STA 6366, Homework.5

Carson Slater *Baylor University*

STA 6366-Unit 5 Homework

1) CG-Islands: As discussed in class, the CG nucleotide pair shows up quite infrequently. The reason for this is due to methylation. However, some areas of the genome are protected from methylation, which results in CG pairs showing up more frequently (these are called "CG-islands"). We will explore this problem using the first $\sim$1.1 million nucleotides from a human X chromosome (the file is called *chrX_HW5.txt* on canvas).

a. First, determine the probability of observing a CG pair if you randomly picked two sequential nucleotides in this region. Is it more or less than what you would expect if you only knew the percentage of G and C in the region? You can ignore any dependence due to overlap in your calculation and in the below problems.

```
## # A tibble: 1 x 2
##   prob_CG_obs prob_CG_exp
##         <dbl>       <dbl>
## 1      0.0243      0.0591
```

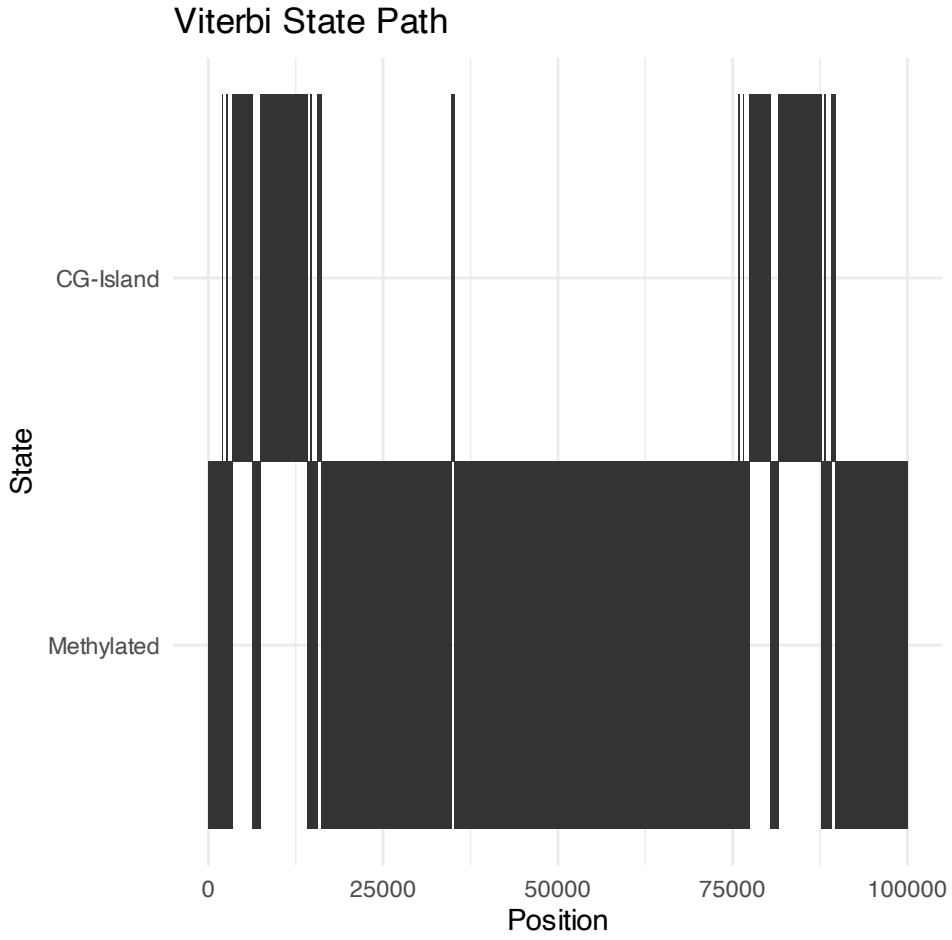
The observed probability of a CG dinucleotide is 0.0243 while the expected probability based on independent C and G frequencies is 0.0591, meaning CG pairs occur at less than half the expected rate. This substantial depletion (observed/expected ratio $\approx$ 0.41) provides strong evidence for CG methylation throughout the genome. The suppression reflects evolutionary deamination of methylated cytosines, which converts CG pairs to TG pairs over time.

b. Assume that CGs occur according to the following model: There are two possible states that exist at any point in the sequence, call them the "methylated" and the "CG-Island" states, respectively. In the methylated state, there is only a 0.5% chance of emitting a CG pair. Alternatively, the CG-Island has a much higher probability of outputting a CG pair at 6.25%. Assume that transitioning from the CG-Island to methylation state is rare (0.1% probability) and transitioning from the methylation to the CG-Island state is rarer still (0.01% probability). Finally, assume there is a 90% chance that the sequence begins in the methylation state. Write a version of the forward algorithm in R that can calculate the log probability of a sequence under this model. Use it to calculate the log-likelihood of the first 100,000 nucleotides of the X chromosome. Do not use any specialized HMM packages for this problem.

```
## [1] -7615.258
```

The forward algorithm computed a log-likelihood of -7615.258 for the first 100,000 nucleotides under the specified HMM with two states (methylated and CG-island).

c. Now, under this model, use the Viterbi algorithm to identify the location of CG-Islands in the first 100,000 nucleotides. Note that this is equivalent to finding the optimal path. In a table or graph, show the location, size, and CG-content (as a proportion) for each CG-Island. Do not use any specialized HMM packages for this problem.



The Viterbi algorithm identified 13 distinct CG-islands ranging from 114 to 6,815 nucleotides in length, with CG proportions between 4.43% and 7.93%. The largest islands span thousands of base pairs (positions 3,489-6,245 and 7,391-14,205), while smaller islands tend to have higher CG content approaching the model's assumed 6.25%. Table 1 shows the details for each CG island.

d. Implement the Baum-Welch algorithm on the entire region (not just the first 100k) and determine the likelihood-maximizing parameters for a 2-state HMM. Make sure that you test it with multiple starting parameters to ensure that you aren't getting caught at local maxima (hint: at the global maximum, the transition matrix will not have an absorbing state, but might have one at certain local maxima). Report the parameter estimates and compare them to those from (b). Note: For the sake of computational efficiency, I would encourage you to use one of the packages we discussed in class for this problem, however you are more than welcome to write this from scratch.

##

Table 1: Viterbi-Identified CG-Islands (First 100,000 Nucleotides)

Start Index	End Index	Length	CG Proportion
1963	2076	114	0.0789
2561	2702	142	0.0704
3489	6245	2757	0.0526
7391	14205	6815	0.0483
14631	14776	146	0.0685
15568	16154	587	0.0443
34836	35226	391	0.0793
75881	75994	114	0.0789
76479	76620	142	0.0775
77407	80403	2997	0.0527
81551	87719	6169	0.0451
88145	88290	146	0.0685
89081	89667	587	0.0477

```
## === ESTIMATED PARAMETERS ===


##
## Initial State Probabilities:


##   P(Methylated) = 0


##   P(CG-Island)  = 1


##
## Transition Matrix:


##   From Methylated:
## to Methylated = 0.9984 ,
## to CG-Island = 0.0016


##   From CG-Island:
## to Methylated = 0.0015 ,
## to CG-Island = 0.9985


##
## Emission Probabilities:


##   State 1: P(CG) = 0.0413
```

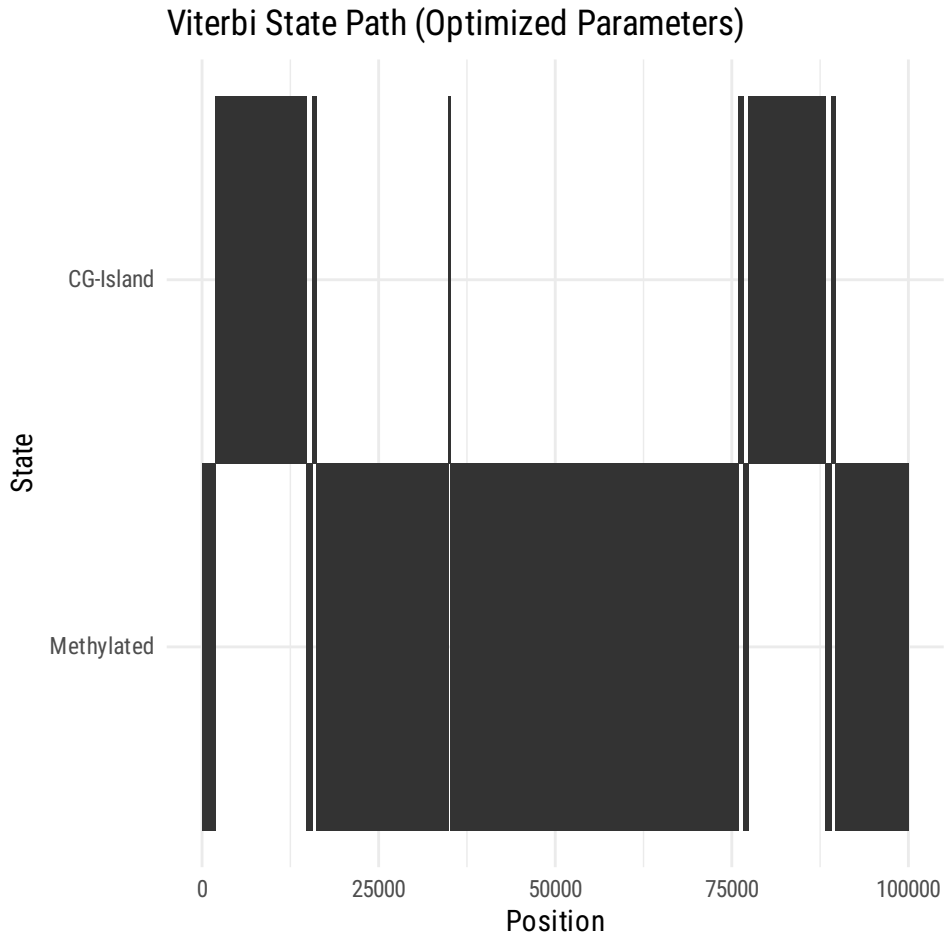
```
##   State 2: P(CG) = 0.0085

##
## === COMPARISON TO PART (b) ===

## Part (b) parameters:

##   Methylated: P(CG) = 0.0050

##   CG-Island:  P(CG) = 0.0625
```

The Baum-Welch algorithm fitted the HMM to the full sequence and converged to a log-likelihood of -124,676.7, with estimated emission probabilities of P(CG) = 0.0412 for State 1 and P(CG) = 0.0085 for State 2. The fitted parameters differ substantially from initial assumptions, showing that the methylated background has even lower CG content (0.85% vs. initial 0.5%) while CG-islands have lower enrichment than assumed (4.12% vs. initial 6.25%). The transition probabilities became nearly symmetric (approximately 0.15-0.16% in either direction), indicating more frequent state switching than initially assumed.

e. Re-run your analysis from (c), this time with the optimized parameters. How did the properties of the identified CG-Islands change?

Table 2: Viterbi-Identified CG-Islands (Optimized Parameters)

Start Index	End Index	Length	CG Proportion
1963	14776	12814	0.0418
15568	16154	587	0.0443
34836	35226	391	0.0793
75881	76620	740	0.0338
77407	88290	10884	0.0426
89081	89667	587	0.0477



Using the optimized parameters, the Viterbi algorithm identified only 6 CG-islands compared to 13 with the original parameters, and these islands are substantially larger (ranging from 391 to 12,814 nucleotides). The largest island now spans positions 1,963 to 14,776, consolidating several smaller islands from the original analysis into one continuous region. The CG proportions in these optimized islands (3.38% to 7.93%) are more moderate overall, reflecting the fitted model's lower CG-island emission probability of 4.12%. Table 2 shows the details for each CG island found using the optimized parameter estimates.

f. If you compare the log-likelihoods of the optimal paths using the initial parameters and the optimized parameters, you will notice that the log likelihood is slightly worse when using the fitted parameters. Explain how this can be true if the fitted parameters were chosen to maximize

the likelihood.

This result occurs because the algorithms optimize two different objective functions. The Baum-Welch algorithm maximizes the overall marginal likelihood of the observations by summing the probabilities across *all* possible hidden state paths, ensuring the model fits the data better as a whole. In contrast, the Viterbi algorithm maximizes the joint likelihood of the data for only a *single* specific path. When Baum-Welch optimizes the parameters, it typically relaxes rigid initial assumptions, causing the probability mass to spread out across a wider distribution of plausible sequences rather than being heavily concentrated into just one. Consequently, while the total likelihood of the model increases, the specific fraction of probability belonging to the single "best" Viterbi path shrinks.

Appendix

```
knitr::opts_chunk$set(
  dev = "cairo_pdf",
  fig.width = 5,
  fig.height = 5,
  fig.align = 'center',
  echo = FALSE,
  message = FALSE,
  warning = FALSE,
  error = FALSE,
  results = 'markup'
)

# Load required libraries
library("tidyverse")
library("patchwork")
library("glue")
library("scales", warn.conflicts = FALSE)
library("extrafont")
library("tinytex")
library("knitr")
library("tidyr")
library("latex2exp")
library("MASS")
library("depmixS4")
library("dplyr")
library("purrr")
library("kableExtra")

theme_set(theme_minimal(base_family = "Roboto Condensed"))

conflicted::conflicts_prefer(
  readr::col_factor(),
  purrr::discard(),
  dplyr::lag(),
  readr::parse_date(),
  kableExtra::group_rows(),
  dplyr::select
)
chrX <- read_file(
  "/Users/carson/Documents/Baylor_Work/2026_Spring/Statistical_Bioinformatics/
)

# ------------ PART a ------------
# Calculate the observed and expected probabilities
prob_results <- tibble(
```

```
  sequence = chrX
) |>
  mutate(
    # Total nucleotides and total sequential pairs
    n_nuc = nchar(sequence),
    total_pairs = n_nuc - 1,

    # Counts of the individual nucleotides and the CG pair
    n_C = str_count(sequence, "C"),
    n_G = str_count(sequence, "G"),
    n_CG = str_count(sequence, "CG"),

    # Calculate probabilities
    prob_CG_obs = n_CG / total_pairs,
    prob_CG_exp = (n_C / n_nuc) * (n_G / n_nuc)
  ) |>
  select(prob_CG_obs, prob_CG_exp)

# View the results
prob_results

# ------------ PART b ------------

# Log-sum-exp trick for numerical stability
log_sum_exp <- function(x) {
  m <- max(x)
  m + log(sum(exp(x - m)))
}

forward_log <- function(y, pi, A, B) {
  T <- length(y)
  N <- length(pi)

  log_alpha <- matrix(-Inf, nrow = T, ncol = N)

  # Initialization
  log_alpha[1, ] <- log(pi) + log(B[, y[1]])

  # Recursion
  for (t in 2:T) {
    for (j in 1:N) {
      log_alpha[t, j] <- log(B[j, y[t]]) +
        log_sum_exp(log_alpha[t - 1, ] + log(A[, j]))
    }
  }

  # Termination
  log_sum_exp(log_alpha[T, ])
```

```r
}
pi <- c(0.9, 0.1)

# Transition Matrix
A <- matrix(
  c(
    0.9999,
    0.0001,
    0.0010,
    0.9990
  ),
  nrow = 2,
  byrow = TRUE
)

#
B <- matrix(
  c(
    0.9950,
    0.0050,
    0.9375,
    0.0625
  ),
  nrow = 2,
  byrow = TRUE
)

# Take first 100,000 nucleotides
seq_sub <- substr(chrX, 1, 100000)

# Create overlapping dinucleotides
pairs <- substring(seq_sub, 1:(nchar(seq_sub) - 1), 2:nchar(seq_sub))

# Encode observations:
# 1 = not CG, 2 = CG
y <- ifelse(pairs == "CG", 2, 1)

(log_lik <- forward_log(y, pi, A, B))

# ------------ PART c ------------

# Log-sum not needed here (we use max instead)

viterbi_log <- function(y, pi, A, B) {
  T <- length(y)
  N <- length(pi)

  log_delta <- matrix(-Inf, nrow = T, ncol = N)
```

```
  psi <- matrix(0, nrow = T, ncol = N)

  # Initialization
  log_delta[1, ] <- log(pi) + log(B[, y[1]])

  # Recursion
  for (t in 2:T) {
    for (j in 1:N) {
      vals <- log_delta[t - 1, ] + log(A[, j])
      psi[t, j] <- which.max(vals)
      log_delta[t, j] <- max(vals) + log(B[j, y[t]])
    }
  }

  # Backtracking
  path <- numeric(T)
  path[T] <- which.max(log_delta[T, ])

  for (t in (T - 1):1) {
    path[t] <- psi[t + 1, path[t + 1]]
  }

  list(path = path, log_prob = max(log_delta[T, ]))
}

vit <- viterbi_log(y, pi, A, B)
state_path <- vit$path


island_df <- tibble(
  pos = 1:length(state_path),
  state = state_path
) |>
  mutate(
    island = cumsum(c(1, diff(state) != 0))
  ) |>
  group_by(island) |>
  summarise(
    state = first(state),
    start = min(pos),
    end = max(pos),
    length = n(),
    .groups = "drop"
  ) |>
  filter(state == 2)


island_df <- island_df |>
```

```
  rowwise() |>
  mutate(
    cg_count = sum(y[start:end] == 2),
    cg_prop = cg_count / length
  ) |>
  ungroup()
state_df <- tibble(
  pos = 1:length(state_path),
  state = factor(state_path, labels = c("Methylated", "CG-Island"))
)

ggplot(state_df, aes(x = pos, y = state)) +
  geom_tile() +
  labs(
    title = "Viterbi State Path",
    x = "Position",
    y = "State"
  ) +
  theme_minimal()

island_table <- island_df |>
  transmute(
    `Start Index` = start,
    `End Index` = end,
    `Length` = length,
    `CG Proportion` = round(cg_prop, 4)
  ) |>
  arrange(`Start Index`)

kable(
  island_table,
  caption = "Viterbi-Identified CG-Islands (First 100,000 Nucleotides)",
  align = "c",
  format = "latex"
) |>
  kable_styling(
    full_width = FALSE,
    position = "center"
  )

# ------------ PART d ------------

# Create observation sequence
chars <- str_split(chrX, "")[[1]]
pairs <- str_c(chars[-length(chars)], chars[-1])

y_full <- factor(
  ifelse(pairs == "CG", "CG", "notCG"),
```

```
  levels = c("notCG", "CG")
)

hmm_data_full <- data.frame(obs = y_full)

# Function to set parameters manually with better initialization
set_hmm_params <- function(pi_vec, A_mat, B_mat) {
  # Create model
  mod <- depmix(
    obs ~ 1,
    data = hmm_data_full,
    nstates = 2,
    family = multinomial("identity"),
    ntimes = nrow(hmm_data_full)
  )

  # Parameter order for depmixS4:
  # initial probs (2), transition matrix row-wise (4), emission probs for eac
  pars <- c(
    pi_vec, # initial state probabilities
    A_mat[1, ], # transition from state 1
    A_mat[2, ], # transition from state 2
    B_mat[1, ], # emissions from state 1 (notCG, CG)
    B_mat[2, ] # emissions from state 2 (notCG, CG)
  )

  setpars(mod, pars)
}

# Fit with multiple good starting values
fit_multiple_starts <- function(n_starts = 10) {
  results <- list()

  # Starting configurations
  starts <- list(
    # Configuration 1: from problem statement
    list(
      pi = c(0.9, 0.1),
      A = matrix(c(0.9999, 0.0001, 0.001, 0.999), 2, 2, byrow = TRUE),
      B = matrix(c(0.995, 0.005, 0.9375, 0.0625), 2, 2, byrow = TRUE)
    ),
    # Configuration 2: more CG in state 2
    list(
      pi = c(0.95, 0.05),
      A = matrix(c(0.9995, 0.0005, 0.005, 0.995), 2, 2, byrow = TRUE),
      B = matrix(c(0.99, 0.01, 0.90, 0.10), 2, 2, byrow = TRUE)
    ),
    # Configuration 3: even stronger difference
```

```
  list(
    pi = c(0.85, 0.15),
    A = matrix(c(0.999, 0.001, 0.002, 0.998), 2, 2, byrow = TRUE),
    B = matrix(c(0.998, 0.002, 0.85, 0.15), 2, 2, byrow = TRUE)
  )
)

# Add random starts
for (i in 4:n_starts) {
  pi_rand <- c(runif(1, 0.7, 0.95), runif(1, 0.05, 0.3))
  pi_rand <- pi_rand / sum(pi_rand)

  # Keep transition probabilities strongly diagonal
  diag1 <- runif(1, 0.995, 0.9999)
  diag2 <- runif(1, 0.99, 0.999)

  A_rand <- matrix(
    c(
      diag1,
      1 - diag1,
      1 - diag2,
      diag2
    ),
    2,
    2,
    byrow = TRUE
  )

  # Make state 1 have low CG, state 2 have high CG
  cg1 <- runif(1, 0.001, 0.01)
  cg2 <- runif(1, 0.03, 0.15)

  B_rand <- matrix(
    c(
      1 - cg1,
      cg1,
      1 - cg2,
      cg2
    ),
    2,
    2,
    byrow = TRUE
  )

  starts[[i]] <- list(pi = pi_rand, A = A_rand, B = B_rand)
}

for (i in 1:n_starts) {
```

```
cat("Fitting model", i, "of", n_starts, "\n")

mod_init <- set_hmm_params(
  starts[[i]]$pi,
  starts[[i]]$A,
  starts[[i]]$B
)

fit_result <- tryCatch(
  {
    fit(
      mod_init,
      verbose = FALSE,
      emcontrol = em.control(maxit = 1000, tol = 1e-8)
    )
  },
  error = function(e) {
    cat("Error:", e$message, "\n")
    NULL
  }
)

if (!is.null(fit_result)) {
  ll <- as.numeric(logLik(fit_result))

  # Check if states are degenerate (too similar)
  pars <- getpars(fit_result)
  B1 <- pars[7:8] # State 1 emissions
  B2 <- pars[9:10] # State 2 emissions

  # Only keep if states are sufficiently different
  if (abs(B1[2] - B2[2]) > 0.01) {
    # CG probabilities differ by >1%
    results[[i]] <- list(
      logLik = ll,
      model = fit_result,
      params = pars
    )
    cat(
      "  -> Log-likelihood:",
      ll,
      "| CG probs:",
      round(B1[2], 4),
      "vs",
      round(B2[2], 4),
      "\n"
    )
  } else {
```

```
        cat("  -> Degenerate solution, skipping\n")
      }
    }
  }

  # Filter and find best
  results <- results[!sapply(results, is.null)]

  if (length(results) == 0) {
    stop("All fitting attempts failed or were degenerate")
  }

  best_idx <- which.max(sapply(results, function(x) x$logLik))
  cat("\n=== Best model: iteration", best_idx, "===\n")
  cat("Log-likelihood:", results[[best_idx]]$logLik, "\n\n")

  results[[best_idx]]$model
}

# Fit the model
best_model <- fit_multiple_starts(n_starts = 10)

# Display results
summary(best_model)

# Extract and format parameters nicely
pars <- getpars(best_model)
cat("\n=== ESTIMATED PARAMETERS ===\n")
cat("\nInitial State Probabilities:\n")
cat("  P(Methylated) =", round(pars[1], 4), "\n")
cat("  P(CG-Island)  =", round(pars[2], 4), "\n")

cat("\nTransition Matrix:\n")
cat(
  "  From Methylated:\nto Methylated =",
  round(pars[3], 4),
  ",\nto CG-Island =",
  round(pars[4], 4),
  "\n"
)
cat(
  "  From CG-Island:\nto Methylated =",
  round(pars[5], 4),
  ",\nto CG-Island =",
  round(pars[6], 4),
  "\n"
)
```

```r
cat("\nEmission Probabilities:\n")
cat("  State 1: P(CG) =", round(pars[8], 4), "\n")
cat("  State 2: P(CG) =", round(pars[10], 4), "\n")

cat("\n=== COMPARISON TO PART (b) ===\n")
cat("Part (b) parameters:\n")
cat("  Methylated: P(CG) = 0.0050\n")
cat("  CG-Island:  P(CG) = 0.0625\n")

# ------------ PART e ------------
# 1. Define the optimized parameters from the Baum-Welch (depmixS4) output
pi_opt <- c(1.0, 0.0)

A_opt <- matrix(
  c(
    0.9985,
    0.0015,
    0.0016,
    0.9984
  ),
  nrow = 2,
  byrow = TRUE
)

# Remember column 1 is notCG, column 2 is CG based on your factor setup
B_opt <- matrix(
  c(
    1 - 0.0085,
    0.0085,
    1 - 0.0413,
    0.0413
  ),
  nrow = 2,
  byrow = TRUE
)

# 2. Run the Viterbi algorithm using the custom function from part (b/c)
vit_opt <- viterbi_log(y, pi_opt, A_opt, B_opt)
state_path_opt <- vit_opt$path

# 3. Recycle the data wrangling to identify the CG-Islands
island_df_opt <- tibble(
  pos = 1:length(state_path_opt),
  state = state_path_opt
) |>
  mutate(
    island = cumsum(c(1, diff(state) != 0))
  ) |>
```

```
  group_by(island) |>
  summarise(
    state = first(state),
    start = min(pos),
    end = max(pos),
    length = n(),
    .groups = "drop"
  ) |>
  filter(state == 2) |>
  rowwise() |>
  mutate(
    cg_count = sum(y[start:end] == 2),
    cg_prop = cg_count / length
  ) |>
  ungroup()

# 4. Plot the optimized Viterbi state path
state_df_opt <- tibble(
  pos = 1:length(state_path_opt),
  state = factor(state_path_opt, labels = c("Methylated", "CG-Island"))
)

ggplot(state_df_opt, aes(x = pos, y = state)) +
  geom_tile() +
  labs(
    title = "Viterbi State Path (Optimized Parameters)",
    x = "Position",
    y = "State"
  )

# 5. Generate the updated table for comparison
island_table_opt <- island_df_opt |>
  transmute(
    `Start Index` = start,
    `End Index` = end,
    `Length` = length,
    `CG Proportion` = round(cg_prop, 4)
  ) |>
  arrange(`Start Index`)

kable(
  island_table_opt,
  caption = "Viterbi-Identified CG-Islands (Optimized Parameters)",
  align = "c",
  format = "latex"
) |>
  kable_styling(
    full_width = FALSE,
```

```
  position = "center"
)
```