Student Number: [redacted] Name: Bryan Hoang

1. (6 points) **Answer:**

$\mathcal{V}(d, \beta)$	$\beta = 0.4$	$\beta = 0.8$	$\beta = 1.2$
$d = 60$	35.261	40.701	45.341
$d = 120$	71.188	82.062	91.594

Code:

```r
# Shows a progress bar in the R terminal.
if (!require(progress)) {
  install.packages("progress")
}

library(progress)

#' m
#'
#' Computes the "smoothness" of a 1-D vector.
#'
#' @param x The vector to compute the smoothness of.
#'
#' @return the number of adjacent componenents of the vector that have the same
#' value.
#'
#' @examples
#' x <- c(0, 1, 1, 0, 1, 0, 0)
#' m(x) # returns 2, from the consecutive 1,1 and 0,0 in the vector `x`
m <- function(x) {
  # The dimension of the vector.
  d <- length(x)

  # Vector representing the indicator on the equality of ajacent elements in
  # `x`.
  smoothness_indicator <- rep(0, d - 1)

  for (i in seq_len(d - 1)) {
    smoothness_indicator[i] <- x[i] == x[i + 1]
  }

  return(sum(smoothness_indicator))
}

#' 𝒱
#'
#' Computes a value using the Markov Chain Monte Carlo method, with a Markov
#' Chain generated using the Metropolis-Hastings (M-H) algorithm.
#'
#' @param d The dimension of sample space.
#'
#' @param β Parameter that changes the distribution of mass coming from
#' "smoothness".
#'
```

```r
#' @param burnin_value Optionally specifies the number of iterations to discard
#' before sampling. Defaults to 3000.
#'
#' @return $\mathcal{V}(d, \beta)$
#'
#' @examples
#' print(v(10, 1.2)) # Prints approximately 6.916
#' print(v(12, 0.8)) # Prints approximately 7.590
v <- function(d, beta, burnin_value = NULL) {
  # Default value based on the assignment's recommendations.
  if (is.null(burnin_value)) {
    burnin_value <- 3000
  }

  summation <- 0
  iteration_count <- 2 * burnin_value
  pb <- progress_bar$new(
    total = iteration_count,
    format = sprintf("Computing V(%d, %.1f) - [:bar] :percent", d, beta)
  )

  # Generate X_1 as a vector of 0's and 1's randomly.
  x_current <- sample(c(0, 1), d, replace = TRUE)

  # Generate the rest of the Markov chain.
  for (n in 1:iteration_count) {
    pb$tick()
    # Generate the next `d` elements of the Markov chain.
    for (i in seq_len(d)) {
      # Generate the next proposal state by randomly flipping one of the
      # current vector's components.
      x_next <- x_current
      component_to_flip <- sample(seq_len(d), 1)
      x_next[component_to_flip] <- 1 - x_next[component_to_flip]

      # Calculate the "smoothness" of the current vector and the proposal
      # vector (with the flipped component).
      m_0 <- m(x_current)
      m_1 <- m(x_next)

      # Calculate the acceptance probability.
      #
      # NOTE: `min`` is unecessary here for the implementation, but helps with
      # readability when referencing the lecture notes.
      p_accept <- min(1, exp(beta * (m_1 - m_0)))

      # Decide if we want to accept the proposal. If not, keep the current
      # state for the next proposal.
      if (runif(1) <= p_accept) {
        x_current[component_to_flip] <- 1 - x_current[component_to_flip]
      }
    }

```

Student Number: [redacted] Name: Bryan Hoang

```
    # After enough burn-in iterations, begin computing V(d,beta).
    if (n > burnin_value) {
      summation <- summation + m(x_current)
    }
  }

  return(summation / burnin_value)
}

main <- function() {
  # Combinations of parameters from assignment.
  dimensions <- c(60, 120)
  betas <- c(0.4, 0.8, 1.2)

  for (i in seq_len(length(dimensions))) {
    for (j in seq_len(length(betas))) {
      print(
        sprintf(
          "V(%d, %.1f) = %.3f",
          dimensions[i],
          betas[j],
          v(dimensions[i], betas[j])
        )
      )
    }
  }
}

main()
```