load("../../datasets/ICHS/ICHS.RData")

# using glmer to validate my method

true\_res = lme4::glmer(infection ~ age + xerop +(1|id) ,data = ichs, family = binomial, nAGQ = 3)

sample.dmtx = model.matrix(infection ~ age + xerop ,data = ichs)

# Function to calculate the Hermite polynomial coefficients using the recurrence relation

hermite\_poly\_coefs <- function(n) {

if (n == 0) {

return(c(1))

} else if (n == 1) {

return(c(2, 0))

} else {

H\_n\_minus\_two <- c(1)

H\_n\_minus\_one <- c(2, 0)

for (i in 2:n) {

H\_n <- c(2 \* H\_n\_minus\_one, 0) - c(0, 0, 2 \* (i - 1) \* H\_n\_minus\_two)

H\_n\_minus\_two <- H\_n\_minus\_one

H\_n\_minus\_one <- H\_n

}

return(H\_n)

}

}

# Function to calculate the roots of the Hermite polynomial

hermite\_roots <- function(n) {

coefs <- hermite\_poly\_coefs(n)

# Reverse the coefficients for polyroot function

coefs <- rev(coefs)

# Find the roots of the polynomial

roots <- polyroot(coefs)

# Return only the real parts of the roots

return(Re(roots))

}

# Function to calculate the weights for the Gauss-Hermite quadrature

gauss\_hermite\_weights <- function(m) {

# Calculate the roots of the Hermite polynomial of degree m

roots <- hermite\_roots(m)

weights <- numeric(m)

for (j in 1:m) {

H\_m\_derivative <- hermite\_poly\_coefs(m - 1)

H\_m\_value <- sum(rev(H\_m\_derivative) \* roots[j]^(0:(m-1)))

weights[j] <- 2^(m-1) \* factorial(m) \* sqrt(pi) / (m^2 \* H\_m\_value^2)

}

return(weights)

}

# wrapper

gauss\_hermite\_point <- function(m){

x = hermite\_roots(m)

w = gauss\_hermite\_weights(m)

return(list(x = x, w = w))

}

# using fastGHQuad package to check the validity of the weights

# fastGHQuad::gaussHermiteData(5)

# gauss\_hermite\_point(5)

# two outputs are the same

# logit function

logit <- function(linear\_predictor){

p = 1/(1 + exp(linear\_predictor))

return(p)

}

# conditional log\_likelihood given random effect, fixed effect for a given subject

conditional\_log\_likelihood <- function(random\_effect, fixed\_effects, response, X\_matrix) {

linear\_predictor <- X\_matrix %\*% fixed\_effects + random\_effect

p <- logit(linear\_predictor)

log\_lik <- dbinom(response, size = rep(1, length(response)), prob = p, log = T)

return(sum(log\_lik))

}

# Gauss-Hermite approximation for the contribution for likelihood of each subject, given estimate of variance component sigma\_gamma

gh\_approx <- function(fixed\_effects, response, X\_matrix, gh\_points, sigma\_gamma){

integral <- 0

gh\_x = gh\_points$x

gh\_w = gh\_points$w

for (i in seq\_along(gh\_x)) {

# Scale the node by the random effect standard deviation

node\_scaled <- gh\_x[i] \* sqrt(2) \* sigma\_gamma

# Calculate the log-likelihood contribution for the current node and weight

ll\_contrib <- conditional\_log\_likelihood(node\_scaled, fixed\_effects, response, X\_matrix)

# Update the integral approximation

integral <- integral + exp(ll\_contrib) \* gh\_w[i]

}

integral\_scaled <- integral/ sqrt(pi)

return(integral\_scaled)

}

# log-likelihood for the whole data

# parameters to be optimized

## fixed\_effects : beta

## sigma\_gamma : variance component

calculate\_log\_likelihood <- function(data, idVar, responseVar, fixedVars, fixed\_effects, sigma\_gamma, gh\_points) {

unique\_ids <- unique(data[[idVar]])

total\_log\_likelihood <- 0

# Ensure fixed\_effects is a numeric vector

fixed\_effects <- as.numeric(fixed\_effects)

# Pre-calculate the model matrix for fixed effects

formula <- as.formula(paste(responseVar, "~", paste(fixedVars, collapse = "+")))

X\_matrix <- model.matrix(formula, data)

# Loop over each subject

for (subj\_id in unique\_ids) {

# Subset the data and model matrix for the subject

subj\_indices <- which(data[[idVar]] == subj\_id)

subject\_data <- data[subj\_indices, ]

subj\_X\_matrix <- X\_matrix[subj\_indices, , drop = FALSE]

#print(subj\_X\_matrix)

#print(fixed\_effects)

response <- subject\_data[[responseVar]]

# Calculate subject-specific likelihood contribution using gh\_approx

subject\_log\_likelihood <- gh\_approx(fixed\_effects, response, subj\_X\_matrix, gh\_points, sigma\_gamma)

# Sum the log-likelihood contributions

total\_log\_likelihood <- total\_log\_likelihood + log(subject\_log\_likelihood) # Log is taken here as gh\_approx returns the scaled integral, not the log likelihood

}

return(total\_log\_likelihood)

}

# Optimizer for maximize likelihood

optimize\_parameters <- function(data, idVar, responseVar, fixedVars, gh\_points) {

# Initial guesses for parameters

initial\_fixed\_effects <- rep(0, length(fixedVars)+1)

initial\_sigma\_gamma <- 1

print(initial\_fixed\_effects)

# Objective function to minimize (negative log-likelihood)

objective\_func <- function(params) {

#print(params)

#print(fixedVars)

fixed\_effects <- params[1:(length(fixedVars)+1)]

#print(fixed\_effects)

sigma\_gamma <- params[(length(fixedVars) + 2)]

-calculate\_log\_likelihood(data, idVar, responseVar, fixedVars, fixed\_effects, sigma\_gamma, gh\_points)

}

# Combine initial guesses into a single vector

initial\_params <- c(initial\_fixed\_effects, initial\_sigma\_gamma)

#print(length(initial\_fixed\_effects))

# Use optim to maximize the log-likelihood (minimize its negative)

optim\_results <- optim(initial\_params, objective\_func, method = "BFGS")

return(optim\_results)

}

# Formula wrapper

parse\_formula <- function(formula) {

# Extract the response variable

responseVar <- as.character(formula[[2]])

# Extract the fixed effects variables

fixed\_effects <- attr(terms(formula), "term.labels")

fixedVar <- fixed\_effects[!grepl("\\|", fixed\_effects)]

# Identify the random effect and ID variable

random\_effects <- fixed\_effects[grepl("\\|", fixed\_effects)]

if (length(random\_effects) > 0) {

randomVar <- stringr::str\_trim(gsub("(.\*)\\|.\*", "\\1", random\_effects))

idVar <- stringr::str\_trim(gsub(".\*\\|(.\*)", "\\1", random\_effects))

} else {

randomVar <- NA

idVar <- NA

}

return(list(responseVar = responseVar, fixedVar = fixedVar, randomVar = randomVar, idVar = idVar))

}

# main function

fit\_glmm <- function(formula, data, nAGQ){

modelVars = parse\_formula(formula)

response\_var = modelVars$responseVar

fixed\_vars = c(modelVars$fixedVar)

random\_var = as.numeric(modelVars$randomVar) # only accepts random intercepts model

id\_var = modelVars$idVar

subject\_ids <- unique(data$id\_var)

random\_effects <- rep(0, length(subject\_ids))

gh\_points = gauss\_hermite\_point(nAGQ)

# minimizing the negative log-likelihood, need to put a minus sign in front of all estimates

optim\_res = optimize\_parameters(data, id\_var, response\_var, fixed\_vars, gh\_points)

return(optim\_res)

}

fit\_glmm(infection ~ age + xerop +(1|id) ,data = ichs, nAGQ = 3)