE ", col =
    "black", ylim = c(0, 0.2))
lines(MSE_test_seq, type = "l", col = "red")
legend("topright", legend = c("Training MSE", "Test MSE"), col =
    c("black", "red"), lty = 1)
grid()
dev.off()

pdf("True_v_Predict.pdf")
# Plot true vs. predicted values
plot(x_test, y_test, col = "blue", main = "True vs. Predicted Values",
    xlab = "x_test", ylab = "y")
points(x_test, y_pred_test, col = "red")
legend("topright", legend = c("True Data", "Predicted Values"), col =
    c("blue", "red"), lty = 1)
grid()
dev.off()
# Calculate residuals
residuals <- abs(y_test - y_pred_test)

# # Plot residuals vs. predicted values
# plot(y_pred_test, residuals, col = "blue", main = "Residuals vs.
    Predicted Values", xlab = "Predicted Values", ylab = "Residuals")
# abline(h = 0, col = "red")

# Histogram of residuals
pdf("Residual_histo.pdf")
hist(residuals, col = "lightblue", main = "Histogram of Residuals", xlab
    = "Residuals", ylab = "Frequency")
grid()
dev.off()
# Scatterplot of residuals
# plot(y_pred_test, residuals, col = "blue", main = "Scatterplot of
    Residuals", xlab = "Predicted Values", ylab = "Residuals")
# abline(h = 0, col = "red")
```

Listing 12: Exercise 5b