BIS 567 HW05

Question 1

Part (A)

```
library(knitr)
library(kableExtra)
library(magrittr)
load(here::here("./bis567/homework/hw05/HW5.RData"))
set.seed(123)

# values that we need for matrix size etc
n <- length(y)
p <- ncol(x)
n_iter <- 100000 # num of iterations
burn_in <- 2000 # burn in
thin_interval <- 10 # thinning interval

# function to calculate the log posterior
log_posterior <- function(beta, y, x) {
    eta <- x %*% beta
    log_lik <- sum(y * eta - log(1 + exp(eta)))
    log_prior <- -0.5 * t(beta) %*% beta / 10000
    return(log_lik + log_prior)
}

# initialize the important bits
beta_init <- rep(0, p)
proposal_sd <- 0.2 # 0.2 gets about 50% acceptance
beta_samples <- matrix(0, nrow = n_iter, ncol = p)
beta_samples[1, ] <- beta_init

accept_count <- 0 # init the acceptance counter
for (i in 2:n_iter) {
    current_beta <- beta_samples[i - 1, ]

    # new beta
    proposed_beta <- current_beta + rnorm(p, mean = 0, sd = proposal_sd)

    # this function is pre-written
    log_posterior_current <- log_posterior(current_beta, y, x)
    log_posterior_proposed <- log_posterior(proposed_beta, y, x)

    # acceptance prob
    accept_prob <- exp(log_posterior_proposed - log_posterior_current)

    # reject or
```

```
    if (runif(1) < accept_prob) {
        beta_samples[i, ] <- proposed_beta
        accept_count <- accept_count + 1 # inc
    } else {
        beta_samples[i, ] <- current_beta
    }

    # for testing
    if (i %% 1000 == 0) {
        acceptance_rate <- accept_count / i
        cat("Iteration", i, "Acceptance Rate:", round(acceptance_rate, 4), "\n")
    }
}
```

```
## Iteration 1000 Acceptance Rate: 0.528
## Iteration 2000 Acceptance Rate: 0.515
## Iteration 3000 Acceptance Rate: 0.5197
## Iteration 4000 Acceptance Rate: 0.5182
## Iteration 5000 Acceptance Rate: 0.516
## Iteration 6000 Acceptance Rate: 0.5208
## Iteration 7000 Acceptance Rate: 0.5224
## Iteration 8000 Acceptance Rate: 0.5215
## Iteration 9000 Acceptance Rate: 0.5223
## Iteration 10000 Acceptance Rate: 0.5217
## Iteration 11000 Acceptance Rate: 0.5216
## Iteration 12000 Acceptance Rate: 0.523
## Iteration 13000 Acceptance Rate: 0.524
## Iteration 14000 Acceptance Rate: 0.5252
## Iteration 15000 Acceptance Rate: 0.5263
## Iteration 16000 Acceptance Rate: 0.5254
## Iteration 17000 Acceptance Rate: 0.5261
## Iteration 18000 Acceptance Rate: 0.5255
## Iteration 19000 Acceptance Rate: 0.5266
## Iteration 20000 Acceptance Rate: 0.5256
## Iteration 21000 Acceptance Rate: 0.5268
## Iteration 22000 Acceptance Rate: 0.5261
## Iteration 23000 Acceptance Rate: 0.5265
## Iteration 24000 Acceptance Rate: 0.5265
## Iteration 25000 Acceptance Rate: 0.5264
## Iteration 26000 Acceptance Rate: 0.5261
## Iteration 27000 Acceptance Rate: 0.526
## Iteration 28000 Acceptance Rate: 0.5248
## Iteration 29000 Acceptance Rate: 0.5248
## Iteration 30000 Acceptance Rate: 0.5248
## Iteration 31000 Acceptance Rate: 0.5247
## Iteration 32000 Acceptance Rate: 0.524
## Iteration 33000 Acceptance Rate: 0.524
## Iteration 34000 Acceptance Rate: 0.5244
## Iteration 35000 Acceptance Rate: 0.5239
## Iteration 36000 Acceptance Rate: 0.5246
## Iteration 37000 Acceptance Rate: 0.5256
## Iteration 38000 Acceptance Rate: 0.5258
## Iteration 39000 Acceptance Rate: 0.5257
## Iteration 40000 Acceptance Rate: 0.5258
```

```
## Iteration 41000 Acceptance Rate: 0.5252
## Iteration 42000 Acceptance Rate: 0.5246
## Iteration 43000 Acceptance Rate: 0.5247
## Iteration 44000 Acceptance Rate: 0.525
## Iteration 45000 Acceptance Rate: 0.5259
## Iteration 46000 Acceptance Rate: 0.526
## Iteration 47000 Acceptance Rate: 0.5259
## Iteration 48000 Acceptance Rate: 0.5256
## Iteration 49000 Acceptance Rate: 0.5258
## Iteration 50000 Acceptance Rate: 0.526
## Iteration 51000 Acceptance Rate: 0.5266
## Iteration 52000 Acceptance Rate: 0.5267
## Iteration 53000 Acceptance Rate: 0.5267
## Iteration 54000 Acceptance Rate: 0.5262
## Iteration 55000 Acceptance Rate: 0.5257
## Iteration 56000 Acceptance Rate: 0.5257
## Iteration 57000 Acceptance Rate: 0.5262
## Iteration 58000 Acceptance Rate: 0.5261
## Iteration 59000 Acceptance Rate: 0.5263
## Iteration 60000 Acceptance Rate: 0.5262
## Iteration 61000 Acceptance Rate: 0.5262
## Iteration 62000 Acceptance Rate: 0.5266
## Iteration 63000 Acceptance Rate: 0.5267
## Iteration 64000 Acceptance Rate: 0.5262
## Iteration 65000 Acceptance Rate: 0.5264
## Iteration 66000 Acceptance Rate: 0.5261
## Iteration 67000 Acceptance Rate: 0.5264
## Iteration 68000 Acceptance Rate: 0.5265
## Iteration 69000 Acceptance Rate: 0.5264
## Iteration 70000 Acceptance Rate: 0.5266
## Iteration 71000 Acceptance Rate: 0.5266
## Iteration 72000 Acceptance Rate: 0.5266
## Iteration 73000 Acceptance Rate: 0.5268
## Iteration 74000 Acceptance Rate: 0.5269
## Iteration 75000 Acceptance Rate: 0.5268
## Iteration 76000 Acceptance Rate: 0.5268
## Iteration 77000 Acceptance Rate: 0.5268
## Iteration 78000 Acceptance Rate: 0.5268
## Iteration 79000 Acceptance Rate: 0.5264
## Iteration 80000 Acceptance Rate: 0.5268
## Iteration 81000 Acceptance Rate: 0.5267
## Iteration 82000 Acceptance Rate: 0.5268
## Iteration 83000 Acceptance Rate: 0.5267
## Iteration 84000 Acceptance Rate: 0.5267
## Iteration 85000 Acceptance Rate: 0.5266
## Iteration 86000 Acceptance Rate: 0.5266
## Iteration 87000 Acceptance Rate: 0.5265
## Iteration 88000 Acceptance Rate: 0.5268
## Iteration 89000 Acceptance Rate: 0.5269
## Iteration 90000 Acceptance Rate: 0.527
## Iteration 91000 Acceptance Rate: 0.5268
## Iteration 92000 Acceptance Rate: 0.5268
## Iteration 93000 Acceptance Rate: 0.527
## Iteration 94000 Acceptance Rate: 0.5269
```

```
## Iteration 95000 Acceptance Rate: 0.5264
## Iteration 96000 Acceptance Rate: 0.5261
## Iteration 97000 Acceptance Rate: 0.526
## Iteration 98000 Acceptance Rate: 0.5262
## Iteration 99000 Acceptance Rate: 0.5265
## Iteration 100000 Acceptance Rate: 0.5267
```

```
# now discard the burn-in
post_burn_samples <- beta_samples[(burn_in + 1):n_iter, ]
# thin the samples!
thinned_samples <- post_burn_samples[seq(1, nrow(post_burn_samples), by = thin_interval), ]

# provide the inference on ONLY the thinned parts
summary_stats_thinned <- apply(thinned_samples, 2, function(x) {
    c(Mean = mean(x), SD = sd(x), `2.5%` = quantile(x, 0.025), `97.5%` = quantile(x, 0.975))
})

summary_stats_df <- as.data.frame(t(summary_stats_thinned))
summary_stats_df$parameter <- c("beta1", "beta2")
summary_stats_df_metrop <- summary_stats_df %>%
    dplyr::select(parameter, names(.)[1:4])

knitr::kable(summary_stats_df_metrop,
    caption = "Summary Statistics for Thinned Samples for Question 1a)",
    col.names = c("Parameter", "Estimate", "SD", "2.5%", "97.5%")
) %>%
    kableExtra::kable_classic()
```

Table 1: Summary Statistics for Thinned Samples for Question 1a)

Parameter	Estimate	SD	2.5%	97.5%
beta1	-0.1376366	0.1563357	-0.4452936	0.1659304
beta2	1.7644822	0.2339350	1.3232356	2.2465797

```
# diagnostics!
geweke_results <- coda::geweke.diag(thinned_samples)
ess_results <- coda::effectiveSize(thinned_samples)

diags_df_metrop <- data.frame(
    parameter = c("beta1", "beta2"),
    ESS = unname(ess_results),
    geweke = unname(geweke_results$z)
)
knitr::kable(diags_df_metrop,
    caption = "Convergence Diagnostics for Thinned Samples for Question 1a)",
    col.names = c("Parameter", "Effective Sample Size", "Geweke Diagnostic")
) %>%
    kableExtra::kable_classic()
```

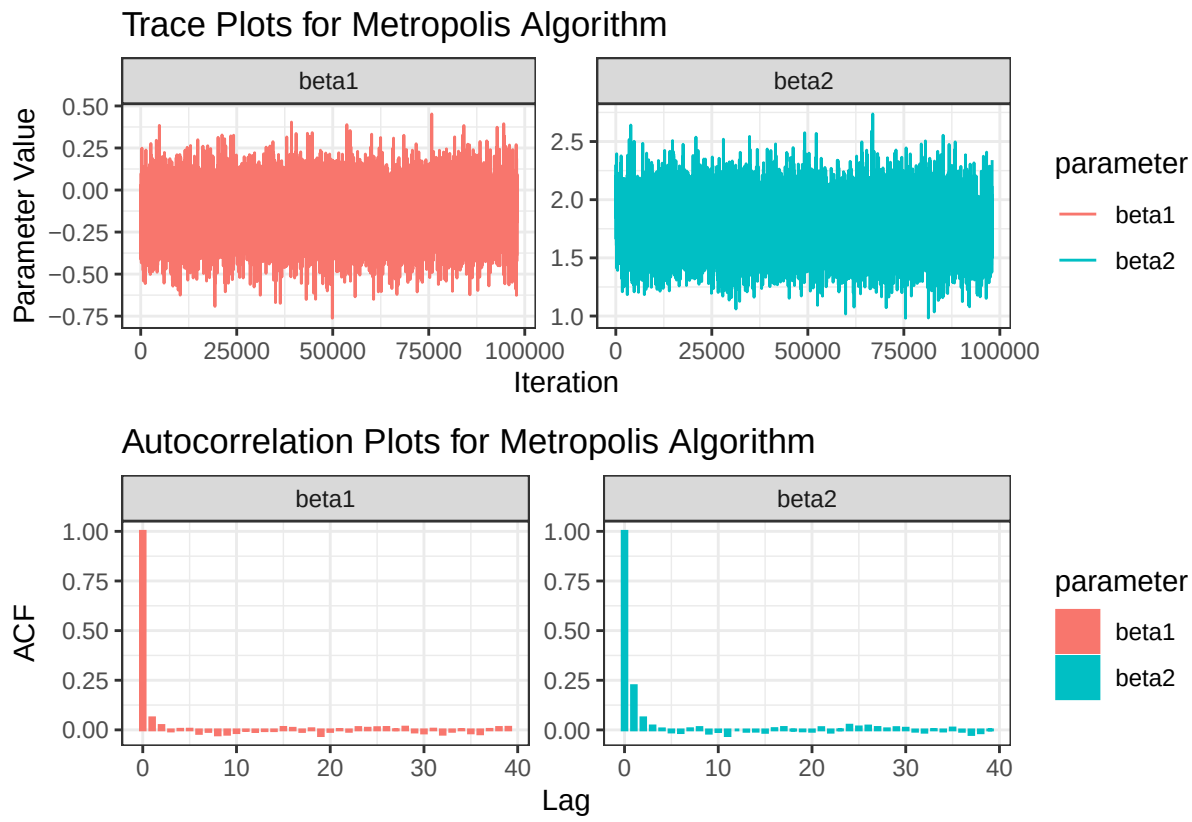
Table 2: Convergence Diagnostics for Thinned Samples for Question 1a)

Parameter	Effective Sample Size	Geweke Diagnostic
beta1	8362.623	1.0273850
beta2	6228.419	-0.0890246

```
# plotting
df_metropolis <- dplyr::as_tibble(data.frame(thinned_samples)) %>%
    dplyr::rename(beta1 = X1, beta2 = X2) %>%
    dplyr::mutate(
        iteration = seq(1, nrow(post_burn_samples), by = thin_interval),
        method = "Metropolis"
    )
df_metropolis_long <- dplyr::as_tibble(df_metropolis %>%
    tidyr::pivot_longer(
        cols = c("beta1", "beta2"),
        names_to = "parameter",
        values_to = "value"
    ))
trace_plot_metropolis <- ggplot2::ggplot(
    df_metropolis_long,
    ggplot2::aes(x = iteration, y = value, colour = parameter)
) +
    ggplot2::geom_line() +
    ggplot2::facet_wrap(~parameter, scales = "free_y") +
    ggplot2::labs(title = "Trace Plots for Metropolis Algorithm", y = "Parameter Value", x = "Iteration
    ggplot2::theme_bw()

acf_metropolis_beta1 <- acf(thinned_samples[, 1], plot = FALSE)
acf_metropolis_beta2 <- acf(thinned_samples[, 2], plot = FALSE)
acf_df_metropolis_beta1 <- dplyr::as_tibble(data.frame(
    lag = acf_metropolis_beta1$lag,
    acf = acf_metropolis_beta1$acf, parameter = "beta1"
))
acf_df_metropolis_beta2 <- dplyr::as_tibble(data.frame(
    lag = acf_metropolis_beta2$lag,
    acf = acf_metropolis_beta2$acf, parameter = "beta2"
))
acf_df_metropolis <- dplyr::bind_rows(
    acf_df_metropolis_beta1,
    acf_df_metropolis_beta2
) %>%
    dplyr::mutate(method = "Metropolis")
acf_plot_metropolis <- ggplot2::ggplot(
    acf_df_metropolis, ggplot2::aes(
        x = lag, y = acf,
        colour = parameter, fill = parameter
    )
) +
    ggplot2::geom_bar(stat = "identity", width = 0.5) +
    ggplot2::facet_wrap(~parameter, scales = "free_y") +
    ggplot2::labs(title = "Autocorrelation Plots for Metropolis Algorithm", y = "ACF", x = "Lag") +
    ggplot2::theme_bw()
plot_metropolis <- patchwork::wrap_plots(
    trace_plot_metropolis,
```

```
    acf_plot_metropolis,
    ncol = 1
) +
    patchwork::plot_layout(ncol = 1)
plot_metropolis
```



Part (B)

Part (i)

- see handwritten paper *

Part (ii)

- see handwritten paper *

```
library(pgdraw)
library(mvtnorm)
n <- length(y)
p <- ncol(x)

n_iter <- 100000 # num of iterations
burn_in <- 2000 # burn in
thin_interval <- 10 # thinning interval

# init the values and objects to store things in
```

```r
beta <- rep(0, p)
omega <- rep(1, n)
beta_samples <- matrix(0, nrow = n_iter, ncol = p)

for (t in 1:n_iter) {
    # current beta
    for (i in 1:n) {
        # this is the predicotr
        psi_i <- x[i, ] %*% beta

        # samp omega_i | beta using Polya-Gamma(1, psi_i)
        omega[i] <- pgdraw(1, psi_i)
    }

    # posterior covariance ?
    Omega <- diag(omega)
    Sigma_beta <- solve(t(x) %*% Omega %*% x + diag(1 / 10000, p))
    mu_beta <- Sigma_beta %*% t(x) %*% (y - 0.5)

    # draw beta
    beta <- as.vector(rmvnorm(1, mean = mu_beta, sigma = Sigma_beta))
    beta_samples[t, ] <- beta
}

# discard burnin
post_burn_samples <- beta_samples[(burn_in + 1):n_iter, ]

# thin them samples
thin_interval <- 10
thinned_samples <- post_burn_samples[seq(1, nrow(post_burn_samples), thin_interval), ]
summary_stats_thinned <- apply(thinned_samples, 2, function(x) {
    c(Mean = mean(x), SD = sd(x), `2.5%` = quantile(x, 0.025), `97.5%` = quantile(x, 0.975))
})

summary_stats_df <- as.data.frame(t(summary_stats_thinned))
summary_stats_df$parameter <- c("beta1", "beta2")
summary_stats_df_gibbs <- summary_stats_df %>%
    dplyr::select(parameter, names(.)[1:4])

# nice table
knitr::kable(summary_stats_df_gibbs,
    caption = "Summary Statistics for Thinned Samples for Question 1b-iii)",
    col.names = c("Parameter", "Estimate", "SD", "2.5%", "97.5%")
) %>%
    kableExtra::kable_classic()
```

Part (iii)

Table 3: Summary Statistics for Thinned Samples for Question 1b-iii)

Parameter	Estimate	SD	2.5%	97.5%
beta1	-0.136689	0.1567279	-0.4432636	0.1691957

beta2	1.762736	0.2363699	1.3136549	2.2441134

```
# diagnostics!
geweke_results <- coda::geweke.diag(thinned_samples)
ess_results <- coda::effectiveSize(thinned_samples)

diags_df_gibbs <- data.frame(
    parameter = c("beta1", "beta2"),
    ESS = unname(ess_results),
    geweke = unname(geweke_results$z)
)
knitr::kable(diags_df_gibbs,
    caption = "Convergence Diagnostics for Thinned Samples for
        Question 11b-iii)",
    col.names = c("Parameter", "Effective Sample Size", "Geweke Diagnostic")
) %>%
    kableExtra::kable_classic()
```

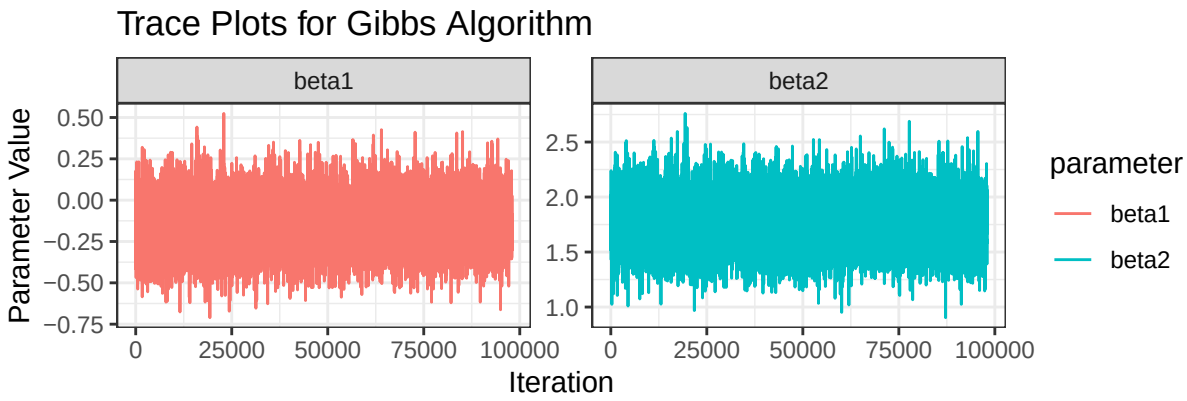
Table 4: Convergence Diagnostics for Thinned Samples for Question 11b-iii)

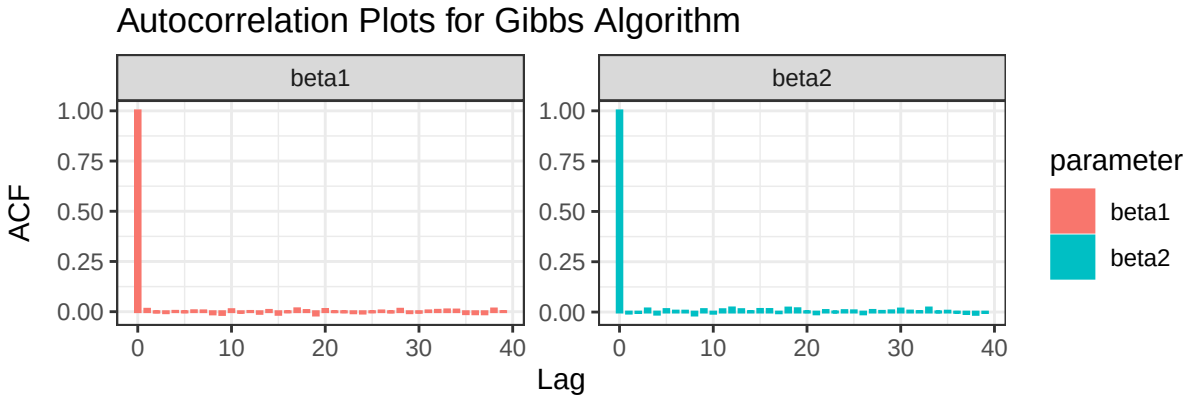
Parameter	Effective Sample Size	Geweke Diagnostic
beta1	9800	-1.906409
beta2	9800	-1.679111

```
df_gibbs <- dplyr::as_tibble(data.frame(thinned_samples)) %>%
    dplyr::rename(beta1 = X1, beta2 = X2) %>%
    dplyr::mutate(
        iteration = seq(1, nrow(post_burn_samples), by = thin_interval),
        method = "Gibbs"
    )
df_gibbs_long <- dplyr::as_tibble(df_gibbs %>%
    tidyr::pivot_longer(
        cols = c("beta1", "beta2"),
        names_to = "parameter",
        values_to = "value"
    ))
trace_plot_gibbs <- ggplot2::ggplot(
    df_gibbs_long,
    ggplot2::aes(x = iteration, y = value, colour = parameter)
) +
    ggplot2::geom_line() +
    ggplot2::facet_wrap(~parameter, scales = "free_y") +
    ggplot2::labs(
        title = "Trace Plots for Gibbs Algorithm",
        y = "Parameter Value", x = "Iteration"
    ) +
    ggplot2::theme_bw()

acf_gibbs_beta1 <- acf(thinned_samples[, 1], plot = FALSE)
acf_gibbs_beta2 <- acf(thinned_samples[, 2], plot = FALSE)
acf_df_gibbs_beta1 <- dplyr::as_tibble(data.frame(
```

```
    lag = acf_gibbs_beta1$lag,
    acf = acf_gibbs_beta1$acf, parameter = "beta1"
))
acf_df_gibbs_beta2 <- dplyr::as_tibble(data.frame(
    lag = acf_gibbs_beta2$lag,
    acf = acf_gibbs_beta2$acf, parameter = "beta2"
))
acf_df_gibbs <- dplyr::bind_rows(
    acf_df_gibbs_beta1,
    acf_df_gibbs_beta2
) %>%
    dplyr::mutate(method = "gibbs")
acf_plot_gibbs <- ggplot2::ggplot(
    acf_df_gibbs, ggplot2::aes(
        x = lag, y = acf,
        colour = parameter, fill = parameter
    )
) +
    ggplot2::geom_bar(stat = "identity", width = 0.5) +
    ggplot2::facet_wrap(~parameter, scales = "free_y") +
    ggplot2::labs(
        title = "Autocorrelation Plots for Gibbs Algorithm",
        y = "ACF", x = "Lag"
    ) +
    ggplot2::theme_bw()
plot_gibbs <- patchwork::wrap_plots(
    trace_plot_gibbs,
    acf_plot_gibbs,
    ncol = 1
) +
    patchwork::plot_layout(ncol = 1)
plot_gibbs
```





Question 1

Part (C)

```
metropolis_sampling <- function(y, x) {
    n <- length(y)
    p <- ncol(x)
    n_iter <- 100000

    beta_init <- rep(0, p)
    proposal_sd <- 0.2 # 0.2 gets about 50% acceptance
    beta_samples <- matrix(0, nrow = n_iter, ncol = p)
    beta_samples[1, ] <- beta_init

    for (i in 2:n_iter) {
        current_beta <- beta_samples[i - 1, ]

        # new beta
        proposed_beta <- current_beta + rnorm(p, mean = 0, sd = proposal_sd)

        # this function is pre-written
        log_posterior_current <- log_posterior(current_beta, y, x)
        log_posterior_proposed <- log_posterior(proposed_beta, y, x)

        # acceptance prob
        accept_prob <- exp(log_posterior_proposed - log_posterior_current)
```

```
        # reject or
        if (runif(1) < accept_prob) {
            beta_samples[i, ] <- proposed_beta
        } else {
            beta_samples[i, ] <- current_beta
        }
    }
}

gibbs_sampling <- function(y, x) {
    n <- length(y)
    p <- ncol(x)

    n_iter <- 100000 # num of iterations
    burn_in <- 2000 # burn in
    thin_interval <- 10 # thinning interval

    # init the values and objects to store things in
    beta <- rep(0, p)
    omega <- rep(1, n)
    beta_samples <- matrix(0, nrow = n_iter, ncol = p)

    for (t in 1:n_iter) {
        # current beta
        for (i in 1:n) {
            # this is the predicotr
            psi_i <- x[i, ] %*% beta

            # samp omega_i | beta using Polya-Gamma(1, psi_i)
            omega[i] <- pgdraw::pgdraw(1, psi_i)
        }

        # posterior covariance ?
        Omega <- diag(omega)
        Sigma_beta <- solve(t(x) %*% Omega %*% x + diag(1 / 10000, p))
        mu_beta <- Sigma_beta %*% t(x) %*% (y - 0.5)

        # draw beta
        beta <- as.vector(mvtnorm::rmvnorm(
            1,
            mean = mu_beta, sigma = Sigma_beta
        ))
        beta_samples[t, ] <- beta
    }
}
timing_results <- microbenchmark::microbenchmark(
    Metropolis_Hastings = metropolis_sampling(y, x),
    Gibbs_Sampling = gibbs_sampling(y, x),
    times = 1
)
```

```
## Warning in microbenchmark::microbenchmark(Metropolis_Hastings =
## metropolis_sampling(y, : less accurate nanosecond times to avoid potential
```

```
## integer overflows
time_metropolis <- median(
    timing_results$time[timing_results$expr == "Metropolis_Hastings"]
) / 1e9
time_gibbs <- median(
    timing_results$time[timing_results$expr == "Gibbs_Sampling"]
) / 1e9

comparison_df <- data.frame(
    method = c(rep("Metropolis", 2), rep("Gibbs", 2)),
    parameter = c("beta1", "beta2", "beta1", "beta2"),
    estimates = c(
        round(summary_stats_df_metrop$Mean, 3),
        round(summary_stats_df_gibbs$Mean, 3)
    ),
    low_ci = c(
        round(summary_stats_df_metrop$`2.5%.2.5%`, 3),
        round(summary_stats_df_gibbs$`2.5%.2.5%`, 3)
    ),
    hi_ci = c(
        round(summary_stats_df_metrop$`97.5%.97.5%`, 3),
        round(summary_stats_df_gibbs$`97.5%.97.5%`, 3)
    ),
    times = c(
        rep(round(time_metropolis, 2), 2),
        rep(round(time_gibbs, 2), 2)
    ),
    gweke = c(
        round(diags_df_metrop$geweke, 3),
        round(diags_df_gibbs$geweke, 3)
    ),
    ess = c(
        diags_df_metrop$ESS,
        diags_df_gibbs$ESS
    )
)

knitr::kable(comparison_df,
    caption = "Comparison of Diagnostics between Metropolis-Hastings and Gibbs Sampling",
    col.names = c(
        "Method", "Parameter", "Est.", "2.5%", "97.5%",
        "Median Time(s))", "Geweke",
        "ESS"
    )
) %>%
    kableExtra::kable_classic()
```

Table 5: Comparison of Diagnostics between Metropolis-Hastings and Gibbs Sampling

Method	Parameter	Est.	2.5%	97.5%	Median Time(s))	Geweke	ESS
Metropolis	beta1	-0.138	-0.445	0.166	1.74	1.027	8362.623
Metropolis	beta2	1.764	1.323	2.247	1.74	-0.089	6228.419

Gibbs	beta1	-0.137	-0.443	0.169	85.57	-1.906	9800.000
Gibbs	beta2	1.763	1.314	2.244	85.57	-1.679	9800.000

We can see that the comparison table shows us that the two algorithms produce relatively similar results. The Gibbs sampler does take significantly longer, but the autocorrelation (seen in the plots above) is significantly lower, and the ESS is understandably higher. Likely the longer period of time for the Gibbs sampler is due to the fact that the auxilary varible needs to be drawn and since it's from a non-standard distribution that almost certainly adds compute time.