# Assignment 5

anonymous

## 1 General information

> ! Reporting accuracy
>
> **For posterior statistics of interest, only report digits for which the Monte Carlo standard error (MCSE) is zero.**
> *Example:* If you estimate $E(\mu) = 1.234$ with $\text{MCSE}(E(\mu)) = 0.01$, you should report $E(\mu) = 1.2$.
> See lecture video 4.1, the chapter notes, and a case study for more information.

## 2 Generalized linear model: Bioassay model with Metropolis

### 2.1 (a)

**Write your answers/code here!**

```
# Useful functions: runif, rnorm
# bioassaylp, dmvnorm (from aaltobda)

data("bioassay")
# Start by implementing a function called `density_ratio` to
# compute the density ratio function, $r$ in Eq. (11.1) in BDA3:
density_ratio <- function(alpha_propose, alpha_previous, beta_propose, beta_previous, x,
    # Do computation here, and return as below.
    # Below are the correct return values for two different calls of this function:
    q_previous = bioassaylp(alpha_previous, beta_previous, x, y,n) + dmvnorm(c(alpha_prev
    q_propose =  bioassaylp(alpha_propose, beta_propose, x, y,n) + dmvnorm(c(alpha_propos

    out = exp(q_propose - q_previous)
```

```
    out
    # alpha_propose = 1.89, alpha_previous = 0.374,
    # beta_propose = 24.76, beta_previous = 20.04,
    # x = bioassay$x, y = bioassay$y, n = bioassay$n
    # 1.305179

    # alpha_propose = 0.374, alpha_previous = 1.89,
    # beta_propose = 20.04, beta_previous = 24.76,
    # x = bioassay$x, y = bioassay$y, n = bioassay$n
    # 0.7661784

}
# Then implement a function called `metropolis_bioassay()` which
# implements the Metropolis algorithm using the `density_ratio()`:
metropolis_bioassay <- function(alpha_initial, beta_initial, alpha_sigma, beta_sigma, no_
    # Do computation here, and return as below.
  # initial state
  alpha_out = c(alpha_initial)
  beta_out = c(beta_initial)

  # iteration
  for (draw in 1:(no_draws-1)){
    # last value
    alpha_last = alpha_out[length(alpha_out)]
    beta_last = beta_out[length(beta_out)]

    # transition to proposed
    alpha_proposed = rnorm(1, mean=alpha_last, sd=alpha_sigma)
    beta_proposed = rnorm(1, mean=beta_last, sd = beta_sigma)

    # ratio between proposed and last
    ratio  = density_ratio(alpha_proposed, alpha_last,beta_proposed, beta_last, x, y, n)
    # update rule. If ratio is bigger than 1, always update, if lower, only with ratio pr
    update = runif(1) < min(1, ratio)

    if (update == TRUE){
      alpha_new = alpha_proposed
      beta_new = beta_proposed
    } else {
      alpha_new = alpha_last
      beta_new = beta_last
    }
    alpha_out = c(alpha_out, alpha_new)
    beta_out = c(beta_out, beta_new)
```

```
  }



    # Below are "wrong" values (unlikely to actually occur)
    # in the "correct" format (such that they work with the plotting functions further do
    # data.frame(
    #     alpha=c(alpha_initial, alpha_initial+alpha_sigma, alpha_initial-alpha_sigma),
    #     beta=c(beta_initial, beta_initial+beta_sigma, beta_initial-beta_sigma)
    # )
  data.frame(
        alpha=alpha_out,
        beta=beta_out
    )

}
df = metropolis_bioassay(0, 0, 1, 5, 1000, bioassay$x, bioassay$y, bioassay$n)
```

## 2.2 (b)

```
library(dplyr)
```

```
Warning: package 'dplyr' was built under R version 4.0.5
```

```
library(magrittr)
```

```
Warning: package 'magrittr' was built under R version 4.0.5
```

```
n_chains = 4
result = list()
for (chain in 1:n_chains){
  result[[chain]] = metropolis_bioassay(runif(1,0,10), runif(1,0,40), 1, 5, 1000, bioassa
  result[[chain]]['Chain'] = chain

}

chains_df = bind_rows(result)
```

**Write your answers/code here!**

1) Metropolis algorithm works by proposing a new value for the parameters based on some transition rule (ex: sample from a normal distribution centered in the previous value with some scale to define) and compare the posterior between the previous value and the proposed one. It does so by calculating the ratio, that doesn't require the full posterior, only likelihood and prior. If the ratio is larger than one, we update the parameter, if it's lower, we update it with probability equal to the ratio. The idea is that the parameters move towards the area with higher mass but that they still can move around exploring different areas of the distribution. We do this for some defined number of iterations.

2) The proposal distribution comes from sampling from a normal distribution centered at the previous value with scale 1 and 5 respectively. I tried a few scales and these one seemed to generate traces (mcmc_trace chart) with more variation centered is some value (mean) and not multiple areas with variation around different values. I guess it also means it was updating rapidly enough.

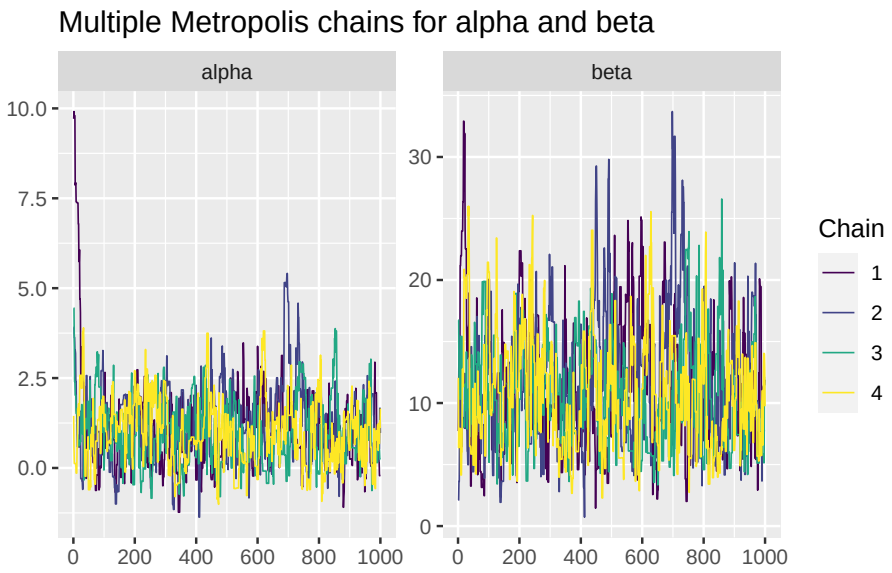
3) The starting points are random numbers between 0-10 for alpha and 0-40 for beta.

4 and 6) 4 chains are run, each of length 1000.

5) Usual convention of using 50% as warm up so 500 are warm up.

6) We can see in the plot below that chains converge for both parameters since all chains are mixed.

```
color_scheme_set("viridis")
p = mcmc_trace(chains_df, pars=c("alpha", "beta"))
p + labs(title="Multiple Metropolis chains for alpha and beta")
```




**The below example plot only includes a single chain, but your report should include a plot with multiple chains overlayed!**

```
# # Useful functions: mcmc_trace (from bayesplot)
# mcmc_trace(df, pars=c("alpha", "beta"))
```

## 2.3 (c)

**Write your answers/code here!** 1) The idea of $\hat{R}$ is to roughly estimate by how much the variance/scale of the value estimated would be reduced if we kept simulating more values going towards $n->$ inf

```
# Useful functions: rhat_basic (from posterior)
warmup = 501
alpha_r = df$alpha[warmup:length(df$alpha)]
beta_r = df$beta[warmup:length(df$beta)]

alpha_rhat = rhat_basic(alpha_r)
beta_rhat = rhat_basic(beta_r)
```

I used rhat_basic() which uses the more recent version. $\hat{R}$ for $\alpha$ is 1 and $\hat{R}$ for $\beta$ is 1.02.

These are the values for the normal distribution used as proposal detailed above.

I had to try a few scale parameters (increasing it ) specially for beta to obtain good results with 1000 iterations.

## 2.4 (c)

**Write your answers/code here!**

Have a look at bayesplot scatter plot examples and tune your plot if wanted/needed. Don't forget to include a title/caption/description.

```
chains_wo_warmup = chains_df %>% group_by(Chain) %>% filter(row_number() > 500)
```

```
# Useful functions: mcmc_scatter (from bayesplot)
p = mcmc_scatter(chains_wo_warmup, pars=c("alpha", "beta"))
p + labs(title = "Samples from the posterior distribution of alpha and beta using Metropo
```