Optimization and Numerical Methods Solutions

Thom Volker

28-10-2022

Table of contents

Preface

This project has two purposes. First, it is an attempt to organize my solutions to the course Optimization and Numerical Methods in a structured way. Second, it provides a justification to try and learn Quarto.

1 Introduction

No exercises.

2 Motivating Problems

Chapter 2 on motivating problems is the first chapter that actually entails exercises.

2.1 Exercises (2.7 in the notes)

1. Consider the multinomial likelihood in Equation 2.1 for a model (for a two-way contingency table) assuming independence. Can you simplify the likelihood?

$$\sum_{j=1}^{R}\sum_{k=1}^{C} n_{jk}\ln(\pi_{jk}) \qquad\qquad \sum_{j=1}^{R}\sum_{k=1}^{C} \pi_{jk} = 1 \tag{2.1}$$

Solution

$$\begin{aligned}
\ell(\pi) &= \sum_{j=1}^{R}\sum_{k=1}^{C} n_{jk}\ln(\pi_{jk}) \\
&= \sum_{j=1}^{R}\sum_{k=1}^{C} n_{jk}\ln(\pi_{j+}\cdot\pi_{+k}) \\
&= \sum_{j=1}^{R}\sum_{k=1}^{C} n_{jk}\ln\pi_{j+} + n_{jk}\ln\pi_{+k} \\
&= \sum_{j=1}^{R} n_{j+}\ln\pi_{j+} + \sum_{k=1}^{C} n_{+k}\ln\pi_{+k}
\end{aligned} \tag{2.2}$$

2. In a mixed model, optimization is carried out using the marginal likelihood (the likelihood with the random effects integrated out). Define the marginal likelihood for the one-way random effects ANOVA model.

One-way random effects ANOVA with group-specific effects $\mu_j \sim \mathcal{N}(0, \sigma_\mu^2)$, and

$$y_{ij} = \beta + \mu_j + \epsilon_{ij},$$

with $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$, with a groups indexed j, and n_j individuals in every group.

Solution

So, the likelihood consists of two components. For the individuals within each group, we have

$$\prod_{i=1}^{n_j} \frac{1}{\sqrt{2\pi\sigma_\epsilon^2}} \exp\left(- \frac{(y_{ij} - \beta - \mu_j)^2}{2\sigma_\epsilon^2} \right),$$

whereas for the groups themselves, we have

$$\prod_{j=1}^{a} \frac{1}{\sqrt{2\pi\sigma_\mu^2}} \exp\left(- \frac{\mu_j^2}{2\sigma_\mu^2} \right).$$

Combining these components, and integrating out the random effects, we obtain the marginal likelihood

$$\prod_{j=1}^{a} \int \prod_{i=1}^{n_j} \frac{1}{\sqrt{2\pi\sigma_\epsilon^2}} \exp\left(- \frac{(y_{ij} - \beta - \mu_j)^2}{2\sigma_\epsilon^2} \right) \frac{1}{\sqrt{2\pi\sigma_\mu^2}} \exp\left(- \frac{\mu_j^2}{2\sigma_\mu^2} \right) d\mu_j.$$

3. Suppose you do a simple linear regression analysis using a t_ν-distribution for the residuals (density: $f_\nu(y) = C\sqrt{\lambda}\left(1+\frac{\lambda(y-\mu)^2}{\nu}\right)^{-\frac{\nu+1}{2}}$ where μ is the mean (for $\nu > 1$), λ is a scale parameter and C is a normalizing constraint that does not depend on μ or λ). Define the (log-)likelihood for n observations (y_i, x_i), such that $\mu_i = \beta_0 + \beta_1 x_i$.

Solution

$$L(\beta) = \prod_{i=1}^{n} C\sqrt{\lambda}\left(1 + \frac{\lambda(y_i - \beta_0 - \beta_1 x_i)^2}{\nu}\right)^{-\frac{\nu+1}{2}},$$

$$\ell(\beta) = N \ln C + \frac{N}{2} \ln \lambda - \sum_{i=1}^{n} \frac{\nu+1}{2} \ln \left(1 + \frac{\lambda(y_i - \beta_0 - \beta_1 x_i)^2}{\nu}\right)$$

3 Basic tools

Chapter 3 introduces basic tools for optimization problems, such as Taylor Series Expansion, and introduces the exponential family.

3.1 Exercises (3.7 in the book)

1. Consider $f(x) = \frac{e^x}{1+e^x}$. Derive the third-order Taylor series expansion of this function at $x = 0$, and make a graph with the function and the third-order Taylor series expansion at $x = 0$.

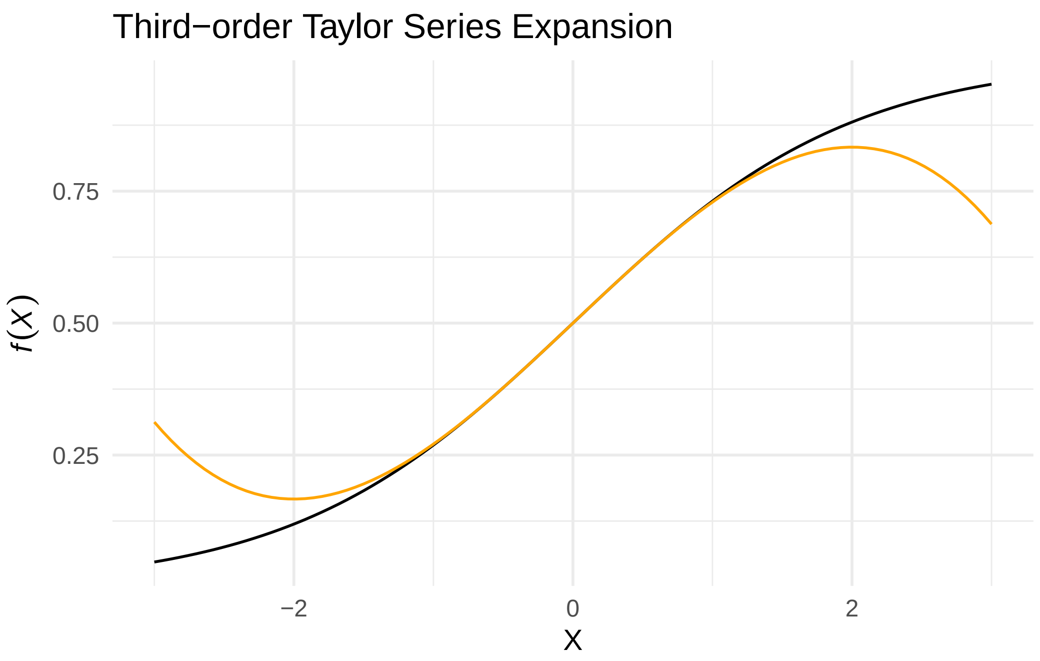
Solution

$$\begin{aligned}
f(x) &= \frac{e^x}{1+e^x} \\
f'(x) &= \frac{e^x(1+e^x)}{(1+e^x)^2} - \frac{e^{2x}}{(1+e^x)^2} = \frac{e^x}{(1+e^x)^2} \\
f''(x) &= \frac{e^x(1+e^x)^2 - e^x 2(1+e^x)e^x}{(1+e^x)^4} \\
&= \frac{e^x(1+e^x)^2 - 2e^{2x}}{(1+e^x)^3} \\
&= \frac{e^x - e^{2x}}{(1+e^x)^3} \\
f'''(x) &= \frac{(e^x - 2e^{2x})(1+e^x)^3 - (e^x - e^{2x})3(1+e^x)^2 e^x}{(1+e^x)^6} \\
&= \frac{e^x - 2e^{2x} + e^{2x} - 2e^{3x} - 3e^{2x} + 3e^{3x}}{(1+e^x)^4} \\
&= \frac{e^x - 4e^{2x} + e^{3x}}{(1+e^x)^4},
\end{aligned}$$

using Taylor's theorem, we get

$$\begin{aligned}
f(x) &\approx \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k \\
&= \frac{1}{2} + \frac{1}{4}x + 0 - \frac{1}{48}x^3.
\end{aligned}$$

```r
library(ggplot2)
fx <- function(x) exp(x) / (1 + exp(x))
fx1 <- function(x) exp(x) / (1 + exp(x))^2
fx2 <- function(x) (exp(x) - exp(2*x)) / (1 + exp(x))^3
fx3 <- function(x) (exp(x) - 4*exp(2*x) + exp(3*x)) / (1 + exp(x))^4

taylor <- function(x, root) {
  fx(root) + fx1(root) * (x - root) + fx2(root) / 2 * (x - root)^2 + fx3(root) / 6 * (x -
}

ggplot() +
  geom_function(fun = fx) +
  geom_function(fun = taylor, args = list(root = 0), col = "orange") +
  xlim(-3, 3) +
  theme_minimal() +
  labs(x = "X", y = expression(italic(f(X))),
       title = "Third-order Taylor Series Expansion")
```

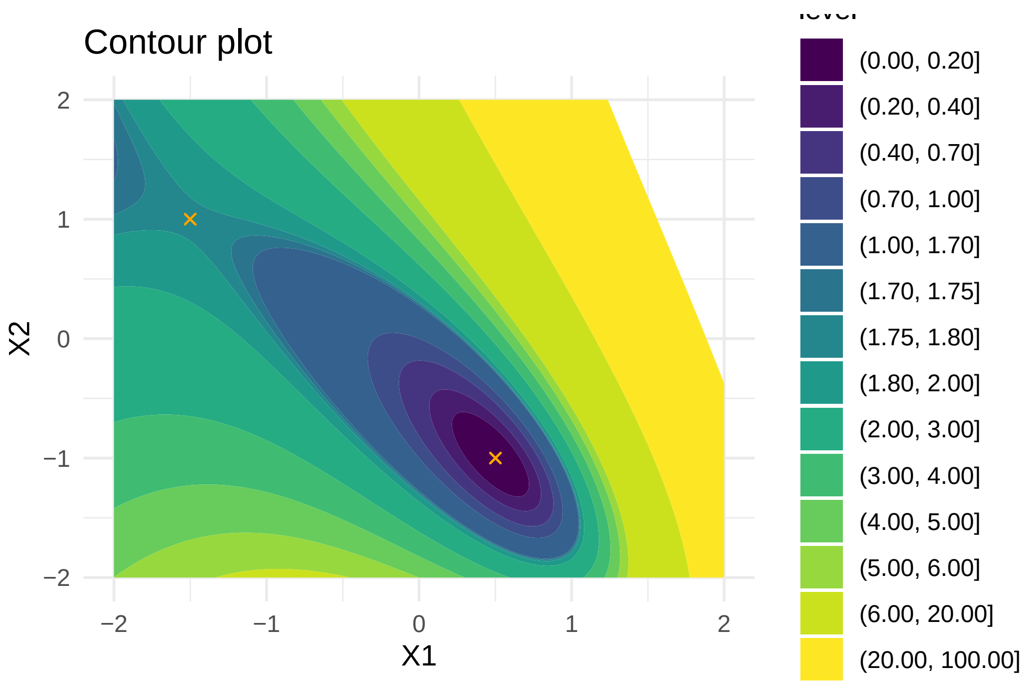


2. Consider the function: $f(x) = e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1)$. Make a contour plot of this function (let both axes run from -2 to 2) at function values $0.2, 0.4, 0.7, 1, 1.7, 1.75, 1.8, 2, 3, 4, 5, 6, 20$. Derive the second-order Taylor series at $x = (0.5, -1)'$ and $x = (-0.75, 1)'$.

Solution

Contour plot

```r
fx12 <- function(x1, x2) {
  exp(x1) * (4*x1^2 + 2*x2^2 + 4*x1*x2 + 2*x2 + 1)
}

expand.grid(x1 = -200:200/100,
            x2 = -200:200/100) |>
  dplyr::mutate(z = fx12(x1, x2)) |>
  ggplot(aes(x = x1, y = x2, z = z)) +
  stat_contour_filled(breaks = c(0, 0.2, 0.4, 0.7, 1, 1.7, 1.75, 1.8, 2, 3, 4, 5, 6, 20, 1
  geom_point(aes(x = 0.5, y = -1), col = "orange", shape = "cross") +
  geom_point(aes(x = -1.5, y = 1), col = "orange", shape = "cross") +
  theme_minimal() +
  labs(x = "X1", y = "X2",
       title = "Contour plot")
```



The second-order Taylor expansion uses the first and second partial derivatives of the function

$f(x)$.

$$\begin{aligned}
f(x) &= e^{x1}(4e_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1),\\
\frac{\partial f}{\partial x_1} &= f(x) + e^{x_1}(8x_1 + 4x_2),\\
\frac{\partial f}{\partial x_2} &= e^{x_1}(4x_2 + 4x_1 + 2),\\
\frac{\partial^2 f}{\partial x_1^2} &= f(x) + 2e^{x1}(8x_1 + 4x_2) + 8e^{x_1},\\
\frac{\partial^2 f}{\partial x_2^2} &= 4e^{x_1},\\
\frac{\partial^2 f}{\partial x_1 \partial x_2} &= 4e^{x_1} + e^{x_1}(4x_2 + 4x_1 + 2).
\end{aligned}$$

Accordingly, the Gradient $\nabla f(x)$ is defined as

$$\nabla f(x) = \begin{pmatrix} f(x) + e^{x_1}(8x_1 + 4x_2) \\ e^{x_1}(4x_2 + 4x_1 + 2) \end{pmatrix},$$

and the Hessian $\nabla^2 f(x)$ is defined as

$$\nabla^2 f(x) = \begin{pmatrix} f(x) + 2e^{x1}(8x_1 + 4x_2) + 8e^{x_1} & 4e^{x_1} + e^{x_1}(4x_2 + 4x_1 + 2) \\ 4e^{x_1} + e^{x_1}(4x_2 + 4x_1 + 2) & 4e^{x_1} \end{pmatrix}.$$

Moreover, the second-order Taylor series at $x = (0.5, -1)'$ and $x = (-0.75, 1)'$ is defined as

$$\begin{aligned}
\nabla f((0.5, -1)) &= \begin{pmatrix} 0 \\ 0 \end{pmatrix}\\
\nabla^2 f((0.5, -1)) &= \begin{pmatrix} 13.19 & 6.59 \\ 6.59 & 6.59 \end{pmatrix},
\end{aligned}$$

and

$$\begin{aligned}
\nabla f((-1.5, 1)) &= \begin{pmatrix} 0 \\ 0 \end{pmatrix}\\
\nabla^2 f((-1.5, 1)) &= \begin{pmatrix} 0 & 0.89 \\ 0.89 & 0.89 \end{pmatrix}.
\end{aligned}$$

As can be seen in the contour plot, the first point is a minimum, while the second point is a saddle point.

3. Consider the likelihood function

$$L = \prod_{i=1}^{N} \frac{e^{(\alpha+\beta x_i)y_i}}{1 + e^{(\alpha+\beta x_i)}}.$$

derive the log-likelihood function, the gradient vector for the parameter vector $\theta = (\alpha, \beta)$ and the Hessian matrix for the parameter vector θ.

Solution

The log-likelihood is defined as

$$\ell = \sum_{i=1}^{N} (\alpha + \beta x_i) y_i - \log(1 + e^{(\alpha+\beta x_i)}),$$

differentiation with respect to α yields

$$\frac{\partial \ell}{\partial \alpha} = \sum_{i=1}^{N} y_i - \frac{e^{(\alpha+\beta x_i)}}{1 + e^{(\alpha+\beta x_i)}} = \sum_{i=1}^{N} y_i - \pi_i,$$

differentiation with respect to β yields

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{N} y_i x_i - x_i \frac{e^{(\alpha+\beta x_i)}}{1 + e^{(\alpha+\beta x_i)}} = \sum_{i=1}^{N} x_i (y_i - \pi_i).$$

Accordingly, the gradient is defined as

$$\nabla \ell = \begin{pmatrix} \sum_{i=1}^{N} y_i - \pi_i \\ \sum_{i=1}^{N} x_i (y_i - \pi_i) \end{pmatrix}.$$

The second partial derivatives are defined as

$$\begin{aligned}
\frac{\partial^2 \ell}{\partial \alpha^2} &= \sum_{i=1}^{N} - \frac{e^{(\alpha+\beta x_i)}(1 + e^{(\alpha+\beta x_i)}) - e^{(\alpha+\beta x_i)} e^{(\alpha+\beta x_i)}}{(1 + e^{(\alpha+\beta x_i)})^2} \\
&= -\sum_{i=1}^{N} \frac{e^{(\alpha+\beta x_i)}}{1 + e^{(\alpha+\beta x_i)}} - \frac{(e^{(\alpha+\beta x_i)})^2}{(1 + e^{(\alpha+\beta x_i)})^2} \\
&= -\sum_{i=1}^{N} \pi_i (1 - \pi_i), \\
\frac{\partial^2 \ell}{\partial \beta^2} &= \sum_{i=1}^{N} -x_i^2 \frac{e^{(\alpha+\beta x_i)}(1 + e^{(\alpha+\beta x_i)}) - e^{(\alpha+\beta x_i)} e^{(\alpha+\beta x_i)}}{(1 + e^{(\alpha+\beta x_i)})^2} \\
&= -\sum_{i=1}^{N} x_i^2 \pi_i (1 - \pi_i), \\
\frac{\partial^2 \ell}{\partial \alpha \partial \beta} &= \sum_{i=1}^{N} -x_i \frac{e^{(\alpha+\beta x_i)}(1 + e^{(\alpha+\beta x_i)}) - e^{(\alpha+\beta x_i)} e^{(\alpha+\beta x_i)}}{(1 + e^{(\alpha+\beta x_i)})^2} \\
&= -\sum_{i=1}^{N} x_i \pi_i (1 - \pi_i).
\end{aligned}$$

Hence, the Hessian $\nabla^2\ell$ is defined as

$$\nabla^2\ell = \begin{pmatrix} -\sum_{i=1}^N \pi_i(1-\pi_i) & -\sum_{i=1}^N x_i\pi_i(1-\pi_i) \\ -\sum_{i=1}^N x_i\pi_i(1-\pi_i) & -\sum_{i=1}^N x_i^2\pi_i(1-\pi_i) \end{pmatrix}.$$

4. Take the Weibull density

$$p(y) = \varphi\rho y^{\rho-1}e^{-\varphi y^\rho}.$$

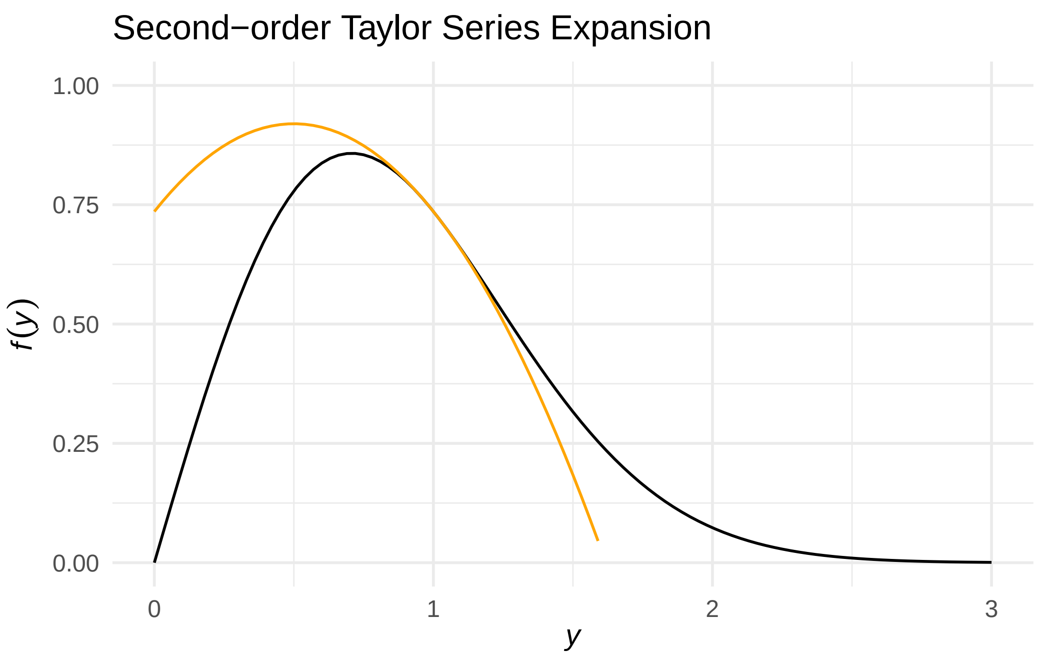
Derive the second-order Taylor series expansion of $p(y)$ about $y = 1$.

Solution

$$\begin{aligned}
\frac{\partial}{\partial y}\left[\varphi\rho y^{\rho-1}e^{-\varphi y^\rho}\right] &= \varphi\rho\Big((\rho-1)y^{\rho-2}e^{-\varphi y^\rho} - \varphi\rho y^{2\rho-2}e^{-\varphi y^\rho}\Big) \\
&= \varphi\rho e^{-\varphi y^\rho}y^{\rho-2}\Big(\rho - 1 - \varphi\rho y^\rho\Big), \\
\frac{\partial^2}{\partial y^2}\left[\varphi\rho y^{\rho-1}e^{-\varphi y^\rho}\right] &= \varphi\rho\left[\frac{\partial}{\partial y}\rho\Big(e^{-\varphi y^\rho}y^{\rho-2}\Big) - \frac{\partial}{\partial y}\Big(e^{-\varphi y^\rho}y^{\rho-2}\Big) - \frac{\partial}{\partial y}\varphi\rho\Big(e^{-\varphi y^\rho}y^{2\rho-2}\Big)\right] \\
&= \varphi\rho e^{-\varphi y^\rho}y^{\rho-3}\Big((\rho-1)(\rho-2-\varphi\rho y^\rho) - \varphi\rho y^r ho(2\rho-2-\varphi\rho y^r ho)\Big)
\end{aligned}$$

```
fx <- function(phi, rho, y) {
  e <- exp(-phi*y^rho)
  phi * rho * y^{rho-1} * e
}

fx1 <- function(phi, rho, y) {
  e <- exp(-phi*y^rho)
  phi*rho*e*y^{rho-2}*((rho-1) - phi*rho*y^rho)
}

fx2 <- function(phi, rho, y) {
  e <- exp(-phi*y^rho)
  phi*rho*e*y^{rho-3} * ((rho-1)*(rho-2-phi*rho*y^rho) - phi*rho*y^rho*(2*rho-2-phi*rho*y^
}

taylor <- function(phi, rho, y, root) {
  fx(phi, rho, root) + fx1(phi, rho, root) * (y - root) + fx2(phi, rho, root)/2 * (y - roo
}
```

```r
ggplot() +
  geom_function(fun = fx, args = list(phi = 1, rho = 2)) +
  geom_function(fun = taylor,
                args = list(phi = 1, rho = 2, root = 1),
                col = "orange") +
  lims(x = c(0, 3), y = c(0, 1)) +
  theme_minimal() +
  labs(x = expression(italic(y)), y = expression(italic(f(y))),
       title = "Second-order Taylor Series Expansion")
```



5. Consider the Weibull-based likelihood function:

$$L = \prod_{i=1}^{n} \rho y_i^{\rho-1} e^{(\alpha+\beta x_i)} e^{-(y_i^{\rho} e^{(\alpha+\beta x_i)})},$$

with y_i the outcome (time-to-event), x_i is a continuous covariate, and α and β are regression parameters. Derive the log-likelihood function for an i.i.d. sample of n observations $(y_1, y_2, ..., y_n)$, the gradient of the log-likelihood function for the parameters (ρ, α, β) and the Hessian of the log-likelihood function for the parameter vector (ρ, α, β).

Solution

The log-likelihood is defined as

$$\ell = \sum_{i=1}^{n} \log(\rho) + (\rho - 1)\log(y_i) + \alpha + \beta x_i - y_i^{\rho} e^{(\alpha+\beta x_i)}.$$

The first-order partial derivatives with respect to ρ, α, β are given by

$$\begin{aligned}
\frac{\partial \ell}{\partial \rho} &= \sum_{i=1}^{n} \rho^{-1} + \log(y_i) - y_i^{\rho} e^{(\alpha+\beta x_i)} \log(y_i), \\
\frac{\partial \ell}{\partial \alpha} &= \sum_{i=1}^{n} 1 - y_i^{\rho} e^{(\alpha+\beta x_i)}, \\
\frac{\partial \ell}{\partial \alpha} &= \sum_{i=1}^{n} x_i (1 - y_i^{\rho} e^{(\alpha+\beta x_i)}),
\end{aligned}$$

such that the gradient is defined as

$$\nabla \ell = \begin{pmatrix} \sum_{i=1}^{n} \rho^{-1} + \log(y_i) - y_i^{\rho} e^{(\alpha+\beta x_i)} \log(y_i), \\ \sum_{i=1}^{n} 1 - y_i^{\rho} e^{(\alpha+\beta x_i)}, \\ \sum_{i=1}^{n} x_i (1 - y_i^{\rho} e^{(\alpha+\beta x_i)}), \end{pmatrix}.$$

Additionally, the second-order partial derivatives are defined by

$$\begin{aligned}
\frac{\partial^2 \ell}{\partial \rho^2} &= \sum_{i=1}^{n} -\rho^{-2} - y_i^{\rho} e^{(\alpha+\beta x_i)} (\log(y_i))^2, \\
\frac{\partial^2 \ell}{\partial \alpha^2} &= \sum_{i=1}^{n} -y_i^{\rho} e^{(\alpha+\beta x_i)}, \\
\frac{\partial^2 \ell}{\partial \beta^2} &= \sum_{i=1}^{n} -x_i^2 y_i^{\rho} e^{(\alpha+\beta x_i)}, \\
\frac{\partial^2 \ell}{\partial \rho \partial \alpha} &= \sum_{i=1}^{n} -\log(y_i) y_i^{\rho} e^{(\alpha+\beta x_i)}, \\
\frac{\partial^2 \ell}{\partial \rho \partial \beta} &= \sum_{i=1}^{n} -x_i \log(y_i) y_i^{\rho} e^{(\alpha+\beta x_i)}, \\
\frac{\partial^2 \ell}{\partial \alpha \partial \beta} &= \sum_{i=1}^{n} -x_i y_i^{\rho} e^{(\alpha+\beta x_i)},
\end{aligned}$$

such that the Hessian is defined as

$$\nabla^2 \ell(\rho, \alpha, \beta) = \begin{pmatrix} \sum_{i=1}^{n} -\rho^{-2} - y_i^{\rho} e^{(\alpha+\beta x_i)} (\log(y_i))^2 & & \\ \sum_{i=1}^{n} -\log(y_i) y_i^{\rho} e^{(\alpha+\beta x_i)} & \sum_{i=1}^{n} -y_i^{\rho} e^{(\alpha+\beta x_i)} & \\ \sum_{i=1}^{n} -x_i \log(y_i) y_i^{\rho} e^{(\alpha+\beta x_i)} & \sum_{i=1}^{n} -x_i y_i^{\rho} e^{(\alpha+\beta x_i)} & \sum_{i=1}^{n} -x_i^2 y_i^{\rho} e^{(\alpha+\beta x_i)} \end{pmatrix}.$$

6. Consider a logistic regression

$$\text{logit}[P(Y_i = 1|x_i)] = \alpha + \beta x_i,$$

and a small set of data

i	x_i	y_i
1	0.5	0
2	1.0	0
3	1.5	1
4	2.0	0
5	2.5	1

Construct the log-likelihood function and the gradient function.

Solution

Constructing the logit function requires an expression for $P(Y_i = 1|x_i)$, which is defined as follows.

$$\begin{aligned}
\text{logit}[P(Y_i = 1|x_i)] &= \alpha + \beta x_i, \\
\log\left(\frac{P(Y_i = 1|x_i)}{1 - P(Y_i = 1|x_i)}\right) &= e^{(\alpha+\beta x_i)}, \\
P(Y_i = 1|x_i) &= e^{(\alpha+\beta x_i)} - e^{(\alpha+\beta x_i)}(P(Y_i = 1|x_i)), \\
1 &= \frac{e^{(\alpha+\beta x_i)}}{P(Y_i = 1|x_i)} - e^{(\alpha+\beta x_i)}, \\
1 + e^{(\alpha+\beta x_i)} &= \frac{e^{(\alpha+\beta x_i)}}{P(Y_i = 1|x_i)}, \\
P(Y_i = 1|x_i) &= \frac{e^{(\alpha+\beta x_i)}}{1 + e^{(\alpha+\beta x_i)}}.
\end{aligned}$$

Plugging this into a binomial likelihood function yields

$$
\begin{aligned}
L &= \prod_{i=1}^{5} \pi_i^{y_i}(1-\pi_i)^{(1-y_i)}, \\
\ell &= \sum_{i=1}^{5} y_i \log \pi_i + (1-y_i)\log(1-\pi_i) \\
&= \sum_{i=1}^{5} y_i \log\left(\frac{e^{(\alpha+\beta x_i)}}{1+e^{(\alpha+\beta x_i)}}\right) + \log\left(\frac{1}{1+e^{(\alpha+\beta x_i)}}\right) - y_i \log\left(\frac{1}{1+e^{(\alpha+\beta x_i)}}\right) \\
&= \sum_{i=1}^{5} y_i \log\left(\frac{e^{(\alpha+\beta x_i)}}{1+e^{(\alpha+\beta x_i)}} \Big/ \frac{1}{1+e^{(\alpha+\beta x_i)}}\right) + \log\left(\frac{1}{1+e^{(\alpha+\beta x_i)}}\right) \\
&= \sum_{i=1}^{n} y_i(\alpha+\beta x_i) - \log(1+e^{(\alpha+\beta x_i)}).
\end{aligned}
$$

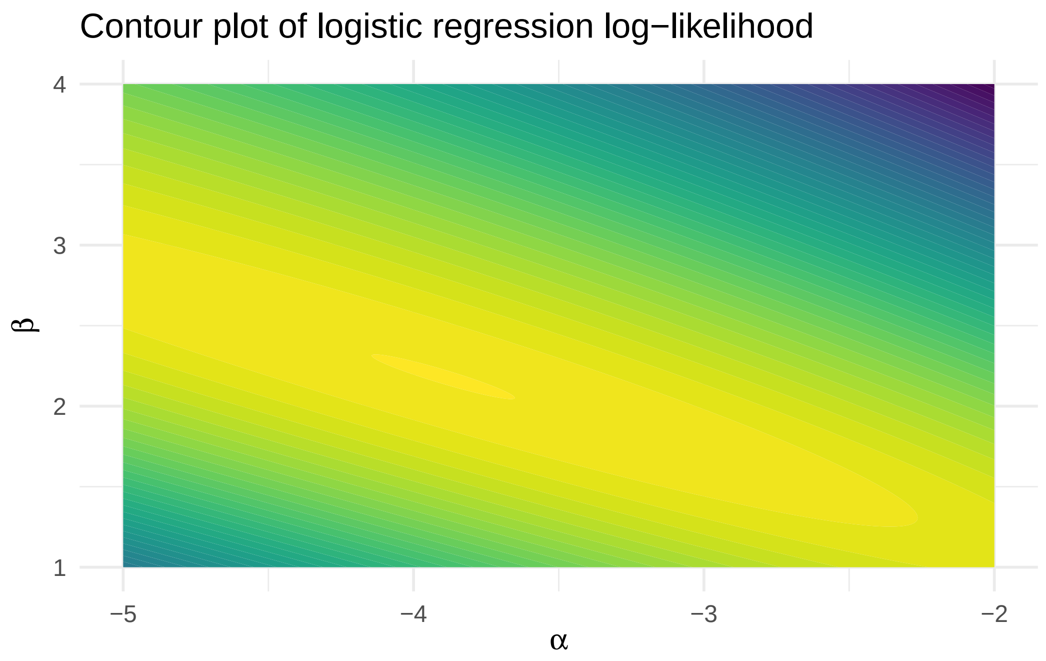
Accordingly, we can define the Gradient as

$$
\nabla \ell = \begin{pmatrix} \sum_{i=1}^{5} y_i - \frac{e^{(\alpha+\beta x_i)}}{1+e^{(\alpha+\beta x_i)}} = \sum_{i=1}^{5} y_i - \pi_i \\ \sum_{i=1}^{5} y_i x_i - x_i \frac{e^{(\alpha+\beta x_i)}}{1+e^{(\alpha+\beta x_i)}} = \sum_{i=1}^{5} x_i(y_i - \pi_i). \end{pmatrix}
$$

Filling in the values for y yields

$$
\begin{aligned}
\frac{\partial \ell}{\partial \alpha} &= 2 - \sum_{i=1}^{5} \pi_i, \\
\frac{\partial \ell}{\partial \beta} &= 4 - \sum_{i=1}^{5} x_i \pi_i.
\end{aligned}
$$

```r
ell <- function(x, y, alpha, beta) {
  sum(y*(alpha + beta*x) - log(1 + exp(alpha + beta*x)))
}

x <- c(0.5, 1, 1.5, 2, 2.5)
y <- c(0, 0, 1, 0, 1)

expand.grid(alpha = -5000:-2000/1000,
            beta = 1000:4000/1000) |>
  dplyr::mutate(l = purrr::map2_dbl(alpha, beta, ~ell(x, y, .x, .y))) |>
  ggplot(aes(x = alpha, y = beta, z = l)) +
  stat_contour_filled(bins = 50, show.legend = FALSE) +
  theme_minimal() +
  labs(x = expression(alpha),
       y = expression(beta),
        title = "Contour plot of logistic regression log-likelihood")
```



7. Consider $f(x_1, x_2, x_3) = (x_1 - 1)^4 + (x_2 - 3)^2 + 4(x_3 + 5)^4$. Find the Gradient and the Hessian and indicate what is special about the point $(1, 3, -5)$.

Solution

The gradient is defined as

$$\nabla f(x_1, x_2, x_3) = \begin{pmatrix} 4(x_1 - 1)^3 \\ 2(x_2 - 3) \\ 16(x_3 + 5)^3 \end{pmatrix}.$$

The Hessian is defined as

$$\nabla^2 f(x_1, x_2, x_3) = \begin{pmatrix} 12(x_1 - 1)^2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 48(x_3 + 5)^2 \end{pmatrix}.$$

In the point $(1, 3, -5)$, the Gradient is $\nabla f(x_1, x_2, x_3) = (0, 0, 0)'$, and the Hessian equals

$$\nabla^2 f(x_1, x_2, x_3) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

In the direction of x_1 and x_3, the function surface is almost flat.

4 From non-iterative to iterative procedures

No exercises.

5 Least squares

TO DO.

6 Iteration-based Function Optimization

Chapter 2 on motivating problems is the first chapter that actually entails exercises.

6.1 Exercises (6.5 in the notes)

1. Suppose for every individual in a small pre-clinical study, it has been recorded how many epileptic seizures are observed (outcome y) and whether the individual is receiving a standard treatment (covariate $x = 0$) or experimental medication (covariate $x = 1$). The data are:

Subject i	Treatment x	# Seizures y
1	1	12
2	1	15
3	1	17
4	0	8
5	0	11
6	0	5

A Poisson regression model is put forward for these data, with linear predictor $\theta_i = \beta_0 + \beta_1 x_i$. Starting from $\beta^{(0)} = (0, 0)'$, do the following: Derive the likelihood equations. Can they be solved analytically in this case? Perform the first five steps of the Newton-Raphson algorithm to find the maximum of the likelihood. Put your results in a table with as columns: Iteration number, current point, and log-likelihood value. Do the same for Fisher-scoring.

Solution

The Poisson model yields

$$Y \sim \text{Poisson}(\lambda), \text{ with } f(y|\theta, \phi) = \frac{e^{-\lambda}\lambda^y}{y!},$$

and thus the likelihood L and log-likelihood ℓ are defined as

$$\begin{aligned}
L &= \prod_{i=1}^{6} \frac{e^{-\lambda}\lambda^{y_i}}{y_i!} = \frac{e^{-e^{(\beta_0+\beta_1 x_i)}} e^{(\beta_0\beta_1 x_i)y_i}}{y_i!} \\
\ell &= \sum_{i=1}^{6} y_i \log\lambda - \lambda - \log(y_i!) \\
&= \sum_{i=1}^{6} y_i(\beta_0 + \beta_1 x_i) - e^{(\beta_0+\beta_1 x_i)} - \log(y_i!).
\end{aligned}$$

Accordingly, the first-order partial derivatives are defined as

$$\begin{aligned}
\frac{\partial \ell}{\partial \beta_0} &= \sum_{i=1}^{6} y_i - e^{(\beta_0-\beta_1 x_i)}, \\
\frac{\partial \ell}{\partial \beta_1} &= \sum_{i=1}^{6} x_i y_i - x_i e^{(\beta_0-\beta_1 x_i)},
\end{aligned}$$

and hence the Gradient (i.e., Score equation) can be written as

$$\nabla\ell(\beta_0,\beta_1) = S(\theta) = \begin{pmatrix} \sum_{i=1}^{6} y_i - e^{(\beta_0-\beta_1 x_i)}, \\ \sum_{i=1}^{6} x_i(y_i - e^{(\beta_0-\beta_1 x_i)}). \end{pmatrix}$$

Additionally, the second-order partial derivates are defined as

$$\begin{aligned}
\frac{\partial^2 \ell}{\partial \beta_0^2} &= \sum_{i=1}^{6} -e^{(\beta_0-\beta_1 x_i)}, \\
\frac{\partial^2 \ell}{\partial \beta_1^2} &= \sum_{i=1}^{6} -x_i^2 e^{(\beta_0-\beta_1 x_i)}, \\
\frac{\partial^2 \ell}{\partial \beta_0 \partial \beta_1} &= \sum_{i=1}^{6} -x_i e^{(\beta_0-\beta_1 x_i)},
\end{aligned}$$

such that the Hessian $\nabla^2\ell(\beta_0,\beta_1)$ can be written as

$$\nabla^2\ell(\beta_0,\beta_1) = \begin{pmatrix} \sum_{i=1}^{6} -e^{(\beta_0-\beta_1 x_i)} & \\ \sum_{i=1}^{6} -x_i e^{(\beta_0-\beta_1 x_i)} & \sum_{i=1}^{6} -x_i^2 e^{(\beta_0-\beta_1 x_i)} \end{pmatrix}.$$

Setting the first-order partial derivatives to zero and filling in the data yields

$$S(\theta) = \begin{pmatrix} 68 - 3e^{(\beta_0+\beta_1)} - 3e^{(\beta_0)} \\ 44 - 3e^{(\beta_0+\beta_1)} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Hence, we have

$$\begin{aligned}
44 - 3e^{(\beta_0+\beta_1)} &= 0 \\
3e^{(\beta_0+\beta_1)} &= 44,
\end{aligned}$$

and thus

$$\begin{aligned} 68 - 3e^{(\beta_0)} &= 44 \\ 3e^{(\beta_0)} &= 24 \\ e^{(\beta_0)} &= 8 \\ \beta_0 &= \log 8 \approx 2.0794. \end{aligned}$$

Filling this into the previous equation yields

$$\begin{aligned} 3e^{(\log 8 + \beta_1)} &= 44 \\ \log 44 - \log 3 - \log 8 &= \beta_1 \approx 0.6061. \end{aligned}$$

Newton-Raphson method

```
NR <- function(formula, data = NULL, start, n.iter) {

  X <- model.matrix(formula, data)
  Y <- model.frame(formula, data)[,1]

  loglikelihood <- function(X, Y, beta) {
    constant <- sapply(Y, function(y) sum(log(1:y))) |> sum()
    sum(y - X %*% beta - exp(X %*% beta) - constant)
  }

  score <- function(X, Y, beta) {
    t(X) %*% (Y - exp(X %*% beta))
  }

  hess <- function(X, beta) {
    - t(X) %*% diag(c(exp(X %*% beta))) %*% X
  }

  out <- matrix(0, n.iter+1, ncol(X))
  out[1, ] <- b <- start

  logL <- numeric(n.iter+1)
  logL[1] <- loglikelihood(X, Y, b)

  for (i in (1:n.iter)+1) {
    b <- b - solve(hess(X, b)) %*% score(X, Y, b)
    out[i, ] <- b
    logL[i] <- loglikelihood(X, Y, b)
  }
  data.frame(iter = 0:n.iter,
```

iter	X1	X2	logL
0	0.000000	0.0000000	-623.7158
1	7.000000	6.6666667	-2589080.4867
2	6.007295	6.6593886	-952918.2151
3	5.026981	6.6397490	-351004.0668
4	4.079450	6.5874061	-129568.7184
5	3.214784	6.4524137	-48104.1538
6	2.536096	6.1320304	-18133.0024
7	2.169495	5.5011536	-7106.9141
8	2.083377	4.5941106	-3051.0568
9	2.079449	3.6165031	-1558.0226
10	2.079442	2.6657840	-1007.2910
11	2.079442	1.7932829	-803.7918
12	2.079442	1.0983733	-729.4703
13	2.079442	0.7096305	-705.1191
14	2.079442	0.6113113	-700.2547
15	2.079442	0.6061492	-700.0115
16	2.079442	0.6061358	-700.0108
17	2.079442	0.6061358	-700.0108
18	2.079442	0.6061358	-700.0108
19	2.079442	0.6061358	-700.0108
20	2.079442	0.6061358	-700.0108

```
             out,
             logL = logL)
}

x <- c(1, 1, 1, 0, 0, 0)
y <- c(12, 15, 17, 8, 11, 5)

NR(y ~ x, start = c(0,0), n.iter = 20) |>
  knitr::kable() |>
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

Fisher scoring

Note that in this case, the expected Hessian equals

$$-x_i' \frac{\partial \mu_i}{\partial \theta_i} \nu_i^{-1} \frac{\partial \mu_i}{\partial \theta_i} x_i.$$

Given that

$$\frac{\partial \mu_i}{\partial \theta_i} = \frac{\partial \mu_i}{\partial \theta_i}\left(\exp\{\theta_i\}\right) = \exp\theta_i,$$

and

$$\nu_i^{-1} = \frac{1}{\exp\{\theta_i\}},$$

it follows that

$$\frac{\partial \mu_i}{\partial \theta_i}\nu_i^{-1}\frac{\partial \mu_i}{\partial \theta_i} = \exp\{\theta_i\} = \exp\{X\beta\}.$$

Hence, for the expected Hessian, we have

$$\mathcal{H} = E\left(\frac{\partial^2 \ell}{\partial\beta\partial\beta'}\right) = X'\text{diag}(\exp X\beta)X,$$

which is equal to the Hessian matrix $H(\beta)$, and thus Fisher scoring and Newton-Raphson are equivalent in this case.

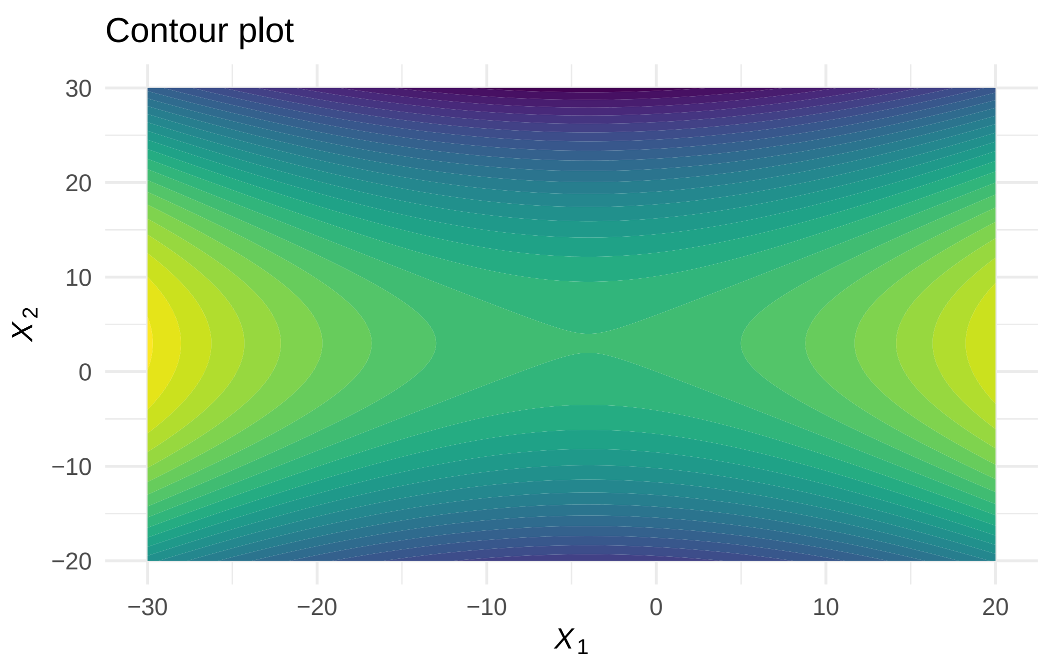
2. Assume the function

$$f(x_1, x_2) = 8x_1 + 12x_2 + x_1^2 - 2x_2^2.$$

Sketch the contour lines of $f(x_1, x_2)$, and find the stationary point of $f(x_1, x_2)$. Does this point correspond to a minimum, a maximum, or something else?

Solution

```r
library(ggplot2)
library(dplyr)
library(purrr)

fx1x2 <- function(x1, x2) 8*x1 + 12*x2 + x1^2 - 2*x2^2

expand.grid(x1 = -300:200/10,
            x2 = -200:300/10) |>
  mutate(f = map2_dbl(x1, x2, fx1x2)) |>
  ggplot(aes(x = x1, y = x2, z = f)) +
  stat_contour_filled(bins = 30, show.legend = FALSE) +
  theme_minimal() +
  labs(x = expression(italic(X[1])),
       y = expression(italic(X[2])),
       title = "Contour plot")
```



The first- and second-order partial derivatives of $f(x_1, x_2)$ are given by

$$\begin{aligned} f(x1, x2) &= 8x_1 + 12x_2 + x_1^2 - 2x_2^2, \\ \frac{\partial f}{\partial x_1} &= 8 + 2x_1, \\ \frac{\partial f}{\partial x_2} &= 12 - 4x_2, \\ \frac{\partial^2 f}{\partial x_1^2} &= 2, \\ \frac{\partial^2 f}{\partial x_2^2} &= -4, \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} &= 0. \end{aligned}$$

The stationary point of $f(x_1, x_2)$ is $f(-4, 3)$, which is a saddle point.

3. Consider the function

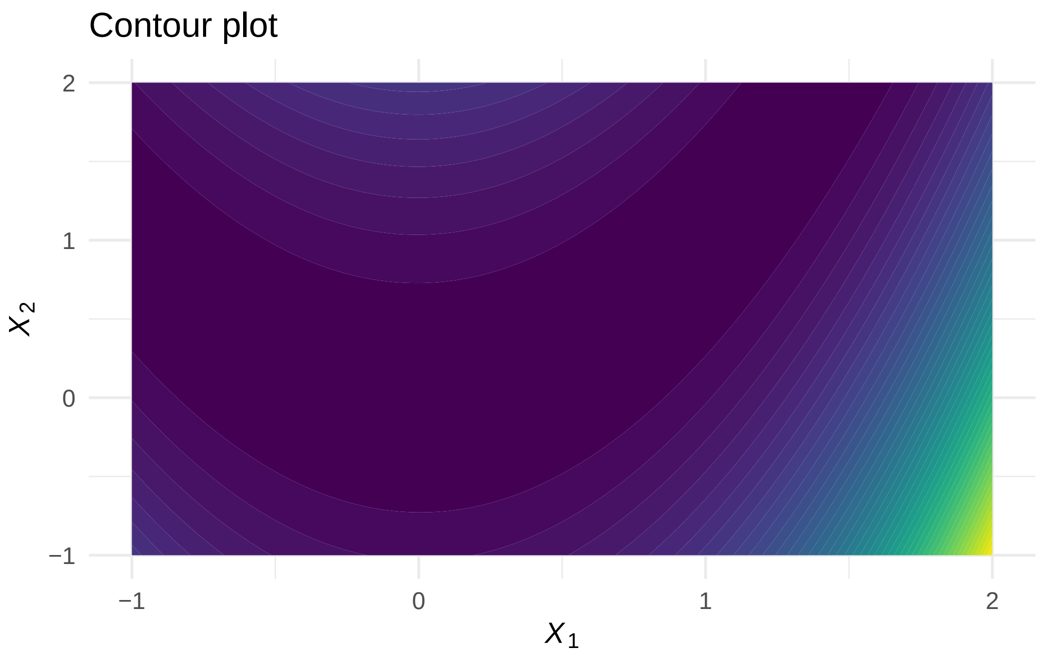
$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

Show that (1, 1)' is a local minimizer of this function. Also, starting from the point $x^{(0)} = (0, 0)'$, perform the first five steps of the steepest descent and the

Newton-Raphson algorithm to minimize the function. Put your results in a table with as columns: iteration number, current point, function value and gradient.

Solution

```r
fx1x2 <- function(x1, x2) 100*(x2 - x1^2)^2 + (1 - x1)^2

expand.grid(x1 = -100:200/100,
            x2 = -100:200/100) |>
  mutate(f = map2_dbl(x1, x2, fx1x2)) |>
  ggplot(aes(x = x1, y = x2, z = f)) +
  stat_contour_filled(bins = 50, show.legend = FALSE) +
  theme_minimal() +
  labs(x = expression(italic(X[1])),
       y = expression(italic(X[2])),
       title = "Contour plot")
```



Showing that the point $(1, 1)'$ is a local minimizer can be done by plugging the $(1, 1)'$ into the

Gradient, and checking whether the Gradient equals zero,

$$\begin{aligned}
f(x_1, x_2) &= 100(x_2 - x_1^2)^2 + (1 - x_1)^2,\\
\nabla f(x_1, x_2) &= \begin{pmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{pmatrix},\\
\nabla f(1, 1) &= \begin{pmatrix} -400(1 - 1) - 2(1 - 1) \\ 200(1 - 1) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},
\end{aligned}$$

which shows that $(1, 1)'$ is a local minimizer. Moreover, the Hessian matrix is defined by

$$\nabla^2 f(x_1, x_2) = \begin{pmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{pmatrix}$$

Steepest-Descent

```
f <- function(x1, x2) 100*(x2 - x1^2)^2 + (1 - x1)^2

score <- function(x1, x2) {
  c(400*x1^3 - 400*x1*x2 + 2*x1 - 2,
    200*x2 - 200*x1^2)
}

hess <- function(x1, x2) {
  matrix(c(1200*x1^2 - 400*x2 + 2, -400*x1, -400*x1, 200),
         nrow = 2, ncol = 2)
}

SD <- function(start, n.iter, alpha, rho, tol = 1e-16) {
  b <- start
  grad <- matrix(0, n.iter + 1, 2)
  grad[1, ] <- score(b[1], b[2])

  out <- matrix(0, n.iter + 1, 2)
  out[1, ] <- b

  i <- 1; conv <- FALSE

  while (!conv) {
    i <- i+1
    fold <- f(out[i-1, 1], out[i-1, 2])
    gradvec <- score(out[i-1, 1], out[i-1, 2])
    out[i, ] <- out[i-1, ] - alpha * gradvec / sum(gradvec^2)
    grad[i, ] <- gradvec
    fnew <- f(out[i,1], out[i,2])
    a <- alpha
    while(fnew > fold) {
      a <- a*rho
      out[i, ] <- out[i-1, ] - a * gradvec / sum(gradvec^2)
      grad[i, ] <- c(score(out[i,1], out[i,2]))
      fnew <- f(out[i,1], out[i,2])
    }
    if (
      i - 1 == n.iter |
      abs(fnew - fold) < tol
    ) {
      conv <- TRUE
    }

  }
  data.frame(iter = 0:(nrow(out)-1),
             out  = out,
             grad = grad,
             fval = f(out[,1], out[,2])) |>
    subset(iter < i)
}

SDout <- SD(c(0,0), 20000, 1, 0.8, 1e-10)

SDout |>
  head(15) |>
  knitr::kable() |>
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

Newton-Raphson

```
NR <- function(start, n.iter, alpha, rho) {

  b <- start
  grad <- matrix(0, n.iter + 1, 2)
  grad[1, ] <- score(b[1], b[2])

  out <- matrix(0, n.iter + 1, 2)
  out[1, ] <- b
```

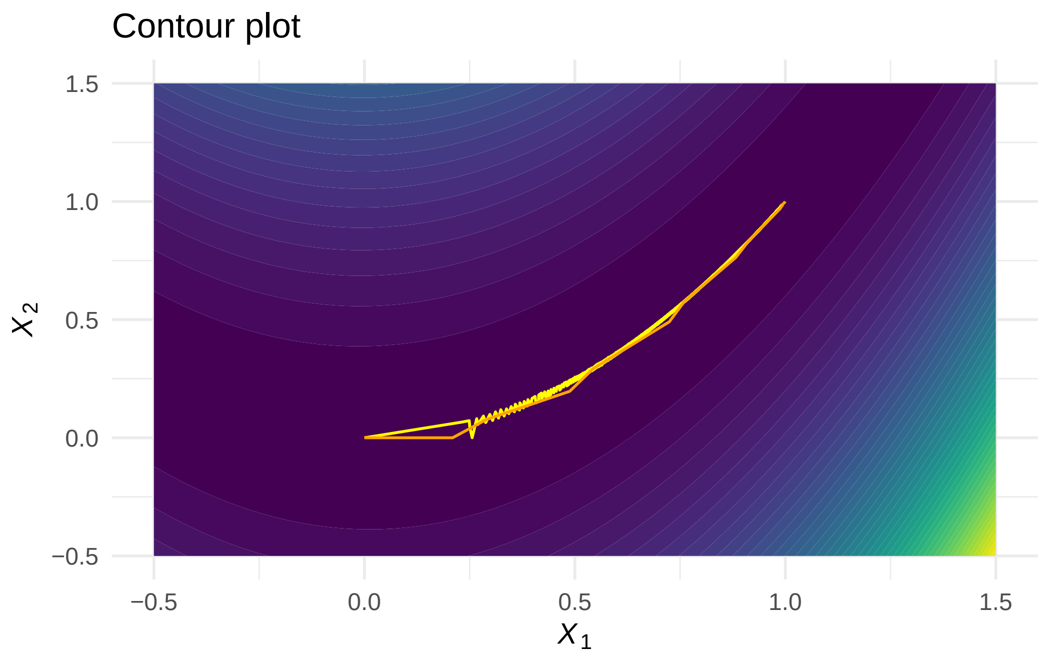
iter	out.1	out.2	grad.1	grad.2	fval
0	0.0000000	0.0000000	-2.0000000	0.000000	1.0000000
1	0.2560000	0.0000000	5.2228864	-13.107200	0.9830327
2	0.2297645	0.0658398	5.2228864	-13.107200	0.6102878
3	0.2503131	0.0462667	0.1416746	-3.277992	0.5888936
4	0.2491826	0.0724227	-2.5313281	2.066142	0.5743991
5	0.2695487	0.0557993	0.3566254	-3.371428	0.5619755
6	0.2668834	0.0809960	-2.5091373	1.953857	0.5470039
7	0.2881952	0.0644006	0.7270034	-3.731174	0.5414702
8	0.2827931	0.0921256	-2.8092036	2.430734	0.5291569
9	0.3046506	0.0732128	0.9976613	-3.919835	0.5219235
10	0.2981029	0.0989389	-2.6049709	2.014701	0.5028071
11	0.3187362	0.0829810	1.0103687	-3.722352	0.4987602
12	0.3114437	0.1098474	-2.9779538	2.570033	0.4906224
13	0.3321087	0.0920131	1.0930160	-3.656631	0.4795061
14	0.3240513	0.1189688	-3.1613443	2.791914	0.4763936

```
  for (i in 1:n.iter + 1) {

    fold <- f(out[i-1,1], out[i-1,2])
    b <- out[i - 1, ]

    out[i, ] <- b - solve(hess(b[1], b[2])) %*% score(b[1], b[2])
    grad[i, ] <- c(score(b[1], b[2]))

    fnew <- f(out[i,1], out[i,2])
    a <- alpha
    while (fnew>fold){
      a <- a*rho
      out[i,] <- out[i-1,] - a * solve(hess(b[1], b[2])) %*% score(b[1], b[2])
      grad[i, ] <- c(score(out[i,1], out[i,2]))
      fnew <- f(out[i,1], out[i,2])
    }
  }
  data.frame(iter = 0:(nrow(out)-1),
             out  = out,
             grad = grad,
             fval = f(out[,1], out[,2]))
}
NRout <- NR(c(0,0), 15, 1, 0.8)
```

iter	out.1	out.2	grad.1	grad.2	fval
0	0.0000000	0.0000000	-2.0000000	0.0000000	1.0000000
1	0.2097152	0.0000000	2.1087792	-8.7960930	0.8179782
2	0.2903887	0.0778174	2.1087792	-8.7960930	0.5077839
3	0.4877049	0.1965794	7.0277391	-8.2553296	0.4328224
4	0.5430563	0.2918463	7.0277391	-8.2553296	0.2097363
5	0.7243882	0.4907541	9.2958913	-6.7968490	0.1914547
6	0.7597374	0.5759513	9.2958913	-6.7968490	0.0578823
7	0.8827605	0.7636815	5.2684843	-3.1169062	0.0380329
8	0.9112381	0.8295438	5.2684843	-3.1169062	0.0079444
9	0.9876125	0.9695454	0.1180717	-0.1621945	0.0035559
10	0.9933299	0.9866717	2.2795435	-1.1666107	0.0000446
11	0.9999567	0.9998694	-0.0003516	-0.0065379	0.0000002
12	0.9999996	0.9999992	0.0174780	-0.0087827	0.0000000
13	1.0000000	1.0000000	0.0000000	-0.0000004	0.0000000
14	1.0000000	1.0000000	0.0000000	0.0000000	0.0000000
15	1.0000000	1.0000000	0.0000000	0.0000000	0.0000000

```r
NRout |>
  knitr::kable() |>
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

```r
expand.grid(x1 = -50:150/100,
            x2 = -50:150/100) |>
  mutate(f = map2_dbl(x1, x2, fx1x2)) |>
  ggplot(aes(x = x1, y = x2, z = f)) +
  stat_contour_filled(bins = 50, show.legend = FALSE) +
  geom_line(data = SDout,
            mapping = aes(x = out.1, y = out.2, z = NULL),
            col = "yellow") +
  geom_line(data = NRout,
            mapping = aes(x = out.1, y = out.2, z = NULL),
            col = "orange") +
  theme_minimal() +
  labs(x = expression(italic(X[1])),
       y = expression(italic(X[2])),
       title = "Contour plot")
```



4. Suppose for an individual during consecutive nights, it is recorded how loudly he snores (covariate x) and whether he wakes up or not (the outcome Y). Consider the following hypothetical data are collected: $x = (0, 1, 2, 3, 4, 5)'$ and $y = (0, 1, 0, 1, 1, 1)'$. A logistic regression model is put forward for these data such that logit$(\Pr(y_i = 1|x_i))$ = logit$(\pi(x_i)) = \beta_0 + \beta_1 x_i$. Starting from $\beta^{(0)} = (0, 0)'$, perform the first five steps of the Newton-Raphson algorithm to find the maximum of the likelihood. Put your results in a table with as columns: iteration number, current point and loglikelihood value. Do the same for iterative reweighted least squares.

Solution

For logistic regression, the likelihood is defined as

$$L = \prod_{i=1}^{N} \frac{e^{y_i(\beta_0+\beta_1 x_i)}}{1 + e^{(\beta_0+\beta_1 x_i)}},$$

$$\ell = \sum_{i=1}^{N} y_i(\beta_0 + \beta_1 x_i) - \log(1 + e^{(\beta_0+\beta_1 x_i)}).$$

Additionally, the Gradient is defined by

$$\nabla\ell(\beta_0, \beta_1) = \begin{pmatrix} \sum_{i=1}^{N} y_i - \frac{e^{(\beta_0+\beta_1 x_i)}}{1+e^{(\beta_0+\beta_1 x_i)}} \\ \sum_{i=1}^{N} x_i(y_i - \frac{e^{(\beta_0+\beta_1 x_i)}}{1+e^{(\beta_0+\beta_1 x_i)}}) \end{pmatrix},$$

while the Hessian is defined as

$$\nabla^2 \ell(\beta_0, \beta_1) = \begin{pmatrix} -\sum_{i=1}^N \frac{e^{(\beta_0+\beta_1 x_i)}}{(1+e^{(\beta_0+\beta_1 x_i)})^2} & -\sum_{i=1}^N x_i \frac{e^{(\beta_0+\beta_1 x_i)}}{(1+e^{(\beta_0+\beta_1 x_i)})^2} \\ -\sum_{i=1}^N x_i \frac{e^{(\beta_0+\beta_1 x_i)}}{(1+e^{(\beta_0+\beta_1 x_i)})^2} & -\sum_{i=1}^N x_i^2 \frac{e^{(\beta_0+\beta_1 x_i)}}{(1+e^{(\beta_0+\beta_1 x_i)})^2} \end{pmatrix}.$$

```r
loglikelihood <- function(X, Y, beta) {
  sum(Y * (X%*%beta) - log(1 + exp(X %*% beta)))
}
score <- function(X, Y, beta) {
  t(X) %*% (Y - 1/(1 + exp(-X%*%beta)))
}
hess <- function(X, Y, beta) {
  - t(X) %*% diag(c(exp(X%*%beta)/(1 + exp(X%*%beta))^2)) %*% X
}
```

Newton-Raphson implementation

```r
NRlogistic <- function(formula, data = NULL, start, n.iter) {
  X <- model.matrix(formula, data)
  Y <- model.frame(formula, data)[, 1]

  out <- matrix(0, n.iter+1, ncol(X))
  out[1, ] <- b <- start

  logL <- numeric(n.iter+1)
  logL <- loglikelihood(X, Y, b)

  for (i in 1:n.iter + 1) {
    b <- b - solve(hess(X, Y, b)) %*% score(X, Y, b)
    out[i, ] <- b
    logL[i] <- loglikelihood(X, Y, b)
  }

  data.frame(iter = 0:n.iter,
             b0 = out[,1],
             b1 = out[,2],
             logL = logL)
}

x <- c(0,1,2,3,4,5)
y <- c(0,1,0,1,1,1)
```

iter	b0	b1	logL
0	0.000000	0.0000000	-4.158883
1	-1.047619	0.6857143	-2.626827
2	-1.444172	0.9933894	-2.457094
3	-1.602433	1.1249532	-2.440395
4	-1.624928	1.1443026	-2.440125
5	-1.625338	1.1446616	-2.440125

```r
NRlogistic(y ~ x, start = c(0,0), n.iter = 5) |>
  knitr::kable() |>
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

```r
glm(y ~ x, family = binomial) |> summary()
```

```

Call:
glm(formula = y ~ x, family = binomial)

Deviance Residuals:
      1        2        3        4        5        6
-0.5995   1.3872  -1.4692   0.5509   0.3189   0.1815

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.6253     1.9284  -0.843    0.399
x             1.1447     0.9278   1.234    0.217

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 7.6382  on 5  degrees of freedom
Residual deviance: 4.8802  on 4  degrees of freedom
AIC: 8.8802

Number of Fisher Scoring iterations: 5
```

Convergence is reached after five iterations!

Iterative re-weighted least squares implementation

iter	b0	b1	logL
0	0.000000	0.0000000	-4.158883
1	-1.047619	0.6857143	-2.626827
2	-1.444172	0.9933894	-2.457094
3	-1.602433	1.1249532	-2.440395
4	-1.624928	1.1443026	-2.440125
5	-1.625338	1.1446616	-2.440125

```r
IRLS <- function(formula, data = NULL, start, n.iter) {

  X <- model.matrix(formula, data)
  Y <- model.frame(formula, data)[,1]
  out <- matrix(0, n.iter+1, ncol(X))
  out[1, ] <- b <- start

  logL <- numeric(n.iter+1)
  logL[1] <- loglikelihood(X, Y, b)

  for (i in 1:n.iter+1) {
    e <- exp(X%*%b) / (1 + exp(X%*%b))
    W <- diag(c(e / (1+exp(X %*% b))))
    Z <- X %*% b + (y - e) * (1 / (e*(1-e)))
    b <- solve(t(X) %*% W %*% X) %*% t(X) %*% W %*% Z
    out[i, ] <- b
    logL[i] <- loglikelihood(X, Y, b)
  }
  data.frame(iter = 0:n.iter,
             b0 = out[,1],
             b1 = out[,2],
             logL = logL)
}
IRLS(y ~ x, start = c(0,0), n.iter = 5) |>
  knitr::kable() |>
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

And again, convergence is reached after five iterations!

5. Consider the function

$$f(x) = \frac{e^x}{(1+e^x)^2}.$$

Using an iterative procedure of your liking, find the optimum of the function, and check whether it is a minimum or a maximum.

Solution

First, calculate we calculate the derivatives.

$$\begin{aligned}
f(x) &= \frac{e^x}{(1+e^x)^2}, \\
f'(x) &= \frac{e^x - e^{2x}}{(1+e^x)^3}, \\
f''(x) &= \frac{e^x - 4e^{2x} + e^{3x}}{(1+e^x)^4}.
\end{aligned}$$

We can first find the optimum analytically. Let's first take the log of the function, which makes it easier to work with:

$$\log f(x) = x - 2\log(1+e^x).$$

Subsequently, we take the derivative of $\log f(x)$ and set it equal to zero to find the optimum.

$$\begin{aligned}
\frac{\partial f}{\partial x} = 1 - \frac{2e^x}{1+e^x} &= 0, \\
\implies \frac{2e^x}{1+e^x} &= 1, \\
1 + e^x &= 2e^x, \\
e^x &= 1, \\
x &= 0.
\end{aligned}$$

So we know the solution must be $x = 0$. Doing the same steps using the Newton-Raphson algorithm yields

```r
fx <- function(x) -exp(x) / (1 + exp(x))^2
f1x <- function(x) -(exp(x) - exp(2*x)) / (1 + exp(x))^3
f2x <- function(x) -(exp(x) - 4*exp(2*x) + exp(3*x)) / (1 + exp(x))^4

NR <- function(start = 0.5, n.iter = 20, alpha = 1, rho = 0.8) {
  out <- matrix(0, n.iter+1, 4)
  out[1, ] <- c(start, fx(start), f1x(start), f2x(start))

  colnames(out) <- c("x", "fx", "f1x", "f2x")

  for (i in 1:n.iter + 1) {
    a <- alpha
    new <- out[i-1, 1] - a * out[i-1, 3] / out[i-1, 4]
    out[i, ] <- c(new, fx(new), f1x(new), f2x(new))
    while(out[i, 2] > out[i-1, 2]) {
      a <- a*rho
```

x	fx	f1x	f2x
0.5000000	-0.2350037	0.0575568	0.0963568
-0.0973301	-0.2494089	-0.0121279	0.1238198
0.0006180	-0.2500000	0.0000773	0.1250000
0.0000000	-0.2500000	0.0000000	0.1250000
0.0000000	-0.2500000	0.0000000	0.1250000
0.0000000	-0.2500000	0.0000000	0.1250000

iter	b0	b1	logL
0	0.000000	0.000000	-3.465736
1	-2.800000	1.600000	-2.479523
2	-3.698907	2.079500	-2.423599
3	-3.886773	2.177155	-2.421969
4	-3.893957	2.180846	-2.421967
5	-3.893967	2.180851	-2.421967

```r
        new <- out[i-1, 1] - a * out[i-1, 3] / out[i-1, 4]
        out[i, ] <- c(new, fx(new), f1x(new), f2x(new))
      }
    }
    out
  }

  NR(0.5, 5) |>
    knitr::kable() |>
    kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

6. Continuation of exercise 6 from chapter 3: implement maximum likelihood estimation for this logistic regression.

Solution

```r
x <- c(0.5, 1, 1.5, 2, 2.5)
y <- c(0,0,1,0,1)

NRlogistic(y ~ x, start = c(0,0), n.iter = 5) |>
  knitr::kable() |>
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

iter	b0	b1	logL
0	0.000000	0.000000	-3.465736
1	-2.800000	1.600000	-2.479523
2	-3.698907	2.079500	-2.423599
3	-3.886773	2.177155	-2.421969
4	-3.893957	2.180846	-2.421967
5	-3.893967	2.180851	-2.421967

```r
IRLS(y ~ x, start = c(0,0), n.iter = 5) |>
  knitr::kable() |>
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

```r
glm(y ~ x, family = binomial) |> summary()
```

```

Call:
glm(formula = y ~ x, family = binomial)

Deviance Residuals:
      1        2        3        4        5  
-0.3430  -0.5758   1.4506  -1.3814   0.6181  

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -3.894      3.465  -1.124    0.261
x              2.181      1.950   1.119    0.263

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6.7301  on 4  degrees of freedom
Residual deviance: 4.8439  on 3  degrees of freedom
AIC: 8.8439

Number of Fisher Scoring iterations: 4
```

7 The MM algorithm with applications to regularized regression

Chapter 7 introduces the MM (Majorize-Minimize in minimization problems, and Minorize-Maximize for maximization problems). The essence of the MM algorithm is to use a surrogate function $g(x|a)$ to minimize a complicating function $f(x)$. The majorizing function $g(x|a)$ has the properties that it touches $f(x)$ at the supporting point: $f(a) = g(a|a)$, and that it lies above $f(x)$, such that $g(x|a) \geq f(x)$.

Example: A majorizing algorithm for the median

```
MM_median <- function(x, start, maxit, tol) {
  theta <- numeric(length = maxit)
  theta[1] <- start
  t <- 1
  conv <- FALSE

  while (!conv) {
    t <- t+1
    theta[t] <- 1/(sum(1/abs(x-theta[t-1]))) * sum(x / abs(x - theta[t-1]))

    if (t == maxit | abs(theta[t] - theta[t-1]) < tol) {
      conv <- TRUE
    }
  }
  theta[1:t]
}

x <- c(1,0,9,5,1)
MM_median(x, mean(x), 50, 1e-10)
```

```
 [1] 3.200000 2.687064 2.221601 1.836210 1.541400 1.331585 1.192418 1.106385
 [9] 1.056601 1.029324 1.014946 1.007548 1.003794 1.001902 1.000952 1.000476
[17] 1.000238 1.000119 1.000060 1.000030 1.000015 1.000007 1.000004 1.000002
[25] 1.000001 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000
[33] 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000
```

```r
median(x)
```

```
[1] 1
```

We obtained the same solution.

Example: Multiple linear lasso regression

```r
set.seed(1)

N <- 10000
P <- 100
B <- c(1:10/5, rep(0, 90))

S <- matrix(0.5, P, P)
diag(S) <- 1
X <- matrix(rnorm(N*P), N, P) %*% chol(S)
Y <- X %*% B + rnorm(N, 0, 10)

X <- scale(X)
Y <- scale(Y)

cv_lasso <- glmnet::cv.glmnet(X, Y, alpha = 1, standardize = F,
                              intercept = FALSE)
lasso <- glmnet::glmnet(X, Y, alpha = 1, standardize = F,
                        intercept = F,
                        lambda = cv_lasso$lambda.min)


MM_lasso <- function(X, Y, start, lambda, threshold, maxit, tol) {

  b <- matrix(0, maxit, ncol(X))
  b[1, ] <- start

  t <- 1
  conv <- FALSE

  while(!conv) {
    t <- t+1
    D <- diag(1/(abs(b[t-1,]) + threshold))
    b[t, ] <- solve(t(X) %*% X + lambda/2 * D) %*% t(X) %*% Y
    if (t == maxit | sum(abs(b[t] - b[t-1])) < tol) conv <- TRUE
```

glmnet	MM_lasso
0.0023482	0.00202
0.0136856	0.01325
0.0226324	0.02220
0.0403210	0.03987
0.0572259	0.05680
0.0715446	0.07115
0.1105939	0.11025
0.1172435	0.11695
0.1465717	0.14635
0.1235894	0.12347
0.0076866	0.00769
0.0000000	0.00000
0.0000000	0.00000
0.0000000	0.00000
0.0000000	0.00000

```r
  }
  round(b[1:t,],5)
}

own_lasso <- MM_lasso(X = X,
                      Y = Y,
                      start = runif(rep(1, P)),
                      lambda = 2*N*cv_lasso$lambda.min,
                      threshold = 1e-7,
                      maxit = 500,
                      tol = 1e-14)

dplyr::bind_cols(glmnet = lasso$beta[1:15,],
                 MM_lasso = own_lasso[nrow(own_lasso), 1:15, drop=F] |> c()) |>
  knitr::kable() |>
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

The results are not the same, but they do come quite close.

```r
n <- 100; p <- 1000
s <- matrix(0.95, p, p)
diag(s) <- 1
```

```
x <- matrix(rnorm(n*p), n, p) %*% chol(s)
b <- runif(p)
y <- x %*% b + rnorm(n, 0, 500)

out <- MM_lasso(x, y, runif(p), 10000, .000001, 500, 1e-15)
out[nrow(out), ][abs(out[nrow(out), ]) > 0.00001]
```

```
 [1]  10.84720   4.01224 106.27432   0.00004  71.19764  40.71499   0.00019
 [8]  72.95655   8.50397   0.00003   0.00003  68.22038   0.00026   7.40789
[15]  76.84439   0.00002   0.00004   0.00003   0.00002   0.00002
```

8 Exercises

1. Write a script to generate data with the following structure: $p = 9$ predictors with the following covariance structure

$$R = \begin{pmatrix} 1 & 0.8 & 0.7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.8 & 1 & 0.6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.7 & 0.6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.2 & 0.4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 1 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0.3 & 1 \end{pmatrix}$$

___and $y_i = x_i'\beta + e_i$ with $\beta = [0, 1, 2, 0, 0, 3, 1.5, 1.5, 0]'$ and $e_i \sim \mathcal{N}(0, 10)$.

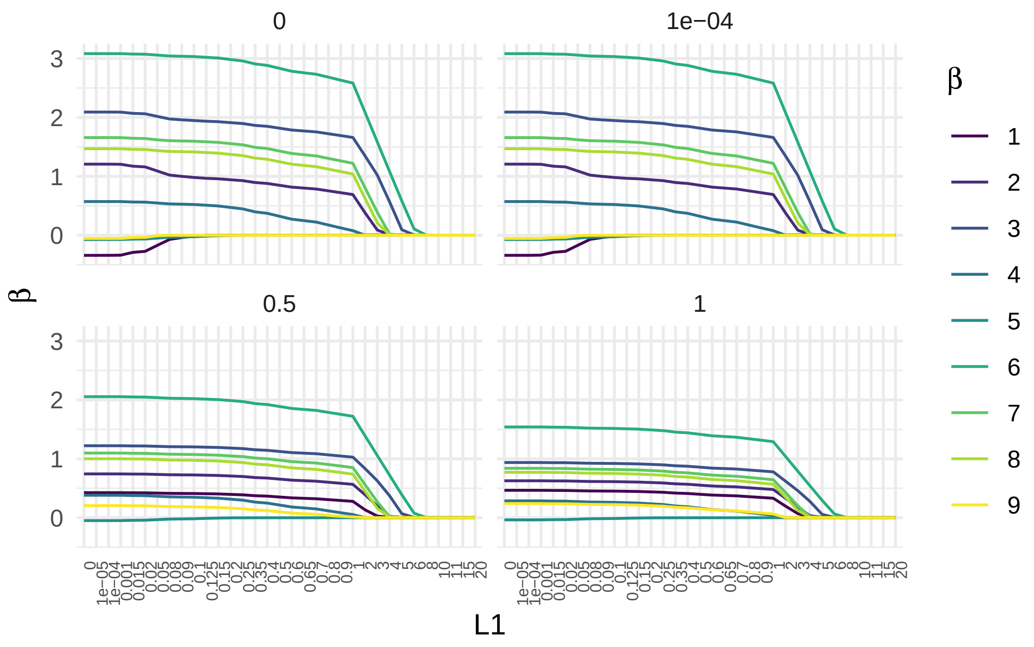
Solution

```r
set.seed(123)
N <- 1000
P <- 9
R <- diag(P)
R[1:3, 1:3] <- c(1, 0.8, 0.7, 0.8, 1, 0.6, 0.7, 0.6, 1)
R[7:9, 7:9] <- c(1, 0.2, 0.4, 0.2, 1, 0.4, 0.4, 0.3, 1)
# X <- matrix(rnorm(N*P), N, P) %*% chol(R)
B <- c(0, 1, 2, 0, 0, 3, 1.5, 1.5, 0)
# Y <- X %*% B + rnorm(N, 0, 10)

Xrand <- matrix(rnorm(N*P), N, P)
Xc <- Xrand - rep(1, N) %*% t(colMeans(Xrand))
S <- svd(Xc)
newX <- sqrt(N-1) * S$u %*% chol(R)
Y <- newX %*% B + 10*rnorm(N)
```

2. Implement the MM algorithm for the elastic net.

```r
library(ggplot2)

MM_elastic_net <- function(X, Y, start, alpha, eps, l1, l2, maxit, tol) {
  b <- matrix(0, maxit, ncol(X))
  b[1, ] <- start
  t <- 1
  conv <- FALSE

  purrr::map2(l1, l2, function(L1, L2) {
    while(!conv) {
      t <- t + 1
      D <- solve(diag(c(abs(b[t-1, ]) + eps)))
      b[t, ] <- solve(t(X) %*% X + alpha * L1/2 * D + (1-alpha) * L2 * diag(ncol(X))) %*%
      b[t, which(abs(b[t, ]) < 1e-5)] <- 0

      if (sum(abs(b[t, ] - b[t-1])) < tol | maxit == t) {
        conv <- T
      }
    }
    b[t, ]
  })
}

L <- expand.grid(L1 = c(20, 15, 11, 10, 8, 6, 5, 4, 3, 2, 1, 0.9, 0.8, 0.7, 0.65,
                        0.6, 0.5, 0.4, 0.35, 0.25, 0.2, 0.15, 0.125, 0.10, 0.09,
                        0.08, 0.05, 0.02, 0.015, 0.001, 0.0001, 0.00001, 0) * 2*N,
                 L2 = c(0, 0.0001, 0.5, 1) * 2*N)
out <- MM_elastic_net(newX, Y, runif(P), alpha = 0.5, eps = 0.001,
                      l1 = L$L1, l2 = L$L2, maxit = 1000, tol = 1e-14)

results <- tibble::tibble(
  L1 = L$L1,
  L2 = L$L2,
  B = out
) |>
  tidyr::unnest(B) |>
  dplyr::mutate(Beta = rep(paste0(1:P), length(out)))

results |>
  dplyr::mutate(L1 = factor(L1 / N / 2),
                L2 = factor(L2 / N / 2)) |>
```

```r
ggplot(aes(x = L1, y = B, col = Beta, group = Beta)) +
geom_line() +
facet_wrap(~L2) +
theme_minimal() +
scale_color_viridis_d(name = expression(beta)) +
theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 6)) +
labs(y = expression(beta))
```



9 Constrained optimization

To be added.

10 Maximum Likelihood Estimation and Inference

1. Continuation of exercise 6 from Chapter 3 and exercise 6 from Chapter 5. Take the data and the logistic regression model. The point estimates are

$$\begin{pmatrix} \hat{\alpha} \\ \hat{\beta} \end{pmatrix} = \begin{pmatrix} -3.89 \\ 2.18 \end{pmatrix}.$$

Obtain the asymptotic variance-covariance matrix and standard errors for $\hat{\alpha}$ and $\hat{\beta}$, and implement a test for $H_0 : \alpha = 1$ and $\beta = 0$.

Solution

Recall that the first- and second-order derivatives are defined as

$$\begin{aligned}
\nabla\ell = S(\alpha, \beta) &= \begin{pmatrix} \sum_{i=1}^n y_i - \frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}} \\ \sum_{i=1}^n x_i(y_i - \frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}) \end{pmatrix} \\
&= X'\left(Y - \frac{e^{X\beta}}{1+e^{X\beta}}\right) \\
\nabla^2\ell = H(\alpha, \beta) &= \begin{pmatrix} -\sum_{i=1}^n \frac{e^{\alpha+\beta x_i}}{(1+e^{\alpha+\beta x_i})^2} & -\sum_{i=1}^n x_i(\frac{e^{\alpha+\beta x_i}}{(1+e^{\alpha+\beta x_i})^2}) \\ -\sum_{i=1}^n x_i(\frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}) & -\sum_{i=1}^n x_i^2(\frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}) \end{pmatrix} \\
&= X'\mathrm{diag}\left(\frac{e^{X\beta}}{1+e^{X\beta}}\right)X.
\end{aligned}$$

The standard errors are defined as $\sqrt{\mathrm{diag}[-H(\alpha, \beta)]'}$. So, we have

```r
loglikelihood <- function(Y, X, beta) {
  sum(Y*(X %*% beta) - log(1 + exp(X %*% beta)))
}
score <- function(Y, X, beta) {
  t(X) %*% (Y - 1 / (1 + exp(-X %*% beta)))
}
hessian <- function(X, beta) {
  - t(X) %*% diag(c(exp(X %*% beta) / (1 + exp(X %*% beta))^2)) %*% X
}

NR <- function(formula, data = NULL, n.iter) {
  X <- model.matrix(formula, data)
  Y <- model.frame(formula, data)[,1]

  beta <- matrix(0, n.iter + 1, ncol(X))

  L <- numeric(n.iter)
  L[1] <- loglikelihood(Y, X, beta[1, ])
  t <- 1; conv <- FALSE

  while(!conv) {
    t <- t + 1
    beta[t, ] <- beta[t-1, ] - solve(hessian(X, beta[t-1, ])) %*% score(Y, X, beta[t-1, ])
    L[t] <- loglikelihood(Y, X, beta[t, ])

    if (abs(L[t] - L[t-1]) < 1e-10 | t == n.iter) conv <- TRUE
  }

  list(b = beta[1:t, ],
       se = sqrt(diag(solve(-hessian(X, beta[t, ])))),
       loglik = L[1:t])
}

x <- c(0.5,1,1.5,2,2.5)
y <- c(0,0,1,0,1)

out <- NR(y ~ x, n.iter = 50)
out
```

```
$b
          [,1]     [,2]
[1,]  0.000000 0.000000
[2,] -2.800000 1.600000
[3,] -3.698907 2.079500
[4,] -3.886773 2.177155
[5,] -3.893957 2.180846
[6,] -3.893967 2.180851

$se
(Intercept)           x
```

```
   3.465687    1.949705

$loglik
[1] -3.465736 -2.479523 -2.423599 -2.421969 -2.421967 -2.421967
```

```
b <- out$b[nrow(out$b),]
summary(glm(y ~ x, family = binomial))
```

```

Call:
glm(formula = y ~ x, family = binomial)

Deviance Residuals:
      1        2        3        4        5  
-0.3430  -0.5758   1.4506  -1.3814   0.6181  

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -3.894      3.465  -1.124    0.261
x              2.181      1.950   1.119    0.263

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6.7301  on 4  degrees of freedom
Residual deviance: 4.8439  on 3  degrees of freedom
AIC: 8.8439

Number of Fisher Scoring iterations: 4
```

```
LT <- b - c(1, 0)
LHL <- t(diag(2)) %*% (-solve(hessian(cbind(1,x), b))) %*% diag(2)

pchisq(t(LT) %*% solve(LHL) %*% LT, 2, lower.tail = FALSE)
```

```
          [,1]
[1,] 0.2949003
```

2. Assume that you maximize $\ell(\alpha; p, n) = p\alpha - n\ln(1 + e^{\alpha})$ to estimate α. In a next step, you want to find the optimal $\pi = \frac{e^{\alpha}}{1+e^{\alpha}}$. What is the MLE of π, and what is the standard error of π (use the delta method).

Solution

$$\begin{aligned}\ell(\alpha; p, n) &= p\alpha - n\ln(1+e^{\alpha})\\ \frac{\partial \ell}{\partial \alpha} &= p - n\frac{e^{\alpha}}{1+e^{\alpha}}\\ \implies 0 &= p - n\pi\\ \implies \pi &= \frac{p}{n}\end{aligned}$$

Using the delta method to obtain the standard error yields

$$\begin{aligned}\mathrm{Var}(F(\hat{\theta})) &= \frac{\partial F(\hat{\theta})}{\partial \theta'}(-\hat{H})^{-1}\frac{\partial F(\hat{\theta})}{\partial \theta}\\ &= \frac{e^{a}}{(1+e^{a})^2}\frac{(1+e^{a})^2}{ne^{a}}\frac{e^{a}}{(1+e^{a})^2}\\ &= \pi(1-\pi)\frac{1}{n\pi(1-\pi)}\pi(1-\pi)\\ &= \frac{\pi(1-\pi)}{n},\\ \mathrm{SE}(F(\hat{\theta})) &= \sqrt{\mathrm{Var}(F(\hat{\theta}))}\\ &= \sqrt{\frac{\pi(1-\pi)}{n}}.\end{aligned}$$

3. Suppose the maximum likelihood estimate of a probability π is $\hat{\pi} = 0.75$ (based on $n = 10$ observations). The (approximate) standard error of this estimate is $SE(\hat{\pi}) \approx \sqrt{\frac{\hat{\pi}(1-\hat{\pi})}{10}}$. What is the MLE of $\log \pi$ and what is the corresponding standard error.

Solution

The MLE of $\log \pi$ equals

$$\begin{aligned}\mathrm{MLE}(\log \pi) &= \log \mathrm{MLE}(\pi)\\ &= \log 0.75\\ &\approx -0.2877.\end{aligned}$$

Additionally, the standard error equals

$$
\begin{aligned}
\mathrm{SE}(F(\hat{\theta})) &= \sqrt{\mathrm{Var}(F(\hat{\theta}))} \\
&\approx \sqrt{\frac{\partial F(\hat{\theta})}{\partial \theta'}(-\hat{H})^{-1}\frac{\partial F(\hat{\theta})}{\partial \theta}} \\
&= \sqrt{\left(1-\frac{e^a}{1+e^a}\right)\frac{ne^a}{(1+e^a)^2}\left(1-\frac{e^a}{1+e^a}\right)} \\
&= \sqrt{(1-\hat{\pi})\frac{1}{n\hat{\pi}(1-\hat{\pi})}(1-\hat{\pi})} \\
&= \sqrt{\frac{1-\hat{\pi}}{n\hat{\pi}}} \\
&= \sqrt{\frac{0.25}{7.5}} \approx 0.1826.
\end{aligned}
$$

11 Numerical integration

1. Assume the following integral

$$I = \int_{-\infty}^{\infty} \frac{e^x}{(1+e^x)^2} \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} dx.$$

The true value of the integral can be assumed to be $I = 0.20662096414....$ Below, we ask you to use several methods to approximate I and compare their accuracy. It is advisable to put the results in a single table as to facilitate comparison between methods. Comment on these methods and on the results you find.

a. Use Laplace's method to approximate the integral.

Solution

Laplace's method yields that the integral $f(z)\phi(z)$ is approximated by a normal density. Note that $\int f(z)\phi(z)dz$ can be written as $\int e^{h(z)}dz$, where $\phi(z)$ is the density of the standard normal distribution and $h(z) = \log(f(z)\phi(z))$. Suppose that $h(z)$ has a single maximum $\hat{z}$, such that $h'(\hat{z}) = 0$ and $h''(\hat{z}) < 0$. Then, one can use the following second-order Taylor approximation

$$\begin{aligned} h(z) &\approx h(\hat{z}) + (z-\hat{z})h'(\hat{z}) + \frac{1}{2}(z-\hat{z})^2 h''(\hat{z}) \\ &= h(\hat{z}) + \frac{1}{2}(z-\hat{z})^2 h''(\hat{z}). \end{aligned}$$

Accordingly, we can define

$$\begin{aligned} \int f(z)\phi(z)dz &= e^{h(z)}dz \\ &\approx e^{h(\hat{z})+\frac{1}{2}(z-\hat{z})^2 h''(\hat{z})}dz \\ &= f(\hat{z})\phi(\hat{z})\sqrt{2\pi}(-h''(\hat{z}))^{-\frac{1}{2}}. \end{aligned}$$

In our case, we have

$$I = \int_{-\infty}^{\infty} \frac{e^x}{(1+e^x)^2} \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} dx,$$

and thus $h(x)$ is defined as

$$\begin{aligned} h(x) &= \log(f(x)\phi(x)) \\ &= x - 2\log(1+e^x) - \log(\sqrt{2\pi}) - \frac{1}{2}x^2, \end{aligned}$$

where $\phi(x)$ denotes the standard normal distribution. Accordingly, we have first- and second-order derivative

$$\begin{aligned} h'(x) &= 1 - \frac{2e^x}{1+e^x} - x, \\ h''(x) &= -2\frac{e^x}{(1+e^x)^2} - 1, \end{aligned}$$

Setting the first derivative equal to zero and solving for x yields

$$\begin{aligned} h'(x) &= 1 - 2\frac{e^x}{1+e^x} - x \\ \implies 1 &= 2\frac{e^x}{1+e^x} - x \\ x &= \frac{1-e^x}{1+e^x} \\ x &= 0. \end{aligned}$$

Subsequently, we can fill in the blanks and evaluate $f(\hat{z})\phi(\hat{z})\sqrt{2\pi}(-h''(\hat{z}))^{-\frac{1}{2}}$:

$$\begin{aligned} f(\hat{z})\phi(\hat{z})\sqrt{2\pi}(-h''(\hat{z}))^{-\frac{1}{2}} &= f(0)\phi(0)\sqrt{2\pi}(-h''(0))^{-\frac{1}{2}} \\ &= \frac{e^0}{(1+e^0)^2}\frac{1}{\sqrt{2\pi}}e^0 \\ &= 0.2041241 \end{aligned}$$

In R, this gives is done as follows:

```
fx <- function(x) (exp(x))/(1+exp(x))^2 * dnorm(x, 0, 1)
fffx <- function(x) -(2*exp(x)/(1+exp(x))^2) - 1
laplace <- fx(0) * sqrt(2*pi) * (-fffx(0))^{-1/2}
```

b. Use a Riemannian approximation (with 2, 3 and 4 nodes).

Solution

The Riemannian approximation yields that $g(z)$ is approximated by dividing the parameter space into several regions, and summing the areas of each region. This yields that $g(z)$ is approximated in each interval by a constant function (usually $g(z_q)$), such that $\int g(z)dz \approx h\sum_{q=1}^{Q} g(z_q)$ with weights $w_q = h$ for all q.

```
riemann <- function(n, a, b) {
  h <- (b-a)/n
  z <- seq(a, b, h)
  zq <- fx(z)
  h*sum(zq)
}

(riemann2 <- riemann(2, -4, 4))
```

```
[1] 0.3989612
```

```
(riemann3 <- riemann(3, -4, 4))
```

```
[1] 0.1444211
```

```
(riemann4 <- riemann(4, -4, 4))
```

```
[1] 0.2221554
```

c. Use a trapezoidal approximation (with 2, 3, and 4 nodes).

Solution

The trapezoidal approximation is very closely related to Riemannian approximation, but with a slightly different different sum.

```
trapezoidal <- function(n, a, b) {
  h <- (b - a)/n
  z <- seq(a, b, h)
  zq <- fx(z)
  h/2 * fx(a) + h * sum(zq[-1]) + h/2 * fx(b)
}

(trap2 <- trapezoidal(2, -4, 4))
```

```
[1] 0.3989612
```

```
(trap3 <- trapezoidal(3, -4, 4))
```

```
[1] 0.1444211
```

```
(trap4 <- trapezoidal(4, -4, 4))
```

```
[1] 0.2221554
```

d. Use Gauss-Hermite quadrature (with 2, 3, and 4 nodes), with the following nodes and weights.

2 nodes

- Nodes: -1.0; 1.0
- Weights: 0.5; 0.5

3 nodes

- Nodes: -1.732051; 0; 1.732051
- Weights: 0.1666667; 0.666667; 0.1666667

4 nodes

- Nodes: -2.3344142; -0.741938; 0.7419638; 2.3344142
- Weights: 0.04587585; 0.45412415; 0.4512415; 0.04587585

Solution

```
n2 <- c(-1, 1)
w2 <- c(0.5, 0.5)
n3 <- c(-1.732051, 0, 1.732051)
w3 <- c(0.1666667, 0.666667, 0.1666667)
n4 <- c(-2.3344142, -0.741938, 0.7419638, 2.3344142)
w4 <- c(0.04587585, 0.45412415, 0.4512415, 0.04587585)

gauss <- function(n, w) {
  if (length(n) != length(w)) stop("n and w must have the same length!")
  sum(w * exp(n)/(1+exp(n))^2)
}

(gauss2 <- gauss(n2, w2))
```

```
[1] 0.1966119
```

```
(gauss3 <- gauss(n3, w3))
```

Riemannian	Trapezoidal	Gaussian.Quadrature
0.3989612	0.3989612	0.1966119
0.1444211	0.1444211	0.2092427
0.2221554	0.2221554	0.2052290

```
[1] 0.2092427
```

```
(gauss4 <- gauss(n4, w4))
```

```
[1] 0.205229
```

Putting everything together, we have:

It is clear that Gaussian quadrature is the most accurate method.

2. The Laplace distribution has as pdf $f(x) = \frac{1}{2}e^{-|x|}$. Consider the integral $I = \int \frac{1}{2}e^{-|x|}dx$.

a. Approximate the integral I with a 4-point Riemannian approximation between -2 and 2.

```
laplace <- function(x) 1/2 * exp(-abs(x))
```

Let's first rely on R's best guess for this interval, by using `integrate()`.

```
integrate(laplace, -Inf, Inf)
```

```
1 with absolute error < 5.7e-05
```

It is a proper density function, and should thus integrate to 1. Now, let's see how approximating it with a Riemannian approximation works.

```
riemann <- function(n, a, b) {
  h <- (b - a)/n
  z <- seq(a, b, h)
  zq <- 1/2 * exp(-abs(z))
  h*sum(zq)
}

riemann(10000000, -1000, 1000)
```

```
[1] 1
```

Breaking the integration region up in small enough functions, we can get quite close to the true answer. Yet, with only 4 points, we're not doing a great job.

```
integrate(laplace, -2, 2)
```

```
0.8646647 with absolute error < 9.6e-15
```

```
riemann(4, -2, 2)
```

```
[1] 1.003215
```

That is, we do an okayish job when the aim is to approximate the entire density, but we cut quite some part of the tails, and increasing the number of nodes shows that we indeed miss part of the distribution.

```
riemann(4, -2, 2)
```

```
[1] 1.003215
```

b. Approximate the integral I using a 3-point Gauss-Hermite approximation. (Hint: You need to rewrite the integral).

The Gauss-Hermite approximation works by factoring the density of interest into a some density $f(x)$ and a standard normal density $\phi(x)$, which can be done by multiplying and dividing the Laplace density by a normal density. Hence, we have

$$\begin{aligned}\int f(x)\phi(x)dx &= \int \frac{1}{2}e^{-|x|} \\ &= \int \left(\frac{e^{-|x|}}{2} \bigg/ \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \right) \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \\ &= \frac{\sqrt{2\pi}}{2} \int e^{\frac{x^2}{2}-|x|}\phi(x),\end{aligned}$$

where $\phi(x)$ again denotes the standard normal distribution. Subsequently, we can use the R-package `npmlreg` to obtain the nodes and weights for the Gauss-Hermite approximation.

```r
fx <- function(x) sqrt(2*pi)/2 * exp(x^2/2 - abs(x))
n <- npmlreg::gqz(4)$location
w <- npmlreg::gqz(4)$weight

gauss <- function(n, w) {
  sum(w * fx(n))
}

gauss(n, w)
```

```
[1] 0.883698
```

Then, showing how well the methods work respectively, we can create a plot for an increasing number of nodes.

```r
library(ggplot2)
library(dplyr)
```

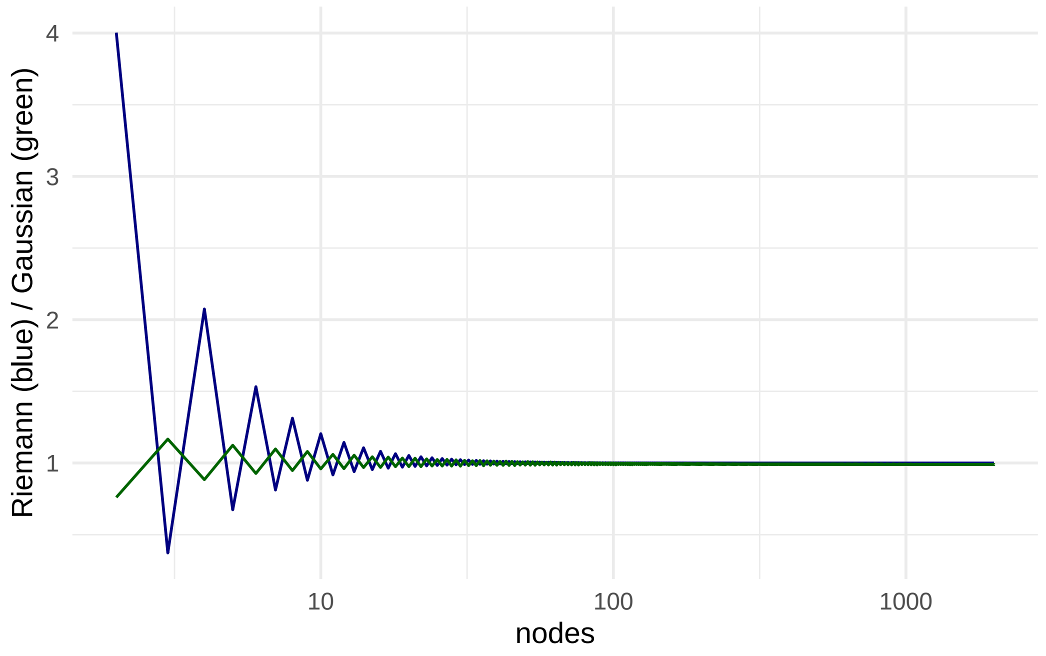
```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(npmlreg)
```

```
Warning: package 'npmlreg' was built under R version 4.2.2
```

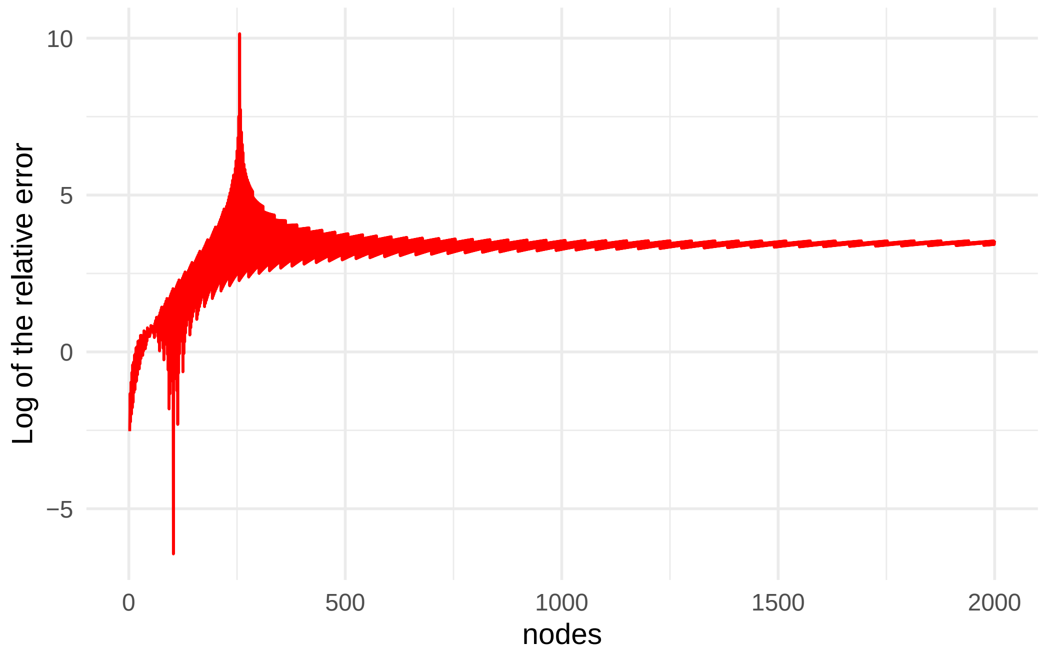
```r
perf <- data.frame(nodes = 2:2000) |>
  mutate(Riemann = sapply(nodes, riemann, a = -8, b = 8),
         Gauss = sapply(nodes, function(x) {
           n <- npmlreg::gqz(x)$location
```

```
            w <- npmlreg::gqz(x)$weight
            gauss(n, w)
           }))

perf |>
  ggplot(aes(x = nodes)) +
  geom_line(aes(y = Riemann), col = "navy") +
  geom_line(aes(y = Gauss), col = "dark green") +
  theme_minimal() +
  labs(y = "Riemann (blue) / Gaussian (green)") +
  scale_x_log10()
```



```
perf |>
  ggplot(aes(x = nodes, y = log(abs(1-Gauss)/abs(1-Riemann)))) +
  geom_line(col = "red") +
  theme_minimal() +
  labs(y = "Log of the relative error")
```

	nodes	Riemann	Gauss
1994	1995	0.9996632	0.9897202
1995	1996	0.9996712	0.9887065
1996	1997	0.9996632	0.9896955
1997	1998	0.9996712	0.9886816
1998	1999	0.9996632	0.9896707
1999	2000	0.9996712	0.9886568



Surprisingly, for a sufficient number of nodes, the relative approximation error of the Riemannian approximation is smaller than of the Gauss-Hermite approximation, although the error is small in an absolute sense.

```
tail(perf) |>
  knitr::kable() |>
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

12 Expectation-Maximization algorithm

Consider the classical example from Dempster, Laird and Rubin (1977). They give the complete (but unobserved) multinomial data model:

Z_{11}	Z_{12}	Z_2	Z_3	Z_4
$\frac{1}{2}$	$\frac{1}{4}\theta$	$\frac{1}{4}(1-\theta)$	$\frac{1}{4}(1-\theta)$	$\frac{1}{4}\theta$

Additionally, we have the observed, but coarsened, version of the data that is called Y, which corresponding observed counts per category:

Y_1	Y_2	Y_3	Y_4	
$\frac{1}{2}+\frac{1}{4}\theta$	$\frac{1}{4}(1-\theta)$	$\frac{1}{4}(1-\theta)$	$\frac{1}{4}\theta$	
125	18	20	34	

There are now (at least) three options to calculate the likelihood of the parameter θ for the complete data Z: (1) calculating the likelihood directly and obtaining a non-iterative solution, (2) calculating the likelihood and obtaining an iterative solution, (3) applying the EM algorithm. We will walk through each of these steps.

Direct calculation; obtaining a non-iterative solution

The log-likelihood for the complete data is defined as

$$\begin{aligned}\ell_c(\theta) =& \sum_{i=1}^{5} Z_j \log[\pi_j^c(\theta)] \\ =& Z_{11}(125;\theta)\log\left(\frac{1}{2}\right) + Z_{12}(125;\theta)\log\left(\frac{1}{4}\theta\right) \\ &+ 18\log\left(\frac{1}{4}(1-\theta)\right) + 20\log\left(\frac{1}{4}(1-\theta)\right) + 34\log\left(\frac{1}{4}\theta\right),\end{aligned}$$

and the log-likelihood for the observed data is defined as

$$\begin{aligned}\ell_c(\theta) =& \sum_{i=1}^{4} Y_j \log[\pi_j(\theta)] \\ =& 125 \log\left(\frac{1}{2} + \frac{1}{4}\theta\right) + 18 \log\left(\frac{1}{4}(1-\theta)\right) \\ &+ 20 \log\left(\frac{1}{4}(1-\theta)\right) + 34 \log\left(\frac{1}{4}\theta\right).\end{aligned}$$

Calculating the first derivative of the observed-data log-likelihood yields

$$S(\theta) = \frac{y_1}{2+\theta} - \frac{y_2}{1-\theta} - \frac{y_3}{1-\theta} + \frac{y_4}{\theta} = 0.$$

The Score function can be rewritten as a quadratic equation

$$S(\theta) = -197 \cdot \theta^2 + 15 \cdot \theta + 68 = 0,$$

which yields two solutions

$$\hat{\theta} = 0.6268 \ \vee \hat{\theta} = -0.5507.$$

Of course, the second one is not possible, so we obtain

$$\hat{\theta} = 0.626821497871.$$

Iterative solution of the observed likelihood

We first define a coarsening matrix that relates the observed data to the complete data

$$C = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

such that $\pi(\theta) = C\pi^c(\theta)$. Subsequently, we can write

$$\pi^c(\theta) = \begin{pmatrix} \frac{1}{2} \\ 0 \\ \frac{1}{4} \\ \frac{1}{4} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{4} \\ -\frac{1}{4} \\ -\frac{1}{4} \\ \frac{1}{4} \end{pmatrix} \theta = X_0 + X_1\theta.$$

As a next step, we can write the score function as

$$S(\theta) = X_1' C' (C \ \mathrm{cov}(Z) C')^- (Y - nC\pi^c),$$

and the Fisher information matrix equals

$$\mathcal{I}(\theta) = nX_1'C'(C\ \mathrm{cov}(Z)C')^{-}CX_1,$$

where $(C\ \mathrm{cov}(Z)C')^{-}$ denotes the generalized inverse of the covariance matrix of Y. The updating algorithm is defined as

$$\theta^{(t+1)} = \theta^{(t)} + S(\theta^{(t)})/\mathcal{I}(\theta^{(t)}).$$

```
X0 <- c(1/2, 0, 1/4, 1/4, 0)
X1 <- c(0, 1/4, -1/4, -1/4, 1/4)

Z <- function(th) c(Z11=1/2, Z12=th/4, Z2=(1-th)/4, Z3=(1-th)/4, Z4=th/4)
covZ <- function(th, n) n * (diag(Z(th)) - Z(th) %*% t(Z(th)))
pi_c <- function(theta, X0, X1) X0 + X1 * theta

Y <- c(125, 18, 20, 34)

C <- matrix(c(1,1,0,0,0,
              0,0,1,0,0,
              0,0,0,1,0,
              0,0,0,0,1), 4, 5, byrow=T)

conv <- F; maxit <- 50; t <- 1
theta <- numeric(maxit); theta[1] <- 0.5
n <- sum(Y)

X1tCt <- t(X1) %*% t(C)

while(!conv) {

  t <- t+1

  CZCtinv <- matlib::Ginv(C %*% covZ(theta[t-1], n) %*% t(C))
  YnCpi <- (Y - n * C %*% pi_c(theta[t-1], X0, X1))

  S <- X1tCt %*% CZCtinv %*% YnCpi
  H <- n * X1tCt %*% CZCtinv %*% t(X1tCt)
  theta[t] <- theta[t-1] + S / H

  if (t == maxit | abs(theta[t] - theta[t-1]) < 1e-20) {
    conv <- T
  }
}
theta[1:t]
```

```
[1] 0.5000000 0.6332487 0.6265341 0.6268344 0.6268209 0.6268215 0.6268215
[8] 0.6268215
```

Expectation-Maximization algorithm

```
EM <- function(start, Y, maxit, tol) {
  n <- sum(Y)
  theta <- numeric(maxit)
  theta[1] <- start
  t <- 1
  conv <- F
  Z <- function(theta, Y) {
    c(250/(2+theta), 125*theta/(2+theta), Y[2:4]/n)
  }

  while(!conv) {
    t <- t+1
    Z <- c(250/(2+theta[t-1]), 125*theta[t-1]/(2+theta[t-1]), Y[2:4])
    theta[t] <- (Z[2] + Z[5]) / sum(Z[2:5])
    if (t == maxit | abs(theta[t] - theta[t-1]) < tol) conv <- T
  }
  theta[1:t]
}

EM(0.5, Y, 50, 1e-50)
```

```
 [1] 0.5000000 0.6082474 0.6243211 0.6264889 0.6267773 0.6268156 0.6268207
 [8] 0.6268214 0.6268215 0.6268215 0.6268215 0.6268215 0.6268215 0.6268215
[15] 0.6268215 0.6268215 0.6268215 0.6268215 0.6268215 0.6268215
```

References