Sixth Week Report & Failture Management

Summer 2024

In this report, I want to deal with several problems for the current maximum likelihood approach. Now, the complexity may lead to the non-convexity of the likelihood function. The high-dimensional parameter space may also lead to the slow convergence of the optimization algorithm. The most important thing is that the aghQuad function is still a blackbox, which needs to be analyzed.

The problem

(1) The instability of two optim method:

It can be solved by adding more iterations.

(2) The aghQuad function is still a blackbox, which needs to be analyzed.

(3) We need to accelerate the current functions.

```
library(Rcpp)
library(ggplot2)
library(fastGHQuad)
library(optimx)
```

The Mathematical Background of Mu_hat and Sigma_hat

The study of AghQuad function unfortunately tells us that the mu_hat and sigma_hat are not constants.

$$\mu_{\text{hat},i} = \arg\max_{v} \left[\log \left(\prod_{a=1}^{A} \left[\Phi(\tau_a - X_i'\beta - v)^{1-y_{ia}} \cdot (1 - \Phi(\tau_a - X_i'\beta - v))^{y_{ia}} \right] \right) - \frac{v^2}{2\sigma_v^2} \right]$$

$$\sigma_{\text{hat},i} = \sqrt{ - \left(\frac{\partial^2}{\partial v^2} \left[\log \left(\prod_{a=1}^{A} \left[\Phi(\tau_a - X_i'\beta - v)^{1-y_{ia}} \cdot (1 - \Phi(\tau_a - X_i'\beta - v))^{y_{ia}} \right] \right) - \frac{v^2}{2\sigma_v^2} \right] \right)^{-1} \Bigg| v = \mu\text{hat}, i }$$

In brief, we need to give up the Aghquad and use the ghQuad function, which is another function in fastGHQuad package. This function provides the standard Gauss-Hermite distribution.

Transfrom the Gauss-Hermite Data Point

In the first step, we need to transform the Gauss-Hermite data point. The transformation is as follows:

$$x^* = x \cdot \sqrt{2}, \quad w^* = w/\sqrt{\pi}.$$

This is achieved by the following codes

```
rule <- gaussHermiteData(100)
rule$x <- rule$x * sqrt(2)
rule$w <- rule$w / sqrt(pi)
rule_trans <- rule
#
```

Data Generation

The Optimization Function

In the optimization part, we first need to write the function to the form

$$g'(x) = \int_{-\infty}^{\infty} e^{-x^2} f(x)dx \approx \sum_{i=1}^{M} w_i^* f(x_i^*)$$

But our function is

$$g(v_i) = \int_{-\infty}^{+\infty} \frac{e^{-v_i^2/2\sigma_v^2}}{\sqrt{2\pi\sigma_v^2}} \left\{ \prod_{a=1}^{A} \left[d_i^a \cdot \Phi\left(-\tau^a + X_i'\beta + v_i\right) \right.\right.$$
$$\left.\left. + \left(1 - d_i^a\right) \cdot \Phi\left(\tau^a - X_i'\beta - v_i\right)\right]\right\} dv_i$$

The corresponding r function writes as

```r
library(fastGHQuad)

g <- function(v_i, X_i, beta, tau, d_i, sigma_v) {
  A <- length(d_i)

  #normal_density <- exp(-v_i^2 / (2 * sigma_v^2)) / sqrt(2 * pi * sigma_v^2)

  X_i_beta <- as.vector(X_i %*% beta)

  product_term <- 1

  for (a in 1:A) {
    phi_term <- d_i[a] * pnorm(-tau[a] + X_i_beta + v_i) +
                (1 - d_i[a]) * pnorm(tau[a] - X_i_beta - v_i)
    product_term <- product_term * phi_term
  }

  return( product_term)
}
```

To transform this function, we need to use the following steps:

First, let $x_i = v_i/(\sigma_v\sqrt{2})$, then we in turn have $v_i = x_i \cdot \sigma_v\sqrt{2}$. In this case, we can have

$$g'(x_i) = \int_{-\infty}^{+\infty} e^{-x_i^2} \left\{ \prod_{a=1}^{A} \left[d_i^a \cdot \Phi\left(-\tau^a + X_i'\beta + x_i \cdot \sigma_v\sqrt{2}\right) \; + \left(1 - d_i^a\right) \cdot \Phi\left(\tau^a - X_i'\beta - x_i \cdot \sigma_v\sqrt{2}\right)\right]\right\} dx$$

With this function, we can design a g_prime function in terms of

```r
g_prime <- function(x_i, X_i, beta, tau, d_i, sigma_v) {
  A <- length(d_i)

  # Include e^(-x_i^2) as a backup
  # exp_term <- exp(-x_i^2)

  X_i_beta <- as.vector(X_i %*% beta)

  product_term <- 1

  for (a in 1:A) {
    phi_term <- d_i[a] * pnorm(-tau[a] + X_i_beta + x_i * sigma_v * sqrt(2)) +
```

```
                (1 - d_i[a]) * pnorm(tau[a] - X_i_beta - x_i * sigma_v * sqrt(2))
    product_term <- product_term * phi_term
  }

  return(product_term)
}
```

Test the log-likelihood estimation

```
# Here is the original log-likelihood
compute_log_prob <- function(X_i, d_i, beta, tau, sigma_v, rule) {
  prob <- ghQuad(
    f = function(v_i) g(v_i, X_i = X_i, beta = beta, tau = tau, d_i = d_i, sigma_v = sigma_v),
    rule = rule
  )
  return(log(prob))
}
```

```
# Here is the original log-likelihood
compute_log_prob_trans <- function(X_i, d_i, beta, tau, sigma_v, rule) {
  prob <- ghQuad(
    f = function(x_i) g(x_i*sigma_v, X_i = X_i, beta = beta, tau = tau, d_i = d_i, sigma_v = sigma_v),
    rule = rule
  )
  return(log(prob))
}
```

```
log_likelihood <- function(params, X, y, rule) {
  n_beta <- ncol(X)
  # print(n_beta)
  n_tau <- ncol(y)
  # print(n_tau)

  beta <- params[1:n_beta]
  # print(beta)
  tau <- params[(n_beta+1):(n_beta+n_tau)]
  # The Sigma v

  sigma_v <- exp(params[n_beta+n_tau+1])  # Ensure sigma_v is positive
  # print(sigma_v)

  log_probs <- sapply(1:nrow(X), function(i) {
    compute_log_prob(X[i,], y[i,], beta, tau, sigma_v, original_rule)
  })
  # print(log_probs)

  return(-sum(log_probs))  # Return negative log-likelihood for minimization
}
```

```
log_likelihood_trans <- function(params, X, y, rule) {
  n_beta <- ncol(X)
  # print(n_beta)
  n_tau <- ncol(y)
  # print(n_tau)
```

```r
  beta <- params[1:n_beta]
  # print(beta)
  tau <- params[(n_beta+1):(n_beta+n_tau)]
  # The Sigma v

  sigma_v <- exp(params[n_beta+n_tau+1])  # Ensure sigma_v is positive
  # print(sigma_v)

  log_probs <- sapply(1:nrow(X), function(i) {
    compute_log_prob_trans(X[i,], y[i,], beta, tau, sigma_v, rule)
  })
  # print(log_probs)

  return(-sum(log_probs))  # Return negative log-likelihood for minimization
}
```

```r
# This create a result with better format
set.seed(42)
# Generate the true information about the data
n_clusters <- 200
n_per_cluster <- 1
beta_true <- c(3,-4,5,-6,2,5)  # Added more beta parameters
sigma_true <- 1
tau_true <- c(3,-2,3)  # Added more tau parameters

data <- generate_data(n_clusters, n_per_cluster, beta_true, sigma_true, tau_true)

# Set up Gauss-Hermite quadrature rule
original_rule <- gaussHermiteData(100)

# Prepare data for optimization
X <- cbind(1, as.matrix(data[, grep("^X", names(data))]))
y <- as.matrix(data[, grep("^y_", names(data))])

# Initial parameter values
initial_beta <- rep(0, ncol(X))
initial_tau <- c(1,1,1)
initial_sigma <- 0.5
initial_params <- c(initial_beta, initial_tau, initial_sigma)

options(scipen=999)
```

```r
# Perform optimization
result <- optimr(initial_params,log_likelihood, X = X, y = y, rule = original_rule,
                 method = "BFGS", control = list(maxit = 1e9))
# log-back the result
result$par
```

```
##  [1]  2.341866 -5.957080  7.817950 -9.262627  2.746820  7.750833  2.933758
##  [8] -5.209367  2.933754  0.500000
## attr(,"status")
##  [1] " " " " " " " " " " " " " " " " " " " "
```

```r
# Perform optimization
result <- optimr(initial_params,log_likelihood_trans, X = X, y = y, rule = rule_trans,
                 method = "BFGS", control = list(maxit = 1e9))
# log-back the result
result$par
```

```
##  [1]  1.981396 -4.593697  6.016889 -7.166765  2.097550  5.975883  2.432440
##  [8] -3.846262  2.432426 -5.926667
## attr(,"status")
##  [1] " " " " " " " " " " " " " " " " " " " "
```