UNIVERSITETET I OSLO

Department of Physics

STK4051 - Computational Statistics

Mandatory Assignment Part 2/2

Author:
Jonas Semprini Næss

Date: 1st May 2024

Table of Contents

List of Figures

List of Tables

Problem 1 - (Simulation from 1D distribution)

We will start this exercise by investigating some methods for sampling a univariate distribution. We will assume that we easily obtain samples from a uniform distribution, i.e. $U \sim \text{Unif}[0, 1]$

a.)

Describe the standard transformation rule to sample from a univariate distribution, and derive the expression for sampling from an exponential distribution.

Solution:

Initially we assume to draw a sample U from a uniform distribution Unif$[0, 1]$. Then we turn to the target distribution given by

$$F(x) = \int_{-\infty}^{x} f(u)\ du$$

whereby looking at the inverse of $f(x)$ will transform the uniform sample U into a sample from our desired distribution. Namely

$$X = F^{-1}(U) = \inf\{x \mid F(x) \geq u\}$$

where $\inf\{x \mid F(x) \geq u\}$ is well-known from inverse transform sampling and that F is right-continuous, meaning $\{u \mid F(x) \geq u\}$ is well-defined.

Suppose so that $X \sim \text{Exp}(\lambda)$. We know that $f(x) = \lambda e^{-\lambda x}$, for $x \geq 0$ and accordingly $F(x) = 1 - e^{-\lambda x}$, which by application of the transformation yields

$$F(x) = U$$

$$1 - e^{-\lambda x} = U$$

$$e^{-\lambda x} = 1 - U$$

$$-\lambda x = \ln(1 - U)$$

$$x = -\frac{\ln(1 - U)}{\lambda}$$

Meaning our sample transformation gives $\boxed{x = -\frac{\ln(1 - U)}{\lambda}}$, and we are done. ■

b.)

When the negative log density is convex we can use adaptive rejection sampling to build an approximation to the density. We will now use this to sample from a standard normal distribution. We shall use the sub-gradient of the convex function, i.e. for a continuously differentiable convex function $k(x)$ we have:

$$k(x) \geq k(x_0) + k'(x_0)(x - x_0)$$

Describe how we can use the convex property of the negative log density to find an upper bound on the distribution. Use the linearization around the points -1, 0, and 1, to derive a bounding function for the standard normal. Illustrate the bounding function. Does the bounding function integrate to 1? Show that the probability distribution corresponding to the bounding function is:

$$g(x) = \begin{cases} \frac{1}{3} & \text{if } -0.5 < x \leq 0.5 \\ \frac{\exp(-|x|+0.5)}{3} & \text{else} \end{cases} \tag{1}$$

Solution:

Upper Bound:

Suppose we have found the log-density of some data x, which we denote by $\ell(x)$. Now since $-\ell(x)$ is convex we have from the result of the sub-gradient that

$$-\ell(x) \geq -\ell(x_0) - \ell'(x_0)(x - x_0)$$
$$\Updownarrow$$
$$\ell(x) \leq \ell(x_0) + \ell'(x_0)(x - x_0)$$

where reversing the log-transformation gives

$$e^{\ell(x)} \leq e^{\ell(x_0)+\ell'(x_0)(x-x_0)}. \tag{2}$$

This is nothing more than a description of the limitation to the likelihood density function, meaning

$$L(x) \leq e^{\ell(x_0)+\ell'(x_0)(x-x_0)}$$

whereby taking the minimum of the right hand side over several points would yield a sensible upper bound

$$L(x) \leq \min\left\{e^{\ell(x_i)+\ell'(x_i)(x-x_i)}\right\} \tag{3}$$

Linearization:

We start off by computing $\ell(x)$

$$\begin{aligned}\ell(x) &= \log\left(\phi(x)\right)\\ &= \log\left(\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}\right)\\ &= \log\left(\frac{1}{\sqrt{2\pi}}\right) - \frac{x^2}{2}\end{aligned}$$

and then $\ell'(x)$

$$\ell'(x) = -x$$

Case 1: $x = -1$

By linearization we have that

$$\mathcal{B}(x) = \ell(-1) + \ell'(-1)(x - (-1)) \tag{4}$$
$$= \log\left(\frac{1}{\sqrt{2\pi}}\right) - \frac{1}{2} + 1 \cdot (x - (-1)) \tag{5}$$
$$= \log\left(\frac{1}{\sqrt{2\pi}}\right) + x + \frac{1}{2} \tag{6}$$

Case 2: $x = 0$

In like manner

$$\mathcal{B}(x) = \ell(0) + \ell'(0)(x) \tag{7}$$
$$= \log\left(\frac{1}{\sqrt{2\pi}}\right) - 0 \tag{8}$$
$$= \log\left(\frac{1}{\sqrt{2\pi}}\right) \tag{9}$$

Case 3: $x = 1$

And lastly

$$\mathcal{B}(x) = \ell(1) + \ell'(1)(x-1) \tag{10}$$

$$= \log\left(\frac{1}{\sqrt{2\pi}}\right) - \frac{1}{2} - 1 \cdot (x-1) \tag{11}$$

$$= \log\left(\frac{1}{\sqrt{2\pi}}\right) - x + \frac{1}{2} \tag{12}$$

Additionally it is worth nothing that the three resulting linearization lines intercept each other in $x_1 = -\frac{1}{2}, x_2 = 0$ and $x_3 = \frac{1}{2}$ respectively. Which then gives us the following bounding function

$$\mathcal{B}(x) = \begin{cases} \log\left(\frac{1}{\sqrt{2\pi}}\right) + x + \frac{1}{2}, & x \in (-\infty, -\frac{1}{2}) \\ \log\left(\frac{1}{\sqrt{2\pi}}\right) & x \in [-\frac{1}{2}, \frac{1}{2}) \\ \log\left(\frac{1}{\sqrt{2\pi}}\right) - x + \frac{1}{2}, & x \in [\frac{1}{2}, \infty) \end{cases} \tag{13}$$

and is illustrated in the plot below, produced by the code in Appendix (B), from code listings (1)

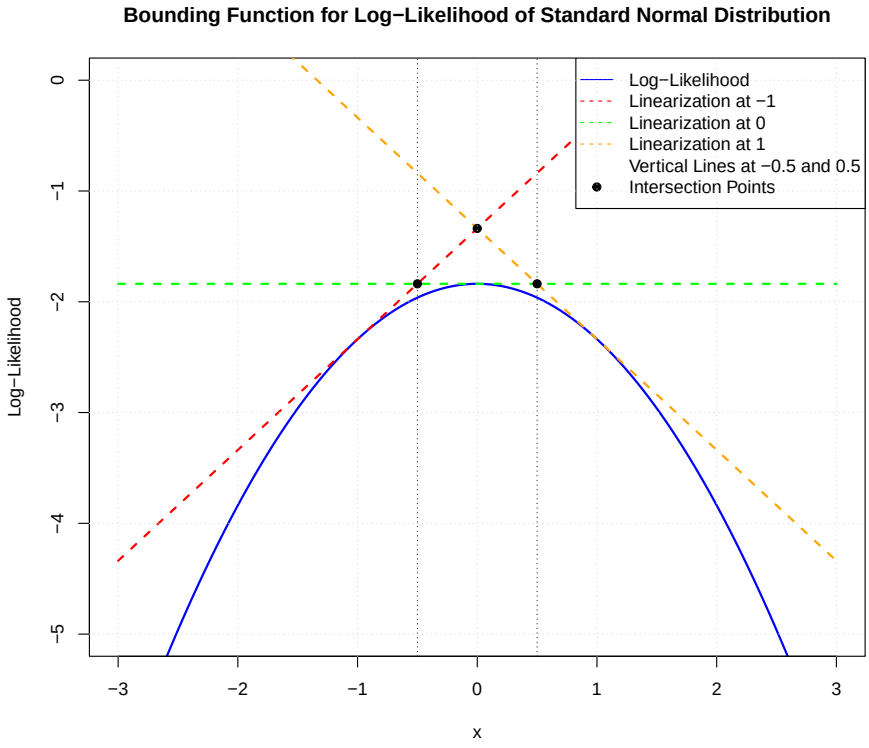


Figure 1: Illustration of the boundaries for the log-density, with the linearization lines and their respective points of intersection.

When integrating $\mathcal{B}(x)$ we observe that the resulting improper integrals are tedious to analyse, but by exponentially transforming $\mathcal{B}(x)$ (i.e $e^{\mathcal{B}(x)}$) the task becomes rather straightforward. We

proceed

$$\begin{aligned}\mathcal{G}(x) &= e^{\mathcal{B}(x)} \\ &= \begin{cases} \frac{1}{\sqrt{2\pi}}\left(e^{x+\frac{1}{2}}\right), & x \in (-\infty, -\frac{1}{2}) \\ \frac{1}{\sqrt{2\pi}} & x \in [-\frac{1}{2}, \frac{1}{2}) \\ \frac{1}{\sqrt{2\pi}}\left(e^{-x+\frac{1}{2}}\right), & x \in [\frac{1}{2}, \infty) \end{cases} \\ &= \frac{1}{\sqrt{2\pi}} \begin{cases} e^{x+\frac{1}{2}}, & x \in (-\infty, -\frac{1}{2}) \\ 1 & x \in [-\frac{1}{2}, \frac{1}{2}) \\ e^{-x+\frac{1}{2}}, & x \in [\frac{1}{2}, \infty) \end{cases}\end{aligned}$$

where then the integral of $\mathcal{G}(x)$ becomes

$$\begin{aligned}\int_{-\infty}^{\infty} \mathcal{G}(x) &= \frac{1}{\sqrt{2\pi}} \left(\int_{-\infty}^{-1/2} e^{x+\frac{1}{2}}\, dx + \int_{-1/2}^{1/2} 1\, dx + \int_{1/2}^{\infty} e^{-x+\frac{1}{2}}\, dx \right) \\ &= \frac{1}{\sqrt{2\pi}} \left(\left[e^{x+\frac{1}{2}} \right]_{-\infty}^{-\frac{1}{2}} + 1 + \left[-e^{-x+\frac{1}{2}} \right]_{\frac{1}{2}}^{\infty} \right) \\ &= \frac{1}{\sqrt{2\pi}} \left(1 + 1 + 1\right) \\ &= \frac{3}{\sqrt{2\pi}}\end{aligned}$$

which is not equal to 1.

Since the bounding function $\mathcal{G}(x)$ did not integrate to 1 we recall from rudimentary probability theory that if we are to find the probability distribution of $\mathcal{G}(x)$ then its integral must equate to 1. This can be done by letting

$$g(x) = k\mathcal{G}(x)$$

for some sufficient $k \in \mathbb{R}$ satisfying the criterion. Where in this case $k = \frac{\sqrt{2\pi}}{3}$, which gives

$$\begin{aligned}g(x) &= \frac{\sqrt{2\pi}}{3} \cdot \mathcal{G}(x) \\ &= \begin{cases} \frac{e^{x+\frac{1}{2}}}{3}, & x \in (-\infty, -\frac{1}{2}) \\ \frac{1}{3} & x \in [-\frac{1}{2}, \frac{1}{2}) \\ \frac{e^{-x+\frac{1}{2}}}{3}, & x \in [\frac{1}{2}, \infty) \end{cases} \\ &= \begin{cases} \frac{1}{3} & \text{if } -0.5 < x \leq 0.5 \\ \frac{\exp(-|x|+0.5)}{3} & \text{else} \end{cases}\end{aligned}$$

and we are done. ■

c.)

Algorithm 1 below uses rejection sampling to sample the standard normal distribution. Show the relation to the distribution in (1), compute the average acceptance rate, and implement the algorithm

Algorithm 1: Rejection Sampling Algorithm

Result: Accepted sample x_p
while *not accepted* **do**
  Sample U_1, U_2 iid from Unif$[0,1]$;
  Sample $i \in \{1,2,3\}$, with probability $p_0 = \{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$;
  if $i = 1$ **then**
    $x_p = U_1 - 0.5$;
  else
    if $i = 2$ **then**
      $x_p = \ln(1 - U_1) - 0.5$;
    else
      $x_p = -\ln(1 - U_1) + 0.5$;
    end
  end
  if $U_2 < \frac{\sqrt{2\pi}\cdot\phi(x_p)}{3\cdot g(x_p)}$ **then**
    Accept proposal and return x_p;
  end
end

Solution:

We start off by finding the relation between the sampling algorithm and $g(x)$ through inverse transform sampling, like it is sketched in (a.).

Finding the cumulative distribution function (CDF):

The first step in our transformation involves finding $G(x)$, which is simply to look at

$$G(x) = \frac{1}{3}\left(\int_{-\infty}^{-1/2} e^{x+\frac{1}{2}}\, dx\right), \quad \text{for } x \in (-\infty, -\frac{1}{2}] \tag{14}$$

$$G(x) = \frac{1}{3}\left(\int_{-\infty}^{-0.5} e^{x+\frac{1}{2}} + \int_{-0.5}^{x} 1\, dx\right), \quad \text{for } x \in (-\frac{1}{2}, \frac{1}{2}] \tag{15}$$

$$G(x) = \frac{1}{3}\left(\int_{-\infty}^{-1/2} e^{x+\frac{1}{2}} + \int_{-1/2}^{1/2} 1\, dx + \int_{1/2}^{\infty} e^{-x+\frac{1}{2}}\right), \quad \text{for } x \in (\frac{1}{2}, \infty) \tag{16}$$

where by doing these calculations give us the following CDF

$$G(x) = \begin{cases} \frac{e^{x+\frac{1}{2}}}{3}, & x \in (-\infty, -\frac{1}{2}] \\ \frac{1}{3} + \frac{x+0.5}{3} & x \in (-\frac{1}{2}, \frac{1}{2}] \\ 1 - \frac{e^{-x+\frac{1}{2}}}{3}, & x \in (\frac{1}{2}, \infty) \end{cases}$$

Inverse transformation:

The next step is hence to find the inverse of $G(x)$, which again must be done to each separate argument in $G(x)$

$$G^{-1}(y) = \log(3y) - \frac{1}{2}, \quad \text{for } y \in (0, \frac{1}{3}] \tag{17}$$

$$G^{-1}(y) = 3y - \frac{3}{2}, \quad \text{for } y \in (\frac{1}{3}, \frac{2}{3}] \tag{18}$$

$$G^{-1}(y) = -\log(-3y + 3) + \frac{1}{2}, \quad \text{for } y \in (\frac{2}{3}, 1). \tag{19}$$

The inverse variable y now spans from 0 to 1 which tells us that we are in the correct domain of

sampling. The full inverse is then expressed as

$$G^{-1}(y) = \begin{cases} \log(3y) - \frac{1}{2}, & y \in (0, \frac{1}{3}] \\ 3y - \frac{3}{2}, & y \in (\frac{1}{3}, \frac{2}{3}] \\ -\log(-3y + 3) + \frac{1}{2}, & y \in (\frac{2}{3}, 1) \end{cases}$$

and if by substituting $u = 3y$ yields

$$G^{-1}(u) = \begin{cases} \log(u) - \frac{1}{2}, & y \in (0, 1] \\ u - \frac{3}{2}, & y \in (1, 2] \\ -\log(-u + 3) + \frac{1}{2}, & y \in (2, 3) \end{cases}$$

making $U \sim \text{Unif}[0, 3]$.

The implementation of the rejection sampling scheme can be found in Appendix (B), from code listings (2) and produces the following output for our acceptance rate:

$$U_{\text{rate}} = 0.8358 \approx \frac{\sqrt{2\pi}}{3}$$

d.)

An alternative to rejection sampling is importance sampling. Implement an importance sampler based on $g(x)$. Give arguments for choices you make when setting up the importance sampler.

Solution:

The implementation of the importance sampling algorithm can be found in Appendix (B), from code listings (3) and produces the following output

$$\hat{\mathbb{E}}(g) = 0.497625803124113$$

In importance sampling, we aim to estimate properties of a **target** distribution by drawing samples from a **proposal** distribution, and assigning weights to each sample based on their likelihood under the target distribution compared to the proposal distribution. Important considerations can include choosing a proposal distribution with heavier tails than the target distribution, making sure there is coverage of the entire target distribution to avoid zero probabilities, normalizing weights to ensure they sum to 1, and estimating the desired property by computing a weighted average of the samples. The number of samples to generate depends on the desired accuracy of the estimate.

e.)

Discuss the differences between rejection sampling and importance sampling. Compare the two approaches in a numerical study where you quantify the Monte Carlo variance in the results. For each rerun of the estimation should be based on 1000 samples from $g(x)$. In the numerical study you should estimate $\mathbb{E}(h(x))$, where:

$$h(x) = x^2 I(x > 0)$$

Solution:

The implementation of the numerical comparison can be found in Appendix (B), from code listings

(4) and produces the following output:

$$\hat{\mathbb{E}}_{\mathrm{RS}}(h) = 0.526384920450183$$

$$\hat{\mathbb{E}}_{\mathrm{IS}}(h) = 0.487551647399552$$

$$\mathrm{SD}_{RS}(h) = 1.13961507092653$$

$$\mathrm{SD}_{IS}(h) = 0.278753890995243$$

where $\hat{\mathbb{E}}_{\mathrm{RS}}(h), \hat{\mathbb{E}}_{\mathrm{IS}}(h)$ are the estimates for the rejection sampling (RS) and importance sampling (IS) respectively. From the results shown above we conclude that importance sampling yield slightly better results than rejection.

Differences:

Rejection sampling involves generating samples from a proposal distribution and accepting or rejecting them based on their likelihood under the target distribution. This method requires a well-defined rejection criterion and may result in low acceptance rates, especially in higher dimensional spaces. Conversely, importance sampling directly estimates properties of the target distribution by weighting samples drawn from a proposal distribution. It provides unbiased estimates, but relies heavily on the choice of proposal distribution, which should cover the entire target distribution (i.e exercise 1d) to avoid significant weights in regions of low probability density.

Problem 2 - (Sequential Monte Carlo)

In spatial statistics there are many different approaches for modelling continuity of discrete properties in a region. One method which is popular due to the intuitive interpretation is the truncated Gaussian model. In this model one generates a Gaussian random field in $\mathbb{R}^d$ and divide the value set into intervals to define the classes. When applying the model, it is challenging to get the parameters of the model right. One way to get the parameters correct is to estimate them from training data. In this case we have a map of discrete values and want to derive the properties of the underlying Gaussian random field. This is the situation we will investigate below. We will however restrict focus to 1D in this exercise.

The model is given as:

$$x_1 \sim \mathcal{N}(0, 1) \tag{20}$$

$$x_t = a x_{t-1} + \lambda + \varepsilon_t, \quad t = 2, 3, \ldots \tag{21}$$

$$\varepsilon_t \sim \mathcal{N}(0, \sigma^2), \quad t = 2, 3, \ldots \tag{22}$$

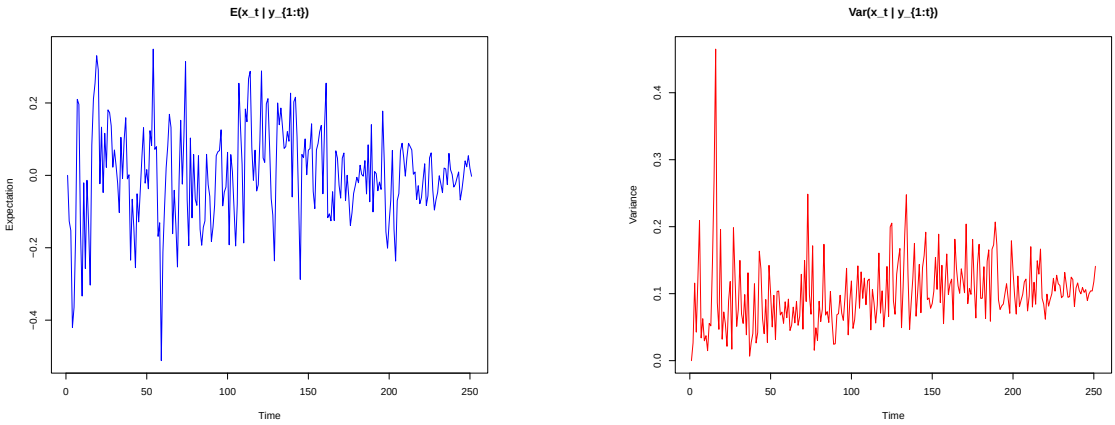
$$y_t = \begin{cases} 1 & \text{if } x < -0.5 \\ 2 & \text{if } -0.5 \leq x < 0.5\ , \\ 3 & \text{if } x \geq 0.5 \end{cases} \quad t = 2, 3, \ldots \tag{23}$$

a.)

Design a sequential Monte Carlo algorithm for inference about $p(x_t|\boldsymbol{y}_{1:t})$, $t = 1, 2, \ldots, n$, assuming that $(a, \lambda, \sigma^2) = (0.85, 0, 0.5^2)$. Display plot of $\mathbb{E}(x_t|\boldsymbol{y}_{1:t})$ and $Var(x_t|\boldsymbol{y}_{1:t})$. Give arguments for the choices you make in the construction.

Solution:

The implementation can be found in Appendix (B), from code listings (5), and produces the following plots:



(a) $\mathbb{E}(x_t|\boldsymbol{y}_{1:t})$ (b) $\text{Var}(x_t|\boldsymbol{y}_{1:t})$

Figure 2: Sequential Monte Carlo Simulation estimation of $\mathbb{E}(x_t|\boldsymbol{y}_{1:t})$ and $\text{Var}(x_t|\boldsymbol{y}_{1:t})$

In construction of the sequential Monte Carlo (SMC) algorithm, the choice of 1000 particles was chosen to balance out the ratio of computational effort and accuracy of estimation. The particles are initialized with zero values for simplicity, and weights are initialized with equal weights. The resampling is applied to address potential particle degeneracy, such that it maintains diverse candidate solutions and make that particles with higher weights are retained for the next iteration. Here, I have used simple multinomial resampling with replacement. Lastly, importance weights are calculated based on the likelihood of observing the data given the particles' states.

b.)

We will use sequential Monte Carlo to perform online learning algorithm for the parameter . For simplicity we will assume that $(\lambda, \sigma^2) = (0, 0.5^2)$ *is fixed and known.*

Consider a prior model for a which is uniform on the interval $[0, 1]$*. Design and implement a sequential Monte Carlo algorithm for inference about a, i.e. estimate* $p(a|\boldsymbol{y}_{1:251})$*. Use a static approach, i.e. sample a initially from the prior distribution, and update it by weighting/resampling throughout the simulation. Why is this approach in general not recommended? Comment on your results, are these acceptable?*

Solution:

The implementation can be found in Appendix (B), from code listings (6), and produces the following output:

$$p(a|\boldsymbol{y}_{1:251}) = 0.41077276906278$$

Problem 3 - (McMC – in Bayesian analysis)

We will in this exercise consider a Bayesian generalized linear model for identifying influential factors for presence vs absence of malaria in blood samples taken from children in Gambia. The data set "gambia" from the geoR package is available on the course page. A description of the data is given in an associated file. These data are often used for spatial analysis, but we will not consider that aspect here. We will assume all observations to be independent and investigate a probit-link between the explanatory variables $\boldsymbol{x}$ and a binary response variable y. For person i the

probit-link between explanatory variables and the response is defined as:

$$\mathbb{P}(y_i = 1) = \Phi(\boldsymbol{\beta}^T \boldsymbol{x}_i) \tag{24}$$

where Φ is the cumulative distribution for a standard normal variable. In the Bayesian analysis below we will use the improper prior $p(\boldsymbol{\beta}) \propto 1$, in which case the posterior $p(\boldsymbol{\beta}|\boldsymbol{y})$ is proportional to the likelihood $L(\boldsymbol{\beta}|\boldsymbol{y})$.

a.)

Define $p_i = \Phi(\boldsymbol{\beta}^T \boldsymbol{x}_i)$ and derive the likelihood function:

$$L\left(\boldsymbol{\beta}|\boldsymbol{y}\right) = \prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1-y_i} \tag{25}$$

Discuss how you can implement a numerically robust evaluation of this likelihood. How should you handle the situation where $\boldsymbol{\beta}^T \boldsymbol{x}_i$ has a large absolute value? In a Metropolis Hastings algorithm, you are asked to evaluate the ratio of two likelihoods, how is this done in a numerically stable way?

Solution:

b.)

Below you will be asked to implement a random walk using a random scan Metropolis Hastings algorithm to investigate the posterior distribution. From the general expression for the Metropolis Hastings (M-H) ratio, deduce the M-H ratio for the random walk. Which criteria need to be met in order for the Markov chain to converge to the stationary distribution? Which of these criteria does the M-H ratio help you to fulfill?

Solution:

c.)

For the Gambia data: Implement a random walk algorithm with a random scan to sample from the posterior distribution $p(\boldsymbol{\beta}|\boldsymbol{y})$, and apply it. Use a model containing the explanatory variables: age, netuse, treated, green, and phc, in addition to the constant term. (Hint: 1-Remember to standardize the design matrix, 2- put some effort into a robust evaluation of the likelihood ratio, see a)

Solution:

d.)

Display plots which illustrate the convergence properties of the method. Comment on the convergence properties of your algorithm. If there are any obvious problems, suggest modifications and revisit c to improve convergence.

Solution:

e.)

A common way to sample the distribution above is to introduce a latent variable, z_i, which is defined such that $\{y_i = 1\} \iff \{z_i > 0\}$, and

$$z_i \sim \mathcal{N}\left(\boldsymbol{\beta}^T \boldsymbol{x}_i, 1^2\right) \tag{26}$$

Argue that the expanded posterior probability distribution

$$p\left(\boldsymbol{\beta}, \boldsymbol{z} | \boldsymbol{y}\right) \propto \prod_{i=1}^{n} \left[I(z_i > 0) \cdot I(y_i = 1) + I(z_i \leq 0) I(y_i = 0)\right] \phi\left(z_i - \boldsymbol{\beta}^T \boldsymbol{x}_i\right)$$

will have the prescribed marginal posterior, $p(\boldsymbol{\beta}|\boldsymbol{y})$ from expression (25). (Hint: consider one data point first and integrate out the latent variable.)

Solution:

f.)

You shall now derive the conditional distributions needed for Gibbs sampling. Show that:

$$p(z_i | \boldsymbol{z}_{-i}, \boldsymbol{\beta}, \boldsymbol{y}) \propto \begin{cases} I(z_i \leq 0)\phi(z_i - \boldsymbol{\beta}\boldsymbol{x}_i), & \text{if } y_i = 0 \\ I(z_i > 0)\phi(z_i - \boldsymbol{\beta}\boldsymbol{x}_i), & \text{if } y_i = 1 \end{cases}$$

and

$$p(\boldsymbol{\beta}|\boldsymbol{z}, \boldsymbol{y}) \propto \exp\left(-0.5 \sum_{i=1}^{n} (z_i - \boldsymbol{\beta}^T \boldsymbol{x}_i)^2\right)$$

Use results from multi linear regression to deduce that the distribution $p(\boldsymbol{\beta}|\boldsymbol{z}, \boldsymbol{y})$, is multi-normal with parameters:

$$\mathbb{E}(\boldsymbol{\beta}|\boldsymbol{z}, \boldsymbol{y}) = \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{z}$$

and

$$\mathrm{Cov}(\boldsymbol{\beta}|\boldsymbol{z}, \boldsymbol{y}) = (\boldsymbol{X}^T \boldsymbol{X})^{-1}$$

State the result from multivariate linear regression you are using and show how you use the connection to derive the results.

Solution:

g.)

Implement the Gibbs sampler, and use this to solve the inference of Gambia data. Use a block update to sample $p(\boldsymbol{\beta}|\boldsymbol{z}, \boldsymbol{y})$. Make illustrations of the convergence of this chain as well (as done for the random walk approach in in d). (Hint: to sample from the multi-normal distribution use code from a library, e.g. `rmvnorm` *from* `mvtnorm`

Solution:

h.)

Compare the two approaches you have implemented for evaluating the posterior. Compare implementation, runtime effects, and results e.g. burn in, convergence, mixing etc.

Solution:

Appendix

A. GitHub repository

B. Source Code

Exercise 1 b): Illustrate the bounding function $\mathcal{B}(x)$

```r
# Define the log-likelihood function
log_likelihood <- function(x) {
  return(log(1/(2*pi)) - 0.5 * x^2)
}

# Define linearization functions
linearization_minus_1 <- function(x) {
  return(log(1/(2*pi)) + 0.5 + x)
}

linearization_0 <- function(x) {
  return(rep(log(1/(2*pi)), length(x)))
}

linearization_plus_1 <- function(x) {
  return(log(1/(2*pi)) + 0.5 - x)
}

# Define function to find intersection point between two lines
find_intersection <- function(a1, b1, a2, b2) {
  x_intersect <- (b2 - b1) / (a1 - a2)
  y_intersect <- a1 * x_intersect + b1
  return(c(x_intersect, y_intersect))
}

# Generate x values
x_values <- seq(-3, 3, length.out = 1000)

# Compute linearization values
linearization_minus_1_values <- linearization_minus_1(x_values)
linearization_0_values <- linearization_0(x_values)
linearization_plus_1_values <- linearization_plus_1(x_values)

# Find intersection points between linearization lines
intersection_minus_1_0 <- find_intersection(1, log(1/(2*pi)) + 0.5, 0,
    log(1/(2*pi)))
intersection_0_plus_1 <- find_intersection(-1, log(1/(2*pi)), 0,
    log(1/(2*pi)) + 0.5)
intersection_minus_1_plus_1 <- find_intersection(1, log(1/(2*pi)) + 0.5,
    -1, log(1/(2*pi)) + 0.5)

# Plot
plot(x_values, log_likelihood(x_values), type='l', col='blue', lwd=2,
    ylim=c(-5, 0),
     xlab='x', ylab='Log-Likelihood', main='Bounding Function for
    Log-Likelihood of Standard Normal Distribution')
lines(x_values, linearization_minus_1_values, col='red', lty='dashed',
    lwd=2)
lines(x_values, linearization_0_values, col='green', lty='dashed', lwd=2)
lines(x_values, linearization_plus_1_values, col='orange', lty='dashed',
    lwd=2)
points(intersection_minus_1_0[1], intersection_minus_1_0[2], col='black',
    pch=19)
points(-intersection_minus_1_0[1], intersection_minus_1_0[2],
    col='black', pch=19)
#points(intersection_0_plus_1[1], intersection_0_plus_1[2], col='black',
    pch=19)
```

```r
points(intersection_minus_1_plus_1[1], intersection_minus_1_plus_1[2],
    col='black', pch=19)

abline(v = -0.5, col='black', lty='dotted')
abline(v = 0.5, col='black', lty='dotted')
legend('topright', legend=c('Log-Likelihood', 'Linearization at -1',
    'Linearization at 0', 'Linearization at 1', 'Vertical Lines at -0.5
    and 0.5', 'Intersection Points'),
       col=c('blue', 'red', 'green', 'orange', 'black', 'black'),
    lty=c(1, 2, 2, 2, 0, NA), pch=c(NA, NA, NA, NA, NA, 19))
grid()
```

Listing 1: Exercise 1b

Exercise 1 c): Rejection Sampling

```r
# Define the standard normal density function
set.seed(123)
phi <- function(x) {
  return(dnorm(x))
}

# Define the target density function g(x)
g <- function(x) {
  ifelse(abs(x) <= 0.5, 1/3, exp(-abs(x) + 0.5) / 3)
}

# Rejection sampling function
rejection_sampling <- function(iterations) {
  accepted <- 0

  for (i in 1:iterations) {
    # Step 1: Sample U1, U2 from Uniform [0,1]
    U1 <- runif(1)
    U2 <- runif(1)

    # Step 2: Sample i from {1, 2, 3} with equal probability
    i <- sample(1:3, 1, prob = c(1/3, 1/3, 1/3))

    # Step 3: Generate proposal x_p based on i
    if (i == 1) {
      x_p <- U1 - 0.5
    } else if (i == 2) {
      x_p <- log(1 - U1) - 0.5
    } else {
      x_p <- -log(1 - U1) + 0.5
    }

    # Step 4: Accept or reject proposal based on U2 and the acceptance
    condition
    if (U2 < sqrt(2 * pi) * phi(x_p) / (3 * g(x_p))) {
      accepted <- accepted + 1
      # Here you can store or print the accepted proposals
    }
  }

  return(accepted / iterations)  # Return the average acceptance rate
}

# Set the number of iterations
iterations <- 10000
```

```r

# Run rejection sampling and compute the average acceptance rate
average_acceptance_rate <- rejection_sampling(iterations)
print(paste("Average Acceptance Rate:", average_acceptance_rate))
```

Listing 2: Exercise 1c

Exercise 1 d): Importance Sampling

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

# Define the standard normal density function
phi <- function(x) {
  return(dnorm(x))
}

g <- function(x) {
  ifelse(abs(x) <= 0.5, 1/3, exp(-abs(x) + 0.5) / 3)
}

# Importance sampling function
importance_sampling <- function(n_samples) {
  # Generate samples from the proposal distribution (standard normal)
  samples <- runif(n_samples)

  # Calculate importance weights
  weights <- sapply(samples, function(x) g(x) / phi(x))

  # Normalize weights
  normalized_weights <- weights / sum(weights)

  # Compute estimate of the expectation
  estimate <- sum(samples * normalized_weights)

  return(estimate)
}

# Number of samples to generate
n_samples <- 10000

# Run importance sampling
estimate <- importance_sampling(n_samples)

# Print estimate of the expectation
print(paste("Estimate of the expectation:", estimate))
```

Listing 3: Exercise 1d

Exercise 1 e): Numerical Comparison (Rejection vs Importance)

```r
set.seed(2000)
# Define the target density function g(x)
g <- function(x) {
  ifelse(abs(x) <= 0.5, 1/3, exp(-abs(x) + 0.5) / 3)
}

# Function h(x)
h <- function(x) {
  return(x^2 * (x > 0))
}

# Rejection sampling function without NA estimates
# Rejection sampling function without NA estimates
rejection_sampling <- function(n_samples) {
  accepted_samples <- numeric(0)
  while (length(accepted_samples) == 0) {
    for (i in 1:n_samples) {
      accepted <- FALSE
      while (!accepted) {
        # Step 1: Sample U1, U2 from Uniform [0,1]
        U1 <- runif(1)
        U2 <- runif(1)

        # Step 2: Sample i from {1, 2, 3} with equal probability
        i <- sample(1:3, 1, prob = c(1/3, 1/3, 1/3))

        # Step 3: Generate proposal x_p based on i
        if (i == 1) {
          x_p <- U1 - 0.5
        } else if (i == 2) {
          x_p <- -log(1 - U1) + 0.5
        } else {
          x_p <- log(1 - U1) - 0.5
        }

        # Step 4: Accept or reject proposal based on U2 and the acceptance condition
        if (U2 < sqrt(2 * pi) * dnorm(x_p) / (3 * g(x_p))) {
          accepted_samples <- c(accepted_samples, x_p)
          accepted <- TRUE
        }
      }
    }
  }

  estimate <- mean(h(accepted_samples))
  return(estimate)
}




importance_sampling <- function(n_samples) {
  # Generate samples from the proposal distribution (standard normal)
  samples <- runif(n_samples)

  # Calculate importance weights
  weights <- sapply(samples, function(x) g(x) / h(x))

  # Normalize weights
  normalized_weights <- weights / sum(weights)

  # Compute estimate of the expectation
  estimate <- sum(samples * normalized_weights)

  return(estimate)
}
```

```r
# Number of iterations
N <- 1000

# Results storage
rejection_estimates <- numeric(N)
importance_estimates <- numeric(N)

# Perform the numerical study for rejection sampling
for (i in 1:N) {
  rejection_estimates[i] <- rejection_sampling(1)
}

# Perform the numerical study for importance sampling
for (i in 1:N) {
  importance_estimates[i] <- importance_sampling(1)
}

# Compute mean estimates
rejection_mean <- mean(rejection_estimates)
importance_mean <- mean(importance_estimates)

rejection_sd <- sd(rejection_estimates, na.rm = TRUE)
importance_sd <- sd(importance_estimates)

# Print results
print("Rejection Sampling:")
print(paste("Mean Estimate:", rejection_mean))
print(paste("Standard Deviation:", rejection_sd))

print("\nImportance Sampling:")
print(paste("Mean Estimate:", importance_mean))
print(paste("Standard Deviation:", importance_sd))
```

Listing 4: Exercise 1e

Exercise 2 a): Sequential Monte Carlo algorithm for inference about $p(x_t|\boldsymbol{y}_{1:t}),\ t = 1, 2, \ldots, n$

```r
set.seed(2000)
# Parameters
a <- 0.85
lambda <- 0
sigma_sq <- 0.5^2

# Load data
data <- read.table("data/tgsim.ascii", header = FALSE)
colnames(data) <- c('y')
y <- data$y
n <- length(y)

# Initialize
n_particles <- 1000
particles <- matrix(0, nrow = n_particles, ncol = n)
weights <- rep(1/n_particles, n_particles)

# Initialize vectors to store estimates
E_xt <- numeric(n)
Var_xt <- numeric(n)

# SIS algorithm
```

```r
for (t in 2:n) {
  # Propagate particles forward according to the state transition model
  particles[, t] <- a * particles[, t-1] + lambda + rnorm(n_particles,
    mean = 0, sd = sqrt(sigma_sq))

  # Sample epsilon from normal distribution
  epsilon <- rnorm(n_particles, mean = 0, sd = sqrt(sigma_sq))

  # Assign discrete values y_t based on intervals
  y <- ifelse(particles[, t] < -0.5, 1, ifelse(particles[, t] < 0.5, 2,
    3))

  # Calculate importance weights
  likelihood <- ifelse(y == 1, dnorm(particles[, t], mean = 0, sd =
    sqrt(sigma_sq)),
                       ifelse(y == 2, dnorm(particles[, t], mean = 0, sd
    = sqrt(sigma_sq)),
                              dnorm(particles[, t], mean = 0, sd =
    sqrt(sigma_sq))))
  weights <- weights * likelihood

  # Normalize weights
  weights <- weights / sum(weights)

  # Resampling
  indices <- sample(1:n_particles, n_particles, replace = TRUE, prob =
    weights)
  particles <- particles[indices, ]
  weights <- rep(1/n_particles, n_particles)

  # Calculate estimates
  E_xt <- colMeans(particles)
  Var_xt <- apply(particles, 2, var)
}

# Plotting
plot(E_xt, type = "l", col = "blue", xlab = "Time", ylab = "Expectation",
    main = "E(x_t | y_{1:t})")
plot(Var_xt, type = "l", col = "red", xlab = "Time", ylab = "Variance",
    main = "Var(x_t | y_{1:t})")
# lines(E_xt + sqrt(Var_xt), col = "red")
# lines(E_xt - sqrt(Var_xt), col = "darkgreen")
```

Listing 5: Exercise 2a

Exercise 2 b): Sequential Monte Carlo algorithm for inference about a, i.e. estimate $p(a|\boldsymbol{y}_{1:251})$.

```r
# Load required libraries
# library("dplyr")
set.seed(2000)
# Define the prior distribution for a (uniform)
prior_sample <- function(n_samples) {
  runif(n_samples, 0, 1)
}


# Likelihood function
likelihood <- function(particles, y, sigma_sq) {
  # Compute likelihood based on the observed class 'y'
```

```r
  likelihood <- ifelse(y == 1, dnorm(particles, mean = 0, sd =
    sqrt(0.5^2)),
                       ifelse(y == 2, dnorm(particles, mean = 0, sd =
    sqrt(0.5^2)),
                              dnorm(particles, mean = 0, sd =
    sqrt(0.5^2))))

  # Return the likelihood
  return(likelihood)
}


# Sequential Monte Carlo algorithm
sequential_monte_carlo <- function(y, n_particles) {
  # Initialize particles with samples from the prior distribution
  particles <- prior_sample(n_particles)

  # Initialize weights
  weights <- rep(1/n_particles, n_particles)

  # Iterate over observations
  for (obs in y) {
    # Compute likelihood for each particle
    particle_likelihood <- likelihood(particles, obs)

    # Compute unnormalized weights
    unnormalized_weights <- weights * particle_likelihood

    # Normalize weights
    normalized_weights <- unnormalized_weights / sum(unnormalized_weights)

    # Resample particles
    resampled_indices <- sample.int(n_particles, size = n_particles,
    replace = TRUE, prob = normalized_weights)
    particles <- particles[resampled_indices]

    # Reset weights
    weights <- rep(1/n_particles, n_particles)
  }

  # Estimate for p(a | y)
  estimate <- mean(particles)

  return(estimate)
}

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

# Number of particles
n_particles <- 1000
# Run sequential Monte Carlo algorithm
estimate <- sequential_monte_carlo(y, n_particles)
print(paste("Estimate of p(a | y):", estimate))
```

Listing 6: Exercise 2b