Asymmetric Model

Siqi

2025-03-06

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
# Load necessary libraries
set.seed(123)
library(ggplot2)
library(splines)
library(minpack.lm)
#install.packages("pracma")
library(pracma)
#install.packages("nls2")
library(nls2)
```

```
## Loading required package: proto
```

```
library(nlstools)
```

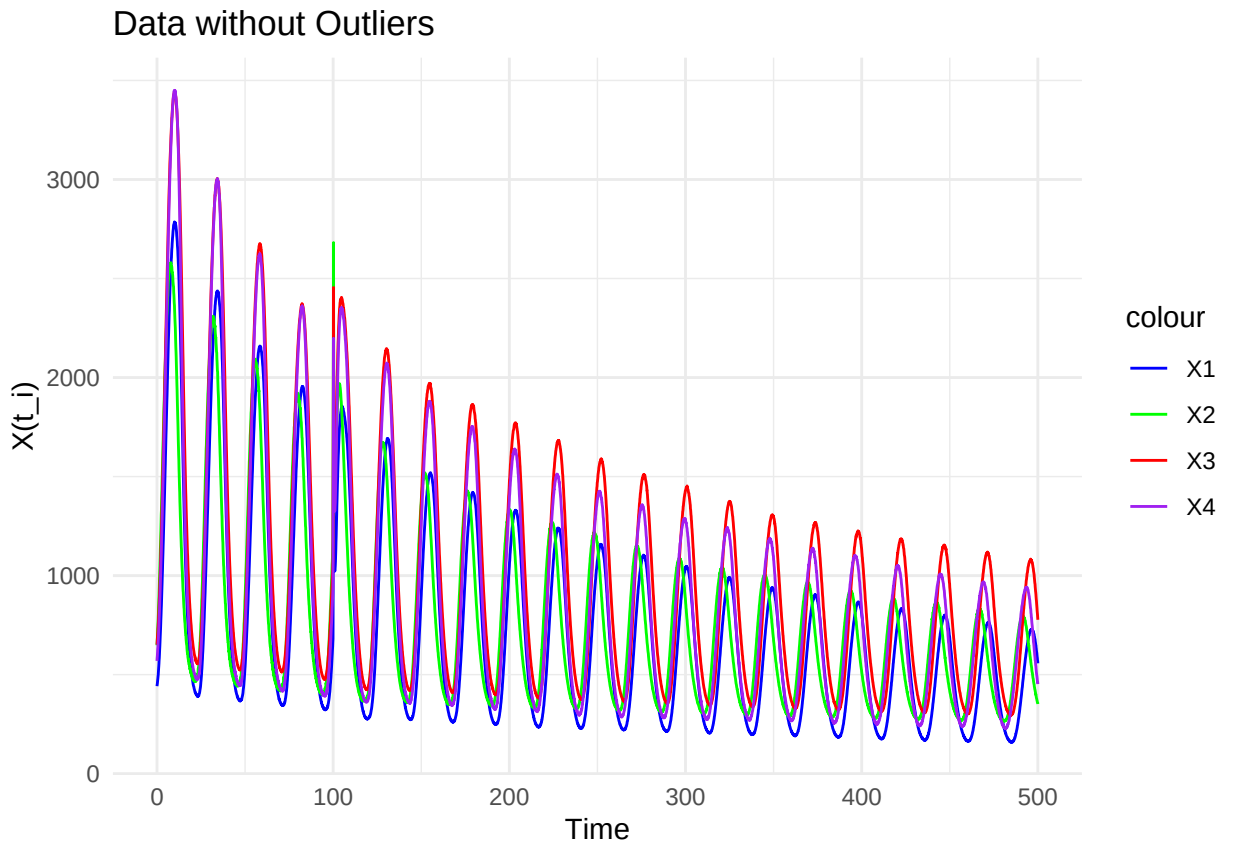
```
##
## 'nlstools' has been loaded.

## IMPORTANT NOTICE: Most nonlinear regression models and data set examples

## related to predictive microbiolgy have been moved to the package 'nlsMicrobio'
```

```
library(stats)
```

```
# Load and process the
library(readxl)
Data_McManus2_WT_P2L_AAVCremCherry <- read_excel("Data_McManus2_WT_P2L_AAVCremCherry.xlsx")
```

```
## New names:
## * `` -> `...1`
```

```r
df_MC <- as.data.frame(Data_McManus2_WT_P2L_AAVCremCherry)

# Rename columns for clarity
colnames(df_MC) <- c("Time", "X1", "X2", "X3", "X4")

# Remove outliers where Time is between 100 and 100.1
df_MC <- df_MC[!(df_MC$Time >= 100 & df_MC$Time <= 100.1), ]

# Plot each species
ggplot(df_MC, aes(x = Time)) +
  geom_line(aes(y = X1, color = "X1")) +
  geom_line(aes(y = X2, color = "X2")) +
  geom_line(aes(y = X3, color = "X3")) +
  geom_line(aes(y = X4, color = "X4")) +
  labs(title = "Data without Outliers", x = "Time", y = "X(t_i)") +
  scale_color_manual(values = c("X1" = "blue", "X2" = "green", "X3" = "red", "X4" = "purple")) +
  theme_minimal()
```



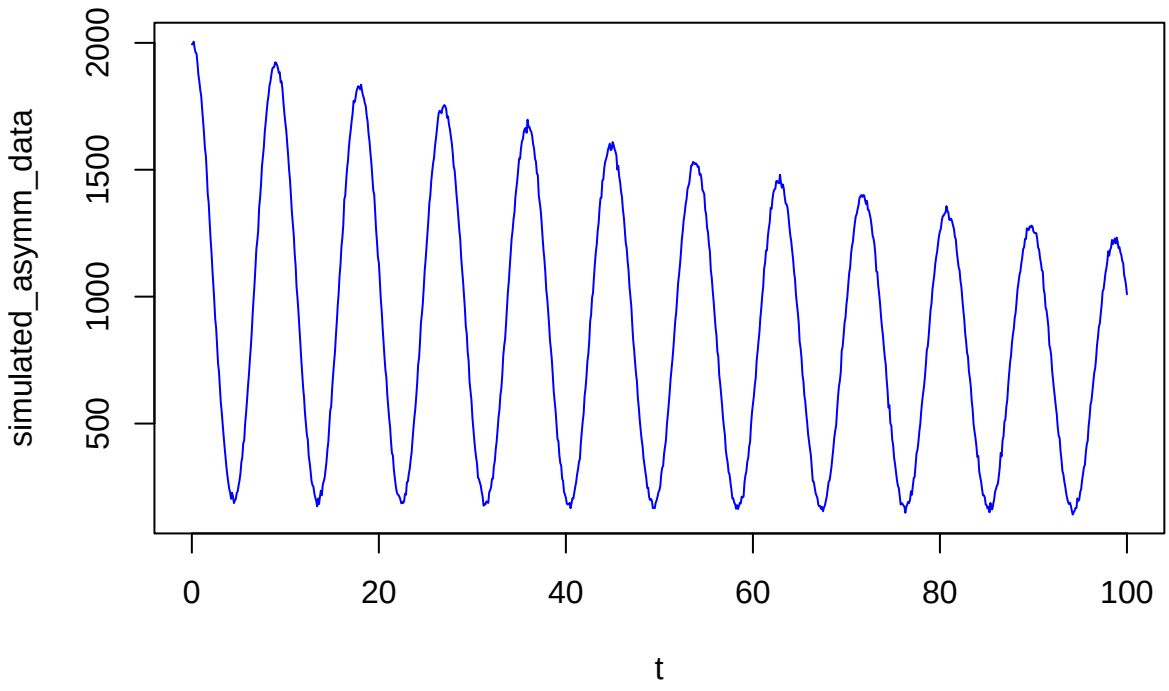

```r
df_MC_early <- subset(df_MC, Time < 100)
df_MC_late <- subset(df_MC, Time >= 124)
```

```r
asym_model_function <- function(t, A_max, eta_max, A_min, eta_min, tau, phi) {
  A_max_t <- A_max * exp(-eta_max * t)
  A_min_t <- A_min * exp(-eta_min * t)
  result <- 0.5 * (A_max_t + A_min_t) + 0.5 * (A_max_t - A_min_t) * cos(tau * t + phi)
```

```
  return(result)
}
```

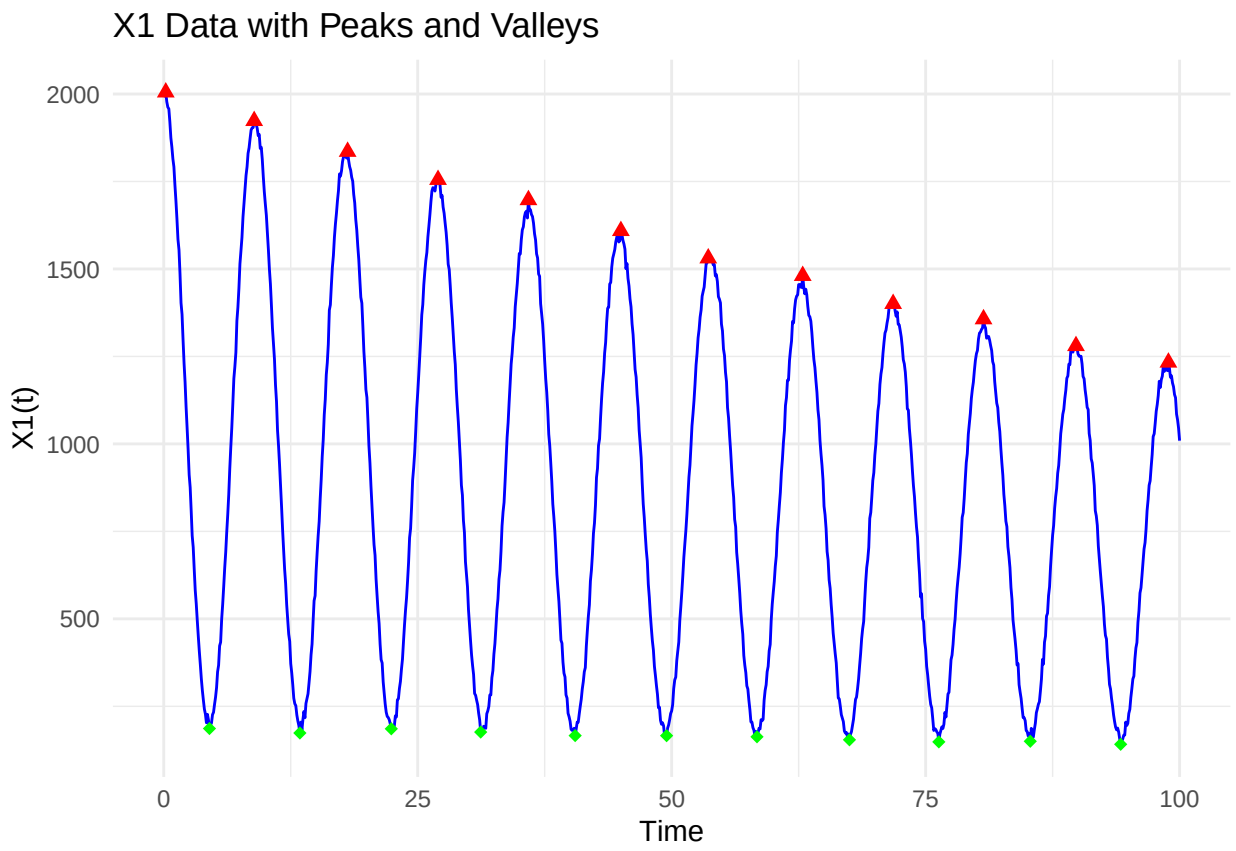
Simulation data for asymmetric model

```
true_params_asym = c(A_max =2000 , eta_max = 0.005, A_min =200, eta_min = 0.0025,tau = 0.7, phi = 0)

simulate_asymm_func<- function(t, params, sigma) {
  predicted <- asym_model_function(t, params[1], params[2], params[3], params[4], params[5],params[6])
  noise <- rnorm(length(t), mean = 0, sd = sigma)  #using sigma0 to produce noise
  simulated_data <- predicted + noise
  return(simulated_data)
}

sigma0 = 10
t <- seq(0, 100, by = 0.1)
simulated_asymm_data <- simulate_asymm_func(t, true_params_asym, sigma0 )
plot(t,simulated_asymm_data,type = "l", col = "blue")
```



find inital parameters for simulated data for asymmetric model

```
peaks <- findpeaks(simulated_asymm_data, minpeakheight = 720, minpeakdistance = 30)
time <- df_MC$Time
t_peaks <- time[peaks[, 2]]
y_peaks<- peaks[, 1]
#delete outlier

valleys <- findpeaks(-simulated_asymm_data, minpeakheight = -600, minpeakdistance = 50)

t_valleys <- time[valleys[, 2]]
y_valleys <- -valleys[, 1]  # restore

df_simulated <- data.frame(Time = t, X1 = simulated_asymm_data)

# Create data frames for peaks and valleys
df_peaks <- data.frame(Time = t_peaks, Value = y_peaks)
df_valleys <- data.frame(Time = t_valleys, Value = y_valleys)

# Plotting using ggplot2

ggplot(df_simulated, aes(x = Time, y = X1)) +
  geom_line(color = "blue") +  # Main line for simulated data
  geom_point(data = df_peaks, aes(x = Time, y = Value), color = "red", size = 2, shape = 17) +  # Peaks
  geom_point(data = df_valleys, aes(x = Time, y = Value), color = "green", size = 2, shape = 18) +  # V
  labs(
    title = "X1 Data with Peaks and Valleys",
    x = "Time",
    y = "X1(t)"
  ) +
  theme_minimal()
```




```r
# exponential fit when t <100
log_y_peaks <- log(y_peaks)
log_y_valleys <- log(y_valleys)

fit_peaks <- lm(log_y_peaks ~ t_peaks)
fit_valleys <- lm(log_y_valleys ~ t_valleys)

A_max_asym_sim <- exp(coef(fit_peaks)[1])
eta_max_asym_sim <- -coef(fit_peaks)[2]
A_min_asym_sim <- exp(coef(fit_valleys)[1])
eta_min_asym_sim <- -coef(fit_valleys)[2]

Delta_asym_sim <- t_peaks[2] - t_peaks[1]  # Assuming the first two peaks are used
tau_asym_sim <- 2 * pi / Delta_asym_sim

# Select the first peak
t_selected <- t_peaks[1]
y_selected <- y_peaks[1]

# Compute A_max(t) and A_min(t) at the first peak
A_max_t <- A_max_asym_sim * exp(-eta_max_asym_sim * t_selected)
A_min_t <- A_min_asym_sim * exp(-eta_min_asym_sim * t_selected)

# Compute the cosine term
cos_input <- (2 * y_selected - (A_max_t + A_min_t)) / (A_max_t - A_min_t)
cos_input <- pmin(pmax(cos_input, -1), 1)  # Ensure input is within valid range
```

```r
# Compute phi
phi_asym_sim <- acos(cos_input) - tau_asym_sim * t_selected

# Store the estimated parameters including phi
initial_params_asymm_sim <- c(A_max_asym_sim, eta_max_asym_sim, A_min_asym_sim, eta_min_asym_sim, tau_as

# Print the estimated parameters
print(initial_params_asymm_sim)
```

```
##   (Intercept)       t_peaks   (Intercept)     t_valleys
##  2.009982e+03  4.956213e-03  1.899828e+02  2.964502e-03  7.222052e-01
##   (Intercept)
## -6.053114e-02
```

```r
cat("Estimated phi:", phi_asym_sim, "\n")
```

```
## Estimated phi: -0.06053114
```

functions for find initial parameters and fit nlsLM model

```r
set.seed(70)
# Function to estimate initial parameters
find_initial_params <- function(t, y) {
  peaks <- findpeaks(y, minpeakheight = 700, minpeakdistance = 30)
  t_peaks <- t[peaks[, 2]]
  y_peaks <- peaks[, 1]

  valleys <- findpeaks(-y, minpeakheight = -600, minpeakdistance = 50)
  t_valleys <- t[valleys[, 2]]
  y_valleys <- -valleys[, 1]

  # Exponential fit for A_max and A_min
  log_y_peaks <- log(y_peaks)
  log_y_valleys <- log(y_valleys)

  fit_peaks <- lm(log_y_peaks ~ t_peaks)
  fit_valleys <- lm(log_y_valleys ~ t_valleys)

  A_max <- exp(coef(fit_peaks)[1])
  eta_max <- -coef(fit_peaks)[2]
  A_min <- exp(coef(fit_valleys)[1])
  eta_min <- -coef(fit_valleys)[2]

  # Estimate tau using peak intervals
  Delta_asym_sim <- t_peaks[2] - t_peaks[1]  # Assuming the first two peaks are used
  tau_asym_sim <- 2 * pi / Delta_asym_sim
  # Select the first peak
  t_selected<-t_peaks[1]
  y_selected<-y_peaks[1]
```

```
  A_max_t<-A_max_asym_sim*exp(-eta_max_asym_sim*t_selected)
  A_min_t<-A_min_asym_sim*exp(-eta_min_asym_sim*t_selected)
  # # Compute the cosine term
  cos_input<-(2*y_selected-(A_max_t+A_min_t))/(A_max_t-A_min_t)
  cos_input<-pmin(pmax(cos_input,-1),1)# Ensure input is within valid range## # Compute phi
  phi_asym_sim<-acos(cos_input)-tau_asym_sim*t_selected
  phi_asym_sim<-((phi_asym_sim+2*pi)%%(4*pi))-2*pi

  return(list(A_max = A_max, eta_max = eta_max, A_min = A_min, eta_min = eta_min,
              tau = tau_asym_sim, phi = phi_asym_sim))
}

compute_phi <- function(t, y, A_max, eta_max, A_min, eta_min, tau_asym_sim) {
  set.seed(123)  # Ensure reproducibility
  selected_indices <- sample(1:length(t), 5, replace = FALSE)
  t_selected <- t[selected_indices]
  y_selected <- y[selected_indices]

  phi_values <- numeric(5)
  for (i in 1:5) {
    A_max_t <- A_max * exp(-eta_max * t_selected[i])
    A_min_t <- A_min * exp(-eta_min * t_selected[i])

    cos_input <- (2 * y_selected[i] - (A_max_t + A_min_t)) / (A_max_t - A_min_t)
    cos_input <- pmin(pmax(cos_input, -1), 1)  # Ensure input is within valid range

    phi_values[i] <- acos(cos_input) - tau_asym_sim * t_selected[i]
  }

  # Compute the median phi value
  phi_asym_sim <- mean(phi_values)
  phi_asym_sim <- ((phi_asym_sim + 2 * pi) %% (4 * pi)) - 2 * pi

  return(phi_asym_sim)
}

lower_bounds <- c(A_max = 500, eta_max = 0, A_min = 50, eta_min = -1, tau = 0, phi = -2*pi)
upper_bounds <- c(A_max = 4000, eta_max = 3, A_min = 600, eta_min = 2, tau = 2, phi = 2*pi)



# Function to fit nlsLM and compute confidence intervals
fit_nls_model <- function(t, y, initial_params) {
  fit <- try(nlsLM(
    y ~ asym_model_function(t, A_max, eta_max, A_min, eta_min, tau, phi),
    start = initial_params,
    data = data.frame(t = t, y = y),
    lower = lower_bounds,
    upper = upper_bounds
  ), silent = TRUE)

  if (inherits(fit, "try-error")) {
    return(NULL)
```

```
  }

  estimates <- coef(fit)
  conf_intervals <- confint2(fit, level = 0.99, method = 'asymptotic')
  return(list(estimates = estimates, conf_intervals = conf_intervals))
}

# Function to extract estimates safely
extract_estimates <- function(results) {
  estimates_list <- lapply(results, function(res) {
    if (!is.null(res)) {
      return(as.numeric(res$estimates))  # Convert to numeric vector
    } else {
      return(rep(NA, 6))  # If NULL, return NA values
    }
  })

  # Convert list to data frame
  estimates_df <- do.call(rbind, estimates_list)

  # Explicitly define column names
  colnames(estimates_df) <- c("A_max", "eta_max", "A_min", "eta_min", "tau", "phi")

  return(as.data.frame(estimates_df))
}
```

Repeat the simulation 100 times (using the above method) to obtain coverage for each estimated parameter, verifying the suitability of the method.

```
# Main function to run simulations and compute coverage
set.seed(223)
run_simulations <- function(n_simulations = 100, sigma = 10) {
  true_params <- c(A_max = 2000, eta_max = 0.005, A_min = 200, eta_min = 0.0025, tau = 0.7, phi = 0)
  t <- seq(0, 100, by = 0.1)

  results <- vector("list", n_simulations)
  for (i in 1:n_simulations) {
    simulated_data <- simulate_asymm_func(t, true_params, sigma)
    initial_params <- find_initial_params(t, simulated_data)

    fit_result <- fit_nls_model(t, simulated_data, initial_params)
    results[[i]] <- fit_result
  }

  # Remove NULL results
  results <- Filter(Negate(is.null), results)

  # Count number of failures
  null_count <- sum(sapply(results, is.null))
```

```
  cat("Number of unsuccessful fits (NULL results):", null_count, "\n")

  # Initialize coverage counter
  coverage_counts <- rep(0, length(true_params))

  for (result in results) {
    conf_intervals <- result$conf_intervals
    for (j in 1:length(true_params)) {
      if (true_params[j] >= conf_intervals[j, 1] && true_params[j] <= conf_intervals[j, 2]) {
        coverage_counts[j] <- coverage_counts[j] + 1
      }
    }
  }
  estimates_df <- extract_estimates(results)
  coverage_proportions <- coverage_counts / n_simulations
  cat("Coverage for A_max:", coverage_proportions[1], "\n")
  cat("Coverage for eta_max:", coverage_proportions[2], "\n")
  cat("Coverage for A_min:", coverage_proportions[3], "\n")
  cat("Coverage for eta_min:", coverage_proportions[4], "\n")
  cat("Coverage for tau:", coverage_proportions[5], "\n")
  cat("Coverage for phi:", coverage_proportions[6], "\n")

  return(list(coverage = coverage_proportions, estimates = estimates_df))
}

# Run the simulation
simulation_results <- run_simulations(n_simulations = 100, sigma = 10 )
```

```
## Number of unsuccessful fits (NULL results): 0
## Coverage for A_max: 1
## Coverage for eta_max: 1
## Coverage for A_min: 1
## Coverage for eta_min: 0.99
## Coverage for tau: 0.99
## Coverage for phi: 0.89
```

```
coverage_results <- simulation_results$coverage
estimates_df <- simulation_results$estimates
```

boxplot:

To compare the true parameters of the simulation data with the interquartile range (IQR) of the estimated parameters obtained from 100 repeated simulations

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
```

```
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(tidyr)
estimates_long <- estimates_df %>%
   pivot_longer(cols = everything(), names_to = "Parameter", values_to = "Estimate")


true_params <- c(A_max = 2000, eta_max = 0.005, A_min = 200, eta_min = 0.0025, tau = 0.7, phi = 0)

# A_max boxplot
plot_A_max <- ggplot(estimates_long %>% filter(Parameter == "A_max"),
                     aes(x = Parameter, y = Estimate)) +
  geom_boxplot(fill = "lightcoral", color = "darkred") +
  geom_hline(yintercept = true_params["A_max"], linetype = "dashed") +
  labs(title = "A_max", y = "Estimate") +
  theme_minimal() +
  ylim(1995, 2005)  #

# A_min boxplot
plot_A_min <- ggplot(estimates_long %>% filter(Parameter == "A_min"),
                     aes(x = Parameter, y = Estimate)) +
  geom_boxplot(fill = "khaki", color = "darkgoldenrod") +
  geom_hline(yintercept = true_params["A_min"], linetype = "dashed") +
  labs(title = "A_min", y = "Estimate") +
  theme_minimal() +
  ylim(195, 205)

# eta_max boxplot
plot_eta_max <- ggplot(estimates_long %>% filter(Parameter == "eta_max"),
                       aes(x = Parameter, y = Estimate)) +
  geom_boxplot(fill = "lightgreen", color = "darkgreen") +
  geom_hline(yintercept = true_params["eta_max"], linetype = "dashed") +
  labs(title = "eta_max", y = "Estimate") +
  theme_minimal() +
  ylim(0.00498, 0.00502)

# eta_min boxplot
plot_eta_min <- ggplot(estimates_long %>% filter(Parameter == "eta_min"),
```

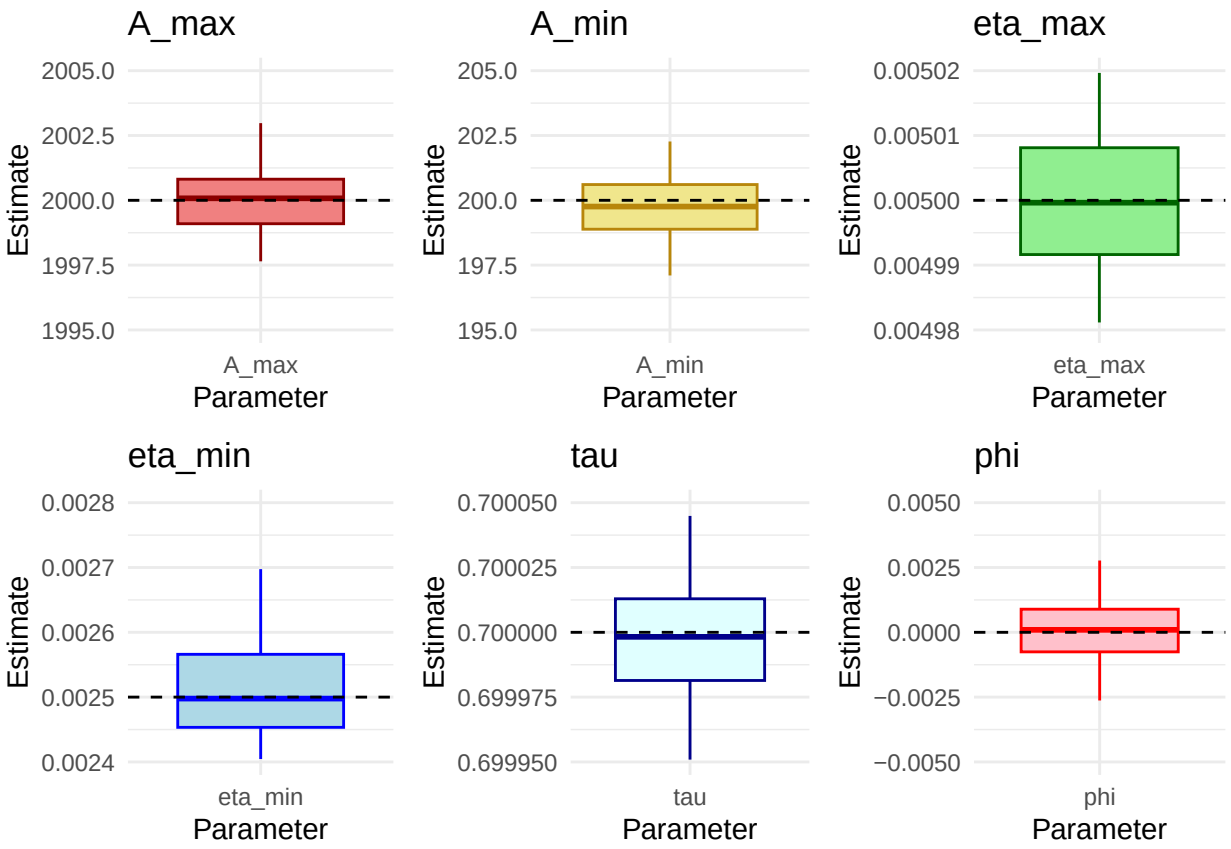
```
                      aes(x = Parameter, y = Estimate)) +
  geom_boxplot(fill = "lightblue", color = "blue") +
  geom_hline(yintercept = true_params["eta_min"], linetype = "dashed") +
  labs(title = "eta_min", y = "Estimate") +
  theme_minimal() +
  ylim(0.0024, 0.0028)

# tau boxplot
plot_tau <- ggplot(estimates_long %>% filter(Parameter == "tau"),
                     aes(x = Parameter, y = Estimate)) +
  geom_boxplot(fill = "lightcyan", color = "darkblue") +
  geom_hline(yintercept = true_params["tau"], linetype = "dashed") +
  labs(title = "tau", y = "Estimate") +
  theme_minimal() +
  ylim(0.69995, 0.70005)

# phi boxplot
plot_phi <- ggplot(estimates_long %>% filter(Parameter == "phi"),
                   aes(x = Parameter, y = Estimate)) +
  geom_boxplot(fill = "pink", color = "red") +
  geom_hline(yintercept = true_params["phi"], linetype = "dashed") +
  labs(title = "phi", y = "Estimate") +
  theme_minimal() +
  ylim(-0.005, 0.005)

#  gridExtra  boxplot
grid.arrange(plot_A_max, plot_A_min, plot_eta_max, plot_eta_min, plot_tau, plot_phi, ncol = 3)
```

```
## Warning: Removed 7 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```

```
## Warning: Removed 21 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_boxplot()`).
```

```
## Warning: Removed 11 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```



separate real data into 2 parts

```
X1 <- df_MC$X1
#  split data for early(t<100) and late segments(t>=100)
time_early <- time[time < 100]
time_late <- time[time >= 124]

y_early <- df_MC$X1[df_MC$Time < 100]

time_late <- df_MC$Time[df_MC$Time >= 124]
y_late <- df_MC$X1[df_MC$Time >= 124]
```

apply the methods verified by simulations to real data

1. get initial parameters for the real data seqparately(time before 100 and after 124), and plot a graph to compare.

```
init_param_asymm_early = find_initial_params(time_early, y_early)
init_param_asymm_late = find_initial_params(time_late, y_late)

init_params <- data.frame(
  Parameter = c("A_max", "eta_max", "A_min", "eta_min", "tau", "phi",
                "A_max", "eta_max", "A_min", "eta_min", "tau", "phi"),
  Time_Period = c("early", "early", "early", "early", "early", "early",
                  "late", "late", "late", "late", "late", "late"),
```

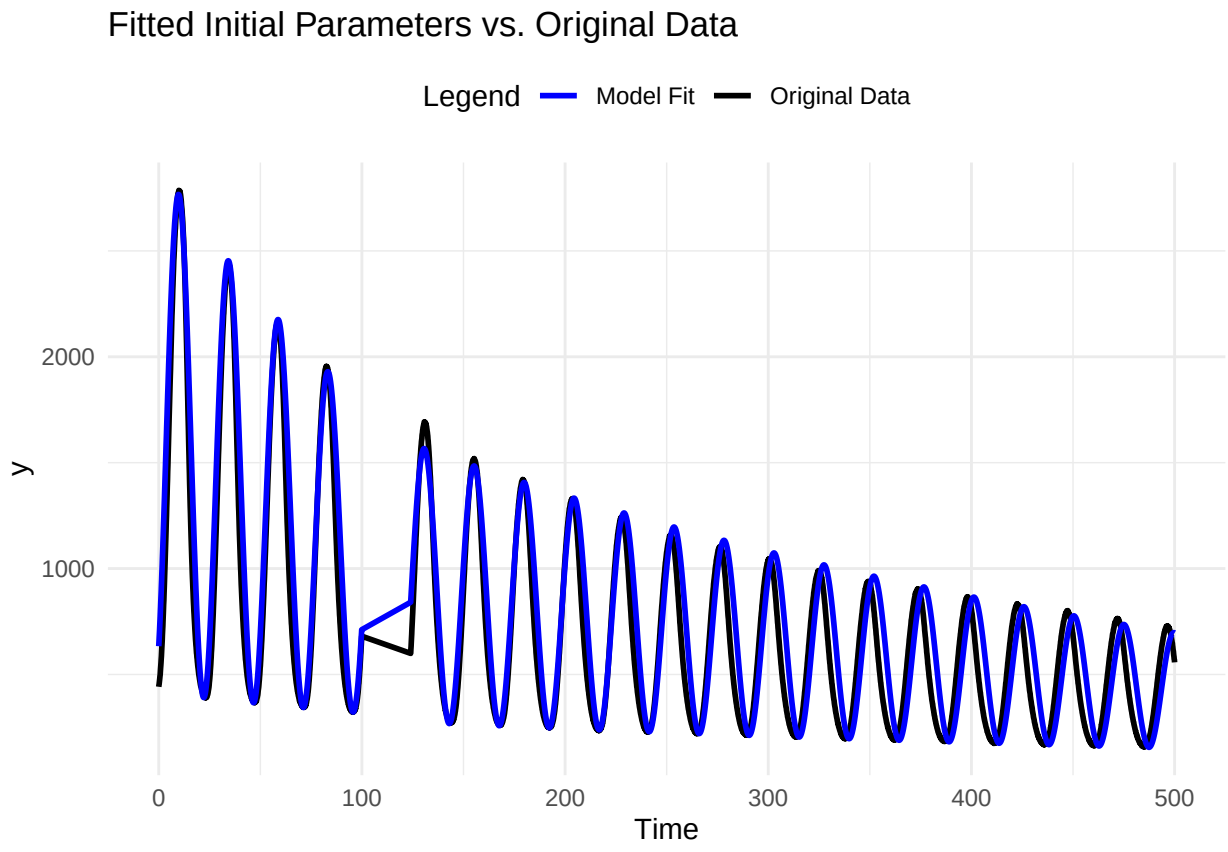
```
  Value = c(init_param_asymm_early$A_max, init_param_asymm_early$eta_max, init_param_asymm_early$A_min,
            init_param_asymm_early$eta_min, init_param_asymm_early$tau, init_param_asymm_early$phi,
            init_param_asymm_late$A_max, init_param_asymm_late$eta_max, init_param_asymm_late$A_min,
            init_param_asymm_late$eta_min, init_param_asymm_late$tau, init_param_asymm_late$phi)
)

# Print the table
print(init_params)
```

```
##    Parameter Time_Period         Value
## 1      A_max       early  2.903053e+03
## 2    eta_max       early  4.902279e-03
## 3      A_min       early  4.143191e+02
## 4    eta_min       early  2.620215e-03
## 5        tau       early  2.564565e-01
## 6        phi       early -2.538920e+00
## 7      A_max        late  2.086986e+03
## 8    eta_max        late  2.193884e-03
## 9      A_min        late  3.356417e+02
## 10   eta_min        late  1.570847e-03
## 11       tau        late  2.554140e-01
## 12       phi        late  4.316497e+00
```

```
y_model_early <- asym_model_function(time_early, init_param_asymm_early$A_max,init_param_asymm_early$eta

y_model_late <- asym_model_function(time_late, init_param_asymm_late$A_max,init_param_asymm_late$eta_max

#
df_plot_early <- data.frame(Time = time_early,
                            Original = y_early,
                            ModelFit = y_model_early,

                            Group = "Early")

df_plot_late <- data.frame(Time = time_late,
                           Original = y_late,
                           ModelFit = y_model_late,

                           Group = "Late")

df_plot <- rbind(df_plot_early, df_plot_late)

ggplot(df_plot, aes(x = Time)) +

  geom_line(aes(y = Original, color = "Original Data", linetype = "Original Data"), size = 1) +

  geom_line(aes(y = ModelFit, color = "Model Fit", linetype = "Model Fit"), size = 1) +

  scale_color_manual(values = c("Original Data" = "black",
                                "Model Fit" = "blue"
                                )) +

  scale_linetype_manual(values = c("Original Data" = "solid",
```

```
                                  "Model Fit" = "solid"
                                  )) +

  #
  labs(title = "Fitted Initial Parameters vs. Original Data",
       x = "Time", y = "y", color = "Legend", linetype = "Legend") +

  #
  theme_minimal() +
  theme(legend.position = "top")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



fit model to. get estimated parameters for real data

```
estimate_early = fit_nls_model(time_early,y_early,init_param_asymm_early)
estimate_late = fit_nls_model(time_late,y_late,init_param_asymm_late)
```

```r
 # Rename estimates
names(estimate_early$estimates) <- c("A_max", "eta_max", "A_min", "eta_min", "tau", "phi")
names(estimate_late$estimates) <- c("A_max", "eta_max", "A_min", "eta_min", "tau", "phi")

# Rename confidence intervals
rownames(estimate_early$conf_intervals) <- c("A_max", "eta_max", "A_min", "eta_min", "tau", "phi")
rownames(estimate_late$conf_intervals) <- c("A_max", "eta_max", "A_min", "eta_min", "tau", "phi")
```

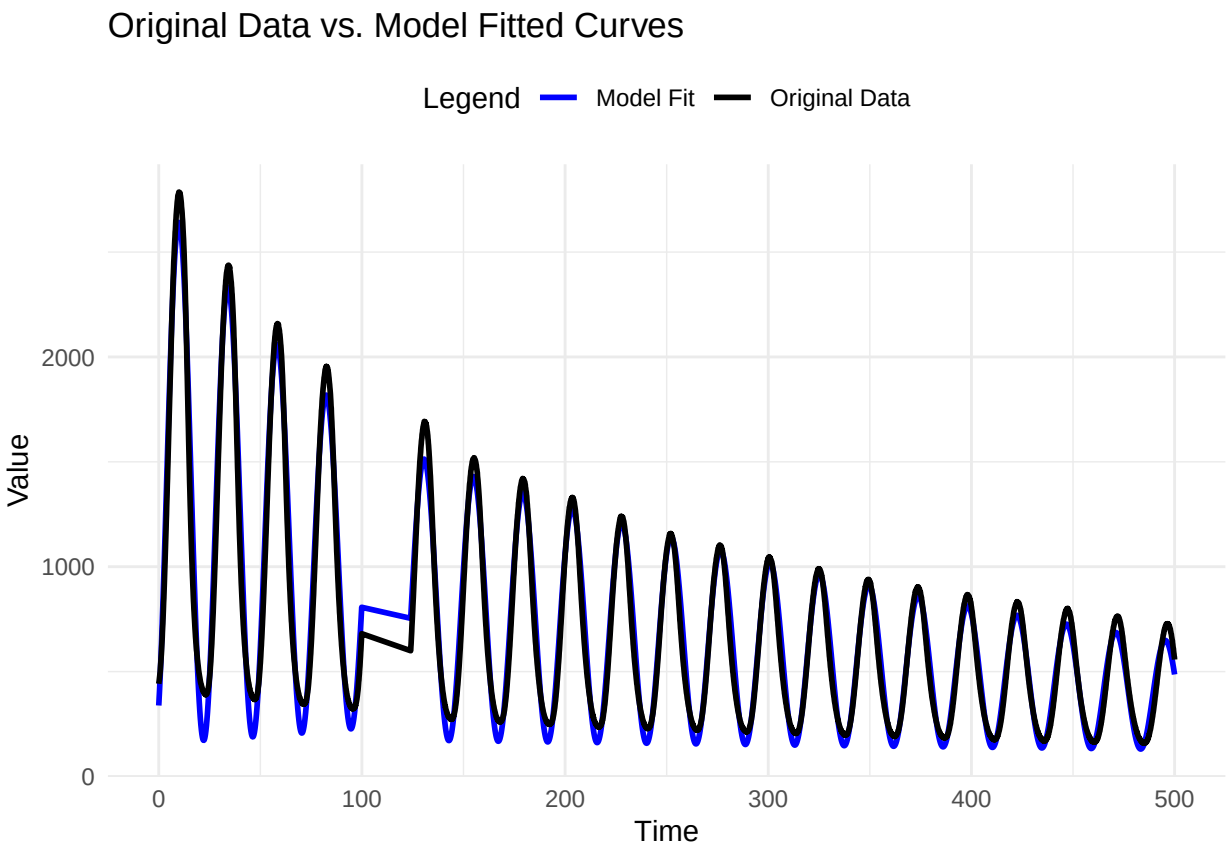
```r
# Create a data frame for early estimates
df_early <- data.frame(
  Parameter = names(estimate_early$estimates),
  Estimate_Early = estimate_early$estimates,
  CI_Lower_Early = estimate_early$conf_intervals[, 1],
  CI_Upper_Early = estimate_early$conf_intervals[, 2]
)

# Create a data frame for late estimates
df_late <- data.frame(
  Parameter = names(estimate_late$estimates),
  Estimate_Late = estimate_late$estimates,
  CI_Lower_Late = estimate_late$conf_intervals[, 1],
  CI_Upper_Late = estimate_late$conf_intervals[, 2]
)

# Merge both into one table
final_table <- merge(df_early, df_late, by = "Parameter")

# Display table
print(final_table)
```

```
##   Parameter Estimate_Early CI_Lower_Early CI_Upper_Early Estimate_Late
## 1     A_max   2.780757e+03   2.744675e+03   2.816839e+03  2.048953e+03
## 2     A_min   1.580924e+02   1.280568e+02   1.881280e+02  1.915146e+02
## 3   eta_max   5.152672e-03   4.880479e-03   5.424865e-03  2.321568e-03
## 4   eta_min  -3.822666e-03  -6.638458e-03  -1.006874e-03  7.888516e-04
## 5       phi  -2.610999e+00  -2.633574e+00  -2.588425e+00  3.945990e+00
## 6       tau   2.600176e-01   2.595499e-01   2.604852e-01  2.583350e-01
##   CI_Lower_Late CI_Upper_Late
## 1  2.028098e+03  2.069808e+03
## 2  1.758962e+02  2.071331e+02
## 3  2.284906e-03  2.358229e-03
## 4  5.285346e-04  1.049169e-03
## 5  3.925877e+00  3.966103e+00
## 6  2.582612e-01  2.584088e-01
```

compare curved generated by estimated parameters and real data

```r
library(ggplot2)
library(dplyr)

library(ggplot2)
```

```r
library(dplyr)

# Extract estimated parameters for early and late
params_early <- as.numeric(estimate_early$estimates)
params_late <- as.numeric(estimate_late$estimates)

# Generate fitted curves using estimated parameters
y_fit_early <- asym_model_function(time_early, params_early[1], params_early[2], params_early[3],
                                   params_early[4], params_early[5], params_early[6])

y_fit_late <- asym_model_function(time_late, params_late[1], params_late[2], params_late[3],
                                  params_late[4], params_late[5], params_late[6])

# Create data frames for ggplot
df_original <- data.frame(Time = c(time_early, time_late),
                          Value = c(y_early, y_late),
                          Type = "Original Data")

df_fitted <- data.frame(Time = c(time_early, time_late),
                        Value = c(y_fit_early, y_fit_late),
                        Type = "Model Fit")

# Combine into one dataset for ggplot
df_plot <- bind_rows(df_original, df_fitted)

# Plot the original data and fitted curves
# Plot the original data and fitted curves
ggplot(df_plot, aes(x = Time, y = Value, color = Type)) +
  geom_line(size = 1) +
  scale_color_manual(values = c("Original Data" = "black", "Model Fit" = "blue")) +  # Explicit color m
  labs(title = "Original Data vs. Model Fitted Curves",
       x = "Time", y = "Value", color = "Legend") +
  theme_minimal() +
  theme(legend.position = "top")
```



Hypothesis Test!

Estimate the variance and plot the histogram to verify the accuracy of the variance, calculate the log-likelihood, obtain the negative Hessian approximation of the Fisher information, then compute the covariance, exclude phi, and use the T-Wald test."

```
# Load necessary libraries
library(numDeriv)
```

```
##
## Attaching package: 'numDeriv'

## The following objects are masked from 'package:pracma':
##
##     grad, hessian, jacobian
```

```
library(nlme)
```

```
##
## Attaching package: 'nlme'

## The following object is masked from 'package:dplyr':
##
##     collapse
```

```r
# Function to compute sigma² (Residual Variance)
estimate_sigma2_asym <- function(y, t, params) {
  A_max <- params["A_max"]
  eta_max <- params["eta_max"]
  A_min <- params["A_min"]
  eta_min <- params["eta_min"]
  tau <- params["tau"]
  phi <- params["phi"]

  # Compute model-predicted values
  f_values <- asym_model_function(t, A_max, eta_max, A_min, eta_min, tau, phi)

  # Compute residuals
  residuals <- y - f_values
  SSR <- sum(residuals^2)
  sigma2 <- SSR / length(t)

  return(sigma2)
}

# Function to compute log-likelihood
log_likelihood_asym <- function(params, t, y, sigma2) {
  A_max <- params["A_max"]
  eta_max <- params["eta_max"]
  A_min <- params["A_min"]
  eta_min <- params["eta_min"]
  tau <- params["tau"]
  phi <- params["phi"]

  # Compute predicted values
  f_values <- asym_model_function(t, A_max, eta_max, A_min, eta_min, tau, phi)

  # Compute residuals
  residuals <- y - f_values
  n <- length(t)

  # Log-likelihood calculation
  log_likelihood_value <- -n / 2 * log(2 * pi * sigma2) - sum(residuals^2) / (2 * sigma2)

  return(log_likelihood_value)
}

# Compute residuals
compute_residuals_asym <- function(y, t, params) {
  A_max <- params["A_max"]
  eta_max <- params["eta_max"]
  A_min <- params["A_min"]
  eta_min <- params["eta_min"]
  tau <- params["tau"]
  phi <- params["phi"]

  f_values <- asym_model_function(t, A_max, eta_max, A_min, eta_min, tau, phi)
  residuals <- y - f_values
```

```r
  return(residuals)
}

# Compute covariance matrix (Fisher Information Inverse)
compute_covariance_asym <- function(params, t, y, sigma2) {
  hessian_matrix <- hessian(func = log_likelihood_asym, x = params, t = t, y = y, sigma2 = sigma2)
  fisher_information <- -hessian_matrix
  covariance_matrix <- solve(fisher_information)
  return(covariance_matrix)
}


wald_test <- function(theta_early, theta_late, covariance_early, covariance_late) {
  theta_diff <- theta_early - theta_late
  cov_total <- covariance_early + covariance_late
  T_Wald <- t(theta_diff) %*% solve(cov_total) %*% theta_diff
  df <- length(theta_diff)
  p_value <- 1 - pchisq(T_Wald, df)
  return(list(T_Wald = T_Wald, p_value = p_value))
}
```

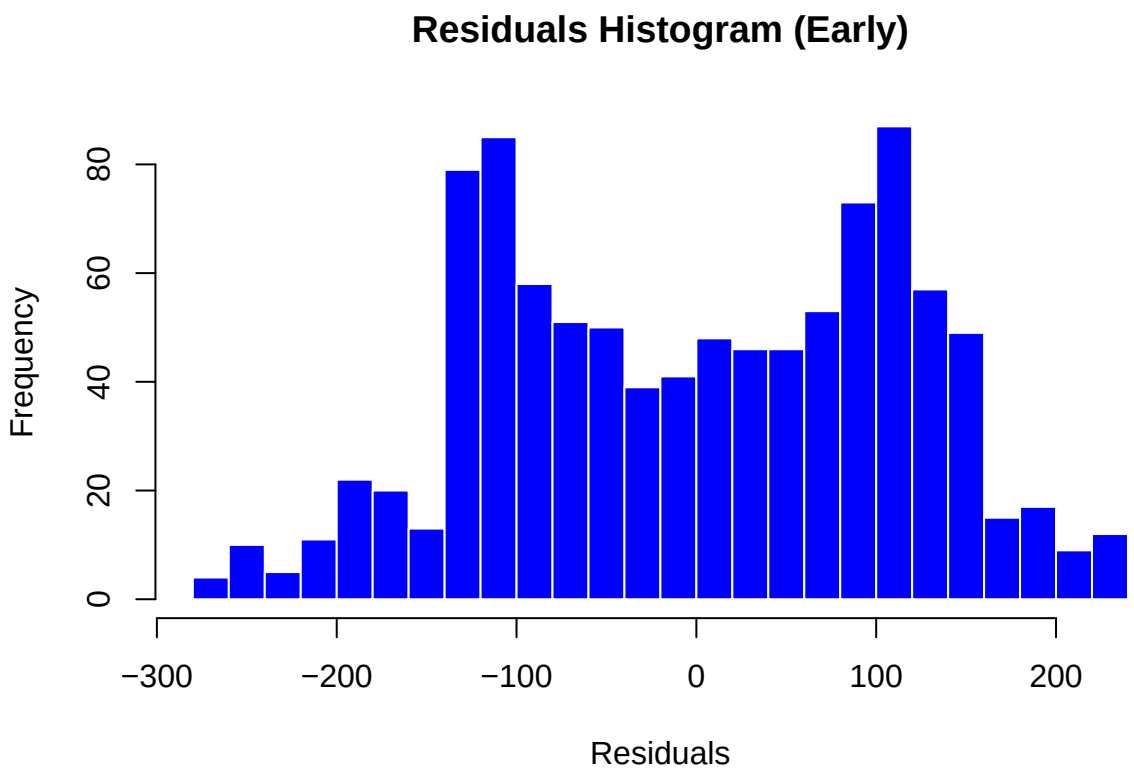
```r
params_early <- setNames(as.numeric(estimate_early$estimates), names(estimate_early$estimates))
params_late <- setNames(as.numeric(estimate_late$estimates), names(estimate_late$estimates))

# Compute sigma² for Early & Late
sigma2_early_asym <- estimate_sigma2_asym(y_early, time_early, params_early)
sigma2_late_asym <- estimate_sigma2_asym(y_late, time_late, params_late)

cat("Sigma² for early:", sigma2_early_asym, "\n")
```

```
## Sigma² for early: 12722.74
```

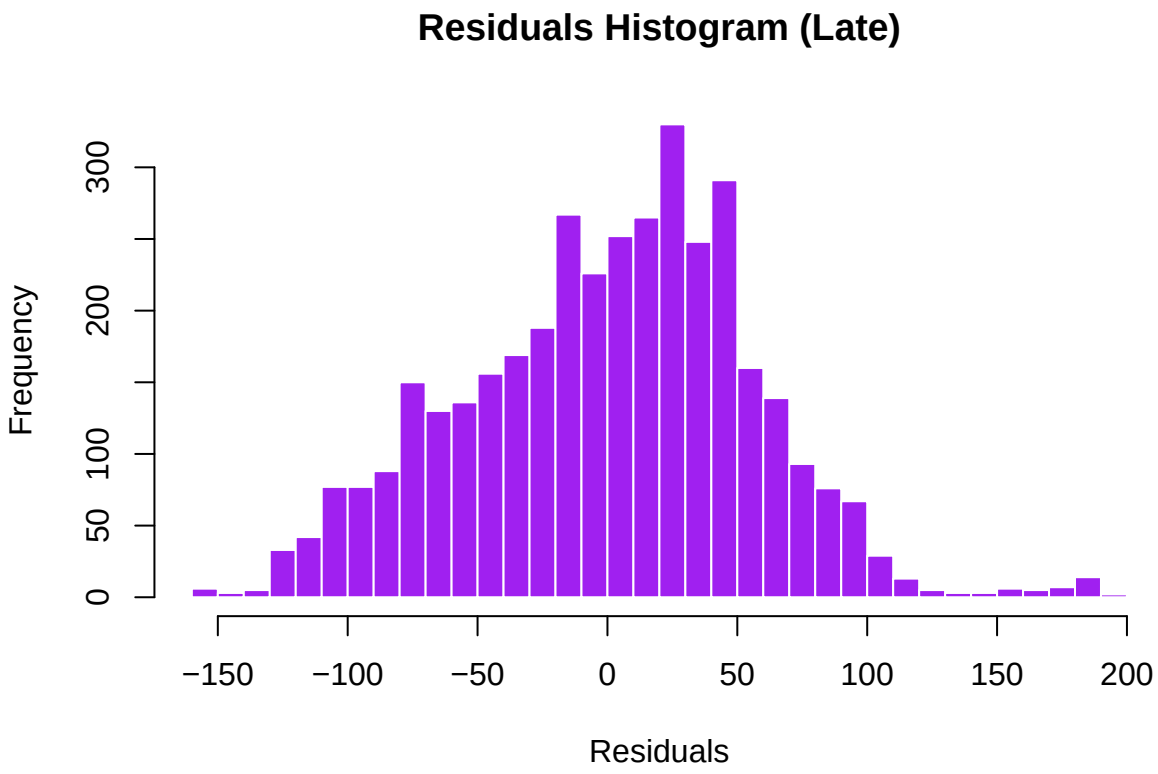
```r
cat("Sigma² for late:", sigma2_late_asym, "\n")
```

```
## Sigma² for late: 3112.272
```

```r
# Compute residuals
residuals_early_asym <- compute_residuals_asym(y_early, time_early, params_early)
residuals_late_asym <- compute_residuals_asym(y_late, time_late, params_late)

# Plot residual histograms

hist(residuals_early_asym, breaks = 30, col = "blue", border = "white",
     main = "Residuals Histogram (Early)", xlab = "Residuals", ylab = "Frequency")
```




```r
hist(residuals_late_asym, breaks = 30, col = "purple", border = "white",
     main = "Residuals Histogram (Late)", xlab = "Residuals", ylab = "Frequency")
```




```r
# Compute covariance matrices
covariance_early_asym <- compute_covariance_asym(params_early, time_early, y_early, sigma2_early_asym)
covariance_late_asym <- compute_covariance_asym(params_late, time_late, y_late, sigma2_late_asym)

# Function to remove phi from parameters and covariance matrix
remove_phi_asym <- function(params, covariance_matrix) {
  param_names <- names(params)
  phi_index <- which(param_names == "phi")

  # Remove phi
  params_reduced <- params[-phi_index]
  covariance_matrix_reduced <- covariance_matrix[-phi_index, -phi_index]

  return(list(params = params_reduced, covariance = covariance_matrix_reduced))
}

# Remove phi from early & late estimates
early_filtered_asym <- remove_phi_asym(params_early, covariance_early_asym)
late_filtered_asym <- remove_phi_asym(params_late, covariance_late_asym)

# Perform Wald Test
wald_result_asym <- wald_test(early_filtered_asym$params, late_filtered_asym$params,
                              early_filtered_asym$covariance, late_filtered_asym$covariance)


# Output Wald Test results
```

```r
cat("Wald Test Statistic:", wald_result_asym$T_Wald, "\n")
```

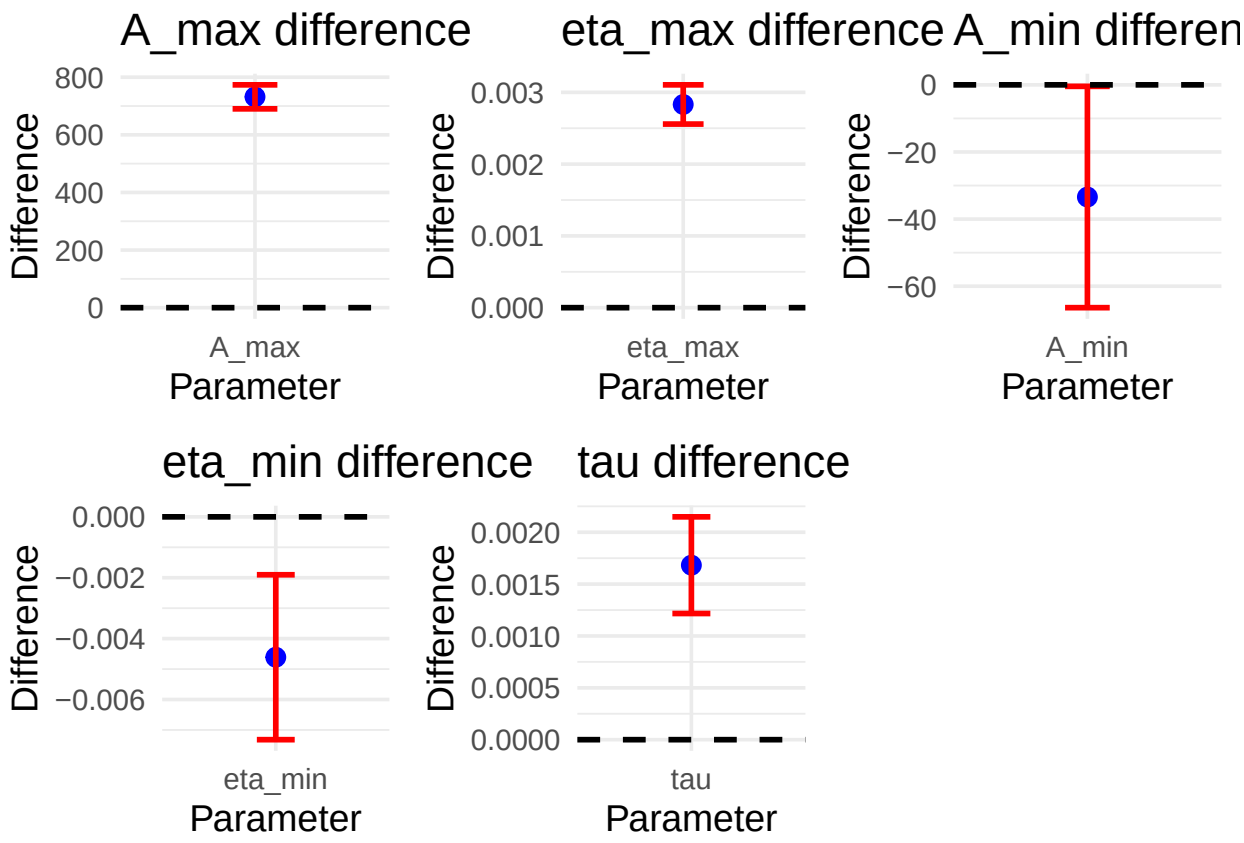
```
## Wald Test Statistic: 2456.654
```

```r
cat("p-value:", wald_result_asym$p_value, "\n")
```

```
## p-value: 0
```

plot difference between time <100 and time>100 for each parameters.

```r
library(ggplot2)
library(dplyr)

library(ggplot2)
library(gridExtra)

#
param_diff <- early_filtered_asym$params - late_filtered_asym$params

# covariance from hessian
var_diff <- diag(early_filtered_asym$covariance + late_filtered_asym$covariance)
se_diff <- sqrt(var_diff)

# 99% CI
Z_99 <- 2.576  #
CI_lower <- param_diff - Z_99 * se_diff
CI_upper <- param_diff + Z_99 * se_diff

#
df_diff <- data.frame(
  Parameter = names(param_diff),
  Difference = param_diff,
  CI_Lower = CI_lower,
  CI_Upper = CI_upper
)

#
plots <- lapply(1:nrow(df_diff), function(i) {
  ggplot(df_diff[i, ], aes(x = Parameter, y = Difference)) +
    geom_point(size = 3, color = "blue") +  #
    geom_errorbar(aes(ymin = CI_Lower, ymax = CI_Upper), width = 0.2, size = 1, color = "red") +  #
    geom_hline(yintercept = 0, linetype = "dashed", color = "black", size = 1) +  # 0
    labs(title = paste0(df_diff$Parameter[i], " difference"),
         x = "Parameter", y = "Difference") +
    theme_minimal(base_size = 14)
})

#
grid.arrange(grobs = plots, ncol = 3)  # 3
```

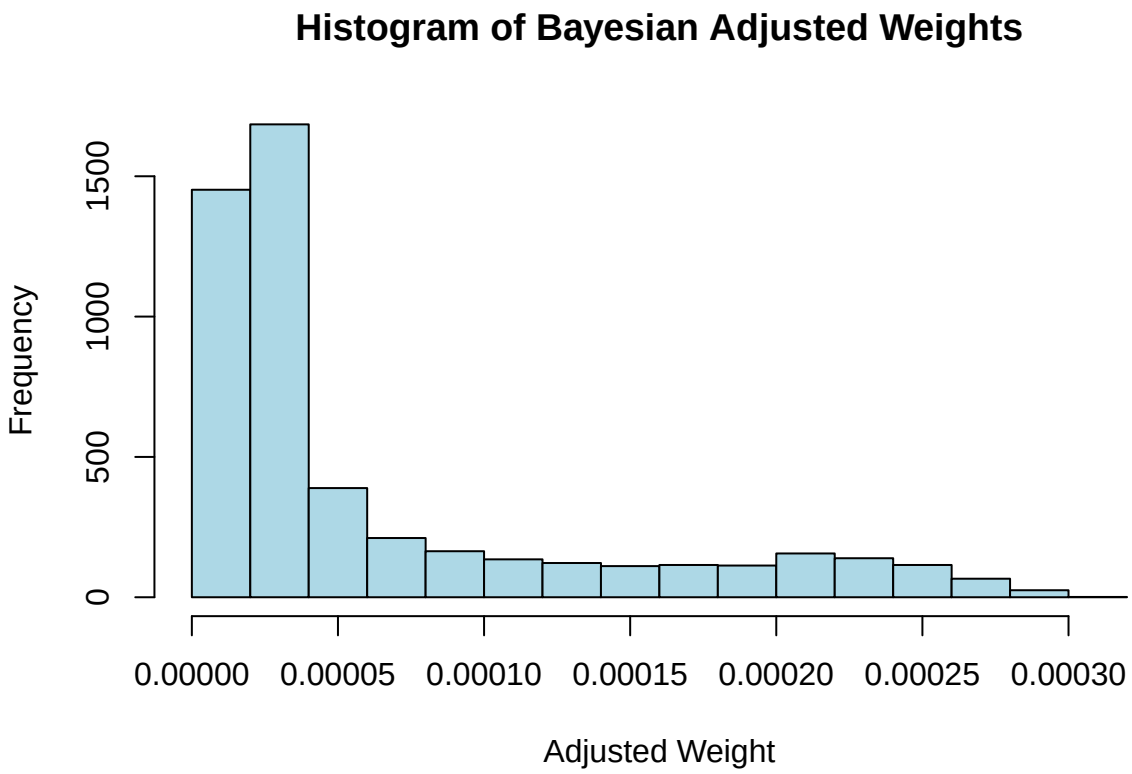


Asymmetric Model with Time dependent Variable

method of moment to get prior; bayesian inference to update posterior:bayesian weight = posterior alpha / posterior beta put bayesian weights in nlsLM. (use the same initial parameters of asymmetric model with constant variance)

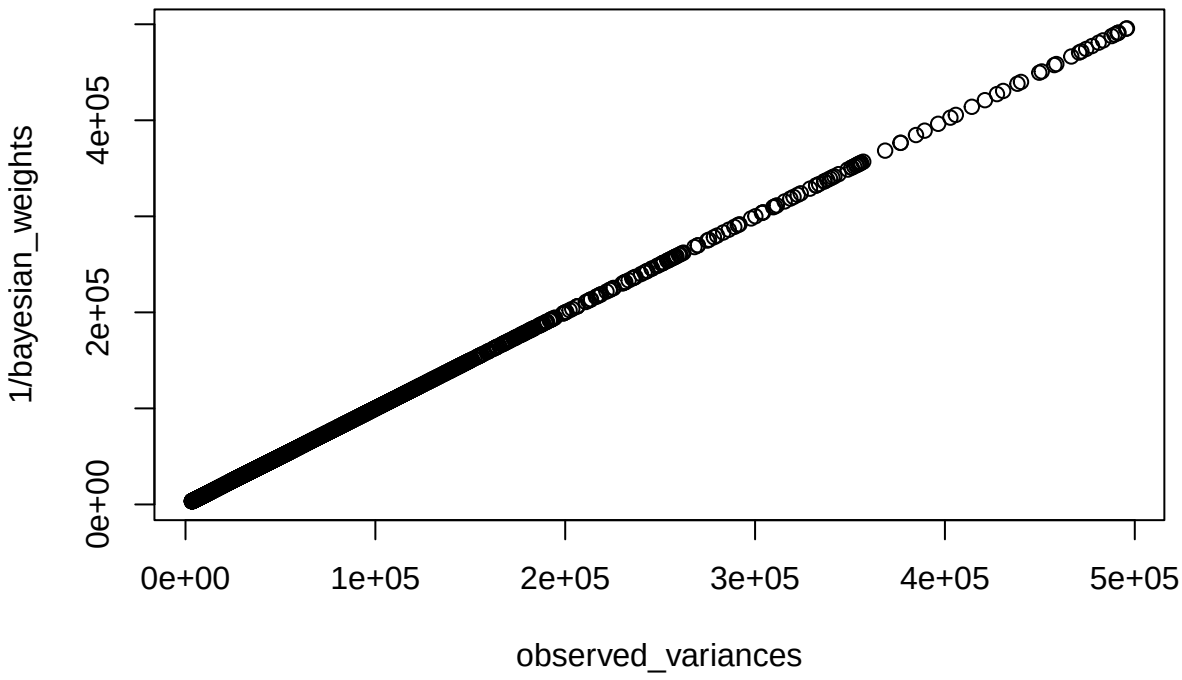
```
# Function to calculate variance at each time point
calculate_variance <- function(replicates) {
  n <- ncol(replicates)#  n=4
  mean_vals <- rowMeans(replicates)  # Mean for each time point
  variance <- rowSums((replicates - mean_vals)^2) / (n - 1)  # Variance formula
  return(variance)
}

# Extract replicate data
replicates <- df_MC[, c("X1", "X2", "X3", "X4")]

# Calculate variance
observed_variances <- calculate_variance(replicates)

# Function to estimate Bayesian weights
bayesian_variance_estimation <- function(observed_variances, n, alpha_prior, beta_prior) {
  weights <- numeric(length(observed_variances))
  for (i in 1:length(observed_variances)) {
    S2_i <- observed_variances[i]
```

```r
    alpha_post <- alpha_prior + (n - 1) / 2
    beta_post <- beta_prior + S2_i * (n - 1) / 2
    posterior_mean <- alpha_post / beta_post  # Expectation for sigma^(-2)
    weights[i] <- posterior_mean
  }
  return(weights)
}

# Step 1: Calculate sample moments (alpha_prior and beta_prior)
calculate_moments <- function(observed_variances) {
  n <- length(observed_variances)
  sample_mean <- mean(observed_variances)
  sample_variance <- sum((observed_variances - sample_mean)^2) / (n - 1)

  beta_prior <- sample_mean / sample_variance
  alpha_prior <- sample_mean^2 / sample_variance

  return(list(alpha_prior = alpha_prior, beta_prior = beta_prior))
}

# Compute prior parameters
prior <- calculate_moments(observed_variances)
alpha_prior <- prior$alpha_prior
beta_prior <- prior$beta_prior

# Compute Bayesian weights
n <- nrow(replicates)
bayesian_weights <- bayesian_variance_estimation(observed_variances, n, alpha_prior, beta_prior)

# Histogram visualization of Bayesian weights
hist(bayesian_weights, main = "Histogram of Bayesian Adjusted Weights",
     xlab = "Adjusted Weight", ylab = "Frequency", col = "lightblue", border = "black")
```




```r
# Split Bayesian weights for early and late phases
bayesian_weights_early <- bayesian_weights[1:length(time_early)]
bayesian_weights_late  <- bayesian_weights[(length(bayesian_weights)-length(time_late)):(length(bayesia
```

```r
plot(observed_variances, 1/bayesian_weights)
```




```
print(alpha_prior)
```

```
## [1] 0.6563677
```

```
print(beta_prior)
```

```
## [1] 1.364639e-05
```

```
library(minpack.lm)
# Function to fit nlsLM using Bayesian weights
fit_nls_model_weighted <- function(t, y, initial_params, bayesian_weights) {
  fit <- try(nlsLM(
    y ~ asym_model_function(t, A_max, eta_max, A_min, eta_min, tau, phi),
    start = initial_params,
    weights = bayesian_weights,  # Incorporate Bayesian weights
    data = data.frame(t = t, y = y),
    lower = lower_bounds,  #   phi
    upper = upper_bounds
  ), silent = TRUE)

  if (inherits(fit, "try-error")) {
    return(NULL)
  }
```

```
  estimates <- coef(fit)
  conf_intervals <- confint2(fit, level = 0.99, method = 'asymptotic')
  return(list(estimates = estimates, conf_intervals = conf_intervals))
}
```

estimated parameters for Asymmetric model with time dependent variance!

```
# Fit models for early and late phases
fit_early_weighted <- fit_nls_model_weighted(time_early, y_early, init_param_asymm_early, bayesian_weigh
fit_late_weighted  <- fit_nls_model_weighted(time_late, y_late, init_param_asymm_late, bayesian_weights_

# Extract fitted parameters
params_early_weighted <- fit_early_weighted$estimates
params_late_weighted  <- fit_late_weighted$estimates

# Extract confidence intervals
conf_early_weighted <- fit_early_weighted$conf_intervals
conf_late_weighted  <- fit_late_weighted$conf_intervals

# Print estimated parameters
cat("Estimated Parameters with Bayesian Weights - Early Phase:\n")
```

```
## Estimated Parameters with Bayesian Weights - Early Phase:
```

```
print(params_early_weighted)
```

```
## A_max.(Intercept)   eta_max.t_peaks A_min.(Intercept) eta_min.t_valleys
##      2.574909e+03      5.026768e-03      3.268677e+02      1.202079e-03
##               tau   phi.(Intercept)
##      2.603120e-01     -2.636483e+00
```

```
cat("Estimated Parameters with Bayesian Weights - Late Phase:\n")
```

```
## Estimated Parameters with Bayesian Weights - Late Phase:
```

```
print(params_late_weighted)
```

```
## A_max.(Intercept)   eta_max.t_peaks A_min.(Intercept) eta_min.t_valleys
##      1.906792e+03      2.194398e-03      2.887954e+02      1.350450e-03
##               tau   phi.(Intercept)
##      2.583116e-01      3.968171e+00
```

```
fit_early_weighted
```

```
## $estimates
## A_max.(Intercept)   eta_max.t_peaks A_min.(Intercept) eta_min.t_valleys
##      2.574909e+03      5.026768e-03      3.268677e+02      1.202079e-03
##               tau   phi.(Intercept)
```

```
##      2.603120e-01     -2.636483e+00
##
## $conf_intervals
##                           0.5 %        99.5 %
## A_max.(Intercept)  2.508163e+03  2.641655e+03
## eta_max.t_peaks    4.554051e-03  5.499486e-03
## A_min.(Intercept)  3.080845e+02  3.456508e+02
## eta_min.t_valleys  2.999586e-04  2.104199e-03
## tau                2.598053e-01  2.608187e-01
## phi.(Intercept)   -2.664707e+00 -2.608259e+00
```

```
fit_late_weighted
```

```
## $estimates
## A_max.(Intercept)   eta_max.t_peaks A_min.(Intercept) eta_min.t_valleys
##      1.906792e+03      2.194398e-03      2.887954e+02      1.350450e-03
##               tau   phi.(Intercept)
##      2.583116e-01      3.968171e+00
##
## $conf_intervals
##                           0.5 %       99.5 %
## A_max.(Intercept) 1.879655e+03 1.933928e+03
## eta_max.t_peaks   2.147478e-03 2.241317e-03
## A_min.(Intercept) 2.795618e+02 2.980290e+02
## eta_min.t_valleys 1.243126e-03 1.457773e-03
## tau               2.582383e-01 2.583849e-01
## phi.(Intercept)   3.947123e+00 3.989218e+00
```

```
# Rename the parameter names
names(params_early_weighted) <- c("A_max", "eta_max", "A_min", "eta_min", "tau", "phi")
names(params_late_weighted) <- c("A_max", "eta_max", "A_min", "eta_min", "tau", "phi")

# Print the renamed parameters
cat("Renamed Estimated Parameters with Bayesian Weights - Early Phase:\n")
```

```
## Renamed Estimated Parameters with Bayesian Weights - Early Phase:
```

```
print(params_early_weighted)
```

```
##         A_max       eta_max         A_min       eta_min           tau
##  2.574909e+03  5.026768e-03  3.268677e+02  1.202079e-03  2.603120e-01
##           phi
## -2.636483e+00
```

```
cat("Renamed Estimated Parameters with Bayesian Weights - Late Phase:\n")
```

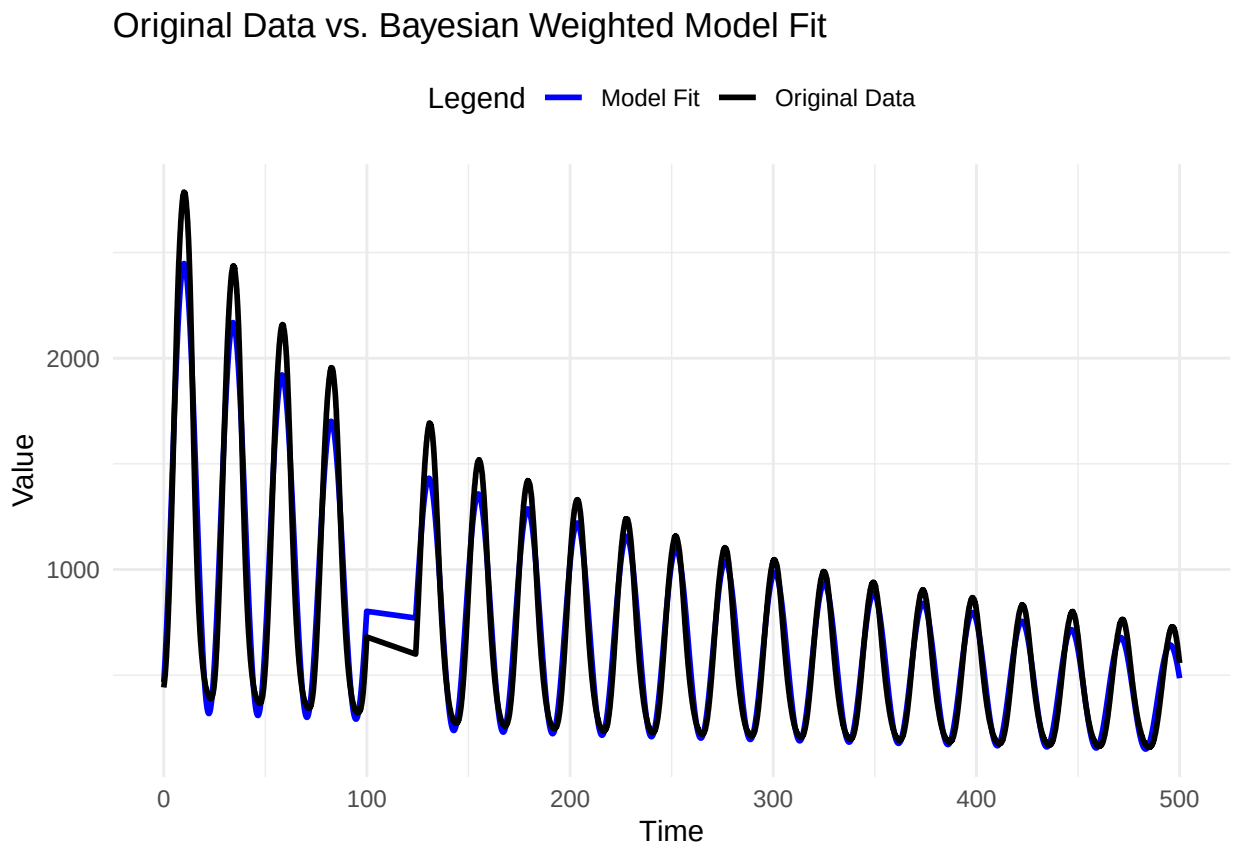
```
## Renamed Estimated Parameters with Bayesian Weights - Late Phase:
```

```
print(params_late_weighted)
```

```
##        A_max      eta_max        A_min      eta_min          tau          phi
## 1.906792e+03 2.194398e-03 2.887954e+02 1.350450e-03 2.583116e-01 3.968171e+00
```

plot curve generated by estimated parameters vs real data

```
library(ggplot2)
# Generate fitted values using the estimated parameters
y_fit_early <- asym_model_function(time_early, params_early_weighted["A_max"],
                                   params_early_weighted["eta_max"],
                                   params_early_weighted["A_min"],
                                   params_early_weighted["eta_min"],
                                   params_early_weighted["tau"],
                                   params_early_weighted["phi"])

y_fit_late <- asym_model_function(time_late, params_late_weighted["A_max"],
                                  params_late_weighted["eta_max"],
                                  params_late_weighted["A_min"],
                                  params_late_weighted["eta_min"],
                                  params_late_weighted["tau"],
                                  params_late_weighted["phi"])

# Create data frames for plotting
df_original <- data.frame(Time = c(time_early, time_late),
                          Value = c(y_early, y_late),
                          Type = "Original Data")

df_fitted <- data.frame(Time = c(time_early, time_late),
                        Value = c(y_fit_early, y_fit_late),
                        Type = "Model Fit")

df_plot <- rbind(df_original, df_fitted)

# Plot using ggplot
ggplot(df_plot, aes(x = Time, y = Value, color = Type)) +
  geom_line(size = 1) +
  scale_color_manual(values = c("blue", "black")) +  # Black for original, blue for model fit
  labs(title = "Original Data vs. Bayesian Weighted Model Fit",
       x = "Time", y = "Value", color = "Legend") +
  theme_minimal() +
  theme(legend.position = "top")
```



hypothesis Test (same method with before)

```
# Load necessary libraries
library(numDeriv)
library(nlme)

# Compute sigma² for Early & Late (with Bayesian weights)
sigma2_early_weighted <- estimate_sigma2_asym(y_early, time_early, params_early_weighted)
sigma2_late_weighted <- estimate_sigma2_asym(y_late, time_late, params_late_weighted)

cat("Sigma² for early (weighted):", sigma2_early_weighted, "\n")
```

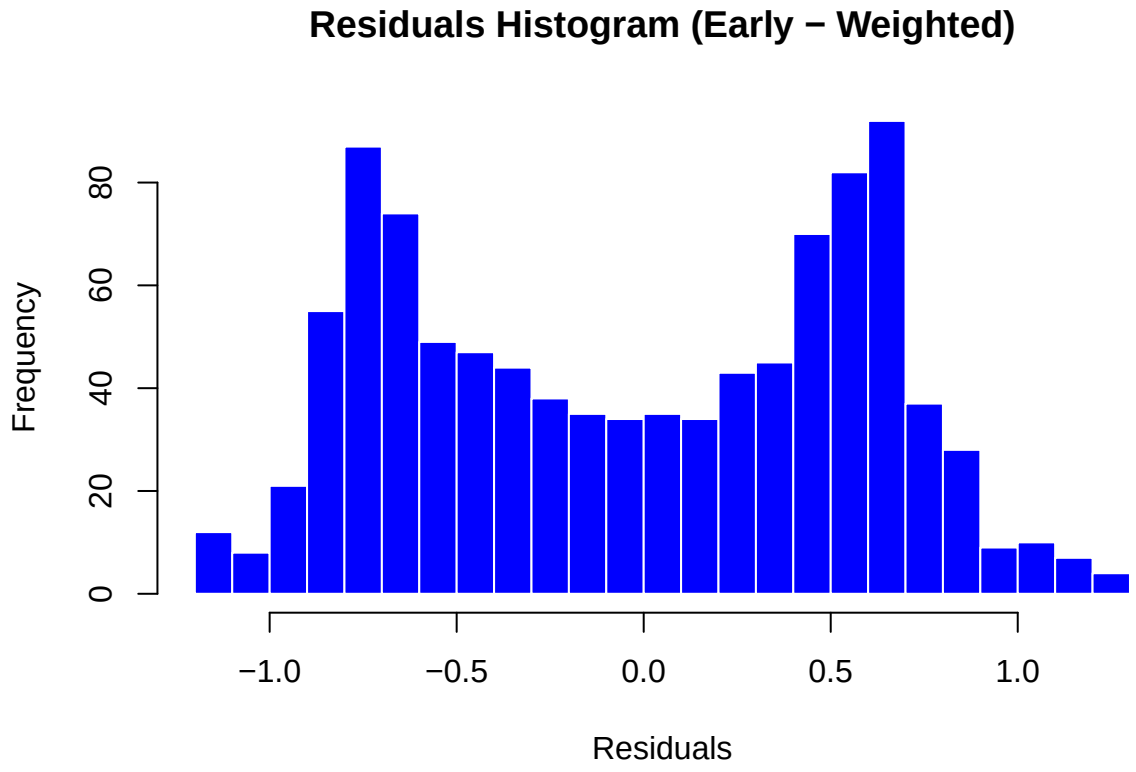
```
## Sigma² for early (weighted): 22097.57
```

```
cat("Sigma² for late (weighted):", sigma2_late_weighted, "\n")
```

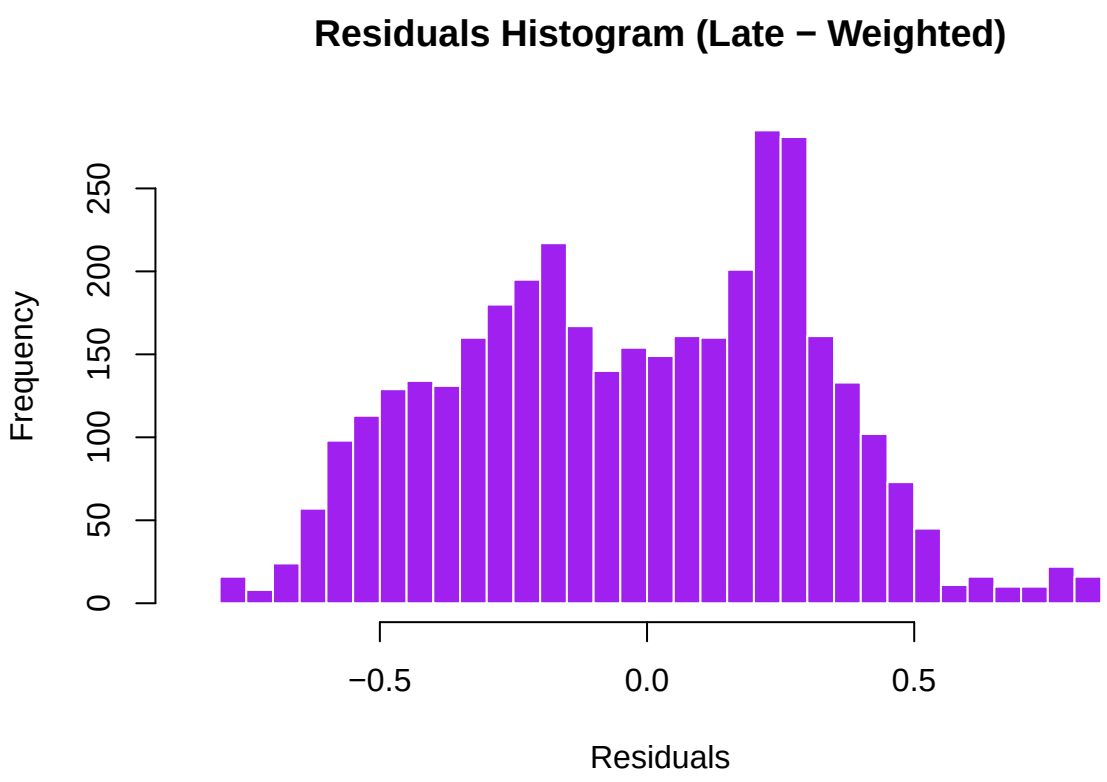
```
## Sigma² for late (weighted): 4074.678
```

```
# Compute residuals (with Bayesian weights)
residuals_early_weighted <- compute_residuals_asym(y_early, time_early, params_early_weighted) *sqrt(ba
residuals_late_weighted <- compute_residuals_asym(y_late, time_late, params_late_weighted) * sqrt(bayes
```

```r
# Plot residual histograms

hist(residuals_early_weighted, breaks = 30, col = "blue", border = "white",
     main = "Residuals Histogram (Early - Weighted)", xlab = "Residuals", ylab = "Frequency")
```




```r
hist(residuals_late_weighted, breaks = 30, col = "purple", border = "white",
     main = "Residuals Histogram (Late - Weighted)", xlab = "Residuals", ylab = "Frequency")
```



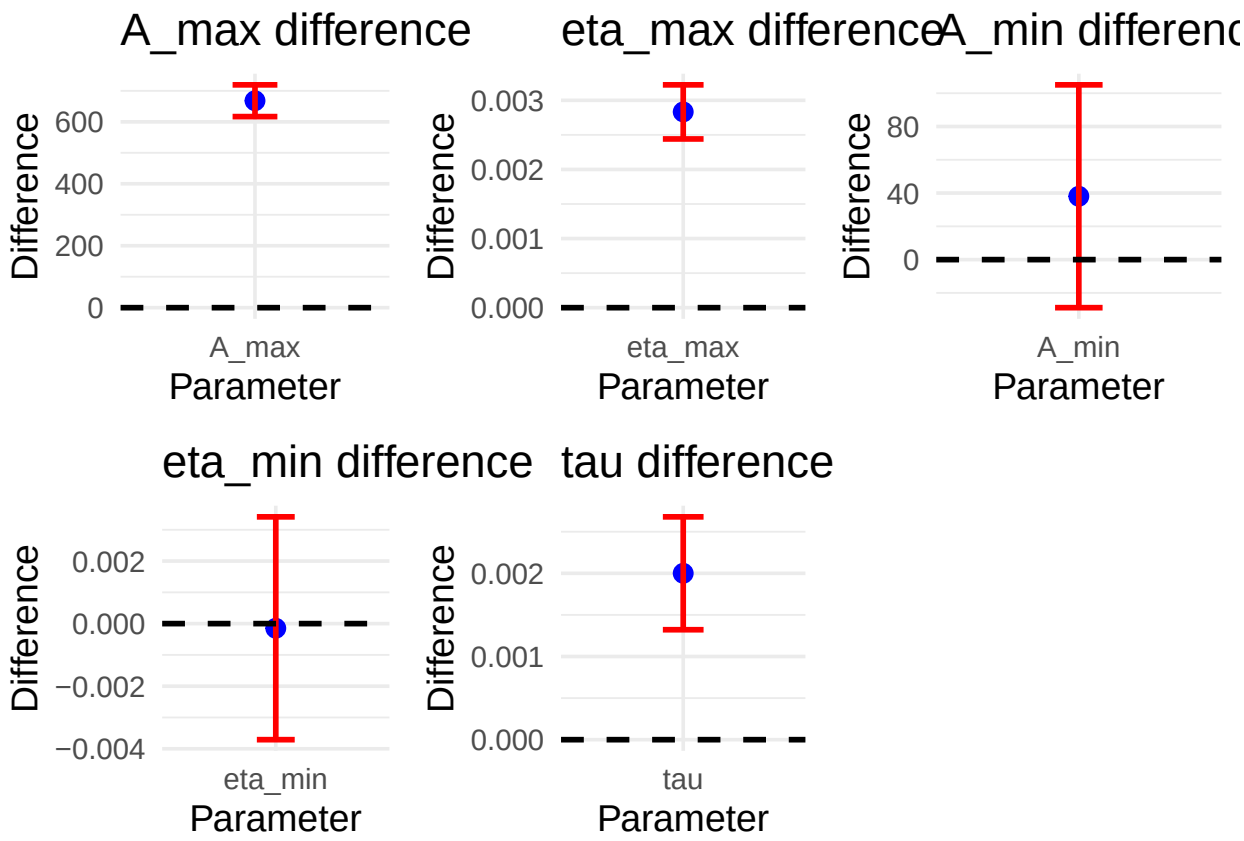

```r
# Compute covariance matrices (with Bayesian weights)
covariance_early_weighted <- compute_covariance_asym(params_early_weighted, time_early, y_early, sigma2_
covariance_late_weighted <- compute_covariance_asym(params_late_weighted, time_late, y_late, sigma2_late

# Function to remove phi from parameters and covariance matrix
remove_phi_asym <- function(params, covariance_matrix) {
  param_names <- names(params)
  phi_index <- which(param_names == "phi")

  # Remove phi
  params_reduced <- params[-phi_index]
  covariance_matrix_reduced <- covariance_matrix[-phi_index, -phi_index]

  return(list(params = params_reduced, covariance = covariance_matrix_reduced))
}

# Remove phi from early & late estimates (with Bayesian weights)
early_filtered_weighted <- remove_phi_asym(params_early_weighted, covariance_early_weighted)
late_filtered_weighted <- remove_phi_asym(params_late_weighted, covariance_late_weighted)

# Perform Wald Test (with Bayesian weights)
wald_result_weighted <- wald_test(early_filtered_weighted$params, late_filtered_weighted$params,
                                  early_filtered_weighted$covariance, late_filtered_weighted$covariance)

# Output Wald Test results (with Bayesian weights)
cat("Wald Test Statistic (Weighted):", wald_result_weighted$T_Wald, "\n")
```

```
## Wald Test Statistic (Weighted): 1549.436
```

```
cat("p-value (Weighted):", wald_result_weighted$p_value, "\n")
```

```
## p-value (Weighted): 0
```

```
library(ggplot2)
library(gridExtra)


weighted_param_diff <- early_filtered_weighted$params - late_filtered_weighted$params

var_diff_weighted <- diag(early_filtered_weighted$covariance + late_filtered_weighted$covariance)
se_diff_weighted <- sqrt(var_diff_weighted)  # (SE)

# 99% CI
Z_99 <- 2.576  #
CI_lower <- weighted_param_diff - Z_99 * se_diff_weighted
CI_upper <- weighted_param_diff + Z_99 * se_diff_weighted

#
df_weighted_diff <- data.frame(
  Parameter = names(weighted_param_diff),
  Difference = weighted_param_diff,
  CI_Lower = CI_lower,
  CI_Upper = CI_upper
)

#
plots <- lapply(1:nrow(df_weighted_diff), function(i) {
  ggplot(df_weighted_diff[i, ], aes(x = Parameter, y = Difference)) +
    geom_point(size = 3, color = "blue") +  #
    geom_errorbar(aes(ymin = CI_Lower, ymax = CI_Upper), width = 0.2, size = 1, color = "red") +  # CI
    geom_hline(yintercept = 0, linetype = "dashed", color = "black", size = 1) +  # 0
    labs(title = paste0(df_weighted_diff$Parameter[i], " difference"),
         x = "Parameter", y = "Difference") +
    theme_minimal(base_size = 14)
})

#
grid.arrange(grobs = plots, ncol = 3)  #
```



The plot below displays the imaginary part of the Fast Fourier Transform (FFT) of df_MC$X1, focusing on frequency indices 2 to 100.

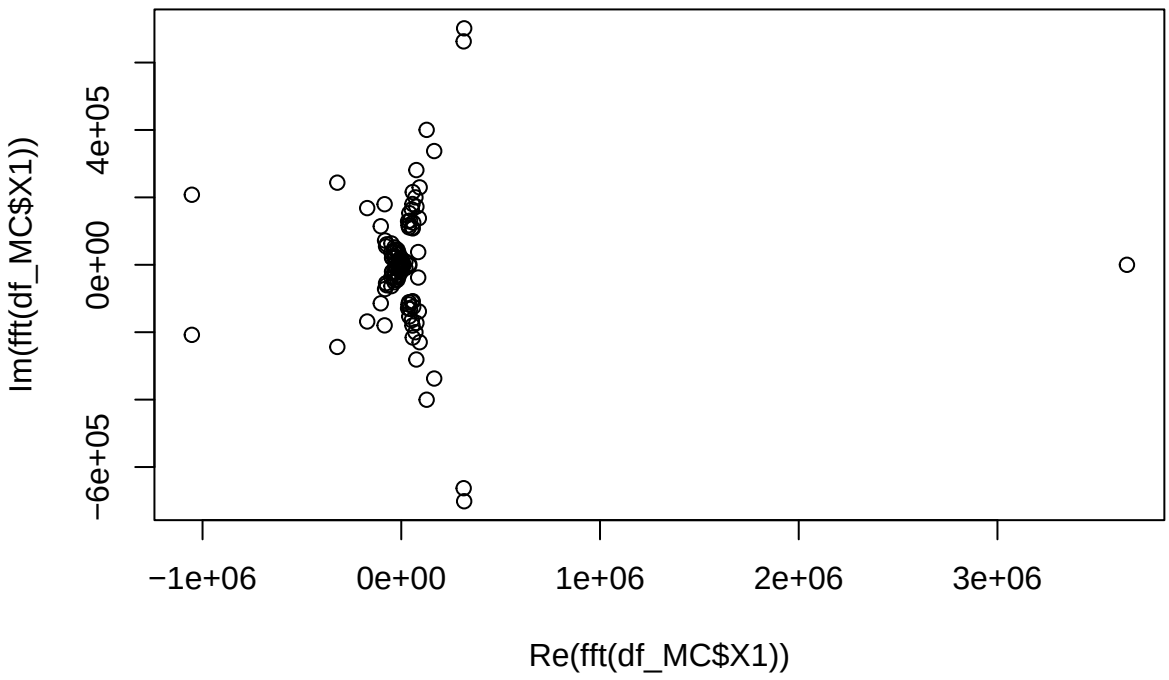
Low-frequency components (Index 1-45) show significant variation, indicating that the primary signal information is concentrated in this range.

High-frequency components (Index >50) approach zero, suggesting that higher frequencies contribute minimally to the signal.

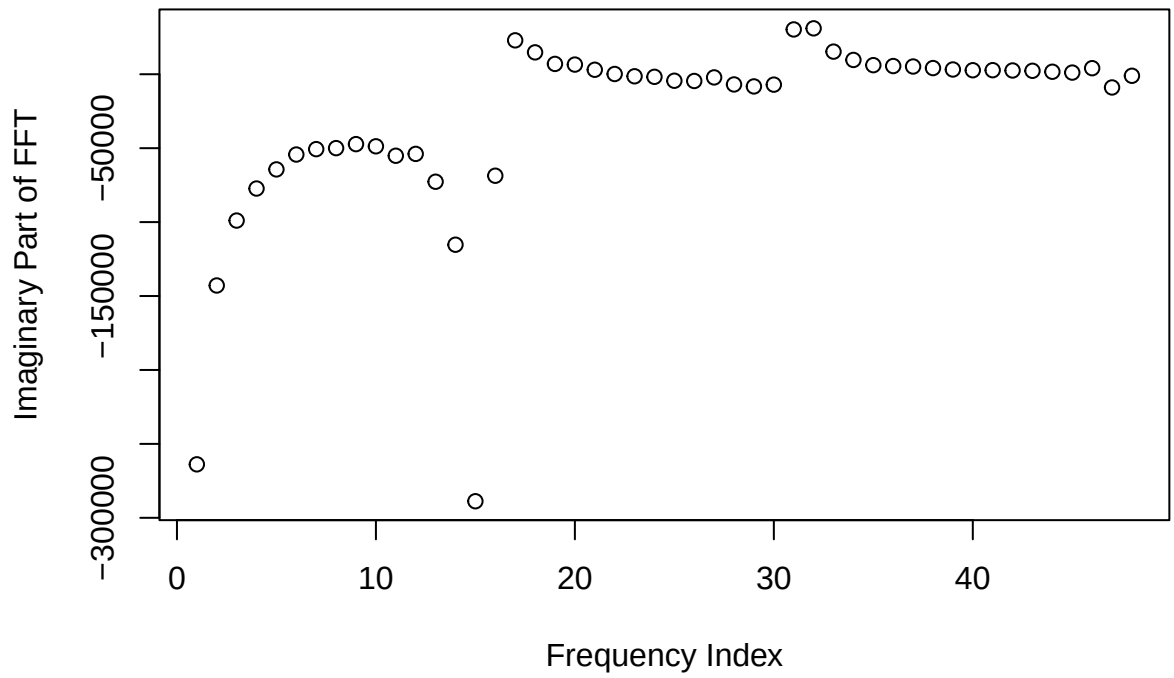
This result implies that the data is predominantly composed of low-frequency components, making it reasonable to retain only the first 45 frequency coefficients and discard higher frequencies to remove noise.

graphs show the reason I choose N=50 for time>124 and N=20 for time<100.

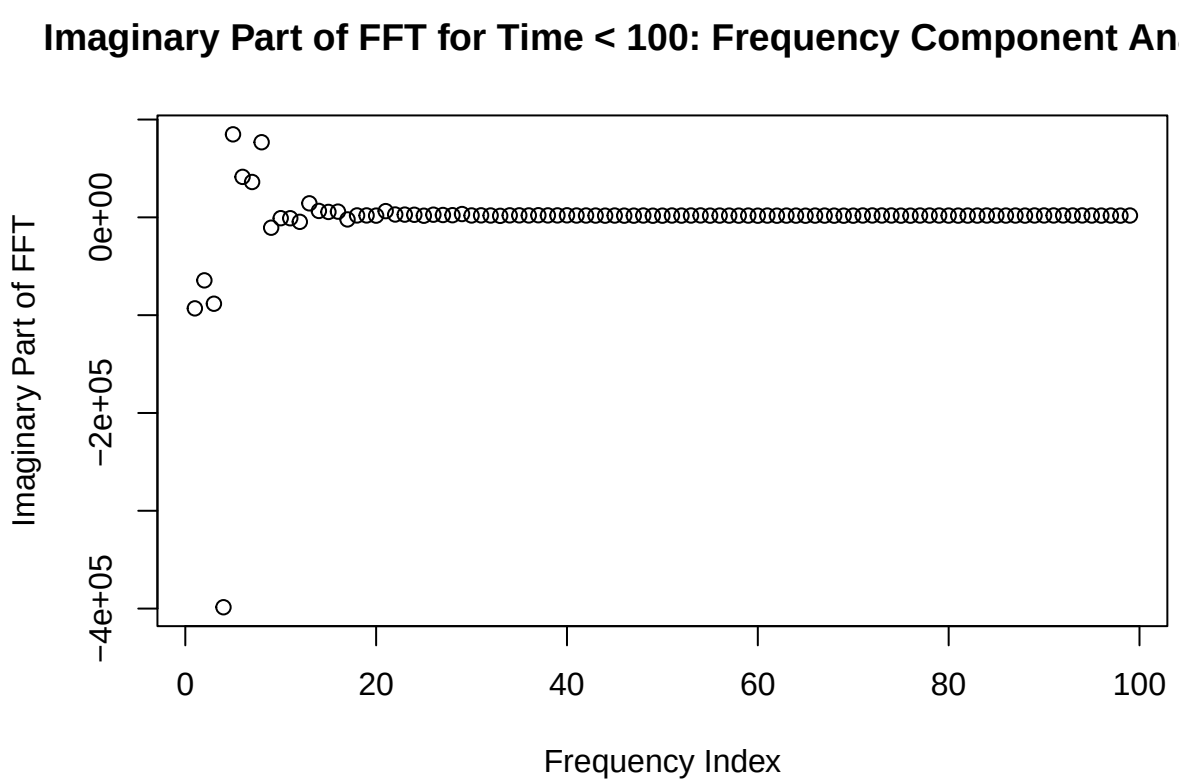
```
plot(fft(df_MC$X1))
```




```r
plot(Im(fft(df_MC$X1[1239:4999]))[2:49],type = "p",
     main = "Imaginary Part of FFT for Time > 124: Frequency Component Analysis",
     xlab = "Frequency Index",
     ylab = "Imaginary Part of FFT")
```




```
plot(Im(fft(df_MC$X1[0:1002]))[2:100],
     type = "p",
     main = "Imaginary Part of FFT for Time < 100: Frequency Component Analysis",
     xlab = "Frequency Index",
     ylab = "Imaginary Part of FFT")
```



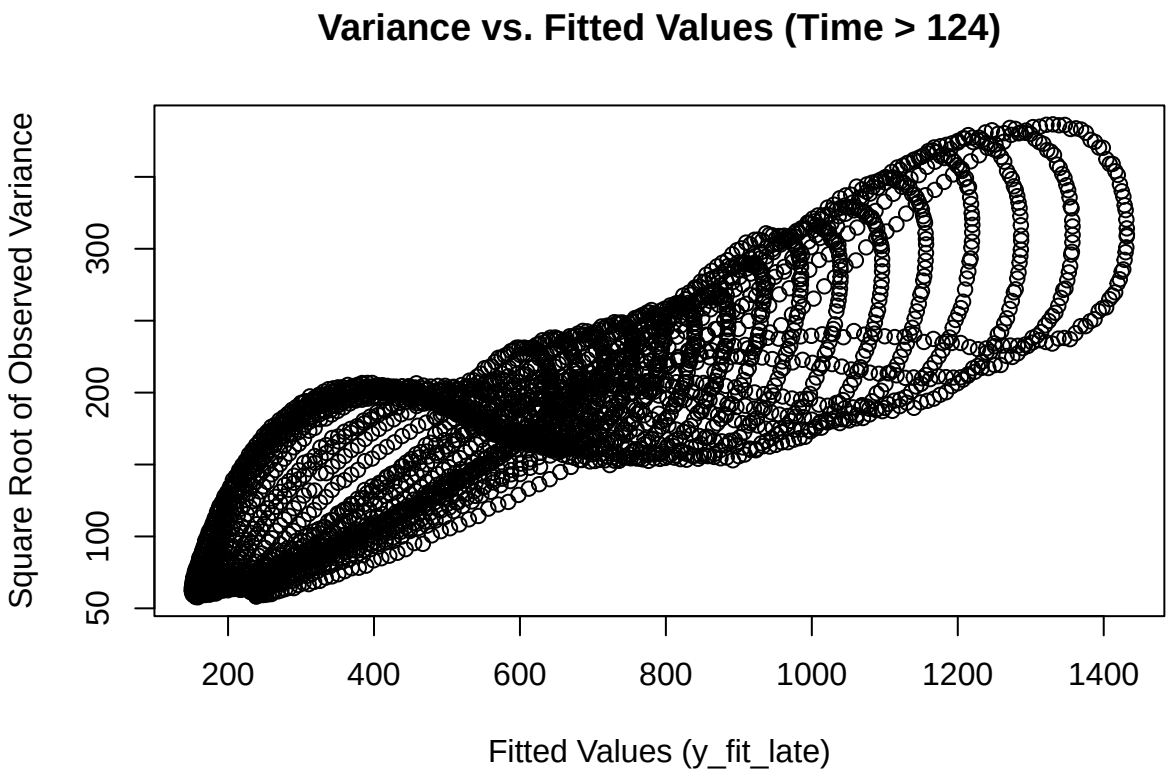
The data shows a clear periodic structure, forming spiral-like patterns, indicating that sqrt(observed_variances) is not randomly distributed. The presence of multiple oscillation cycles suggests that the variance contains multiple frequency components, not just a single 24h rhythm.(Fourier might help to find other frequency components) The variance depends on y_fit_late, meaning the residuals are not independent and have temporal correlation.

Reduce high-frequency noise: Low-pass filtering can remove noise, making residuals more stable.

```
length(y_fit_late)
```

```
## [1] 3761
```

```
plot(y_fit_late,sqrt(observed_variances)[1239:4999], main = "Variance vs. Fitted Values (Time > 124)",
     xlab = "Fitted Values (y_fit_late)",
     ylab = "Square Root of Observed Variance")
```




```
xxx=y_fit_late
yyy=sqrt(observed_variances)[1239:4999]
lm(yyy~xxx)
```

```
##
## Call:
## lm(formula = yyy ~ xxx)
##
## Coefficients:
## (Intercept)          xxx
##     55.3866       0.2021
```

```
lm(yyy~xxx)
```

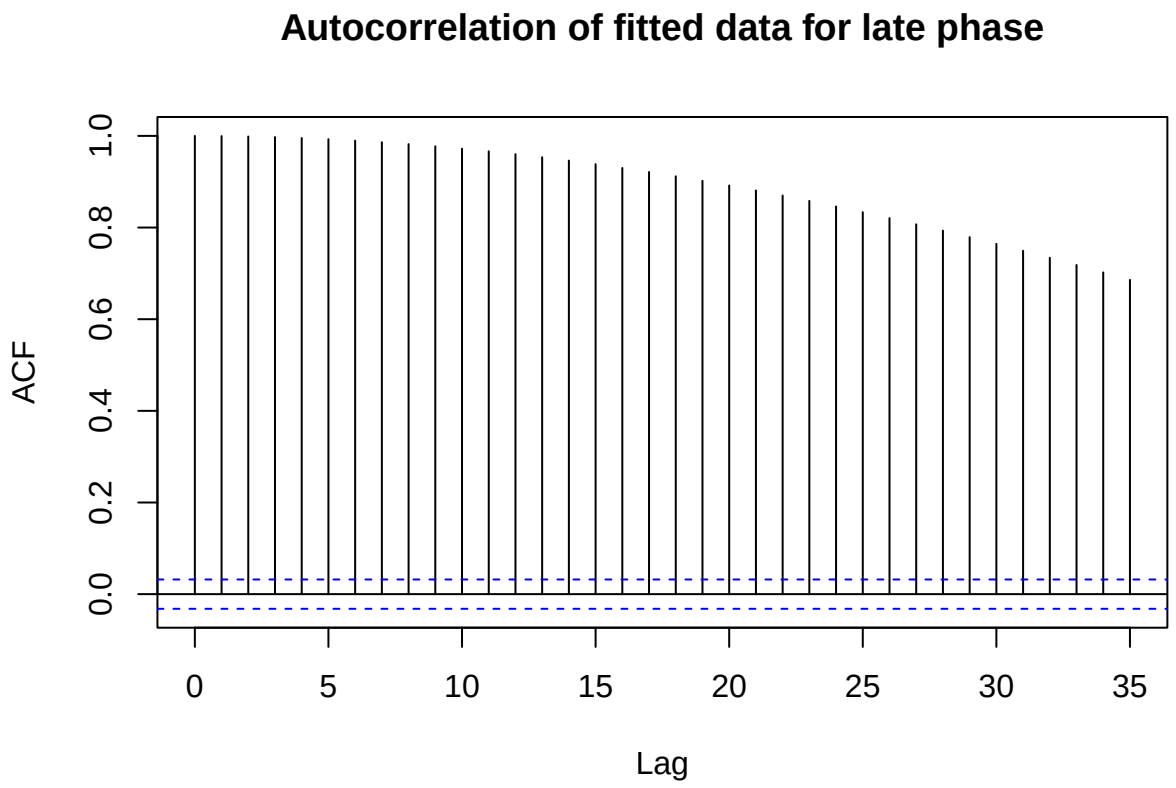
```
##
## Call:
## lm(formula = yyy ~ xxx)
##
## Coefficients:
## (Intercept)          xxx
##     55.3866       0.2021
```

plots generated by acf() illustrate fft() helps decorrelation

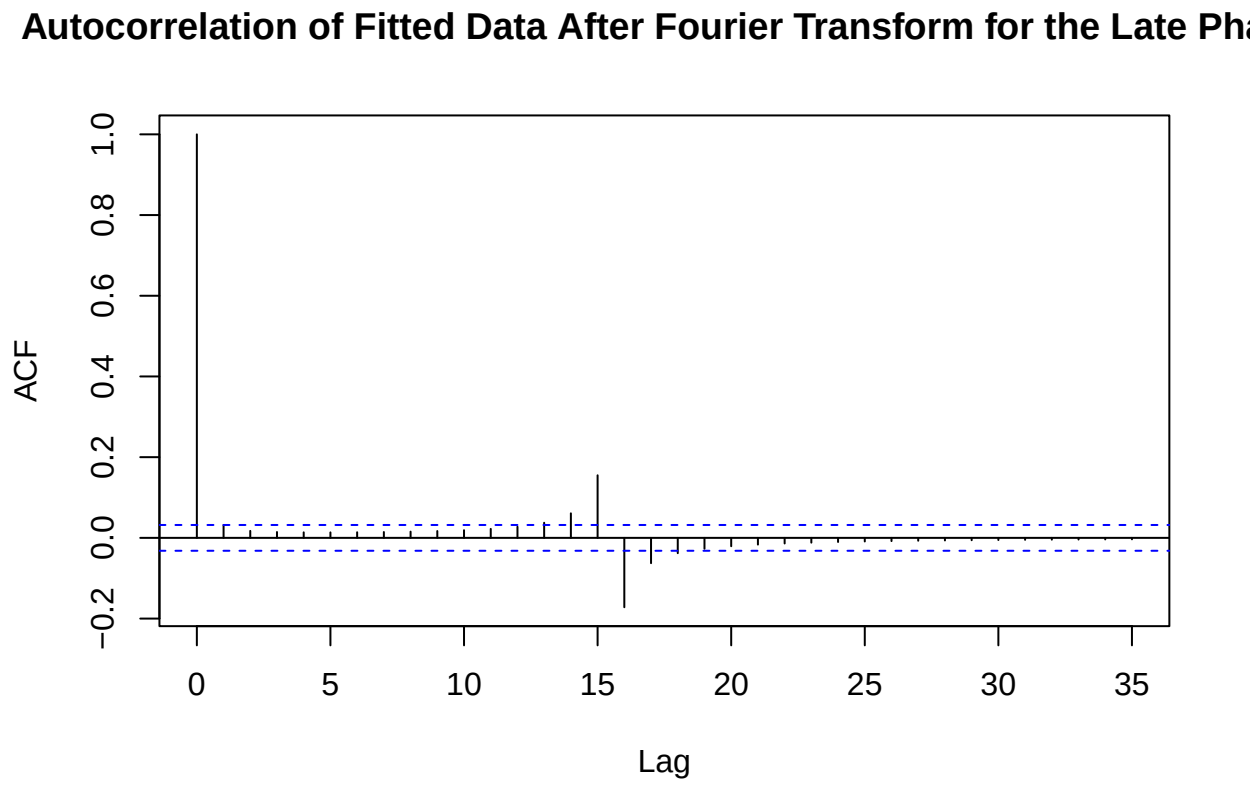
Meaning of acf(y_fit_late) In R, acf(y_fit_late) calculates and plots the autocorrelation function (ACF) of y_fit_late, which measures the correlation of a time series with itself at different lags.

X-axis (Lag): Represents the lag order (number of time steps). Y-axis (ACF): Indicates the correlation of y_fit_late with itself at a given lag (ranging from -1 to 1). Black vertical lines: Represent the autocorrelation coefficients at different lags (the taller the line, the stronger the correlation). Blue dashed lines: Represent significance thresholds (values exceeding these lines indicate statistically significant correlation, typically at a 95% confidence interval). Role of ACF in Your Analysis If ACF decreases slowly (as seen in your first plot), it suggests that y_fit_late exhibits long memory, possibly a stationary series with a trend. If ACF rapidly declines, it indicates that y_fit_late may be a short-range dependent process, such as an AR(1) process. If ACF exhibits periodic fluctuations, it suggests that y_fit_late has an underlying periodic structure. Your second plot, acf(Re(fft(y_fit_late))), computes the ACF of the real part of the Fourier transform (FFT) of y_fit_late. The results show almost no autocorrelation, which could mean that FFT(y_fit_late) has removed the time-dependent structure.

```
acf(y_fit_late,main = "Autocorrelation of fitted data for late phase")
```




```
acf(Re(fft(y_fit_late)),main = "Autocorrelation of Fitted Data After Fourier Transform for the Late Phas
```



simulation about Fourier Transform

about asymmetric model with constant variance get coverage for each estimated parameters.

```
# Define function to perform FFT and extract first N frequency components
fft_extract <- function(y, N) {
  y_fft <- fft(y)
  features <- c(Re(y_fft)[1:(N + 1)], Im(y_fft)[2:(N + 1)])
  return(features)
}

# Define function to fit FFT-based model
fit_fft_model <- function(y_fft, t, N, init_params) {
  fit <- try(nlsLM(
    y_fft ~ fft_extract(asym_model_function(t, A_max, eta_max, A_min, eta_min, tau, phi), N),
    start = init_params,
    data = data.frame(y_fft = y_fft),
    lower = lower_bounds,  #   phi
    upper = upper_bounds,
    control = nls.lm.control(maxiter = 500)

  ), silent = TRUE)

  if (inherits(fit, "try-error")) {
```

```
    return(NULL)
  }
  estimates <- coef(fit)
  conf_intervals <- confint2(fit, level = 0.99, method = 'asymptotic')
  return(list(estimates = estimates, conf_intervals = conf_intervals))
}

# Run simulations to compute coverage
run_fft_simulations <- function(n_simulations = 100, sigma = 10, N = 20) {
  true_params <- c(A_max = 2000, eta_max = 0.005, A_min = 200, eta_min = 0.0025, tau = 0.7, phi = 0)
  t <- seq(0, 100, by = 0.1)

  results <- vector("list", n_simulations)
  for (i in 1:n_simulations) {
    simulated_data <- simulate_asymm_func(t, true_params, sigma)
    y_fft <- fft_extract(simulated_data, N)
    initial_params <- find_initial_params(t, simulated_data)
    fit_result <- fit_fft_model(y_fft, t, N, initial_params)
    results[[i]] <- fit_result
  }
  # Remove NULL results
  results <- Filter(Negate(is.null), results)
  # Compute coverage
  coverage_counts <- rep(0, length(true_params))
  for (result in results) {
    conf_intervals <- result$conf_intervals
    for (j in 1:length(true_params)) {
      if (true_params[j] >= conf_intervals[j, 1] && true_params[j] <= conf_intervals[j, 2]) {
        coverage_counts[j] <- coverage_counts[j] + 1
      }
    }
  }
  coverage_proportions <- coverage_counts / n_simulations
  return(coverage_proportions)
}
# Run FFT-based coverage simulation
set.seed(47)
fft_coverage_results <- run_fft_simulations(n_simulations = 100, sigma = 10, N = 20)
# Print results
print(fft_coverage_results)
```

```
## [1] 1.00 1.00 1.00 1.00 0.99 0.84
```

```
init_param_asymm_early <- list(
    A_max = as.numeric(init_param_asymm_early$A_max),
    eta_max = as.numeric(init_param_asymm_early$eta_max),
    A_min = as.numeric(init_param_asymm_early$A_min),
    eta_min = as.numeric(init_param_asymm_early$eta_min),
    tau = as.numeric(init_param_asymm_early$tau),
    phi = as.numeric(init_param_asymm_early$phi)
)
```

Apply FFT on Real Data (Constant Variance Model)

Early Phase (Time < 100)

For the early phase ($t < 100$), we keep the first **20 cosine (real) parts** and **21 sine (imaginary) parts** in the FFT analysis.

late Phase (Time < 100)

For the early phase ($t > 124$), we keep the first **50 cosine (real) parts** and **51 sine (imaginary) parts** in the FFT analysis.

```
# Apply FFT-based fitting on real data(time<100)
N_early <- 20  # Choosing first 20 components
y_early.fft <- fft_extract(y_early, N_early)

fit_fft <- fit_fft_model(y_early.fft, time_early, N_early, init_param_asymm_early)

# Check if fitting was successful
if (!is.null(fit_fft)) {
  print(summary(fit_fft))  # Print summary of fit

  # Extract estimated parameters
  fft_params_early <- fit_fft$estimates
  print("Estimated Parameters:")
  print(fft_params_early)

  # Compute confidence intervals
  conf_int <- fit_fft$conf_intervals
  print("99% Confidence Intervals:")
  print(conf_int)
} else {
  print("nlsLM fitting in frequency domain failed.")
}
```

```
##                Length Class  Mode
## estimates       6     -none- numeric
## conf_intervals 12     -none- numeric
## [1] "Estimated Parameters:"
##         A_max       eta_max         A_min       eta_min           tau
##  2.782296e+03  5.153225e-03  1.521357e+02 -4.430530e-03  2.600980e-01
##           phi
## -2.613771e+00
## [1] "99% Confidence Intervals:"
##                 0.5 %        99.5 %
## A_max    2.583941e+03  2.980652e+03
## eta_max  3.636517e-03  6.669933e-03
## A_min   -1.084068e+01  3.151120e+02
## eta_min -2.050024e-02  1.163918e-02
## tau      2.574598e-01  2.627363e-01
## phi     -2.740293e+00 -2.487250e+00
```

```r
y_early.fft <- fft_extract(y_early, N_early)
y_early.fft
```

```
##  [1] 1180154.50000    4305.16120   -6443.15409  -16348.92973 -290678.53323
##  [6]  -10139.03641   -7529.55871  -11032.50581  -22178.53612   17455.41683
## [11]    6197.97261    1059.86765   -9382.41651    5967.06901    3706.59093
## [16]    3122.90841    3038.45563    1501.97167     792.53593      51.72905
## [21]    -486.48916  -92448.83596  -63457.11774  -86354.66984 -393276.59684
## [26]   79952.93624   39092.04403   34635.15943   77610.22672  -11014.23705
## [31]   -1585.16824   -1525.76128   -4884.77756   12923.74327    5751.94242
## [36]    4793.99752    5275.32151   -2702.46847    1184.46741    1230.91632
## [41]     984.39819
```

```r
# Apply FFT-based fitting on real data(time>124)
N_late <- 50  # Choosing first 50 components
y_late.fft <- fft_extract(y_late, N_late)

fit_fft_late <- fit_fft_model(y_late.fft, time_late, N_late, init_param_asymm_late)

# Check if fitting was successful
if (!is.null(fit_fft)) {
  print(summary(fit_fft_late))  # Print summary of fit

  # Extract estimated parameters
  estimated_params_fft_late <- fit_fft_late$estimates
  print("Estimated Parameters:")
  print(estimated_params_fft_late)

  # Compute confidence intervals
  conf_intervals_fft_late <- fit_fft_late$conf_intervals
  print("95% Confidence Intervals:")
  print(conf_intervals_fft_late)
} else {
  print("nlsLM fitting in frequency domain failed.")
}
```

```
##                Length Class  Mode
## estimates       6     -none- numeric
## conf_intervals 12     -none- numeric
## [1] "Estimated Parameters:"
## A_max.(Intercept)   eta_max.t_peaks A_min.(Intercept) eta_min.t_valleys
##      2.053249e+03      2.329516e-03      1.951925e+02      8.413235e-04
##               tau   phi.(Intercept)
##      2.583283e-01      3.949971e+00
## [1] "95% Confidence Intervals:"
##                            0.5 %       99.5 %
## A_max.(Intercept)  1.920934e+03 2.185564e+03
## eta_max.t_peaks    2.098034e-03 2.560998e-03
## A_min.(Intercept)  9.533256e+01 2.950525e+02
## eta_min.t_valleys -8.042597e-04 2.486907e-03
## tau                2.578604e-01 2.587962e-01
## phi.(Intercept)    3.822352e+00 4.077589e+00
```

```r
names(fit_fft_late$estimates) <- c("A_max", "eta_max", "A_min", "eta_min", "tau", "phi")
```

```r
fft_params_late = fit_fft_late$estimates
```

```r
fft_params_late
```

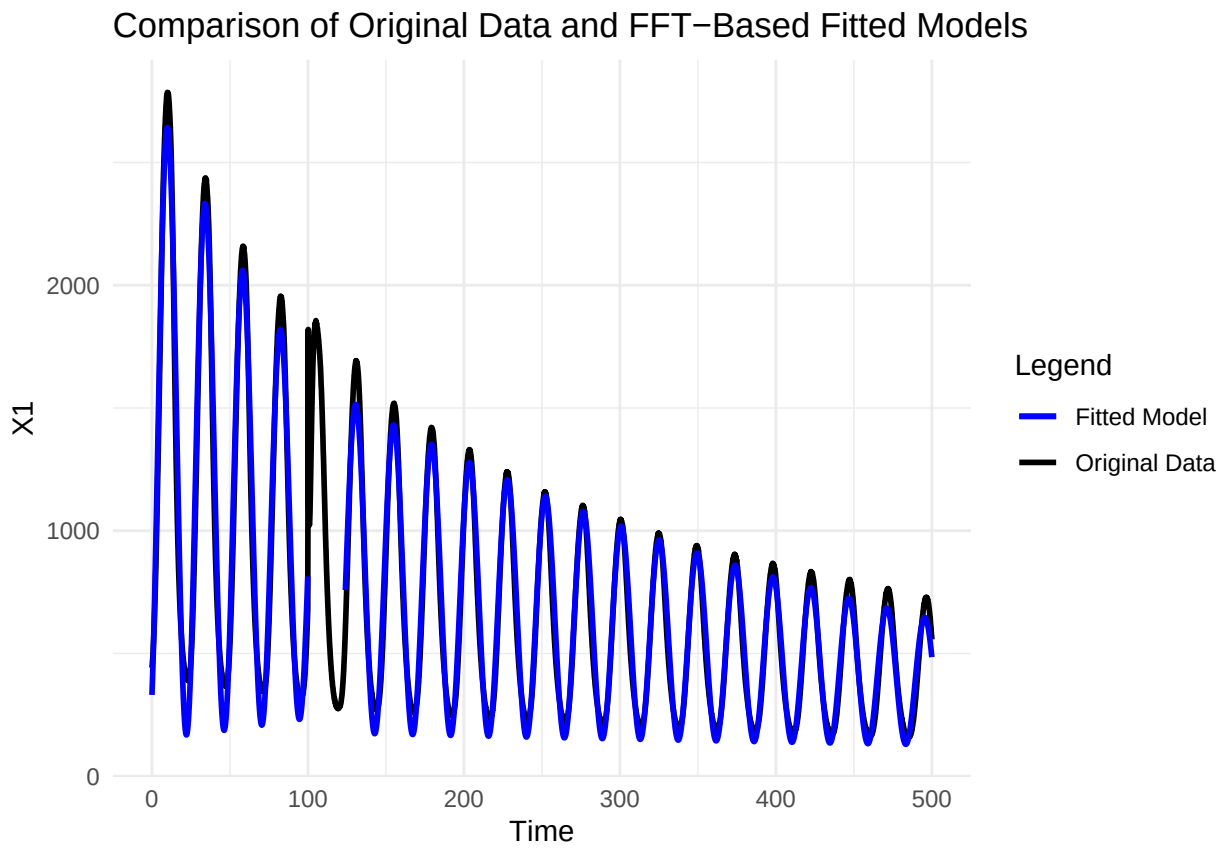
```
##        A_max      eta_max        A_min      eta_min          tau          phi
## 2.053249e+03 2.329516e-03 1.951925e+02 8.413235e-04 2.583283e-01 3.949971e+00
```

Plot fitted curve and original curve to compare

```r
fft_params_early
```

```
##         A_max       eta_max         A_min       eta_min           tau
##  2.782296e+03  5.153225e-03  1.521357e+02 -4.430530e-03  2.600980e-01
##           phi
## -2.613771e+00
```

```r
# Generate model curves using estimated parameters
fft_y_fit_early <- asym_model_function(time_early, fft_params_early["A_max"], fft_params_early["eta_max
                                    fft_params_early["A_min"], fft_params_early["eta_min"],
                                   fft_params_early["tau"], fft_params_early["phi"])

fft_y_fit_late <- asym_model_function(
  time_late,
  as.numeric(fit_fft_late$estimates["A_max"]),
  as.numeric(fit_fft_late$estimates["eta_max"]),
  as.numeric(fit_fft_late$estimates["A_min"]),
  as.numeric(fit_fft_late$estimates["eta_min"]),
  as.numeric(fit_fft_late$estimates["tau"]),
  as.numeric(fit_fft_late$estimates["phi"])
)

df_fit_early <- data.frame(Time = time_early, X1 = fft_y_fit_early, Model = "Fitted Model")
df_fit_late <- data.frame(Time = time_late, X1 = fft_y_fit_late, Model = "Fitted Model")

ggplot() +
  geom_line(data = df_MC, aes(x = Time, y = X1, color = "Original Data"), size = 1) +
  geom_line(data = df_fit_early, aes(x = Time, y = X1, color = Model), size = 1) +
  geom_line(data = df_fit_late, aes(x = Time, y = X1, color = Model), size = 1) +
  scale_color_manual(values = c("Original Data" = "black", "Fitted Model" = "blue")) +
  labs(title = "Comparison of Original Data and FFT-Based Fitted Models", x = "Time", y = "X1", color =
  theme_minimal()
```



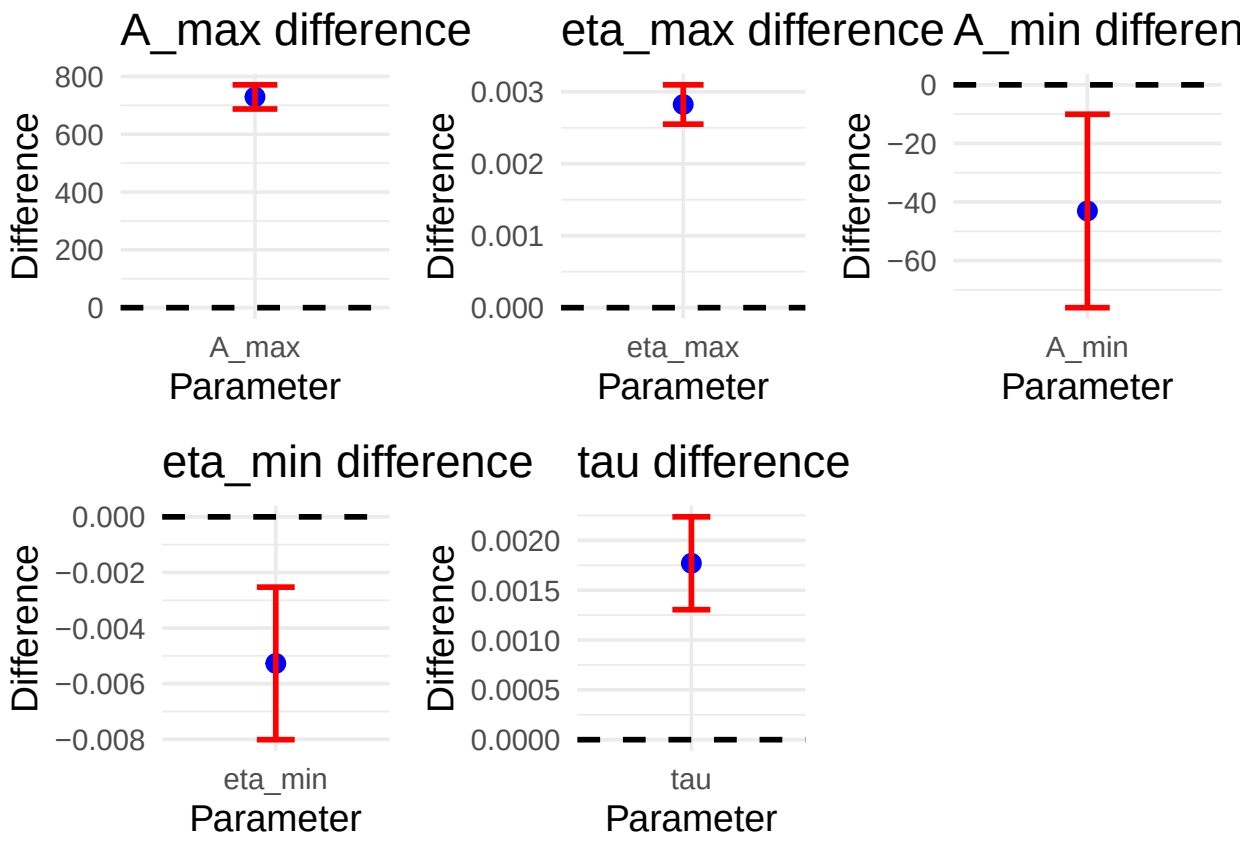

```r
#   sigma²
sigma2_early_fft <- estimate_sigma2_asym(y_early, time_early, fft_params_early)
sigma2_late_fft <- estimate_sigma2_asym(y_late, time_late, fft_params_late)

#
covariance_early_fft <- compute_covariance_asym(fft_params_early, time_early, y_early, sigma2_early_fft)
covariance_late_fft <- compute_covariance_asym(fft_params_late, time_late, y_late, sigma2_late_fft)

#   phi
early_filtered_fft <- remove_phi_asym(fft_params_early, covariance_early_fft)
late_filtered_fft <- remove_phi_asym(fft_params_late, covariance_late_fft)

#   Wald
wald_result_fft <- wald_test(
  early_filtered_fft$params, late_filtered_fft$params,
  early_filtered_fft$covariance, late_filtered_fft$covariance
)

cat("fft Wald Test Statistic :", wald_result_fft$T_Wald, "\n")
```

```
## fft Wald Test Statistic : 2410.584
```

```r
cat("p-value :", wald_result_fft$p_value, "\n")
```

```
## p-value : 0
```

```r
#
fft_param_names <- names(early_filtered_fft$params)
fft_param_diff <- early_filtered_fft$params - late_filtered_fft$params
var_diff <- diag(early_filtered_fft$covariance + late_filtered_fft$covariance)  #
se_diff <- sqrt(var_diff)  #
CI_lower <- fft_param_diff - Z_99 * se_diff
CI_upper <- fft_param_diff + Z_99 * se_diff

#
df_diff_fft <- data.frame(
  Parameter = fft_param_names,
  Difference = fft_param_diff,
  CI_Lower = CI_lower,
  CI_Upper = CI_upper
)

library(gridExtra)

#
plots <- lapply(1:nrow(df_diff_fft), function(i) {
  ggplot(df_diff_fft[i, ], aes(x = Parameter, y = Difference)) +
    geom_point(size = 3, color = "blue") +  #
    geom_errorbar(aes(ymin = CI_Lower, ymax = CI_Upper), width = 0.2, size = 1, color = "red") +  #
    geom_hline(yintercept = 0, linetype = "dashed", color = "black", size = 1) +  # 0
    labs(title = paste0(df_diff_fft$Parameter[i], " difference"),
         x = "Parameter", y = "Difference") +
    theme_minimal(base_size = 14)
})

#   gridExtra
grid.arrange(grobs = plots, ncol = 3)  # 3
```



Simulation for Asymmetric Model with Time-Dependent Variance

In this simulation, the **true parameters** remain the same as in previous models.
We simulate **4 replicates** with:

$$t \leftarrow \text{seq}(0, 100, \text{by} = 0.1)$$

where the variance of the simulated data at each time point is defined as:

$$\sigma_t = 10 + 0.2 \times f_{\text{true}}$$

Here, σ_t depends on f_{true}, which represents the true function values.

Each simulated replicate is transformed into the frequency domain using the `fft_extract()` method (previously defined).
We then apply `calculate_variance()` to estimate the variance between the 4 replicates at each frequency.
Finally, we use 1/**variance as weights** in the nonlinear least squares model (`nlsLM`).

```
# Load required libraries
library(pracma)
library(dplyr)
library(tidyr)
library(gridExtra)
# Function to fit FFT-based model with weights
```

```r
weighted_fit_fft_model <- function(y_fft, t, N, init_params, weights) {
  fit <- try(nlsLM(
    y_fft ~ fft_extract(asym_model_function(t, A_max, eta_max, A_min, eta_min, tau, phi), N),
    start = init_params,
    weights = weights,
    lower = lower_bounds,  #
    upper = upper_bounds,
    data = data.frame(y_fft = y_fft),
    control = nls.lm.control(maxiter = 1000)

  ), silent = TRUE)

  if (inherits(fit, "try-error")) {
    return(NULL)
  }
  estimates <- coef(fit)
  conf_intervals <- confint2(fit, level = 0.99, method = 'asymptotic')
  return(list(estimates = estimates, conf_intervals = conf_intervals))
}
set.seed(28)

# Function to simulate data and compute coverage for Asymmetric Model with Time Dependent Variance
run_asym_time_dep_simulations <- function(n_simulations = 100, sigma_base = 10, scale_factor = 0.2,
                                          N = 20, n_replicates = 4) {
  true_params <- c(A_max = 2000, eta_max = 0.005, A_min = 200, eta_min = 0.0025, tau = 0.7, phi = 0)
  t <- seq(0, 100, by = 0.1)
  f_true <- asym_model_function(t, true_params["A_max"], true_params["eta_max"],
                                true_params["A_min"], true_params["eta_min"],
                                true_params["tau"], true_params["phi"])

  results <- vector("list", n_simulations)
  for (i in 1:n_simulations) {
    # Generate time-dependent noise variance
    sigma_t <- sigma_base + scale_factor * f_true  # _t depends on f_true
    replicates <- matrix(NA, nrow = length(t), ncol = n_replicates)

    for (j in 1:n_replicates) {
      replicates[, j] <- f_true + rnorm(length(t), mean = 0, sd = sigma_t)
    }

    init_param <- find_initial_params(t, rowMeans(replicates))

    # Compute FFT for each replicate
    fft_replicates <- apply(replicates, 2, function(y) fft_extract(y, N))

    # Compute variance in frequency domain
    observed_variances <- calculate_variance(fft_replicates)

    # Compute weights as 1/variance
    weights <- 1 / observed_variances
    #weights[is.infinite(weights)] <- max(weights[!is.infinite(weights)])  # Handle division by zero

    # Perform FFT extraction and fit weighted model
```

```
    y_mean <- rowMeans(fft_replicates)  # Mean of replicates in frequency domain
    fit_result <- weighted_fit_fft_model(y_mean, t, N, init_param, weights)
    results[[i]] <- fit_result
  }

  # Remove NULL results
  results <- Filter(Negate(is.null), results)

  # Compute coverage
  coverage_counts <- rep(0, length(true_params))
  for (result in results) {
    conf_intervals <- result$conf_intervals
    for (j in 1:length(true_params)) {
      if (true_params[j] >= conf_intervals[j, 1] && true_params[j] <= conf_intervals[j, 2]) {
        coverage_counts[j] <- coverage_counts[j] + 1
      }
    }
  }
  coverage_proportions <- coverage_counts / n_simulations
  return(coverage_proportions)
}

# Run simulation for Asymmetric Model with Time Dependent Variance
asym_time_dep_coverage_results <- run_asym_time_dep_simulations(n_simulations = 100, sigma_base = 10, s
                                                                N = 20, n_replicates = 4)

# Print results
print(asym_time_dep_coverage_results)
```

```
## [1] 0.66 0.65 0.69 0.68 0.33 0.47
```

apply method on real data

```
y_early <- replicates[df_MC$Time < 100, ]
y_late <- replicates[df_MC$Time >= 124, ]

# Define FFT components for early and late phases
N_early <- 20
N_late <- 50

# Compute FFT for each replicate in early phase
fft_replicates_early <- apply(y_early, 2, function(y) fft_extract(y, N_early))
observed_variances_early <- sqrt(calculate_variance(fft_replicates_early))
weights_early <- 1 / observed_variances_early
weights_early[is.infinite(weights_early)] <- max(weights_early[!is.infinite(weights_early)])
y_mean_early <- rowMeans(fft_replicates_early)

weighted_fft_fit_early <- weighted_fit_fft_model(y_mean_early, time_early, N_early,
                                                 init_param_asymm_early, sqrt(weights_early))

# Compute FFT for each replicate in late phase
```

```r
fft_replicates_late <- apply(y_late, 2, function(y) fft_extract(y, N_late))
observed_variances_late <- sqrt(calculate_variance(fft_replicates_late))
weights_late <- 1 / observed_variances_late
weights_late[is.infinite(weights_late)] <- max(weights_late[!is.infinite(weights_late)])
y_mean_late <- rowMeans(fft_replicates_late)

weighted_fft_fit_late <- weighted_fit_fft_model(y_mean_late, time_late, N_late,
                                                init_param_asymm_early, sqrt(weights_late))

# Store estimated parameters and confidence intervals
weighted_fft_estimated_params_early <- weighted_fft_fit_early$estimates
weighted_fft_conf_intervals_early <- weighted_fft_fit_early$conf_intervals

weighted_fft_estimated_params_late <- weighted_fft_fit_late$estimates
weighted_fft_conf_intervals_late <- weighted_fft_fit_late$conf_intervals

# Print results
print("Early Phase Estimated Parameters:")
```

```
## [1] "Early Phase Estimated Parameters:"
```

```r
print(weighted_fft_estimated_params_early)
```

```
##         A_max       eta_max         A_min       eta_min           tau
##  3.014461e+03  5.175625e-03  2.658459e+02 -1.444973e-03  2.604960e-01
##           phi
## -2.492798e+00
```

```r
print("Early Phase Confidence Intervals:")
```

```
## [1] "Early Phase Confidence Intervals:"
```

```r
print(weighted_fft_conf_intervals_early)
```

```
##                 0.5 %        99.5 %
## A_max    2.808524e+03  3.220397e+03
## eta_max  3.876717e-03  6.474533e-03
## A_min    7.668949e+01  4.550022e+02
## eta_min -1.137134e-02  8.481394e-03
## tau      2.580605e-01  2.629315e-01
## phi     -2.615019e+00 -2.370577e+00
```

```r
print("Late Phase Estimated Parameters:")
```

```
## [1] "Late Phase Estimated Parameters:"
```

```r
print(weighted_fft_estimated_params_late)
```

```
##         A_max       eta_max         A_min       eta_min           tau
##  2.322913e+03  2.290888e-03  2.500234e+02  1.619466e-05  2.590810e-01
##           phi
## -2.284774e+00
```

```
print("Late Phase Confidence Intervals:")
```

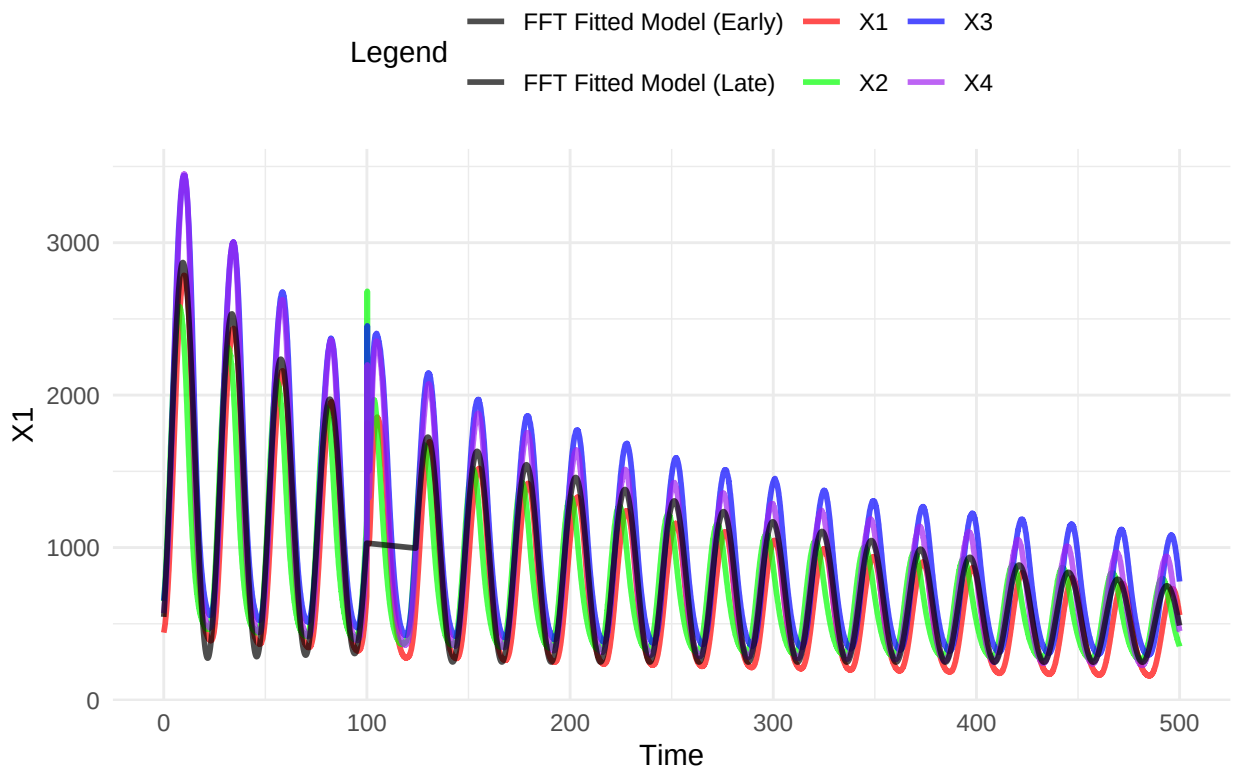
```
## [1] "Late Phase Confidence Intervals:"
```

```
print(weighted_fft_conf_intervals_late)
```

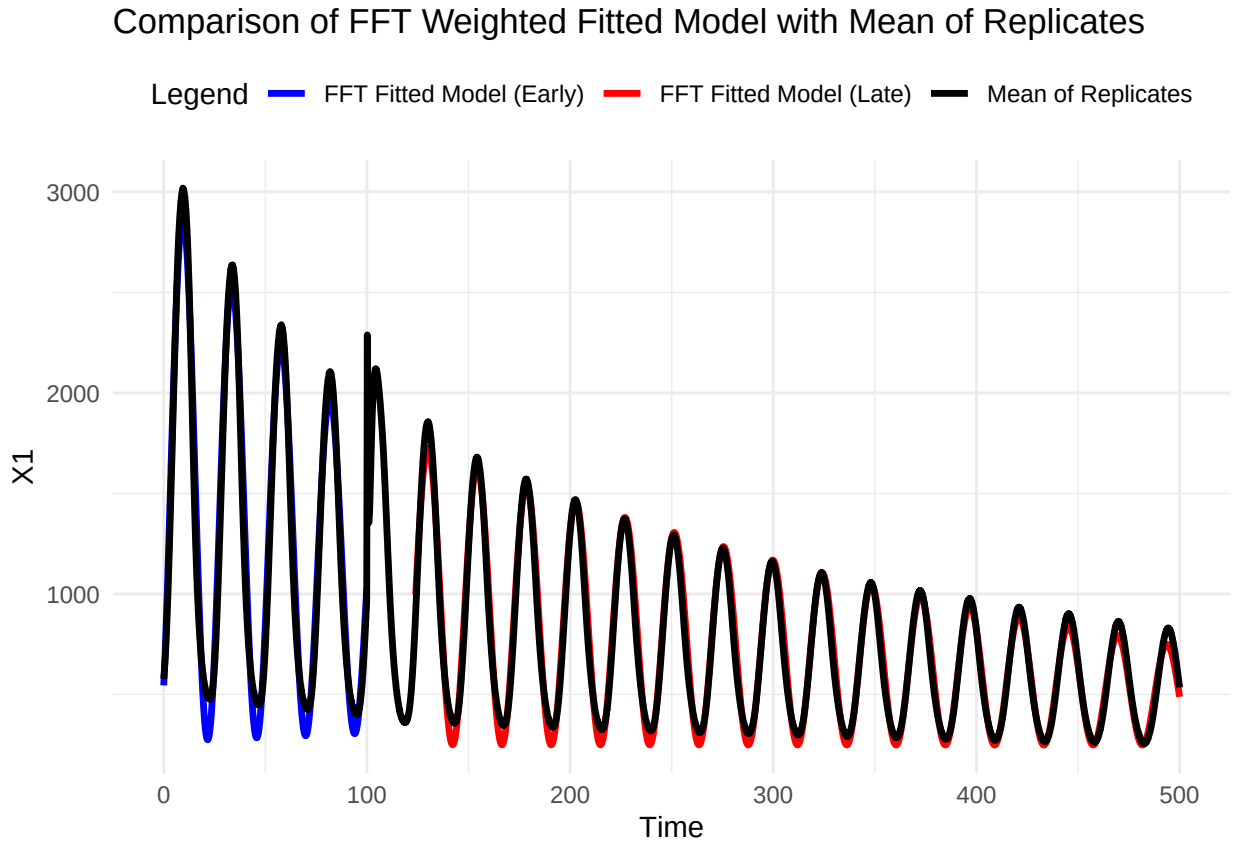
```
##                 0.5 %        99.5 %
## A_max    2.190384e+03  2.455442e+03
## eta_max  2.081777e-03  2.499999e-03
## A_min    1.691109e+02  3.309358e+02
## eta_min -9.835114e-04  1.015901e-03
## tau      2.586007e-01  2.595613e-01
## phi     -2.407515e+00 -2.162034e+00
```

plot to compare

```
# Generate fitted curves using estimated parameters
y_fit_early <- asym_model_function(time_early, weighted_fft_estimated_params_early["A_max"],
                                   weighted_fft_estimated_params_early["eta_max"],
                                   weighted_fft_estimated_params_early["A_min"],
                                   weighted_fft_estimated_params_early["eta_min"],
                                   weighted_fft_estimated_params_early["tau"],
                                   weighted_fft_estimated_params_early["phi"])

y_fit_late <- asym_model_function(time_late, weighted_fft_estimated_params_late["A_max"],
                                  weighted_fft_estimated_params_late["eta_max"],
                                  weighted_fft_estimated_params_late["A_min"],
                                  weighted_fft_estimated_params_late["eta_min"],
                                  weighted_fft_estimated_params_late["tau"],
                                  weighted_fft_estimated_params_late["phi"])

# Create data frames for plotting
df_original <- df_MC %>% pivot_longer(cols = c("X1", "X2", "X3", "X4"), names_to = "Replicate", values_
df_fitted_early <- data.frame(Time = time_early, Value = y_fit_early, Type = "FFT Fitted Model (Early)"
df_fitted_late <- data.frame(Time = time_late, Value = y_fit_late, Type = "FFT Fitted Model (Late)")

df_plot <- bind_rows(df_original %>% mutate(Type = Replicate), df_fitted_early, df_fitted_late)

# Plot with all replicates in different colors
ggplot(df_plot, aes(x = Time, y = Value, color = Type, group = interaction(Replicate, Type))) +
  geom_line(size = 1, alpha = 0.7) +
  scale_color_manual(values = c("X1" = "red", "X2" = "green", "X3" = "blue", "X4" = "purple", "FFT Fitte
  labs(title = "Comparison of FFT Weighted Fitted Model with Original Replicates",
       x = "Time", y = "X1", color = "Legend") +
  theme_minimal() +
  theme(legend.position = "top")
```




```r
# Plot with mean of replicates
df_mean <- df_MC %>%
  rowwise() %>%
  mutate(Mean = mean(c_across(X1:X4))) %>%
  select(Time, Mean)

df_plot_mean <- bind_rows(
  data.frame(Time = df_mean$Time, Value = df_mean$Mean, Type = "Mean of Replicates"),
  df_fitted_early, df_fitted_late
)

# Plot with only mean curve
ggplot(df_plot_mean, aes(x = Time, y = Value, color = Type)) +
  geom_line(size = 1.2) +
  scale_color_manual(values = c("Mean of Replicates" = "black", "FFT Fitted Model (Early)" = "blue", "FF
  labs(title = "Comparison of FFT Weighted Fitted Model with Mean of Replicates",
       x = "Time", y = "X1", color = "Legend") +
  theme_minimal() +
  theme(legend.position = "top")
```




```
# calculate sigma^2
sigma2_early_fft_weighted <- estimate_sigma2_asym(y_early, time_early, weighted_fft_estimated_params_ea
sigma2_late_fft_weighted <- estimate_sigma2_asym(y_late, time_late, weighted_fft_estimated_params_late)

# calculate covariance( from hessian)
covariance_early_fft_weighted <- compute_covariance_asym(weighted_fft_estimated_params_early, time_early
covariance_late_fft_weighted <- compute_covariance_asym(weighted_fft_estimated_params_late, time_late, y

# exclude phi
early_filtered_fft_weighted <- remove_phi_asym(weighted_fft_estimated_params_early, covariance_early_ff
late_filtered_fft_weighted <- remove_phi_asym(weighted_fft_estimated_params_late, covariance_late_fft_w

#  Wald test
wald_result_fft_weighted <- wald_test(
  early_filtered_fft_weighted$params, late_filtered_fft_weighted$params,
  early_filtered_fft_weighted$covariance, late_filtered_fft_weighted$covariance
)

cat("fft Wald Test Statistic :", wald_result_fft_weighted$T_Wald, "\n")
```

```
## fft Wald Test Statistic : 308.3298
```

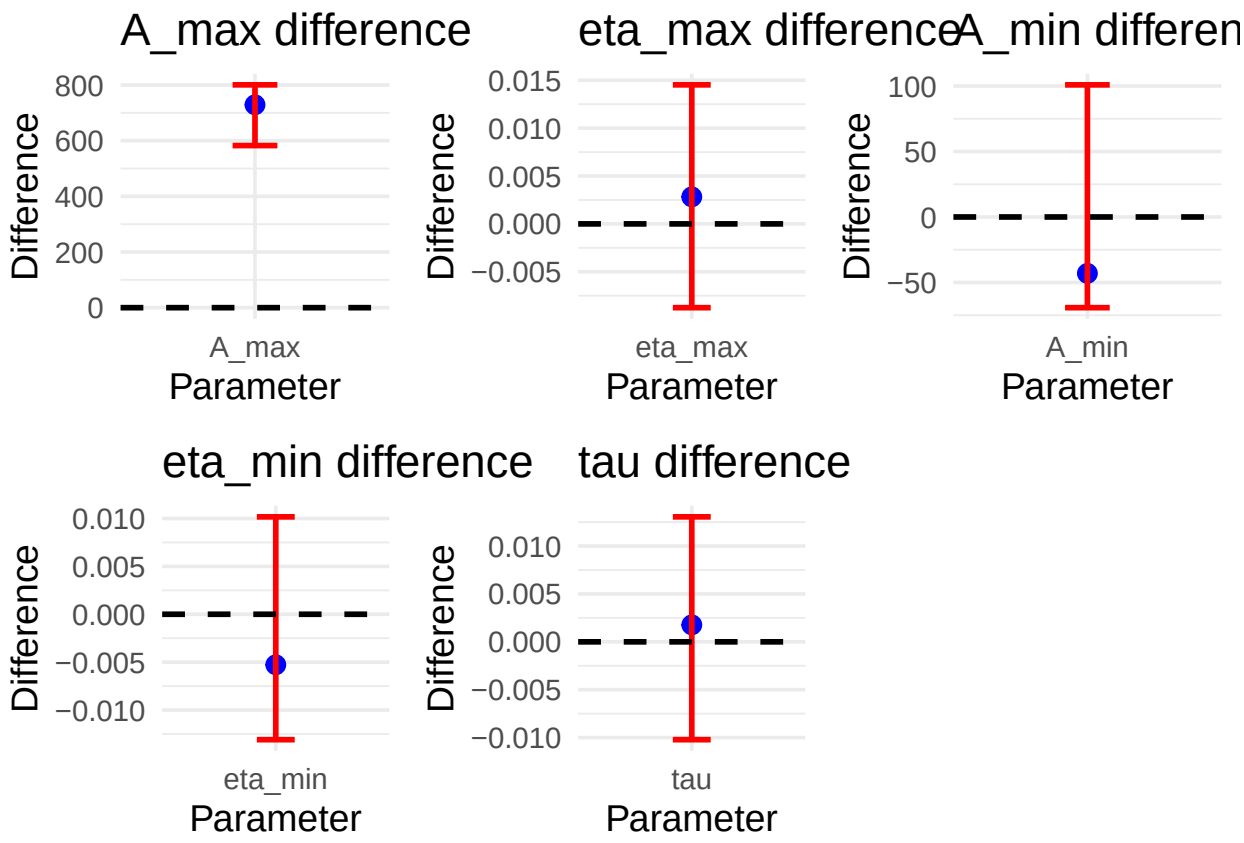
```
cat("p-value :", wald_result_fft_weighted$p_value, "\n")
```

```
## p-value : 0
```

```r
library(Matrix)
```

```
##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## The following objects are masked from 'package:pracma':
##
##     expm, lu, tril, triu
```

```r
covariance_early_fixed <- as.matrix(nearPD(early_filtered_fft_weighted$covariance)$mat)
covariance_late_fixed <- as.matrix(nearPD(late_filtered_fft_weighted$covariance)$mat)


# get CI
fft_param_names <- names(early_filtered_fft_weighted$params)
weighted_fft_param_diff <- early_filtered_fft_weighted$params - late_filtered_fft_weighted$params
var_diff_weighted <- diag(covariance_early_fixed + covariance_late_fixed)  #
se_diff_weighted <- sqrt(var_diff_weighted)  #
CI_lower <- weighted_fft_param_diff - Z_99 * se_diff_weighted
CI_upper <- weighted_fft_param_diff + Z_99 * se_diff_weighted

# plot
df_diff_fft_weighted <- data.frame(
  Parameter = fft_param_names,
  Difference = fft_param_diff,
  CI_Lower = CI_lower,
  CI_Upper = CI_upper
)

library(gridExtra)

# single plot for each parameter
plots <- lapply(1:nrow(df_diff_fft_weighted), function(i) {
  ggplot(df_diff_fft_weighted[i, ], aes(x = Parameter, y = Difference)) +
    geom_point(size = 3, color = "blue") +  #
    geom_errorbar(aes(ymin = CI_Lower, ymax = CI_Upper), width = 0.2, size = 1, color = "red") +  #
    geom_hline(yintercept = 0, linetype = "dashed", color = "black", size = 1) +  # 0
    labs(title = paste0(df_diff_fft$Parameter[i], " difference"),
         x = "Parameter", y = "Difference") +
    theme_minimal(base_size = 14)
})

#   gridExtra
grid.arrange(grobs = plots, ncol = 3)  # 3
```

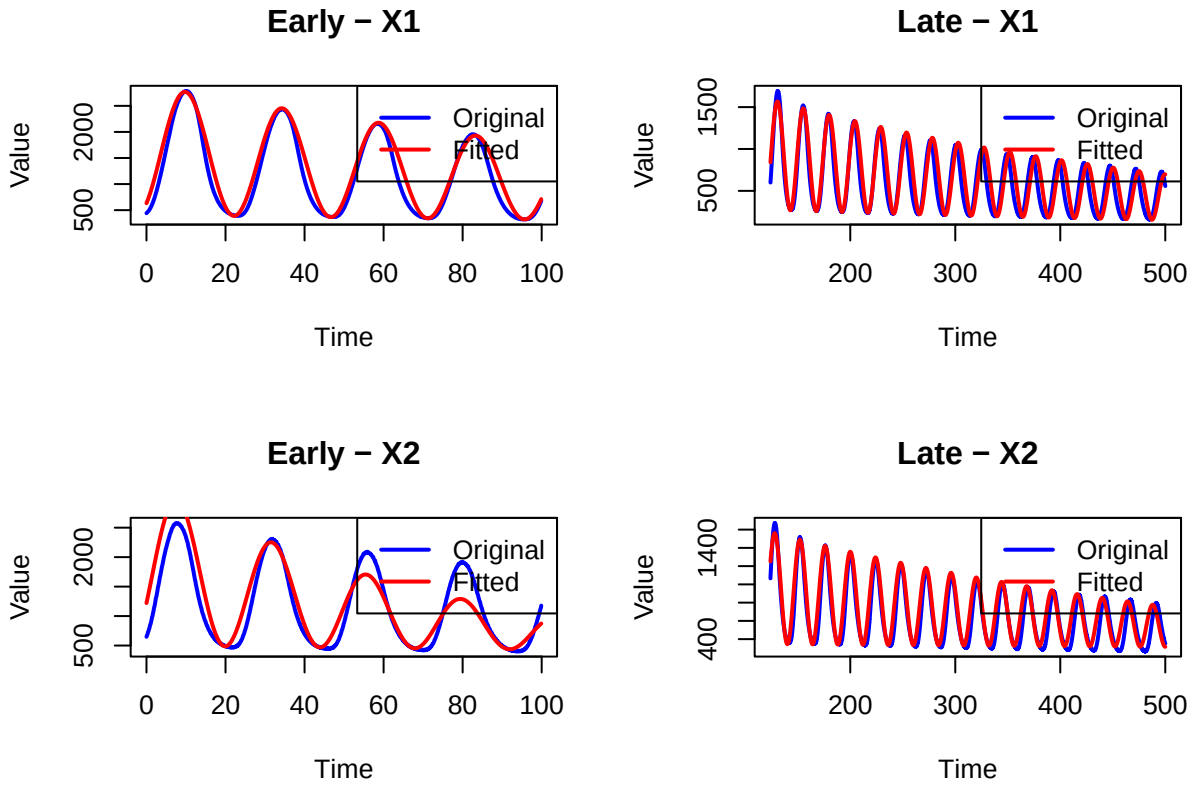


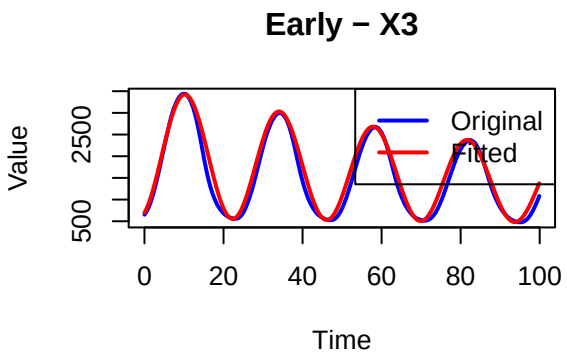
apply model on full dataset

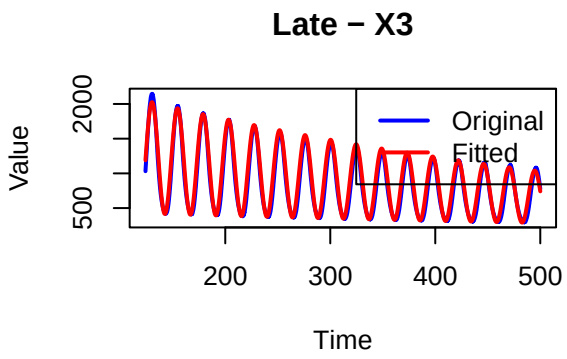
```
# get initial parameters for early phase
early_params <- lapply(1:ncol(y_early), function(i) {
  params <- find_initial_params(time_early, y_early[, i])
  phi <- compute_phi(time_early, y_early[, i], params$A_max, params$eta_max, params$A_min, params$eta_m
  return(list(params = params, phi = phi))
})

#  get initial parameters for early phase
late_params <- lapply(1:ncol(y_late), function(i) {
  params <- find_initial_params(time_late, y_late[, i])
  phi <- compute_phi(time_late, y_late[, i], params$A_max, params$eta_max, params$A_min, params$eta_min
  return( list(params = params, phi = phi))
})
```

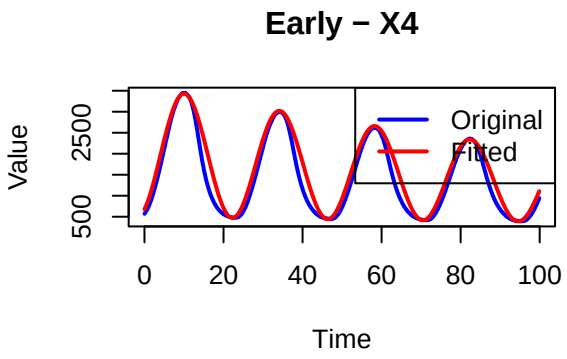
```
# plot fitted curve by initial parameters VS original curves for four replicates
plot_comparison <- function(time, y_original, params, title) {
  #
  y_fitted <- asym_model_function(time,
                                  params$params$A_max,
                                  params$params$eta_max,
                                  params$params$A_min,
```

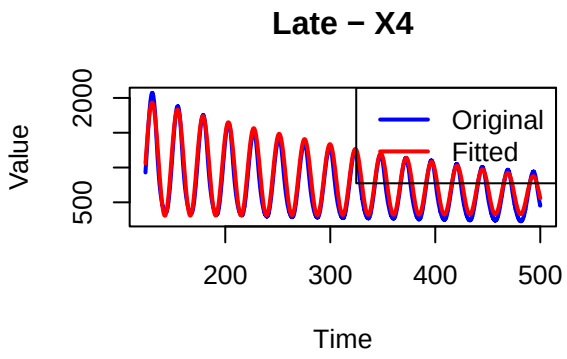
```
                                    params$params$eta_min,
                                    params$params$tau,
                                    params$params$phi)


  plot(time, y_original, type = "l", col = "blue", lwd = 2,
       xlab = "Time", ylab = "Value", main = title)
  lines(time, y_fitted, col = "red", lwd = 2)
  legend("topright", legend = c("Original", "Fitted"),
         col = c("blue", "red"), lwd = 2)
}

par(mfrow = c(2, 2))

for (i in 1:ncol(y_early)) {

  col_name <- colnames(y_early)[i]


  plot_comparison(time_early, y_early[, i], early_params[[i]],
                  paste("Early -", col_name))

  plot_comparison(time_late, y_late[, i], late_params[[i]],
                  paste("Late -", col_name))
}
```












```r
par(mfrow = c(1, 1))
```

```r
phi_initial_early <- lapply(early_params, function(x) x$params$phi)
print("Early - phi from find_initial_params:")
```

```
## [1] "Early - phi from find_initial_params:"
```

```r
print(phi_initial_early)
```

```
## [[1]]
## (Intercept)
##    -2.53892
##
## [[2]]
## (Intercept)
##   -2.068215
##
## [[3]]
## (Intercept)
##   -2.745594
##
## [[4]]
## (Intercept)
##   -2.659273
```

```r
phi_initial_late <- lapply(late_params, function(x) x$params$phi)
print("late - phi from find_initial_params:")
```

```
## [1] "late - phi from find_initial_params:"
```

```r
print(phi_initial_late)
```

```
## [[1]]
## (Intercept)
##    4.316497
##
## [[2]]
## (Intercept)
##     4.16261
##
## [[3]]
## (Intercept)
##    4.007793
##
## [[4]]
## (Intercept)
##    3.842609
```

I find just use find phi: just choose 1 points(method finding phi from find_initial_parmas() is better than using 5 points and taking median

constant variance

```r
# Convert df_MC to long format for fitting the entire dataset
long_format_MC <- df_MC %>%
  pivot_longer(cols = c("X1", "X2", "X3", "X4"), names_to = "Replicate", values_to = "Value")

# Extract early and late phase data using long format
long_MC_early <- long_format_MC %>% filter(Time < 100)
long_MC_late <- long_format_MC %>% filter(Time >= 124)
# Estimate initial parameters for early and late phases
# Aggregate data by computing the mean value for each time point
df_MC_early_avg <- long_MC_early %>%
  group_by(Time) %>%
  summarise(Value = mean(Value, na.rm = TRUE))

df_MC_late_avg <- long_MC_late %>%
  group_by(Time) %>%
  summarise(Value = mean(Value, na.rm = TRUE))

# Use averaged data to estimate initial parameters
init_param_early <- find_initial_params(df_MC_early_avg$Time, df_MC_early_avg$Value)
init_param_late <- find_initial_params(df_MC_late_avg$Time, df_MC_late_avg$Value)
```

```r
init_param_late <- lapply(init_param_late, function(x) as.numeric(unlist(x)))
init_param_early <- lapply(init_param_early, function(x) as.numeric(unlist(x)))


# Fit asymmetric model to the early phase
fit_early <- fit_nls_model(long_MC_early$Time, long_MC_early$Value, init_param_early)

# Fit asymmetric model to the late phase
fit_late <- fit_nls_model(long_MC_late$Time, long_MC_late$Value, init_param_late)

# Extract estimated parameters for early and late phases
params_early <- fit_early$estimates
params_late <- fit_late$estimates

# Extract confidence intervals for early and late phases
conf_early <- fit_early$conf_intervals
conf_late <- fit_late$conf_intervals
```

```r
fit_early
```

```
## $estimates
##         A_max       eta_max         A_min       eta_min           tau
##  3.024166e+03  5.232426e-03  2.485310e+02 -2.572465e-03  2.605693e-01
##           phi
## -2.493583e+00
##
## $conf_intervals
##                  0.5 %        99.5 %
## A_max    2.980382e+03  3.067951e+03
## eta_max  4.925847e-03  5.539006e-03
## A_min    2.099461e+02  2.871159e+02
## eta_min -4.942061e-03 -2.028691e-04
## tau      2.600284e-01  2.611101e-01
## phi     -2.519525e+00 -2.467642e+00
```

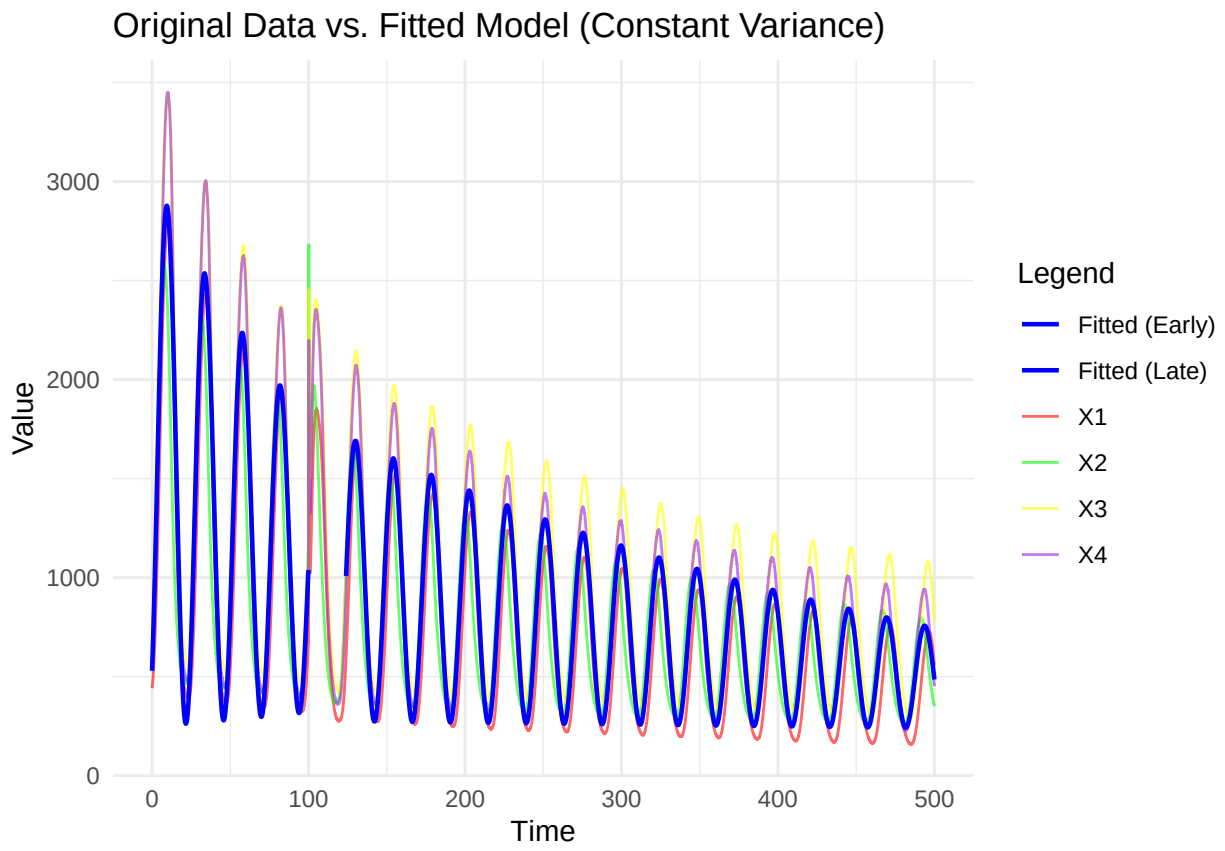
```r
fit_late
```

```
## $estimates
##        A_max      eta_max        A_min      eta_min          tau          phi
## 2.252032e+03 2.206604e-03 2.884473e+02 3.599710e-04 2.590489e-01 4.026005e+00
##
## $conf_intervals
##                 0.5 %       99.5 %
## A_max   2.220843e+03 2.283222e+03
## eta_max 2.157072e-03 2.256137e-03
## A_min   2.667877e+02 3.101069e+02
## eta_min 1.265316e-04 5.934104e-04
## tau     2.589380e-01 2.591597e-01
## phi     3.996072e+00 4.055938e+00
```

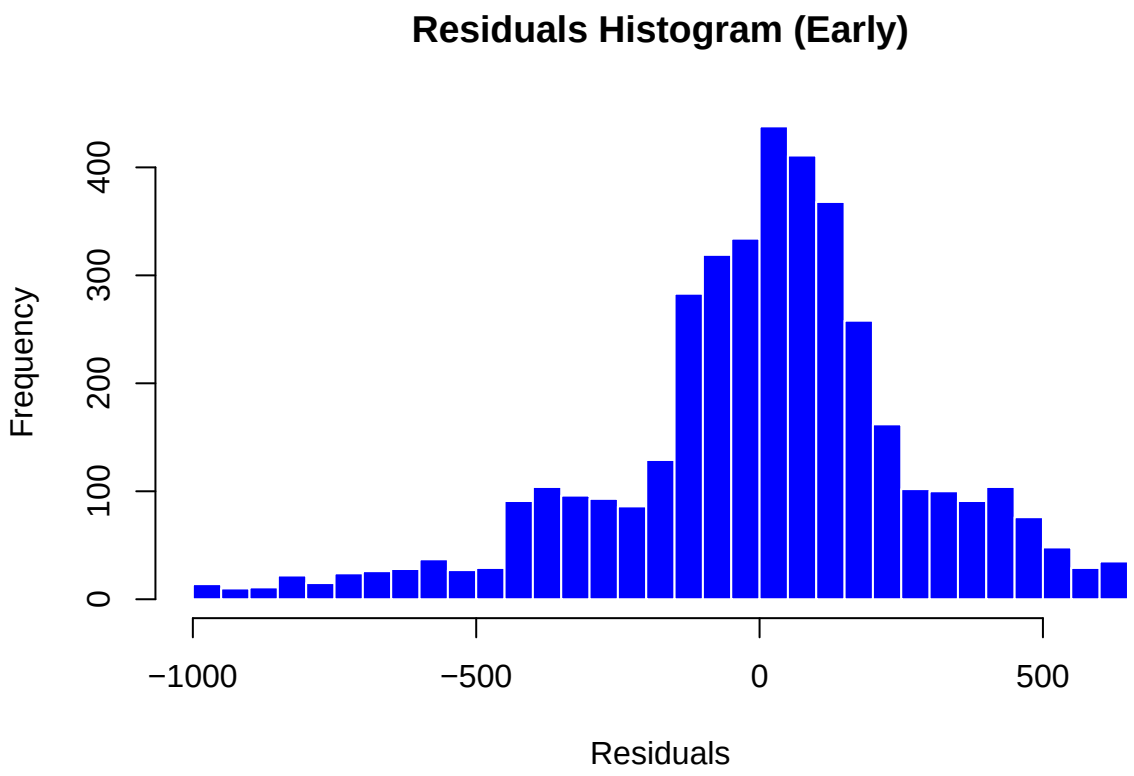
```r
head(df_MC_late_avg)
```

```
## # A tibble: 6 x 2
##    Time Value
##   <dbl> <dbl>
## 1  124   906.
## 2  124.  931.
## 3  124.  951.
## 4  124.  974.
## 5  124.  997.
## 6  124. 1018.
```

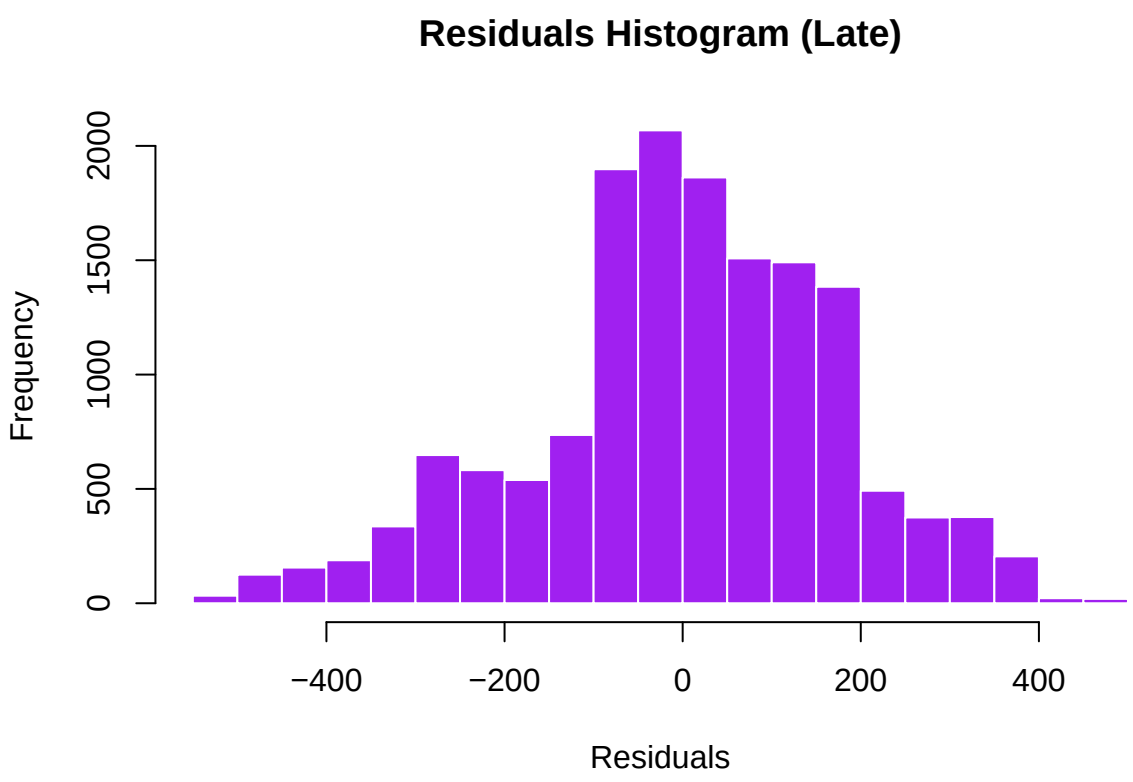
```r
# Compute fitted values for early phase
y_fit_early <- asym_model_function(long_MC_early$Time, params_early["A_max"], params_early["eta_max"],
                                   params_early["A_min"], params_early["eta_min"],
                                   params_early["tau"], params_early["phi"])

# Compute fitted values for late phase
y_fit_late <- asym_model_function(long_MC_late$Time, params_late["A_max"], params_late["eta_max"],
                                  params_late["A_min"], params_late["eta_min"],
                                  params_late["tau"], params_late["phi"])

# Create data frames for fitted curves
df_fitted_early <- data.frame(Time = long_MC_early$Time, Value = y_fit_early, Type = "Fitted (Early)")
df_fitted_late <- data.frame(Time = long_MC_late$Time, Value = y_fit_late, Type = "Fitted (Late)")

# Plot original data vs fitted curves
ggplot(long_format_MC, aes(x = Time, y = Value, color = Replicate)) +
  geom_line(alpha = 0.6) +  # Plot original data
  geom_line(data = df_fitted_early, aes(x = Time, y = Value, color = Type), size = 0.8) +
  geom_line(data = df_fitted_late, aes(x = Time, y = Value, color = Type), size = 0.8) +
  scale_color_manual(values = c("X1" = "red", "X2" = "green", "X3" = "yellow", "X4" = "purple",
                                "Fitted (Early)" = "blue", "Fitted (Late)" = "blue")) +
  labs(title = "Original Data vs. Fitted Model (Constant Variance)",
       x = "Time", y = "Value", color = "Legend") +
  theme_minimal()
```



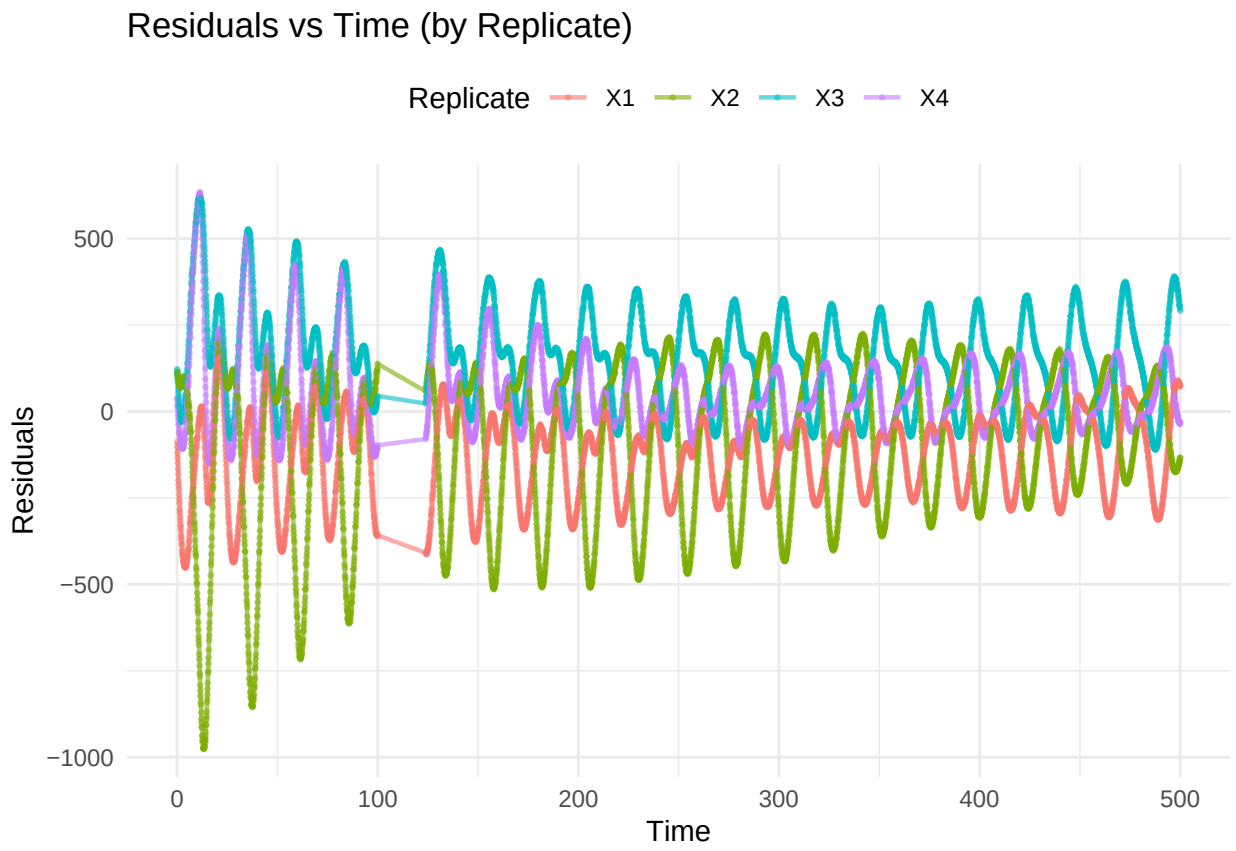

```r
# Compute residuals
residuals_early <- long_MC_early$Value - y_fit_early
residuals_late <- long_MC_late$Value - y_fit_late


hist(residuals_early, breaks = 30, col = "blue", border = "white",
     main = "Residuals Histogram (Early)", xlab = "Residuals", ylab = "Frequency")
```




```r
hist(residuals_late, breaks = 30, col = "purple", border = "white",
     main = "Residuals Histogram (Late)", xlab = "Residuals", ylab = "Frequency")
```




```r
# Compute residuals for early and late phase using the function
residuals_early <- compute_residuals_asym(long_MC_early$Value, long_MC_early$Time, params_early)
residuals_late <- compute_residuals_asym(long_MC_late$Value, long_MC_late$Time, params_late)

# Create a combined dataframe for plotting
df_residuals <- bind_rows(
  data.frame(Time = long_MC_early$Time, Residual = residuals_early, Phase = "Early", Replicate = long_MC
  data.frame(Time = long_MC_late$Time, Residual = residuals_late, Phase = "Late", Replicate = long_MC_la
)
# Plot residuals vs time
ggplot(df_residuals, aes(x = Time, y = Residual, color = Replicate, group = Replicate)) +
  geom_line(alpha = 0.6, size = 0.8) +  # Line plot for residuals
  geom_point(size = 0.5, alpha = 0.5) +  # Scatter plot for residuals

  labs(title = "Residuals vs Time (by Replicate)", x = "Time", y = "Residuals", color = "Replicate") +
  theme_minimal() +
  theme(legend.position = "top")
```



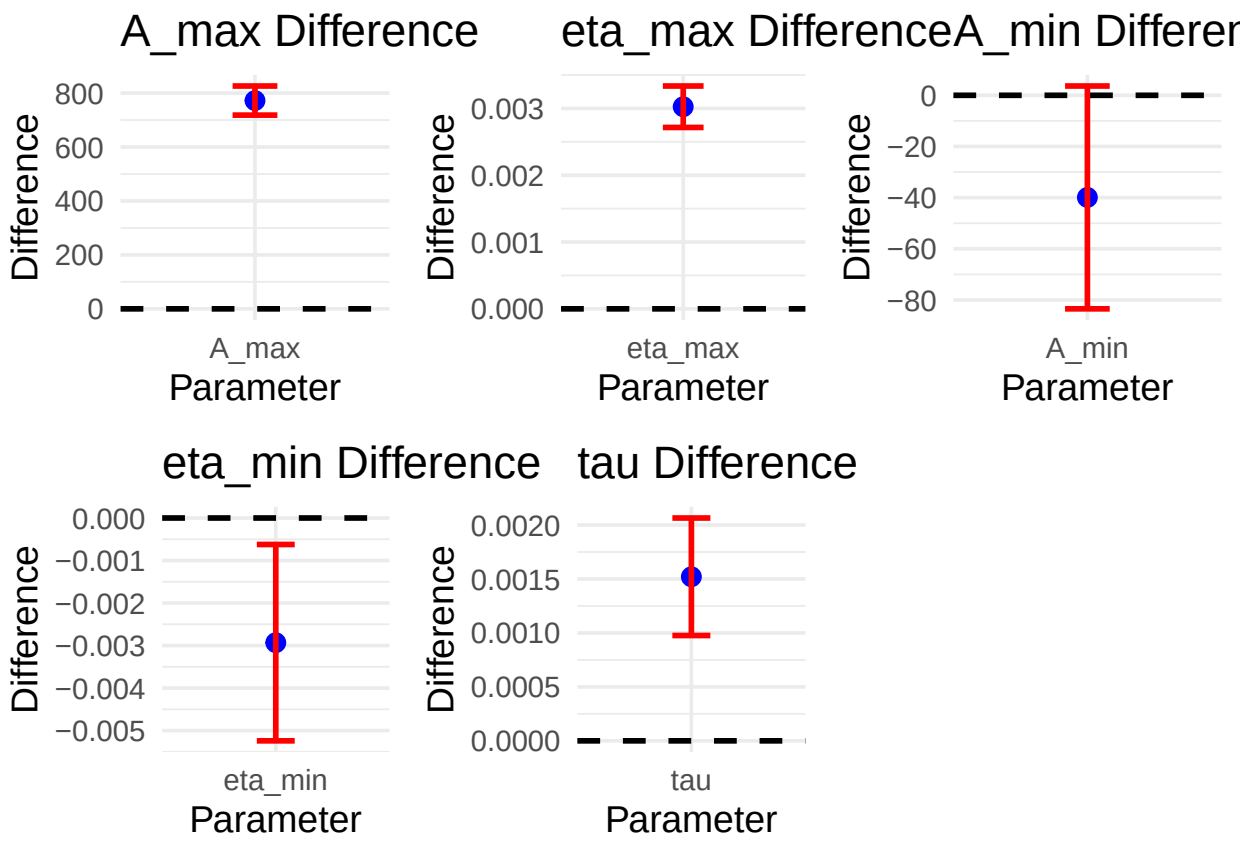

```
# Compute sigma² for early and late phases
sigma2_early <- estimate_sigma2_asym(long_MC_early$Value, long_MC_early$Time, params_early)
sigma2_late <- estimate_sigma2_asym(long_MC_late$Value, long_MC_late$Time, params_late)

# Compute covariance matrices
covariance_early <- compute_covariance_asym(params_early, long_MC_early$Time, long_MC_early$Value, sigma
covariance_late <- compute_covariance_asym(params_late, long_MC_late$Time, long_MC_late$Value, sigma2_la

# Remove phi parameter from estimated parameters and covariance matrix
early_filtered <- remove_phi_asym(params_early, covariance_early)
late_filtered <- remove_phi_asym(params_late, covariance_late)

# Perform Wald Test to compare early and late phases
wald_result <- wald_test(early_filtered$params, late_filtered$params, early_filtered$covariance, late_fi

# Print Wald Test results
cat("Wald Test Statistic:", wald_result$T_Wald, "\n")
```

```
## Wald Test Statistic: 1498.164
```

```
cat("p-value:", wald_result$p_value, "\n")
```

```
## p-value: 0
```

```r
# Compute parameter differences
param_diff <- early_filtered$params - late_filtered$params
var_diff <- diag(early_filtered$covariance + late_filtered$covariance)
se_diff <- sqrt(var_diff)

# Compute 99% confidence intervals
Z_99 <- 2.576
CI_lower <- param_diff - Z_99 * se_diff
CI_upper <- param_diff + Z_99 * se_diff

# Create data frame for plotting
df_diff <- data.frame(Parameter = names(param_diff), Difference = param_diff, CI_Lower = CI_lower, CI_U


# Generate individual plots for each parameter
plots <- lapply(1:nrow(df_diff), function(i) {
  ggplot(df_diff[i, ], aes(x = Parameter, y = Difference)) +
    geom_point(size = 3, color = "blue") +  # Show parameter difference
    geom_errorbar(aes(ymin = CI_Lower, ymax = CI_Upper), width = 0.2, size = 1, color = "red") +  # Con
    geom_hline(yintercept = 0, linetype = "dashed", color = "black", size = 1) +  # Reference line at 0
    labs(title = paste0(df_diff$Parameter[i], " Difference"),
         x = "Parameter", y = "Difference") +
    theme_minimal(base_size = 14)
})
grid.arrange(grobs = plots, ncol = 3)
```



fft with constant variance model (full dataset)

```
N_early <- 20
N_late <- 50

# --- Step 2: FFT on long data ---
y_fft_early <- fft_extract(long_MC_early$Value, N_early)
y_fft_late <- fft_extract(long_MC_late$Value, N_late)

# --- Step 4: FFT on fitted data  ---
fit_early_fft <- fit_fft_model(y_fft_early, long_MC_early$Time, N_early, init_param_early)
fit_late_fft <- fit_fft_model(y_fft_late, long_MC_late$Time, N_late, init_param_late)

fit_late_fft
```

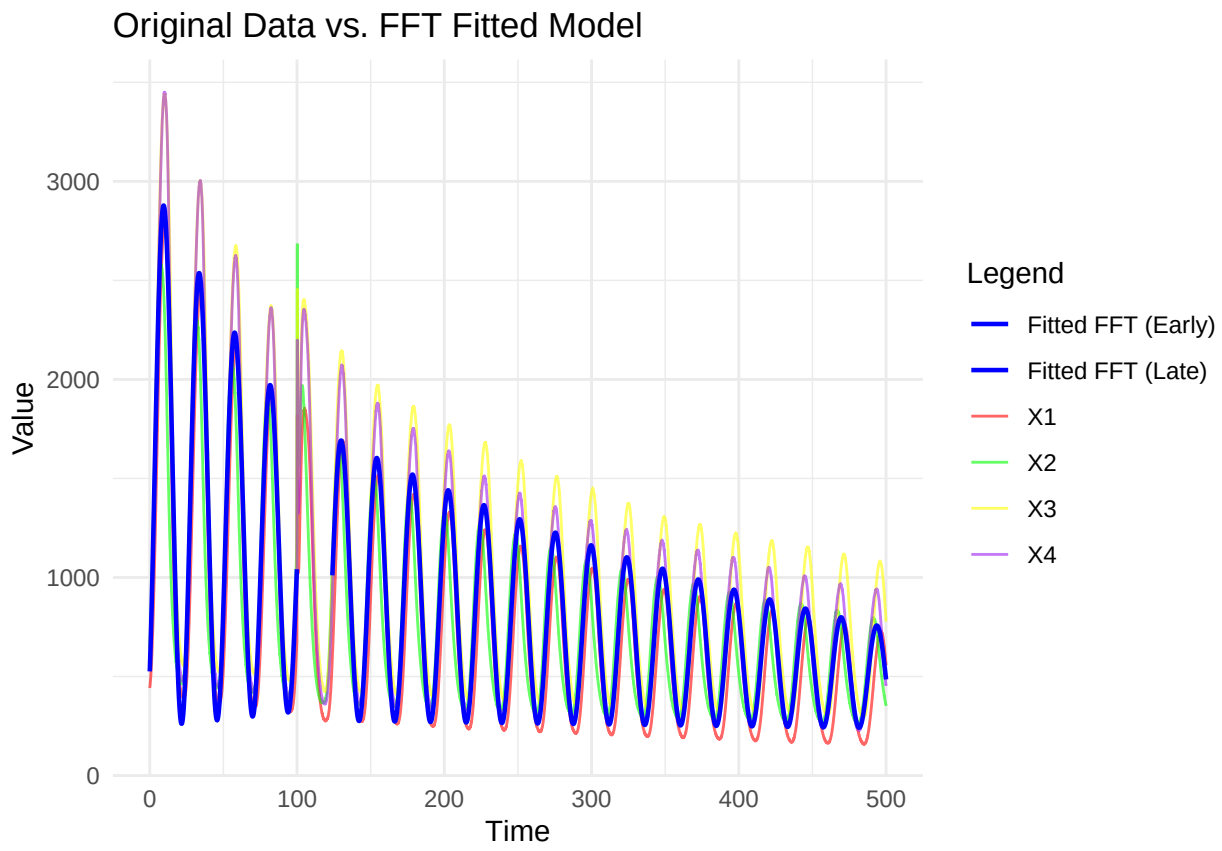
```
## $estimates
##        A_max      eta_max        A_min      eta_min          tau          phi
## 2.253856e+03 2.208845e-03 2.912648e+02 3.900519e-04 2.590435e-01 4.027894e+00
##
## $conf_intervals
##                 0.5 %       99.5 %
## A_max    2.153365e+03 2.354347e+03
## eta_max  2.049855e-03 2.367834e-03
## A_min    2.212836e+02 3.612460e+02
## eta_min -3.634597e-04 1.143563e-03
## tau      2.586866e-01 2.594004e-01
## phi      3.931307e+00 4.124480e+00
```

```
fit_early_fft
```

```
## $estimates
##         A_max       eta_max         A_min       eta_min           tau
##  3.024485e+03  5.230424e-03  2.459611e+02 -2.745716e-03  2.606086e-01
##           phi
## -2.495776e+00
##
## $conf_intervals
##                 0.5 %        99.5 %
## A_max    2.838566e+03  3.210405e+03
## eta_max  3.909811e-03  6.551036e-03
## A_min    8.179902e+01  4.101231e+02
## eta_min -1.311154e-02  7.620111e-03
## tau      2.582465e-01  2.629706e-01
## phi     -2.608455e+00 -2.383098e+00
```

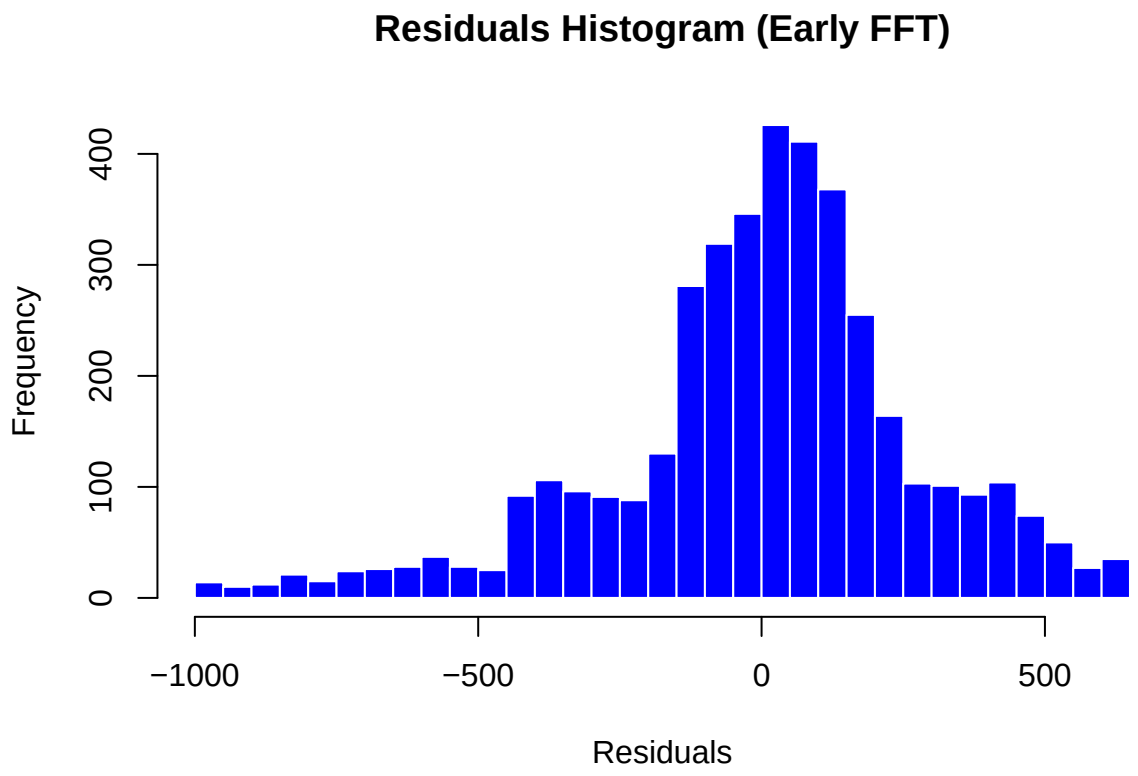
```
params_early_fft <- fit_early_fft$estimates
params_late_fft <- fit_late_fft$estimates

# fitted curve
y_fit_early_fft <- asym_model_function(long_MC_early$Time, params_early_fft["A_max"], params_early_fft[
                                       params_early_fft["A_min"], params_early_fft["eta_min"],
```

```
                                    params_early_fft["tau"], params_early_fft["phi"])

y_fit_late_fft <- asym_model_function(long_MC_late$Time, params_late_fft["A_max"], params_late_fft["eta
                                    params_late_fft["A_min"], params_late_fft["eta_min"],
                                    params_late_fft["tau"], params_late_fft["phi"])


df_fitted_early_fft <- data.frame(Time = long_MC_early$Time, Value = y_fit_early_fft, Type = "Fitted FF
df_fitted_late_fft <- data.frame(Time = long_MC_late$Time, Value = y_fit_late_fft, Type = "Fitted FFT (

# --- Step 8: fitted curve vs original curve ---
ggplot(long_format_MC, aes(x = Time, y = Value, color = Replicate)) +
  geom_line(alpha = 0.6) +
  geom_line(data = df_fitted_early_fft, aes(x = Time, y = Value, color = Type), size = 0.8) +
  geom_line(data = df_fitted_late_fft, aes(x = Time, y = Value, color = Type), size = 0.8) +
  scale_color_manual(values = c("X1" = "red", "X2" = "green", "X3" = "yellow", "X4" = "purple",
                                "Fitted FFT (Early)" = "blue", "Fitted FFT (Late)" = "blue")) +
  labs(title = "Original Data vs. FFT Fitted Model",
       x = "Time", y = "Value", color = "Legend") +
  theme_minimal()
```

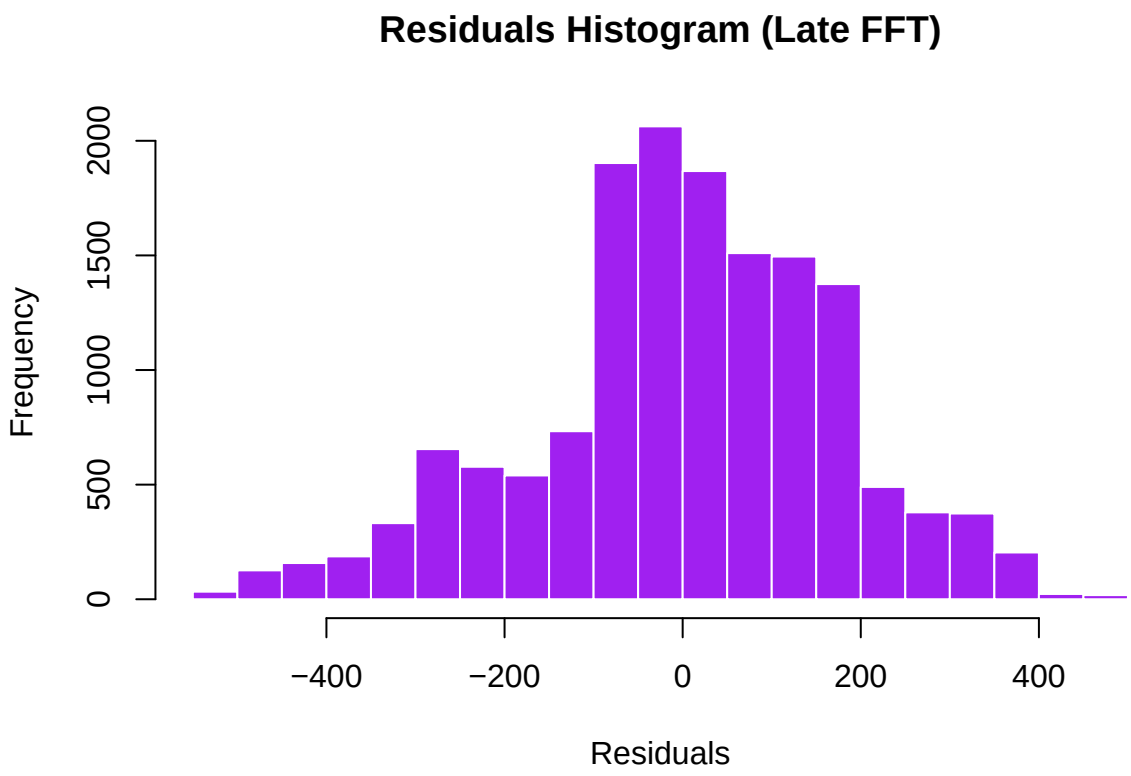



```
# --- Step 9: compute residual(non fft) ---
residuals_early_fft <- long_MC_early$Value - y_fit_early_fft
residuals_late_fft <- long_MC_late$Value - y_fit_late_fft
```

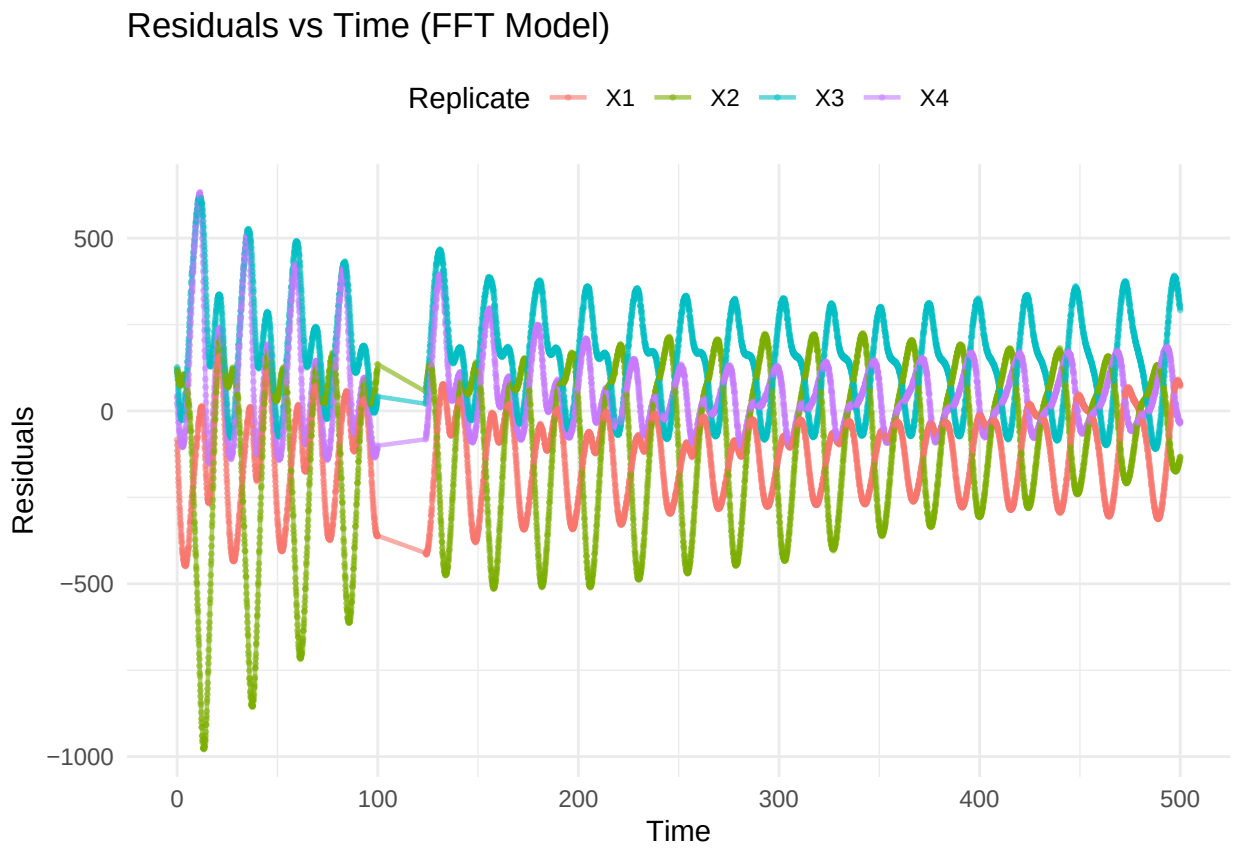
```r
# --- Step 10:  Residual Histogram ---

hist(residuals_early_fft, breaks = 30, col = "blue", border = "white",
     main = "Residuals Histogram (Early FFT)", xlab = "Residuals", ylab = "Frequency")
```




```r
hist(residuals_late_fft, breaks = 30, col = "purple", border = "white",
     main = "Residuals Histogram (Late FFT)", xlab = "Residuals", ylab = "Frequency")
```




```r
# --- Step 11:  Residuals vs Time ---
df_residuals_fft <- bind_rows(
  data.frame(Time = long_MC_early$Time, Residual = residuals_early_fft, Phase = "Early", Replicate = lo
  data.frame(Time = long_MC_late$Time, Residual = residuals_late_fft, Phase = "Late", Replicate = long_
)

ggplot(df_residuals_fft, aes(x = Time, y = Residual, color = Replicate, group = Replicate)) +
  geom_line(alpha = 0.6, size = 0.8) +
  geom_point(size = 0.5, alpha = 0.5) +
  labs(title = "Residuals vs Time (FFT Model)", x = "Time", y = "Residuals", color = "Replicate") +
  theme_minimal() +
  theme(legend.position = "top")
```




```
# --- Step 12:  sigma² ---
sigma2_early_fft <- estimate_sigma2_asym(long_MC_early$Value, long_MC_early$Time, params_early_fft)
sigma2_late_fft <- estimate_sigma2_asym(long_MC_late$Value, long_MC_late$Time, params_late_fft)

# --- Step 13: covariance(from hessian) ---
covariance_early_fft <- compute_covariance_asym(params_early_fft, long_MC_early$Time, long_MC_early$Valu
covariance_late_fft <- compute_covariance_asym(params_late_fft, long_MC_late$Time, long_MC_late$Value, s

# --- Step 14: Wald Test ---
wald_result_fft <- wald_test(params_early_fft, params_late_fft, covariance_early_fft, covariance_late_fi

cat("Wald Test Statistic (FFT Model):", wald_result_fft$T_Wald, "\n")
```

```
## Wald Test Statistic (FFT Model): 361351.1
```

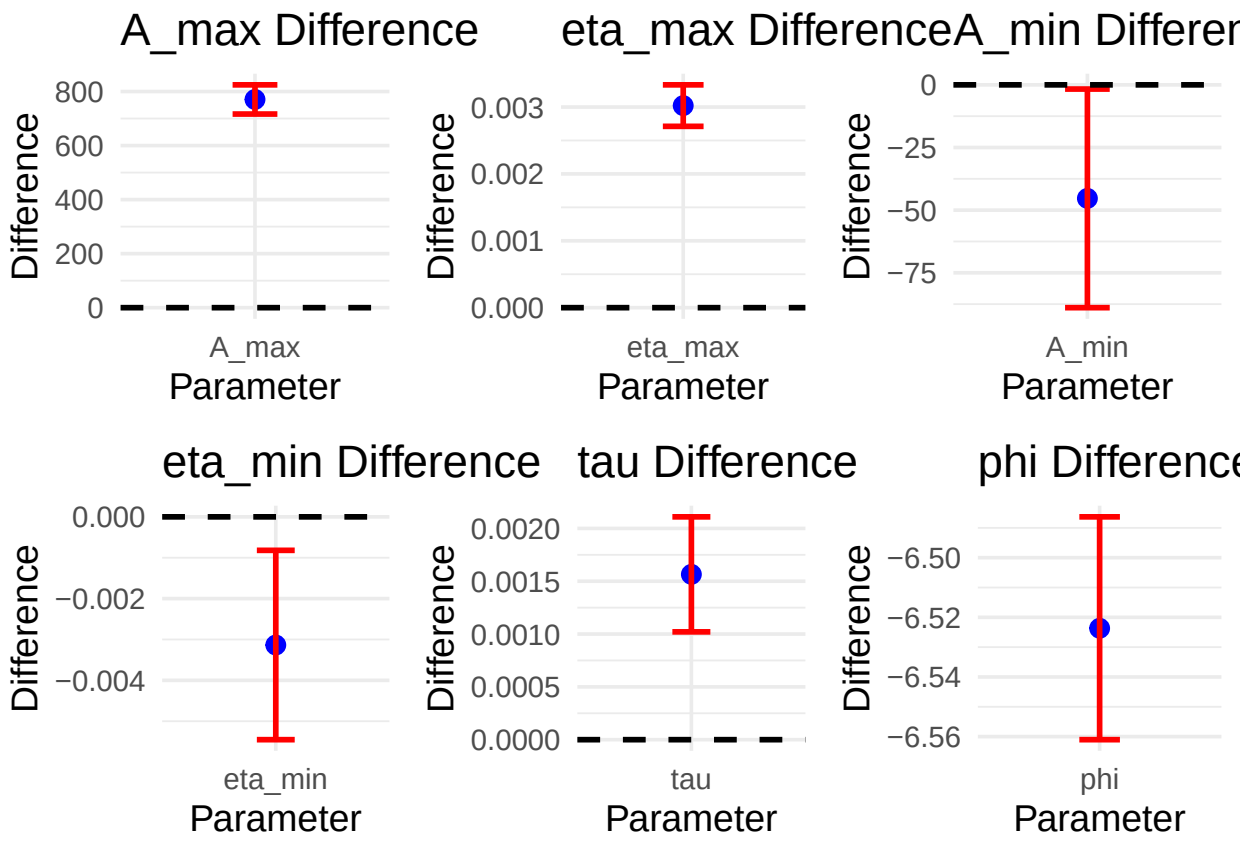
```
cat("p-value:", wald_result_fft$p_value, "\n")
```

```
## p-value: 0
```

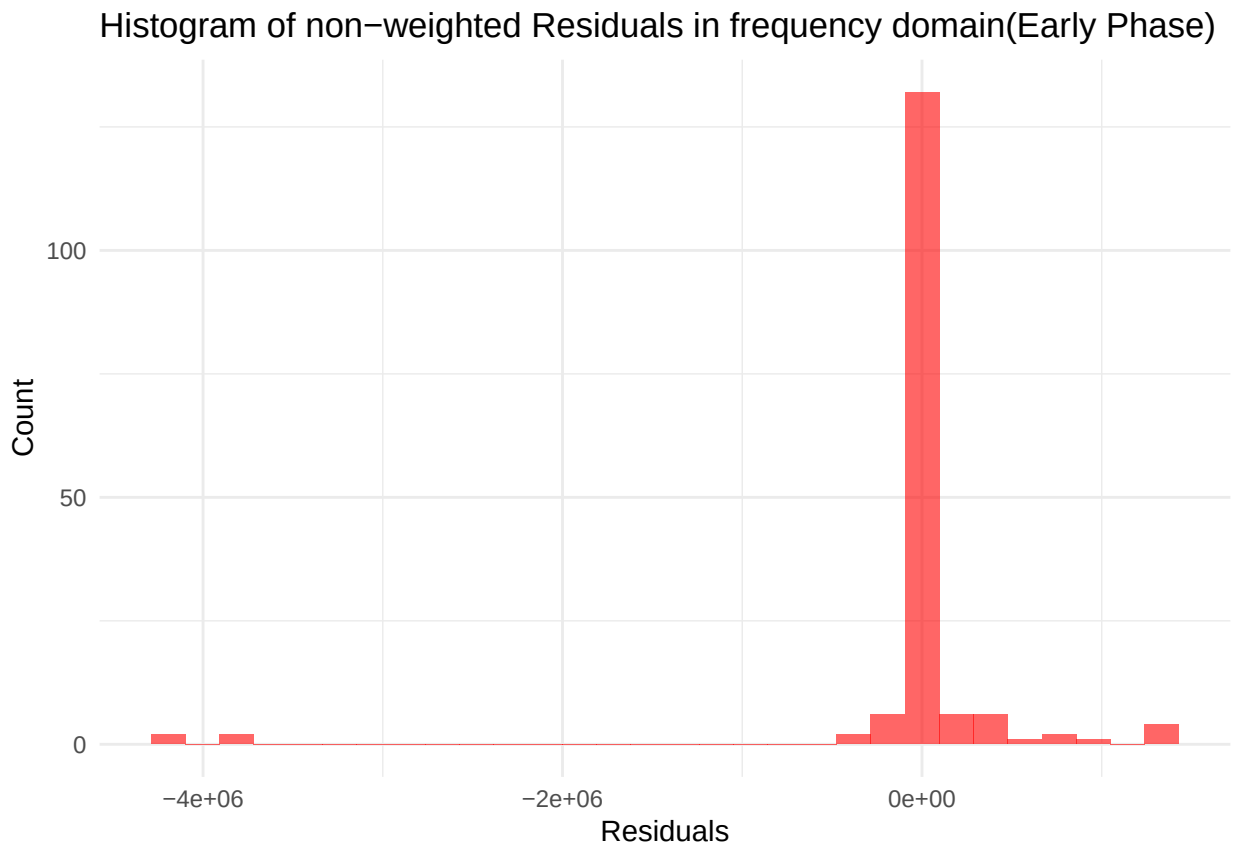
```
param_diff_fft <- params_early_fft - params_late_fft
var_diff_fft <- diag(covariance_early_fft + covariance_late_fft)
se_diff_fft <- sqrt(var_diff_fft)

# --- Step 16: 99% CI ---
```

```r
CI_lower_fft <- param_diff_fft - Z_99 * se_diff_fft
CI_upper_fft <- param_diff_fft + Z_99 * se_diff_fft

# --- Step 17:  ---
df_diff_fft <- data.frame(Parameter = names(param_diff_fft), Difference = param_diff_fft, CI_Lower = CI

plots_fft <- lapply(1:nrow(df_diff_fft), function(i) {
  p <- ggplot(df_diff_fft[i, ], aes(x = Parameter, y = Difference)) +
    geom_point(size = 3, color = "blue") +
    geom_errorbar(aes(ymin = CI_Lower, ymax = CI_Upper), width = 0.2, size = 1, color = "red") +
    geom_hline(yintercept = 0, linetype = "dashed", color = "black", size = 1) +
    labs(title = paste0(df_diff_fft$Parameter[i], " Difference"),
         x = "Parameter", y = "Difference") +
    theme_minimal(base_size = 14)

  #    phi     y
  if (df_diff_fft$Parameter[i] == "phi") {
    p <- p + scale_y_continuous(limits = c(min(df_diff_fft$CI_Lower[i]) * 1, max(df_diff_fft$CI_Upper[i]
  }

  return(p)
})

grid.arrange(grobs = plots_fft, ncol = 3)
```

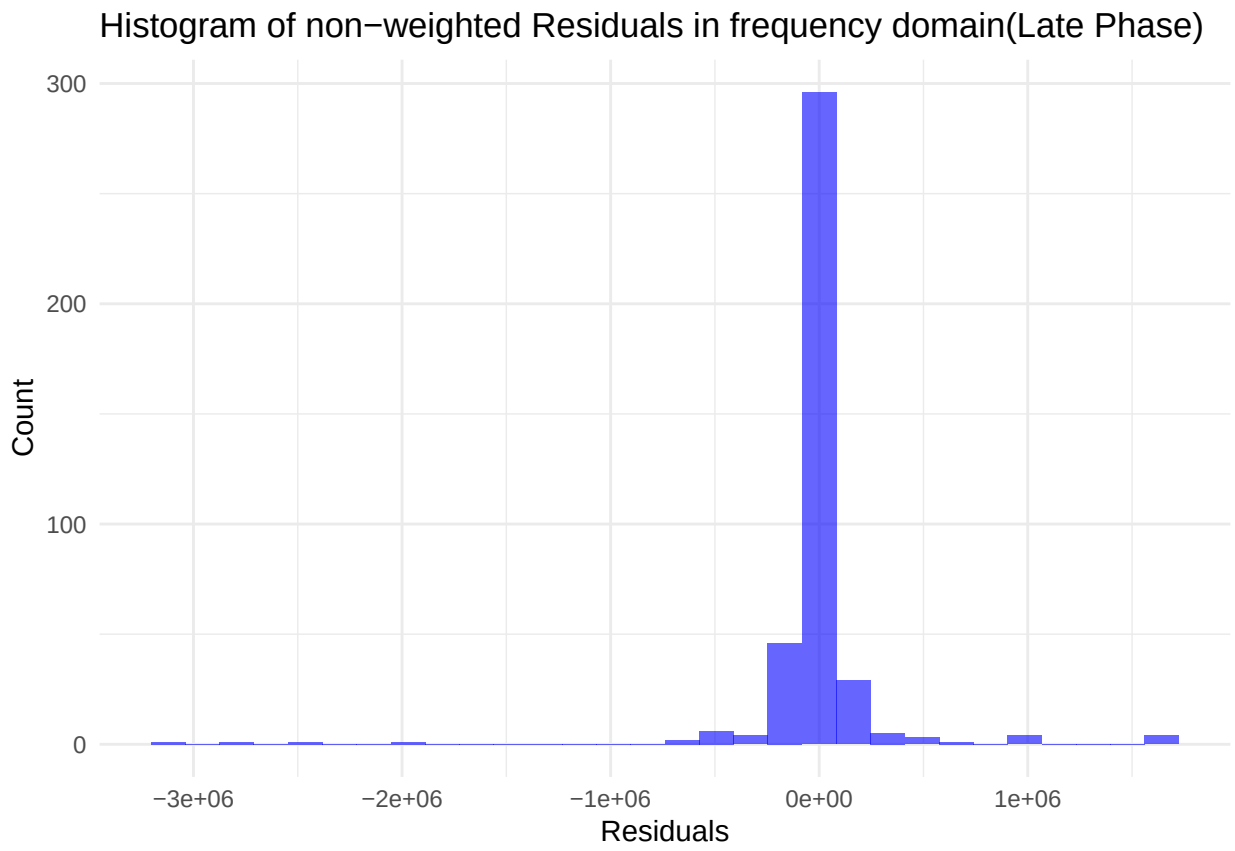
```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_hline()`).
```



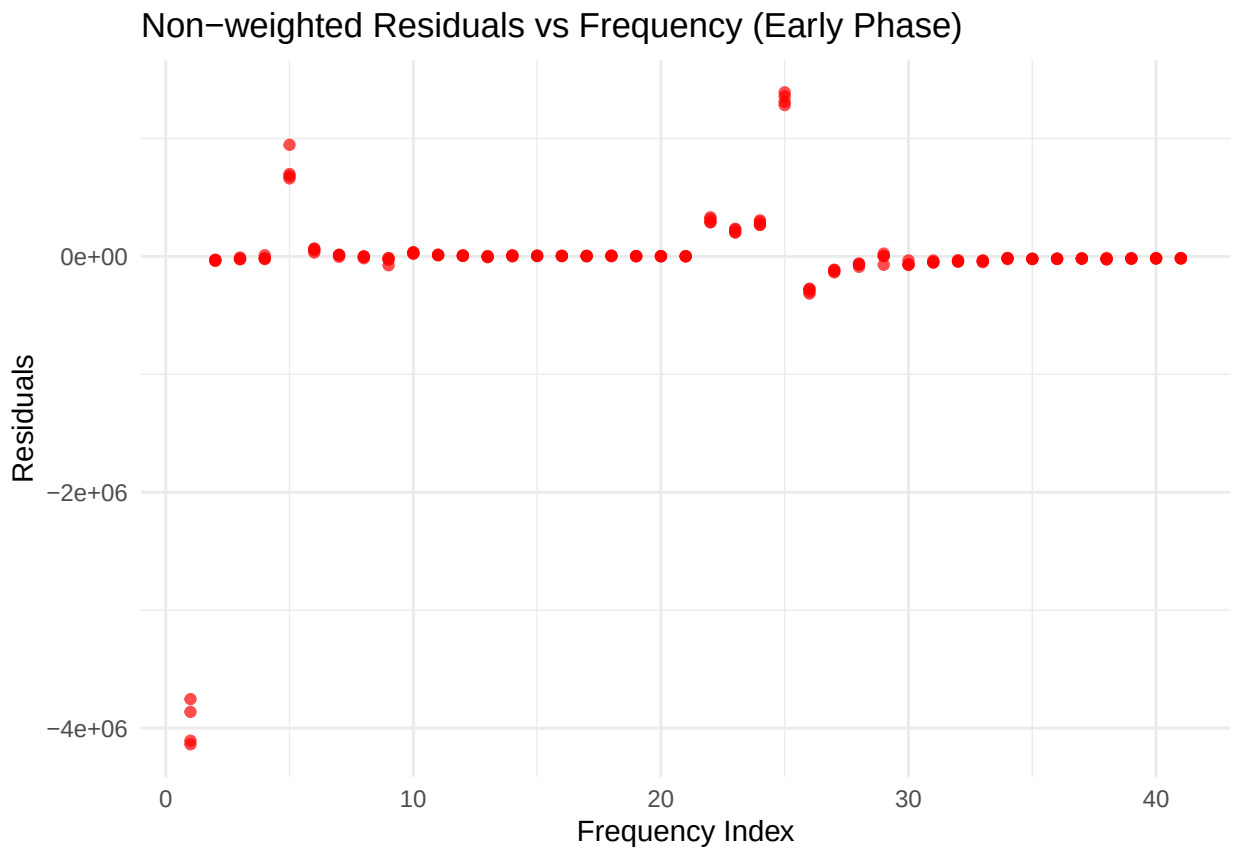

```r
# 2) **computes residuals**
y_early <- replicates[df_MC$Time < 100, ]
y_late <- replicates[df_MC$Time >= 124, ]
fft_replicates_early <- apply(y_early, 2, function(y) fft_extract(y, N_early))
fft_replicates_late <- apply(y_late, 2, function(y) fft_extract(y, N_late))

residuals_early <- fft_replicates_early - fft_extract(y_fit_early_fft,N_early)  #41*4
residuals_late <- fft_replicates_late - fft_extract(y_fit_early_fft,N_late) # 101*4

freq_indices_early <- seq_len(nrow(fft_replicates_early))
freq_indices_late <- seq_len(nrow(fft_replicates_late))

# **plot Early res hist**
df_residuals_early <- data.frame(Residuals = as.vector(residuals_early))
ggplot(df_residuals_early, aes(x = Residuals)) +
  geom_histogram(bins = 30, fill = "red", alpha = 0.6) +
  labs(title = "Histogram of non-weighted Residuals in frequency domain(Early Phase)",
       x = "Residuals", y = "Count") +
  theme_minimal()
```



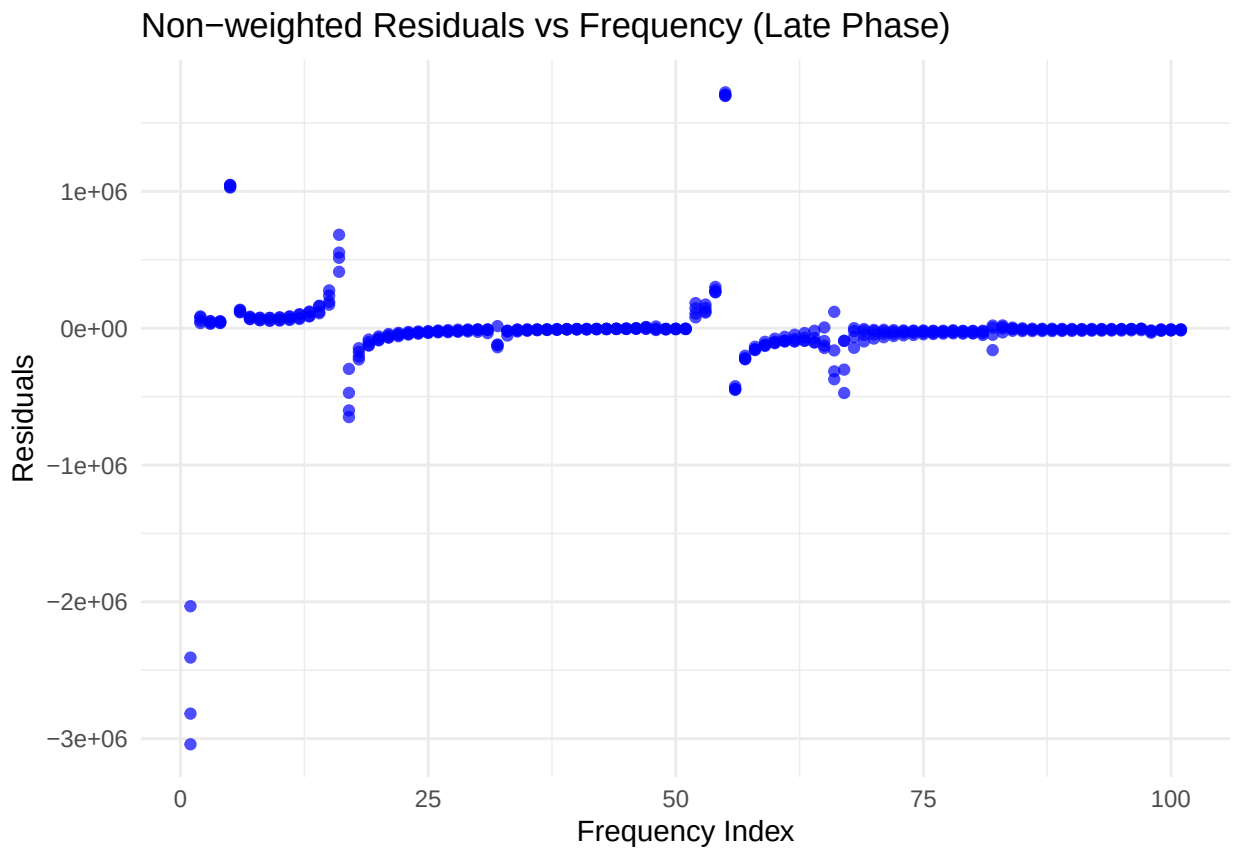

```
# * plot Late res hist**
df_residuals_late <- data.frame(Residuals = as.vector(residuals_late))
ggplot(df_residuals_late, aes(x = Residuals)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.6) +
  labs(title = "Histogram of non-weighted Residuals in frequency domain(Late Phase)",
       x = "Residuals", y = "Count") +
  theme_minimal()
```




```
# **plot Early res vs fre**
df_residual_freq_early <- data.frame(
  Frequency = rep(freq_indices_early, 4),
  Residuals = as.vector(residuals_early)
)

ggplot(df_residual_freq_early, aes(x = Frequency, y = Residuals)) +
  geom_point(color = "red", alpha = 0.7) +
  labs(title = "Non-weighted Residuals vs Frequency (Early Phase)",
       x = "Frequency Index", y = "Residuals") +
  theme_minimal()
```




```
# **plot Late res vs fre**
df_residual_freq_late <- data.frame(
  Frequency = rep(freq_indices_late, 4),
  Residuals = as.vector(residuals_late)
)

ggplot(df_residual_freq_late, aes(x = Frequency, y = Residuals)) +
  geom_point(color = "blue", alpha = 0.7) +
  labs(title = "Non-weighted Residuals vs Frequency (Late Phase)",
       x = "Frequency Index", y = "Residuals") +
  theme_minimal()
```




```
fit_late_fft
```

```
## $estimates
##        A_max      eta_max        A_min      eta_min          tau          phi
## 2.253856e+03 2.208845e-03 2.912648e+02 3.900519e-04 2.590435e-01 4.027894e+00
##
## $conf_intervals
##                  0.5 %       99.5 %
## A_max    2.153365e+03 2.354347e+03
## eta_max  2.049855e-03 2.367834e-03
## A_min    2.212836e+02 3.612460e+02
## eta_min -3.634597e-04 1.143563e-03
## tau      2.586866e-01 2.594004e-01
## phi      3.931307e+00 4.124480e+00
```

```
fit_early_fft
```

```
## $estimates
##         A_max       eta_max         A_min       eta_min           tau
##  3.024485e+03  5.230424e-03  2.459611e+02 -2.745716e-03  2.606086e-01
##           phi
## -2.495776e+00
##
## $conf_intervals
##                 0.5 %        99.5 %
```

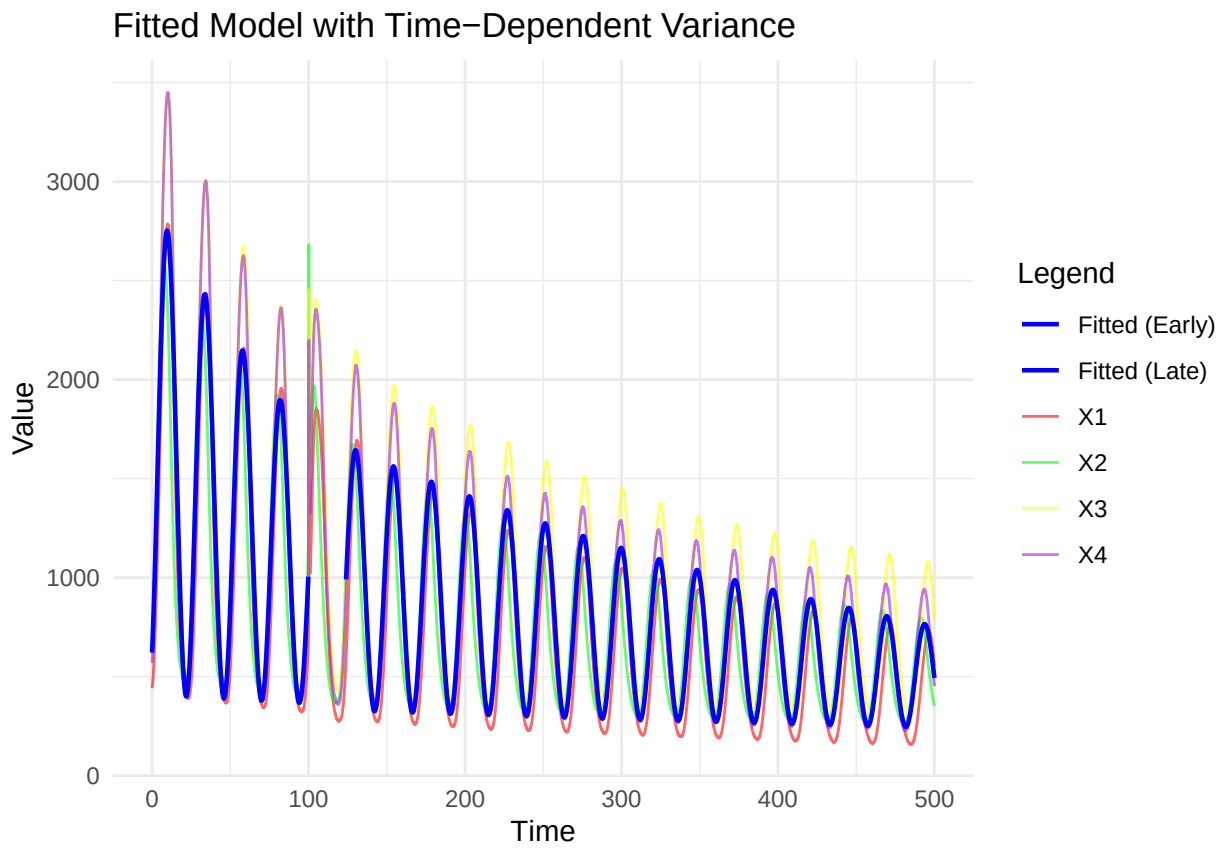
```
## A_max    2.838566e+03  3.210405e+03
## eta_max  3.909811e-03  6.551036e-03
## A_min    8.179902e+01  4.101231e+02
## eta_min -1.311154e-02  7.620111e-03
## tau      2.582465e-01  2.629706e-01
## phi     -2.608455e+00 -2.383098e+00
```

```
# extend bayesian_weights_early match long_MC_early$Value length
bayesian_weights_early_expanded <- rep(bayesian_weights_early, each = 4)
bayesian_weights_late_expanded <- rep(bayesian_weights_late, each = 4)
```

apply model on full dataset

time dependent variance

```
fit_early_weighted <- fit_nls_model_weighted(long_MC_early$Time, long_MC_early$Value, init_param_early,
fit_late_weighted <- fit_nls_model_weighted(long_MC_late$Time, long_MC_late$Value, init_param_late, bay
params_early_weighted <- fit_early_weighted$estimates
params_late_weighted <- fit_late_weighted$estimates
y_fit_early_weighted <- asym_model_function(long_MC_early$Time, params_early_weighted["A_max"], params_
                                            params_early_weighted["A_min"], params_early_weighted["eta_
                                            params_early_weighted["tau"], params_early_weighted["phi"])

y_fit_late_weighted <- asym_model_function(long_MC_late$Time, params_late_weighted["A_max"], params_lat
                                           params_late_weighted["A_min"], params_late_weighted["eta_min
                                           params_late_weighted["tau"], params_late_weighted["phi"])

df_fitted_early_weighted <- data.frame(Time = long_MC_early$Time, Value = y_fit_early_weighted, Type =
df_fitted_late_weighted <- data.frame(Time = long_MC_late$Time, Value = y_fit_late_weighted, Type = "Fi


#
ggplot(long_format_MC, aes(x = Time, y = Value, color = Replicate)) +
  geom_line(alpha = 0.6) +
  geom_line(data = df_fitted_early_weighted, aes(x = Time, y = Value, color = Type), size = 0.8) +
  geom_line(data = df_fitted_late_weighted, aes(x = Time, y = Value, color = Type), size = 0.8) +
  scale_color_manual(values = c("X1" = "red", "X2" = "green", "X3" = "yellow", "X4" = "purple",
                                "Fitted (Early)" = "blue", "Fitted (Late)" = "blue")) +
  labs(title = "Fitted Model with Time-Dependent Variance",
       x = "Time", y = "Value", color = "Legend") +
  theme_minimal()
```




```
fit_early_weighted
```

```
## $estimates
##         A_max       eta_max         A_min       eta_min           tau
##  2.892957e+03  5.150019e-03  4.105308e+02  1.152808e-03  2.607541e-01
##           phi
## -2.549593e+00
##
## $conf_intervals
##                  0.5 %        99.5 %
## A_max     2.835549e+03  2.950364e+03
## eta_max   4.785479e-03  5.514558e-03
## A_min     3.940935e+02  4.269680e+02
## eta_min   5.269720e-04  1.778645e-03
## tau       2.603403e-01  2.611679e-01
## phi      -2.572594e+00 -2.526592e+00
```

```
fit_late_weighted
```

```
## $estimates
##         A_max       eta_max         A_min       eta_min           tau           phi
## 2.160109e+03 2.100972e-03 3.652751e+02 8.141678e-04 2.590978e-01 3.990836e+00
##
## $conf_intervals
##                 0.5 %       99.5 %
```

```
## A_max   2.123946e+03 2.196271e+03
## eta_max 2.046043e-03 2.155902e-03
## A_min   3.545507e+02 3.759995e+02
## eta_min 7.211388e-04 9.071967e-04
## tau     2.590020e-01 2.591936e-01
## phi     3.963655e+00 4.018018e+00
```

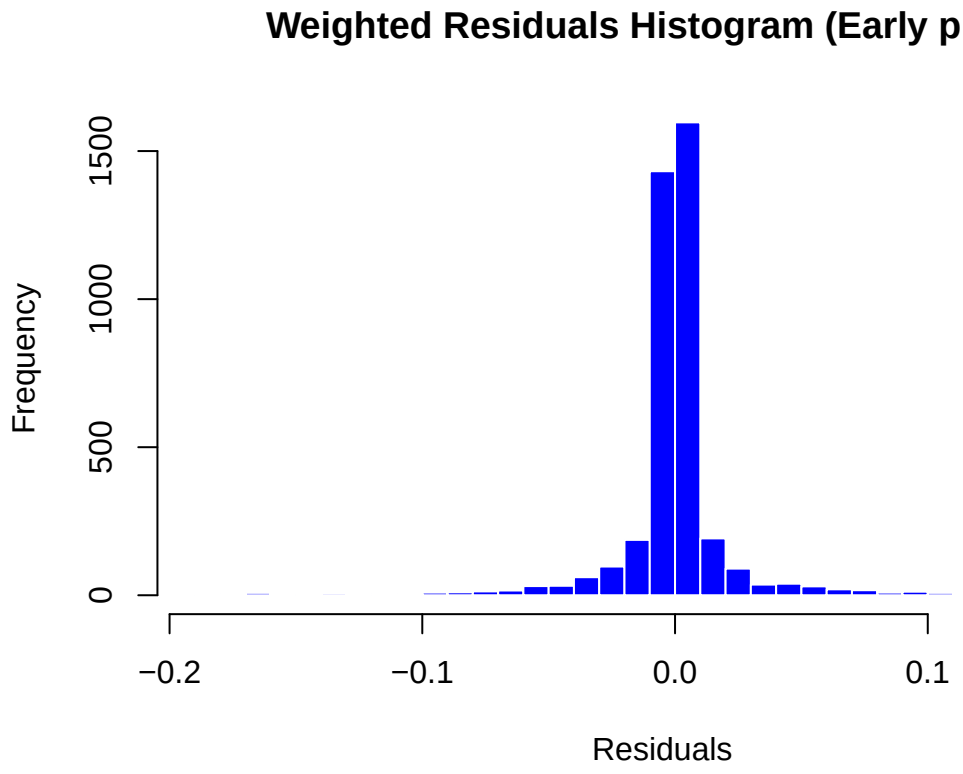
```
param_diff
```

```
##         A_max       eta_max         A_min       eta_min           tau
## 772.133862066   0.003025822 -39.916301870  -0.002932436   0.001520399
```

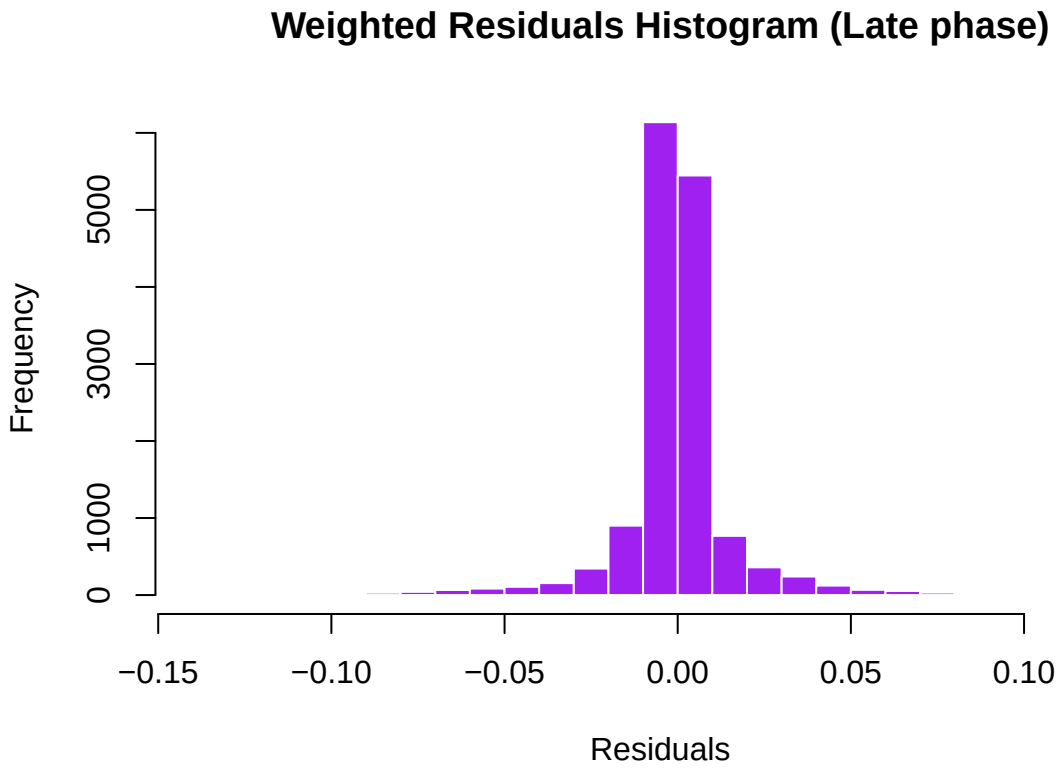
```
var_diff
```

```
## [1] 4.388133e+02 1.450621e-08 2.857823e+02 8.037885e-07 4.468650e-08
```

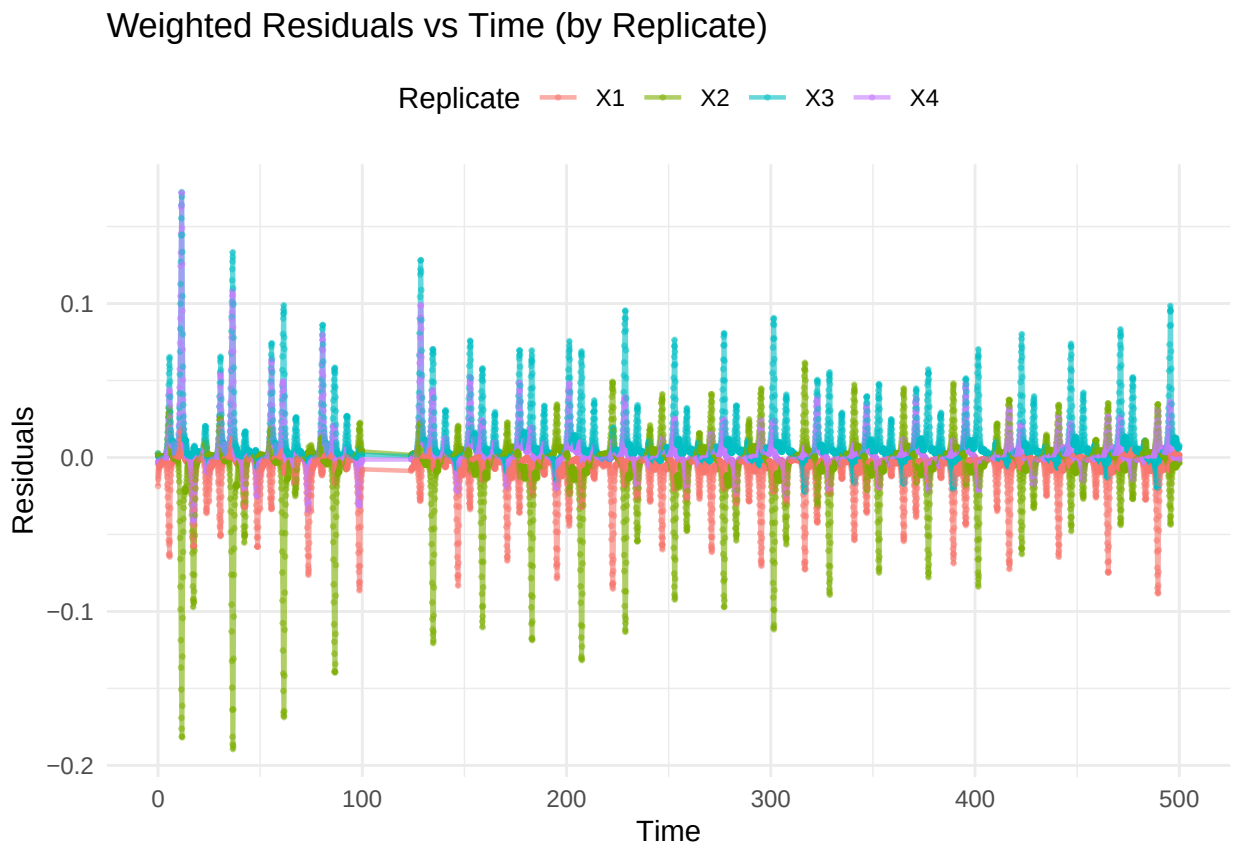
```
residuals_early <- compute_residuals_asym(long_MC_early$Value, long_MC_early$Time, params_early_weighte
residuals_late <- compute_residuals_asym(long_MC_late$Value, long_MC_late$Time, params_late_weighted)

residuals_early_weighted <- residuals_early * bayesian_weights_early
residuals_late_weighted <- residuals_late * bayesian_weights_late

# residual hist

hist(residuals_early_weighted, breaks = 30, col = "blue", border = "white",
     main = "Weighted Residuals Histogram (Early phase)", xlab = "Residuals", ylab = "Frequency")
```



```r
hist(residuals_late_weighted, breaks = 30, col = "purple", border = "white",
     main = "Weighted Residuals Histogram (Late phase)", xlab = "Residuals", ylab = "Frequency")
```




```r
df_residuals_weighted <- bind_rows(
  data.frame(Time = long_MC_early$Time, Residual = residuals_early_weighted, Phase = "Early", Replicate
  data.frame(Time = long_MC_late$Time, Residual = residuals_late_weighted, Phase = "Late", Replicate = 1
)
# residual vs time
ggplot(df_residuals_weighted, aes(x = Time, y = Residual, color = Replicate, group = Replicate)) +
  geom_line(alpha = 0.6, size = 0.8) +
  geom_point(size = 0.5, alpha = 0.5) +
  labs(title = "Weighted Residuals vs Time (by Replicate)", x = "Time", y = "Residuals", color = "Replic
  theme_minimal() +
  theme(legend.position = "top")
```




```
#
sigma2_early_weighted <- estimate_sigma2_asym(long_MC_early$Value, long_MC_early$Time, params_early_weig
sigma2_late_weighted <- estimate_sigma2_asym(long_MC_late$Value, long_MC_late$Time, params_late_weighted

covariance_early_weighted <- compute_covariance_asym(params_early_weighted, long_MC_early$Time, long_MC
covariance_late_weighted <- compute_covariance_asym(params_late_weighted, long_MC_late$Time, long_MC_la

wald_result_weighted <- wald_test(params_early_weighted, params_late_weighted, covariance_early_weighted

cat("Wald Test Statistic:", wald_result_weighted$T_Wald, "\n")
```

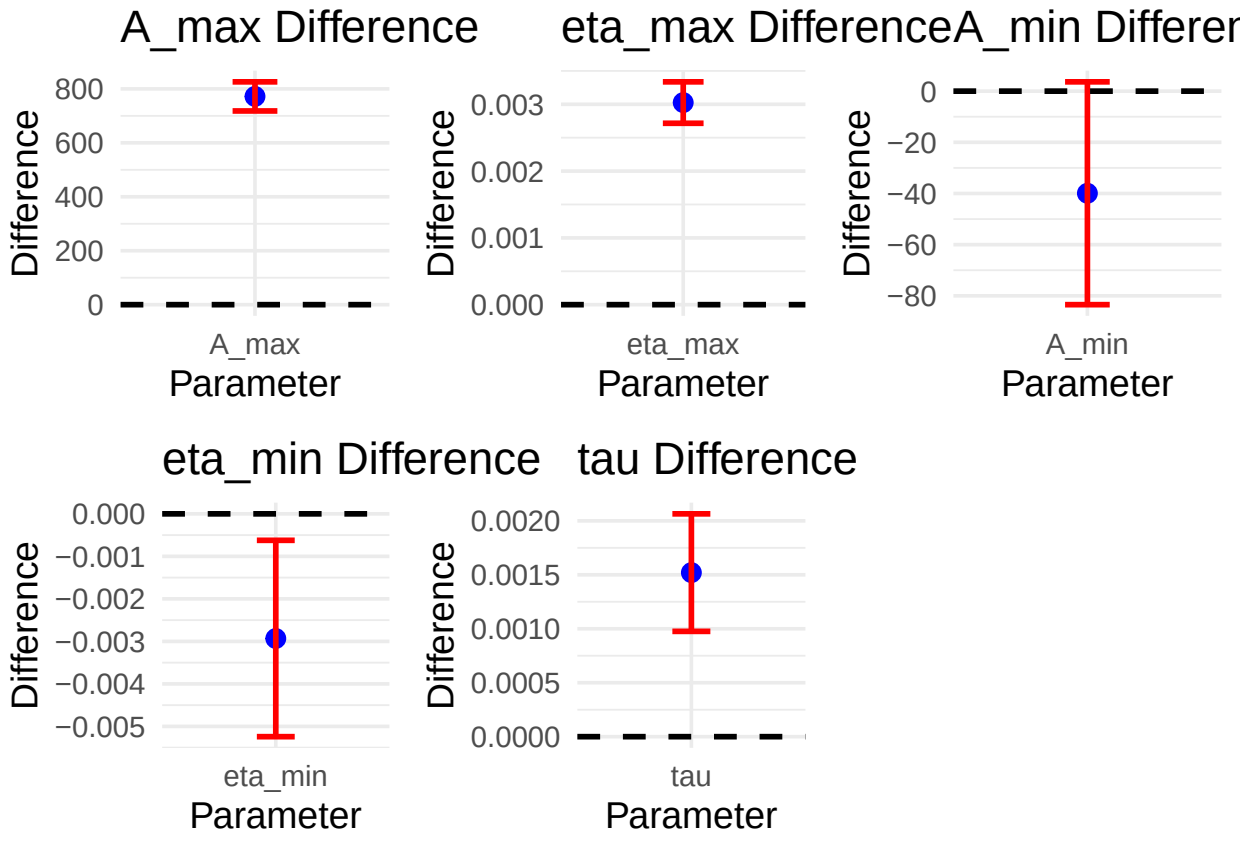
```
## Wald Test Statistic: 327495.7
```

```
cat("p-value:", wald_result_weighted$p_value, "\n")
```

```
## p-value: 0
```

```
# Compute parameter differences
param_diff <- params_early_weighted - params_late_weighted
var_diff <- diag(covariance_early_weighted + covariance_late_weighted)
se_diff <- sqrt(var_diff)

# Compute 99% confidence intervals
Z_99 <- 2.576
```

```r
CI_lower <- param_diff - Z_99 * se_diff
CI_upper <- param_diff + Z_99 * se_diff

# Create data frame for plotting
df_diff_weighted <- data.frame(Parameter = names(param_diff), Difference = param_diff, CI_Lower = CI_lo


# Generate individual plots for each parameter
plots <- lapply(1:nrow(df_diff), function(i) {
  ggplot(df_diff[i, ], aes(x = Parameter, y = Difference)) +
    geom_point(size = 3, color = "blue") +  # Show parameter difference
    geom_errorbar(aes(ymin = CI_Lower, ymax = CI_Upper), width = 0.2, size = 1, color = "red") +  # Con
    geom_hline(yintercept = 0, linetype = "dashed", color = "black", size = 1) +  # Reference line at 0
    labs(title = paste0(df_diff$Parameter[i], " Difference"),
         x = "Parameter", y = "Difference") +
    theme_minimal(base_size = 14)
})
grid.arrange(grobs = plots, ncol = 3)
```



time dependent variance with fft(full dataset)

```r
library(minpack.lm)
```

```r
fft_extract <- function(y, N) {
  y_fft <- fft(y)

  Re_part <- Re(y_fft)[1:(N + 1)]
  Im_part <- Im(y_fft)[2:(N + 1)]
  c(Re_part, Im_part)
}
#  y_early is matrix
y_early <- replicates[df_MC$Time < 100, ]
time_early <- df_MC$Time[df_MC$Time < 100]

N_early <- 20
# fft_replicates_early : (2*N_early+1)  × 4
fft_replicates_early <- apply(y_early, 2, function(y) fft_extract(y, N_early))
#
observed_variances_early <- calculate_variance(fft_replicates_early)
weights_early <- 1 / sqrt(observed_variances_early)
#
weights_early[is.infinite(weights_early)] <- max(weights_early[!is.infinite(weights_early)])
weights_early_stacked <- rep(weights_early, each = 4)
#
y_fft_stacked_early <- c(t(fft_replicates_early))  #

stacked_fft_pred <- function(t, A_max, eta_max, A_min, eta_min, tau, phi, N, n_reps = 4) {
  # 1)  time-domain
  pred_time_domain <- asym_model_function(t, A_max, eta_max, A_min, eta_min, tau, phi)

  # 2) freq-domain
  pred_fft <- fft_extract(pred_time_domain, N)  # lenth =2*N+1

  # 3)  pred_fft become a long vector(replicates 4 times)
  rep_pred <- rep(pred_fft, each = n_reps)

  return(rep_pred)
}

start_early <- unlist(init_param_early)

#  nlsLM fit
fit_early_fft_stacked <- try(
  nlsLM(
    formula = y_fft_stacked_early ~ stacked_fft_pred(
      t = time_early,
      A_max, eta_max, A_min, eta_min, tau, phi,
      N = N_early,
      n_reps = 4
    ),
    start = start_early,
    weights = weights_early_stacked,
    lower = lower_bounds,
    upper = upper_bounds,
    control = nls.lm.control(maxiter = 500)  #
  ),
```

```
  silent = TRUE
)

if (inherits(fit_early_fft_stacked, "try-error")) {
  cat("nlsLM fitting in frequency domain (stacked) failed.\n")
} else {
  summary(fit_early_fft_stacked)

  params_early_fft_stacked <- coef(fit_early_fft_stacked)
  confint_early_fft_stacked <- confint2(fit_early_fft_stacked, level = 0.99, method = 'asymptotic')
    # adjust phi (control between -2pi and 2pi)
  params_early_fft_stacked["phi"] <- ((params_early_fft_stacked["phi"] + 2 * pi) %% (4 * pi)) - 2 * pi

  # adjust phi
  confint_early_fft_stacked["phi", ] <- ((confint_early_fft_stacked["phi", ] + 2 * pi) %% (4 * pi)) - 2

  print("Estimated parameters (stacked, early):")
  print(params_early_fft_stacked)

  print("\n99% confidence intervals (phi adjusted):")
  print(confint_early_fft_stacked)

}
```

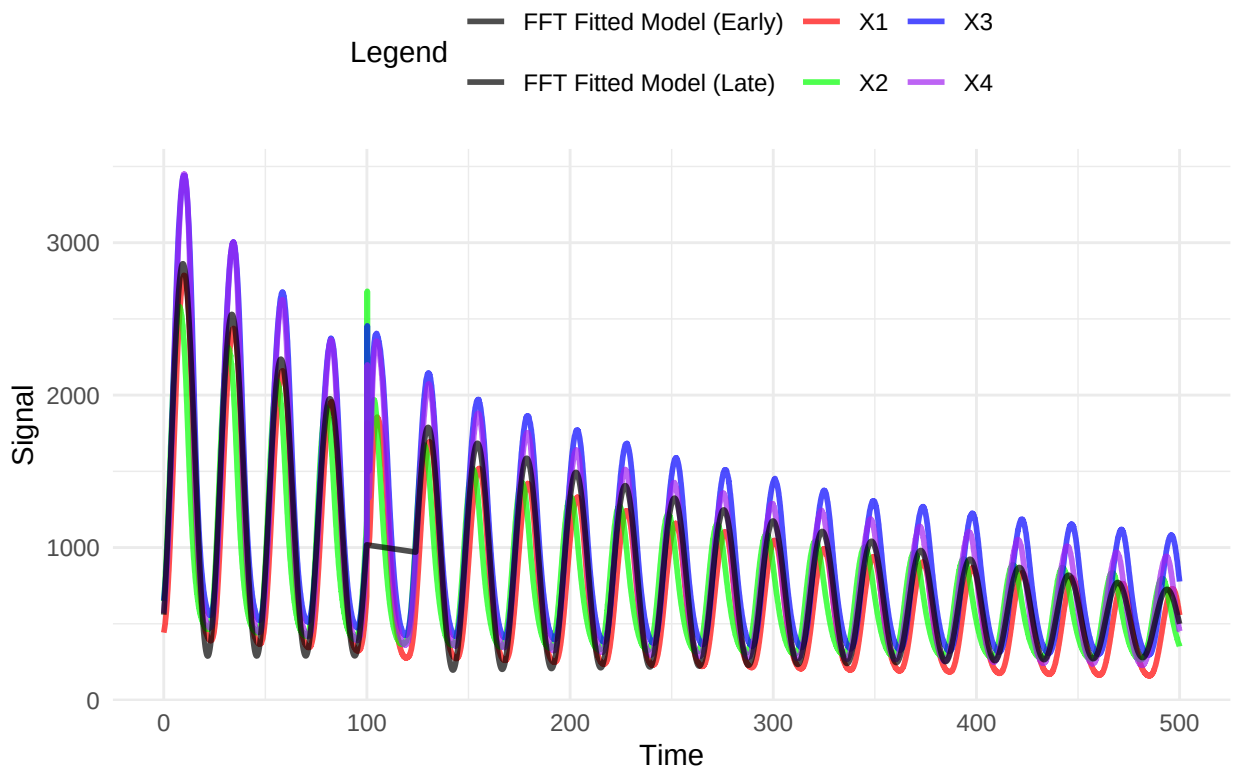
```
## [1] "Estimated parameters (stacked, early):"
##         A_max       eta_max         A_min       eta_min           tau
##  3.004736e+03  5.117529e-03  2.876147e+02 -7.271554e-05  2.604783e-01
##           phi
## -2.497223e+00
## [1] "\n99% confidence intervals (phi adjusted):"
##                  0.5 %        99.5 %
## A_max    2.807011e+03  3.202461e+03
## eta_max  4.114738e-03  6.120319e-03
## A_min    1.011797e+02  4.740496e+02
## eta_min -7.530763e-03  7.385332e-03
## tau      2.584334e-01  2.625231e-01
## phi     -2.606297e+00 -2.388148e+00
```

```
y_late <- replicates[df_MC$Time >= 124, ]
time_late <- df_MC$Time[df_MC$Time >= 124]

N_late <- 50
# fft_replicates_early   (2*N_early+1)  × 4
fft_replicates_late <- apply(y_late, 2, function(y) fft_extract(y, N_late))


observed_variances_late <- calculate_variance(fft_replicates_late)
weights_late <- 1 / sqrt(observed_variances_late)

weights_late[is.infinite(weights_late)] <- max(weights_late[!is.infinite(weights_late)])
weights_late_stacked <- rep(weights_late, each = 4)
#
y_fft_stacked_late <- c(t(fft_replicates_late))  #
```

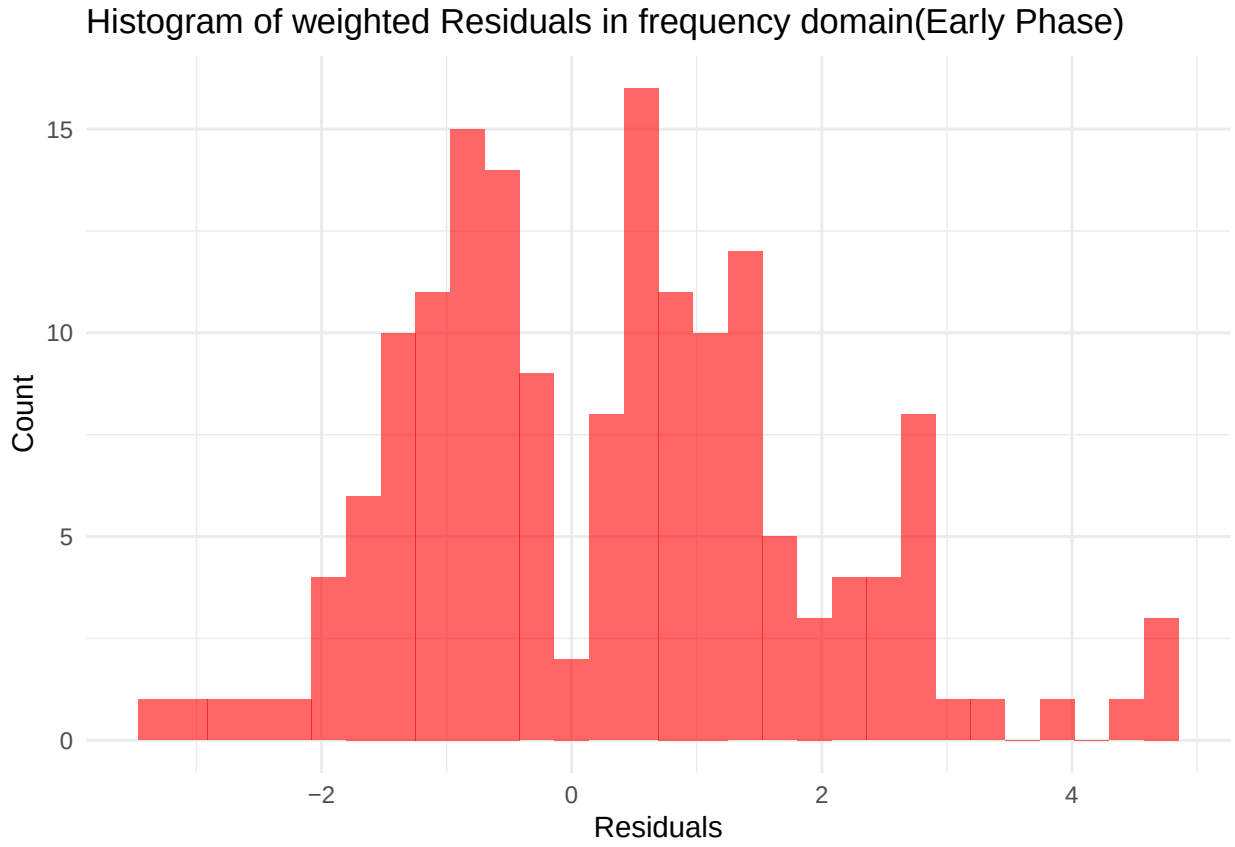
```r
#
fit_late_fft_stacked <- try(
  nlsLM(
    formula = y_fft_stacked_late ~ stacked_fft_pred(
      t = time_late,
      A_max, eta_max, A_min, eta_min, tau, phi,
      N = N_late,
      n_reps = 4
    ),
    start = init_param_late,
    weights = weights_late_stacked,
    lower = lower_bounds,
    upper = upper_bounds,
    control = nls.lm.control(maxiter = 1000)
  ),
  silent = TRUE
)
if (inherits(fit_late_fft_stacked, "try-error")) {
  cat("nlsLM fitting in frequency domain (stacked) failed.\n")
} else {
  summary(fit_late_fft_stacked)
  #
  params_late_fft_stacked <- coef(fit_late_fft_stacked)
  confint_late_fft_stacked <- confint2(fit_late_fft_stacked, level = 0.99, method = 'asymptotic')
    #  adjust phi
  params_late_fft_stacked["phi"] <- ((params_late_fft_stacked["phi"] + 2 * pi) %% (4 * pi)) - 2 * pi


  confint_late_fft_stacked["phi", ] <- ((confint_late_fft_stacked["phi", ] + 2 * pi) %% (4 * pi)) - 2 *

  print("Estimated parameters (stacked, early):")
  print(params_late_fft_stacked)

  print("\n99% confidence intervals (phi adjusted):")
  print(confint_late_fft_stacked)

}
```

```
## [1] "Estimated parameters (stacked, early):"
##         A_max       eta_max         A_min       eta_min           tau
##  2.469698e+03  2.479215e-03  1.695965e+02 -1.027564e-03  2.591294e-01
##           phi
##  3.954056e+00
## [1] "\n99% confidence intervals (phi adjusted):"
##                  0.5 %       99.5 %
## A_max    2.264249e+03 2.675148e+03
## eta_max  2.162924e-03 2.795505e-03
## A_min    7.813878e+01 2.610541e+02
## eta_min -2.578736e-03 5.236085e-04
## tau      2.583105e-01 2.599483e-01
## phi      3.767161e+00 4.140951e+00
```

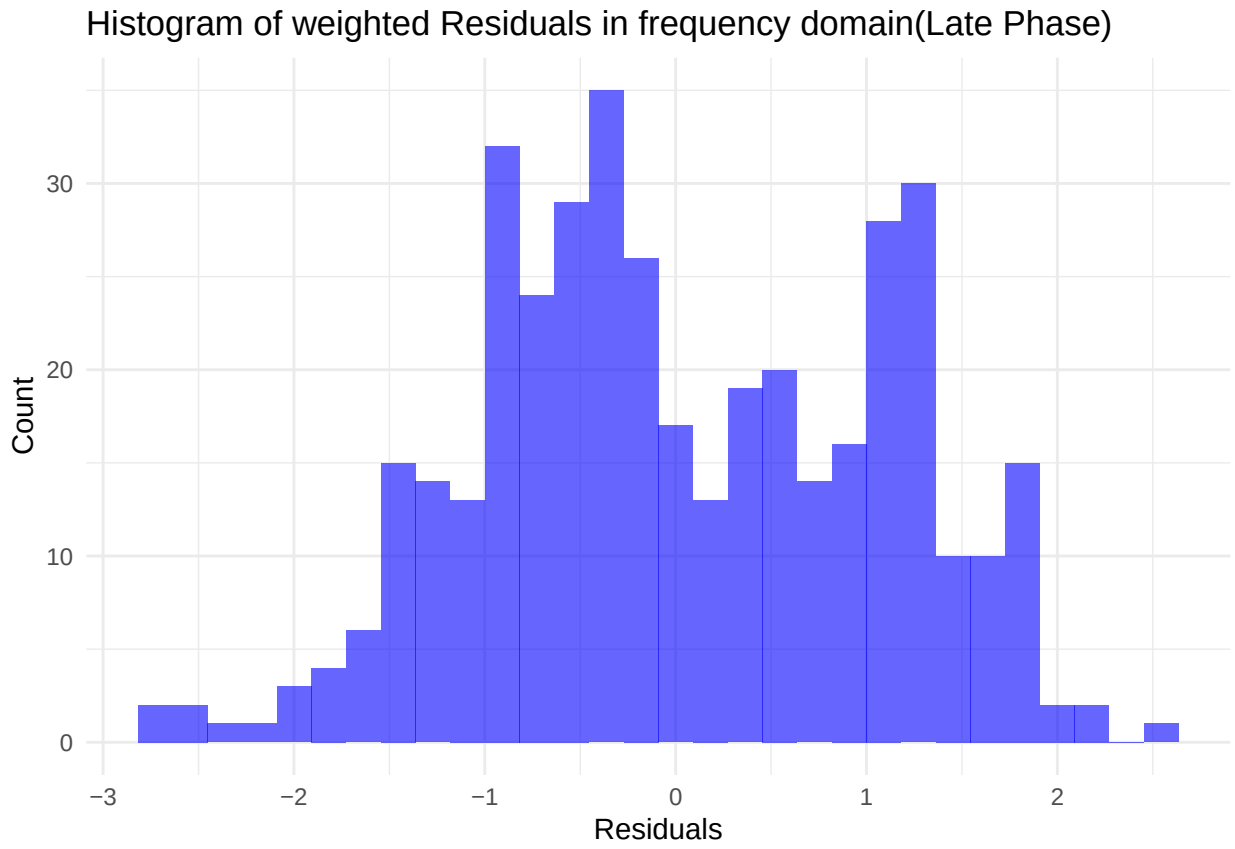
```r
library(ggplot2)
library(dplyr)
library(gridExtra)

# 1) **fitted curve vs original curves**
y_fit_early <- asym_model_function(time_early, params_early_fft_stacked["A_max"],
                                   params_early_fft_stacked["eta_max"],
                                   params_early_fft_stacked["A_min"],
                                   params_early_fft_stacked["eta_min"],
                                   params_early_fft_stacked["tau"],
                                   params_early_fft_stacked["phi"])

y_fit_late <- asym_model_function(time_late, params_late_fft_stacked["A_max"],
                                  params_late_fft_stacked["eta_max"],
                                  params_late_fft_stacked["A_min"],
                                  params_late_fft_stacked["eta_min"],
                                  params_late_fft_stacked["tau"],
                                  params_late_fft_stacked["phi"])

df_fitted_early <- data.frame(Time = time_early, Value = y_fit_early, Type = "FFT Fitted Model (Early)"
df_fitted_late <- data.frame(Time = time_late, Value = y_fit_late, Type = "FFT Fitted Model (Late)")

df_original <- df_MC %>% pivot_longer(cols = c("X1", "X2", "X3", "X4"), names_to = "Replicate", values_

df_plot <- bind_rows(df_original %>% mutate(Type = Replicate), df_fitted_early, df_fitted_late)

ggplot(df_plot, aes(x = Time, y = Value, color = Type, group = interaction(Replicate, Type))) +
  geom_line(size = 1, alpha = 0.7) +
  scale_color_manual(values = c("X1" = "red", "X2" = "green", "X3" = "blue", "X4" = "purple",
                                "FFT Fitted Model (Early)" = "black", "FFT Fitted Model (Late)" = "blac
  labs(title = "Comparison of FFT Weighted Fitted Model with Original Replicates",
       x = "Time", y = "Signal", color = "Legend") +
  theme_minimal() +
  theme(legend.position = "top")
```



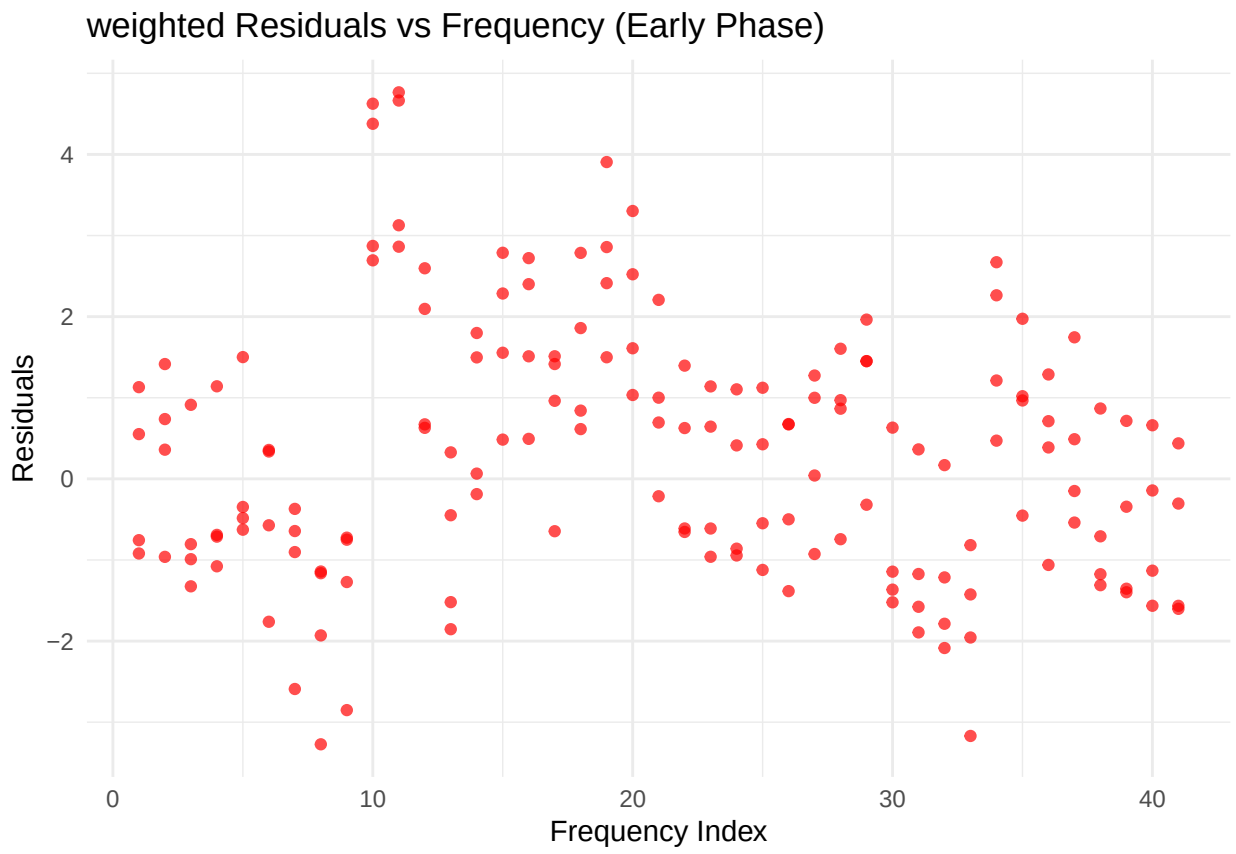

```r
# 2) **computes residuals**
residuals_early <- (fft_replicates_early - fft_extract(y_fit_early,N_early)) * weights_early #41*4
residuals_late <- (fft_replicates_late - fft_extract(y_fit_late,N_late)) * weights_late # 101*4

# **plot Early res hist**
df_residuals_early <- data.frame(Residuals = as.vector(residuals_early))
ggplot(df_residuals_early, aes(x = Residuals)) +
  geom_histogram(bins = 30, fill = "red", alpha = 0.6) +
  labs(title = "Histogram of weighted Residuals in frequency domain(Early Phase)",
       x = "Residuals", y = "Count") +
  theme_minimal()
```



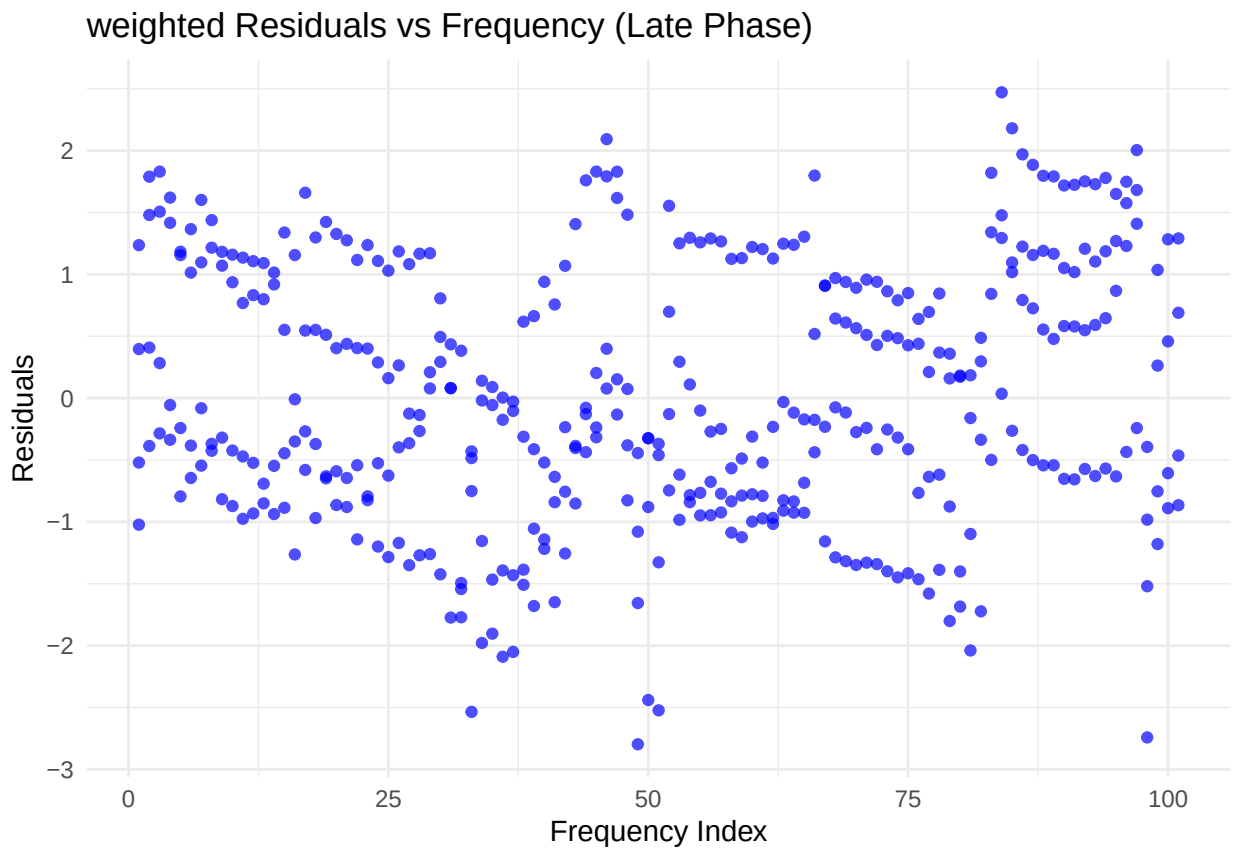

```r
# * plot Late res hist**
df_residuals_late <- data.frame(Residuals = as.vector(residuals_late))
ggplot(df_residuals_late, aes(x = Residuals)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.6) +
  labs(title = "Histogram of weighted Residuals in frequency domain(Late Phase)",
       x = "Residuals", y = "Count") +
  theme_minimal()
```




```r
# **plot Early res vs fre**
df_residual_freq_early <- data.frame(
  Frequency = rep(freq_indices_early, 4),
  Residuals = as.vector(residuals_early)
)

ggplot(df_residual_freq_early, aes(x = Frequency, y = Residuals)) +
  geom_point(color = "red", alpha = 0.7) +
  labs(title = "weighted Residuals vs Frequency (Early Phase)",
       x = "Frequency Index", y = "Residuals") +
  theme_minimal()
```




```r
# **plot Late res vs fre**
df_residual_freq_late <- data.frame(
  Frequency = rep(freq_indices_late, 4),
  Residuals = as.vector(residuals_late)
)

ggplot(df_residual_freq_late, aes(x = Frequency, y = Residuals)) +
  geom_point(color = "blue", alpha = 0.7) +
  labs(title = "weighted Residuals vs Frequency (Late Phase)",
       x = "Frequency Index", y = "Residuals") +
  theme_minimal()
```



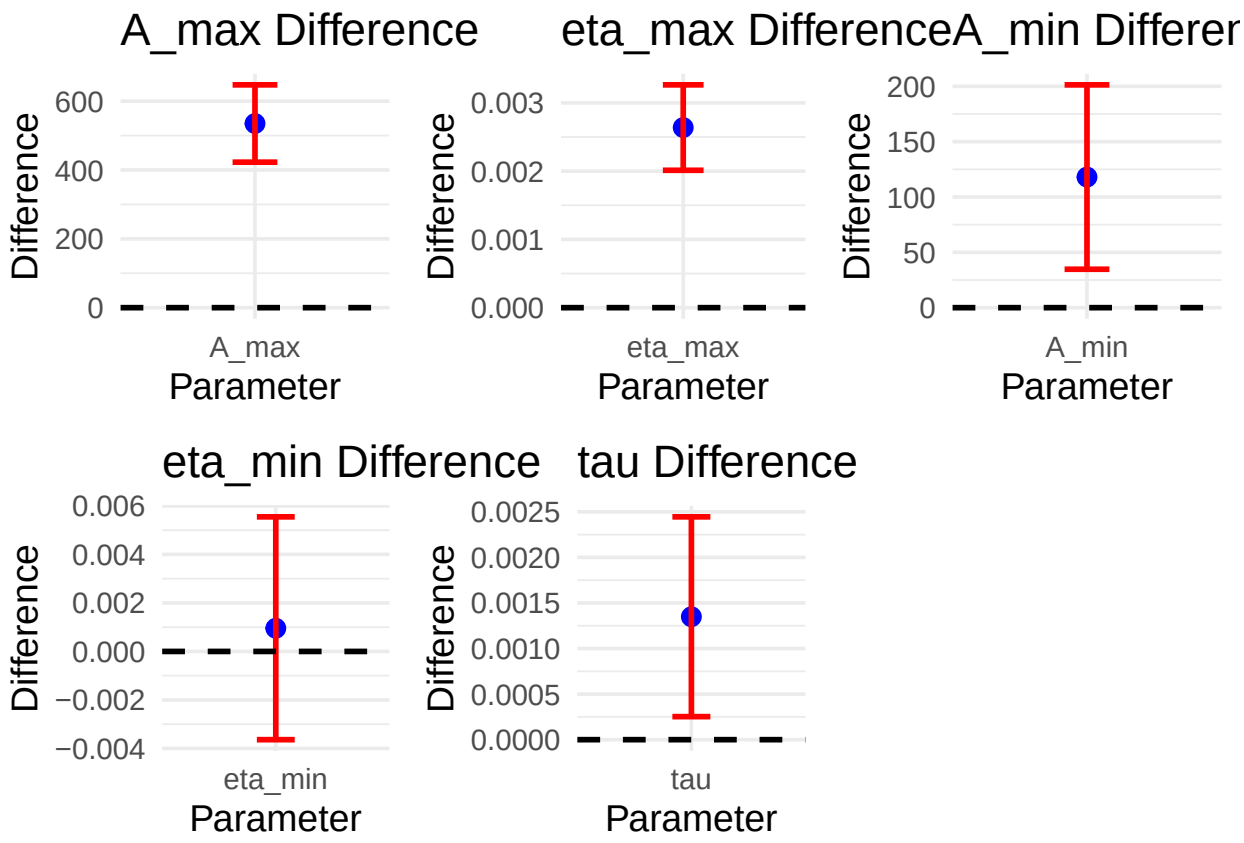

```r
# 3) **T Wald hypothesis test**
sigma2_early_fft_weighted <- estimate_sigma2_asym(y_early, time_early, params_early_fft_stacked)
sigma2_late_fft_weighted <- estimate_sigma2_asym(y_late, time_late, params_late_fft_stacked)

covariance_early_fft_weighted <- compute_covariance_asym(params_early_fft_stacked, time_early, y_early,
covariance_late_fft_weighted <- compute_covariance_asym(params_late_fft_stacked, time_late, y_late, sig

early_filtered_fft_weighted <- remove_phi_asym(params_early_fft_stacked, covariance_early_fft_weighted)
late_filtered_fft_weighted <- remove_phi_asym(params_late_fft_stacked, covariance_late_fft_weighted)

wald_result_fft_weighted <- wald_test(
  early_filtered_fft_weighted$params, late_filtered_fft_weighted$params,
  early_filtered_fft_weighted$covariance, late_filtered_fft_weighted$covariance
)

cat("Wald Test Statistic (FFT Weighted):", wald_result_fft_weighted$T_Wald, "\n")
```

```
## Wald Test Statistic (FFT Weighted): 234.4948
```

```r
cat("p-value:", wald_result_fft_weighted$p_value, "\n")
```

```
## p-value: 0
```

```r
# 4) **99% CI**
covariance_early_weighted_fft <- early_filtered_fft_weighted$covariance
covariance_late_weighted_fft <- late_filtered_fft_weighted$covariance

#
fft_param_names <- names(early_filtered_fft_weighted$params)
weighted_fft_param_diff <- early_filtered_fft_weighted$params - late_filtered_fft_weighted$params
var_diff_weighted <- diag(covariance_early_weighted_fft + covariance_late_weighted_fft)  #
se_diff_weighted <- sqrt(var_diff_weighted)
Z_99 <- qnorm(0.995)
CI_lower <- weighted_fft_param_diff - Z_99 * se_diff_weighted
CI_upper <- weighted_fft_param_diff + Z_99 * se_diff_weighted

df_diff_fft_weighted <- data.frame(
  Parameter = fft_param_names,
  Difference = weighted_fft_param_diff,
  CI_Lower = CI_lower,
  CI_Upper = CI_upper
)

#  99% CI plot
plots <- lapply(1:nrow(df_diff_fft_weighted), function(i) {
  ggplot(df_diff_fft_weighted[i, ], aes(x = Parameter, y = Difference)) +
    geom_point(size = 3, color = "blue") +  #
    geom_errorbar(aes(ymin = CI_Lower, ymax = CI_Upper), width = 0.2, size = 1, color = "red") +  # CI
    geom_hline(yintercept = 0, linetype = "dashed", color = "black", size = 1) +  #
    labs(title = paste0(df_diff_fft_weighted$Parameter[i], " Difference"),
         x = "Parameter", y = "Difference") +
    theme_minimal(base_size = 14)
})

#  use gridExtra to combine multiple graph
grid.arrange(grobs = plots, ncol = 3)
```



NLME

```
# Convert data to long format
df_long <- df_MC %>%
  pivot_longer(cols = c("X1", "X2", "X3", "X4"),
               names_to = "repl",
               values_to = "y")

# Ensure `repl` is a factor variable (important for `nlme()`)
df_long$repl <- as.factor(df_long$repl)
long_early <- df_long %>% filter(Time < 100)
long_late <- df_long %>% filter(Time >= 124)
```

```
# Fit the nonlinear mixed-effects model
mixed_early <- nlme(y ~ asym_model_function(Time, A_max, eta_max, A_min, eta_min, tau, phi),
                random = A_max +  A_min +  tau ~ 1 | repl,  # Allow these parameters to vary across rep
                fixed = A_max + eta_max + A_min + eta_min + tau + phi ~ 1,

# Fixed effects shared across replicates
                data = long_early,  # Use the transformed long-format data
                start = c(init_param_early$A_max, init_param_early$eta_max, init_param_early$A_min, ini
)
```

```
## Warning in nlme.formula(y ~ asym_model_function(Time, A_max, eta_max, A_min, :
```

```
## Iteration 2, LME step: nlminb() did not converge (code = 1). PORT message:
## false convergence (8)
```

```
summary(mixed_early)
```

```
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: y ~ asym_model_function(Time, A_max, eta_max, A_min, eta_min,      tau, phi)
##   Data: long_early
##        AIC      BIC    logLik
##   51585.74 51667.57 -25779.87
##
## Random effects:
##  Formula: list(A_max ~ 1, A_min ~ 1, tau ~ 1)
##  Level: repl
##  Structure: General positive-definite, Log-Cholesky parametrization
##          StdDev       Corr
## A_max    3.840011e+02 A_max  A_min
## A_min    5.680156e+01  0.348
## tau      3.654201e-03 -0.639  0.238
## Residual 1.510414e+02
##
## Fixed effects:  A_max + eta_max + A_min + eta_min + tau + phi ~ 1
##             Value Std.Error   DF   t-value p-value
## A_max   3042.0324 192.36605 3991   15.8138       0
## eta_max    0.0052   0.00006 3991   81.9672       0
## A_min    239.8822  29.54395 3991    8.1195       0
## eta_min   -0.0022   0.00052 3991   -4.3319       0
## tau        0.2612   0.00183 3991  142.5586       0
## phi       -2.5097   0.00540 3991 -464.3329       0
##  Correlation:
##         A_max  eta_mx A_min  eta_mn tau
## eta_max  0.039
## A_min    0.330 -0.060
## eta_min -0.015 -0.298  0.243
## tau     -0.637  0.004  0.231  0.009
## phi     -0.005 -0.090 -0.041 -0.140 -0.050
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -3.83861736 -0.77161084  0.07324958  0.78227563  2.36980665
##
## Number of Observations: 4000
## Number of Groups: 4
```

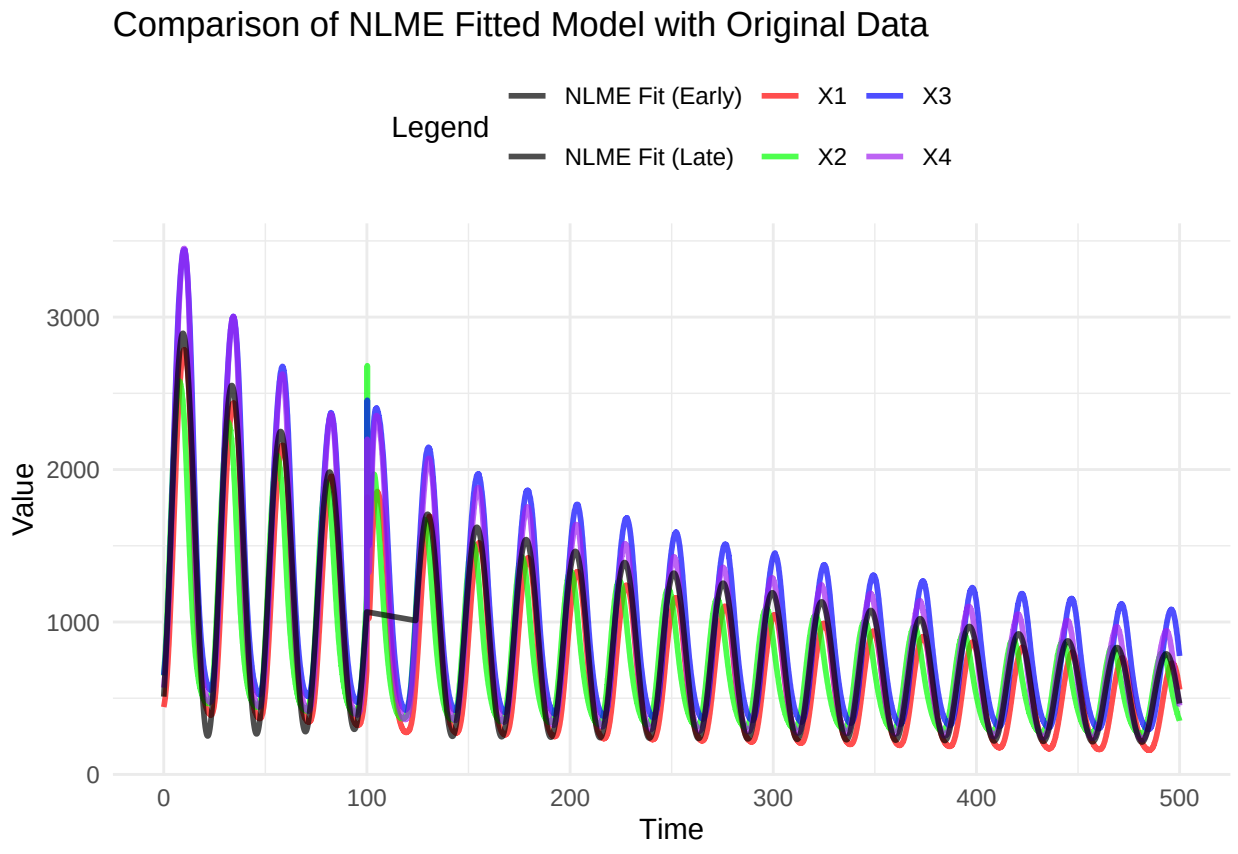
```
mixed_late <- nlme(
  y ~ asym_model_function(Time, A_max, eta_max, A_min, eta_min, tau, phi),
  random = A_max +  A_min +  tau ~ 1 | repl,  # `phi`    random effect
  fixed = A_max + eta_max + A_min + eta_min + tau + phi ~ 1,
  data = long_late,
  start = c(init_param_late$A_max, init_param_late$eta_max,
            init_param_late$A_min, init_param_late$eta_min,
            init_param_late$tau, init_param_late$phi)
)
```

```r
summary(mixed_late)
```

```
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: y ~ asym_model_function(Time, A_max, eta_max, A_min, eta_min,      tau, phi)
##   Data: long_late
##        AIC      BIC    logLik
##   168206.9 168305.9 -84090.44
##
## Random effects:
##  Formula: list(A_max ~ 1, A_min ~ 1, tau ~ 1)
##  Level: repl
##  Structure: General positive-definite, Log-Cholesky parametrization
##          StdDev       Corr
## A_max    3.071586e+02 A_max  A_min
## A_min    5.997867e+01  0.661
## tau      1.245731e-03 -0.412  0.317
## Residual 6.456093e+01
##
## Fixed effects:  A_max + eta_max + A_min + eta_min + tau + phi ~ 1
##             Value Std.Error    DF  t-value p-value
## A_max   2246.9601 153.67306 15035  14.6217       0
## eta_max    0.0021   0.00001 15035 304.4441       0
## A_min    267.5243  30.16103 15035   8.8699       0
## eta_min    0.0005   0.00004 15035  13.0514       0
## tau        0.2592   0.00062 15035 415.9725       0
## phi        4.0111   0.00409 15035 979.5944       0
##  Correlation:
##         A_max  eta_mx A_min  eta_mn tau
## eta_max  0.027
## A_min    0.656 -0.029
## eta_min -0.009 -0.317  0.098
## tau     -0.411  0.000  0.316  0.000
## phi     -0.001 -0.017  0.002  0.014 -0.022
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -3.15020623 -0.66747773  0.05564464  0.65447343  4.13731228
##
## Number of Observations: 15044
## Number of Groups: 4
```

```r
params_early_nlme <- fixef(mixed_early)
params_late_nlme <- fixef(mixed_late)

# Step 2: fitted curve
y_fit_early_nlme <- asym_model_function(time_early,
                                        params_early_nlme["A_max"], params_early_nlme["eta_max"],
                                        params_early_nlme["A_min"], params_early_nlme["eta_min"],
                                        params_early_nlme["tau"], params_early_nlme["phi"])

y_fit_late_nlme <- asym_model_function(time_late,
                                       params_late_nlme["A_max"], params_late_nlme["eta_max"],
                                       params_late_nlme["A_min"], params_late_nlme["eta_min"],
```

```
                                    params_late_nlme["tau"], params_late_nlme["phi"])

# Step 3:
df_MC_long <- df_MC %>%
  pivot_longer(cols = c("X1", "X2", "X3", "X4"), names_to = "Replicate", values_to = "Value")

# Step 4:
df_nlme_early <- data.frame(Time = time_early, Value = y_fit_early_nlme, Type = "NLME Fit (Early)")
df_nlme_late <- data.frame(Time = time_late, Value = y_fit_late_nlme, Type = "NLME Fit (Late)")


df_plot <- bind_rows(df_MC_long %>% mutate(Type = Replicate), df_nlme_early, df_nlme_late)


ggplot(df_plot, aes(x = Time, y = Value, color = Type, group = interaction(Replicate, Type))) +
  geom_line(size = 1, alpha = 0.7) +
  scale_color_manual(values = c("X1" = "red", "X2" = "green", "X3" = "blue", "X4" = "purple",
                                "NLME Fit (Early)" = "black", "NLME Fit (Late)" = "black")) +
  labs(title = "Comparison of NLME Fitted Model with Original Data",
       x = "Time", y = "Value", color = "Legend") +
  theme_minimal() +
  theme(legend.position = "top")
```




```
params_early_nlme
```

```
##         A_max       eta_max         A_min       eta_min           tau
##  3.042032e+03  5.234737e-03  2.398822e+02 -2.232190e-03  2.611685e-01
##           phi
## -2.509700e+00
```

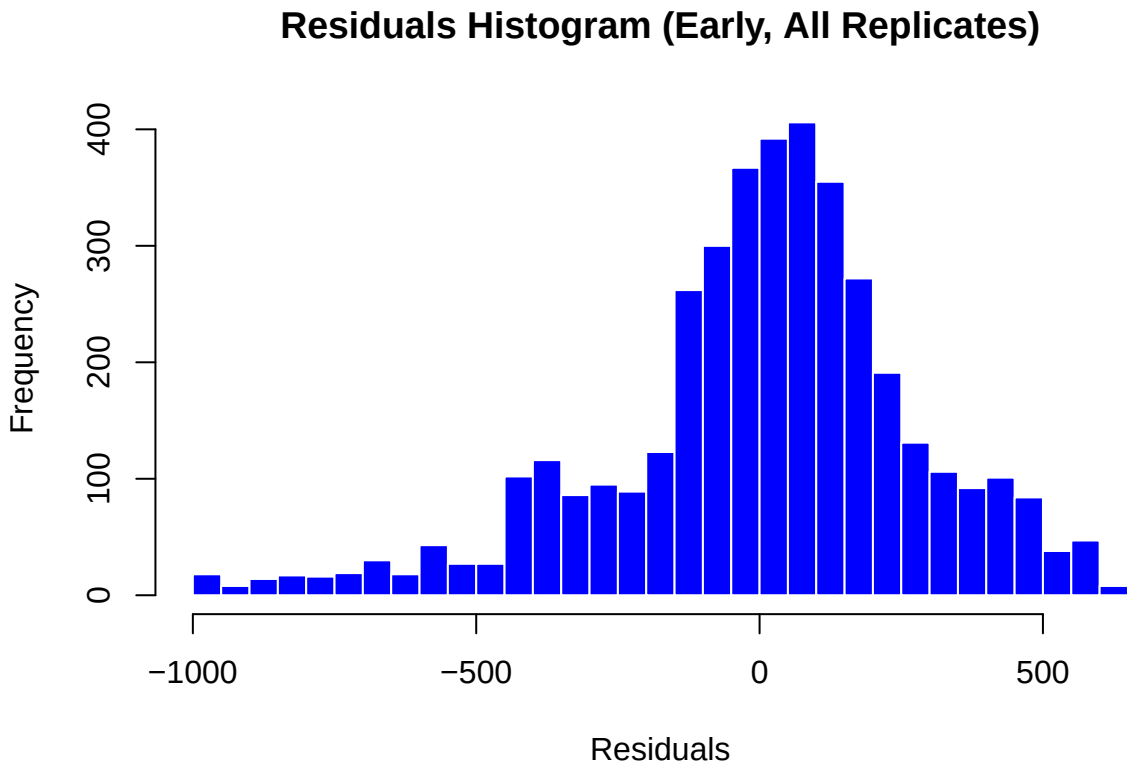
```r
# Compute sigma² for Early & Late
sigma2_early_asym <- estimate_sigma2_asym(df_MC_early, time_early, params_early_nlme)
sigma2_late_asym <- estimate_sigma2_asym(df_MC_late, time_late, params_late_nlme)

cat("Sigma² for early:", sigma2_early_asym, "\n")
```

```
## Sigma² for early: 2547861
```

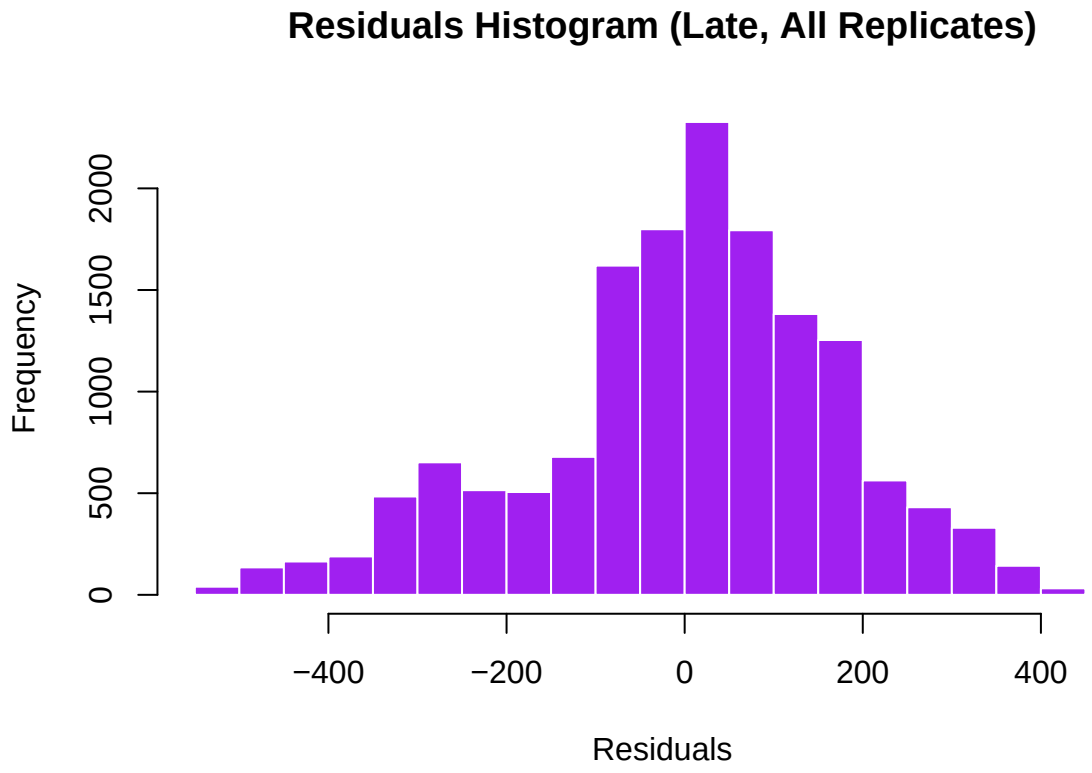
```r
cat("Sigma² for late:", sigma2_late_asym, "\n")
```

```
## Sigma² for late: 479839
```

```r
# Compute residuals
residuals_early_asym <- compute_residuals_asym(df_MC_early[, -1], time_early,params_early_nlme)
residuals_late_asym <- compute_residuals_asym(df_MC_late[, -1], time_late, params_late_nlme)

# Plot residual histograms
 # Arrange in one row
hist(unlist(residuals_early_asym), breaks = 30, col = "blue", border = "white",
     main = "Residuals Histogram (Early, All Replicates)", xlab = "Residuals", ylab = "Frequency")
```



```r
hist(unlist(residuals_late_asym), breaks = 30, col = "purple", border = "white",
     main = "Residuals Histogram (Late, All Replicates)", xlab = "Residuals", ylab = "Frequency")
```




```r
# Compute covariance matrices
covariance_early_asym <- compute_covariance_asym(params_early_nlme, time_early, df_MC_early[, -1], sigma
covariance_late_asym <- compute_covariance_asym(params_late_nlme, time_late, df_MC_late[, -1], sigma2_la

# Function to remove phi from parameters and covariance matrix
remove_phi_asym <- function(params, covariance_matrix) {
  param_names <- names(params)
  phi_index <- which(param_names == "phi")

  # Remove phi
  params_reduced <- params[-phi_index]
  covariance_matrix_reduced <- covariance_matrix[-phi_index, -phi_index]

  return(list(params = params_reduced, covariance = covariance_matrix_reduced))
}

# Remove phi from early & late estimates
early_filtered_asym <- remove_phi_asym(params_early_nlme, covariance_early_asym)
late_filtered_asym <- remove_phi_asym(params_late_nlme, covariance_late_asym)

# Perform Wald Test
wald_result_asym <- wald_test(early_filtered_asym$params, late_filtered_asym$params,
```

```
                                early_filtered_asym$covariance, late_filtered_asym$covariance)


# Output Wald Test results
cat("Wald Test Statistic:", wald_result_asym$T_Wald, "\n")
```

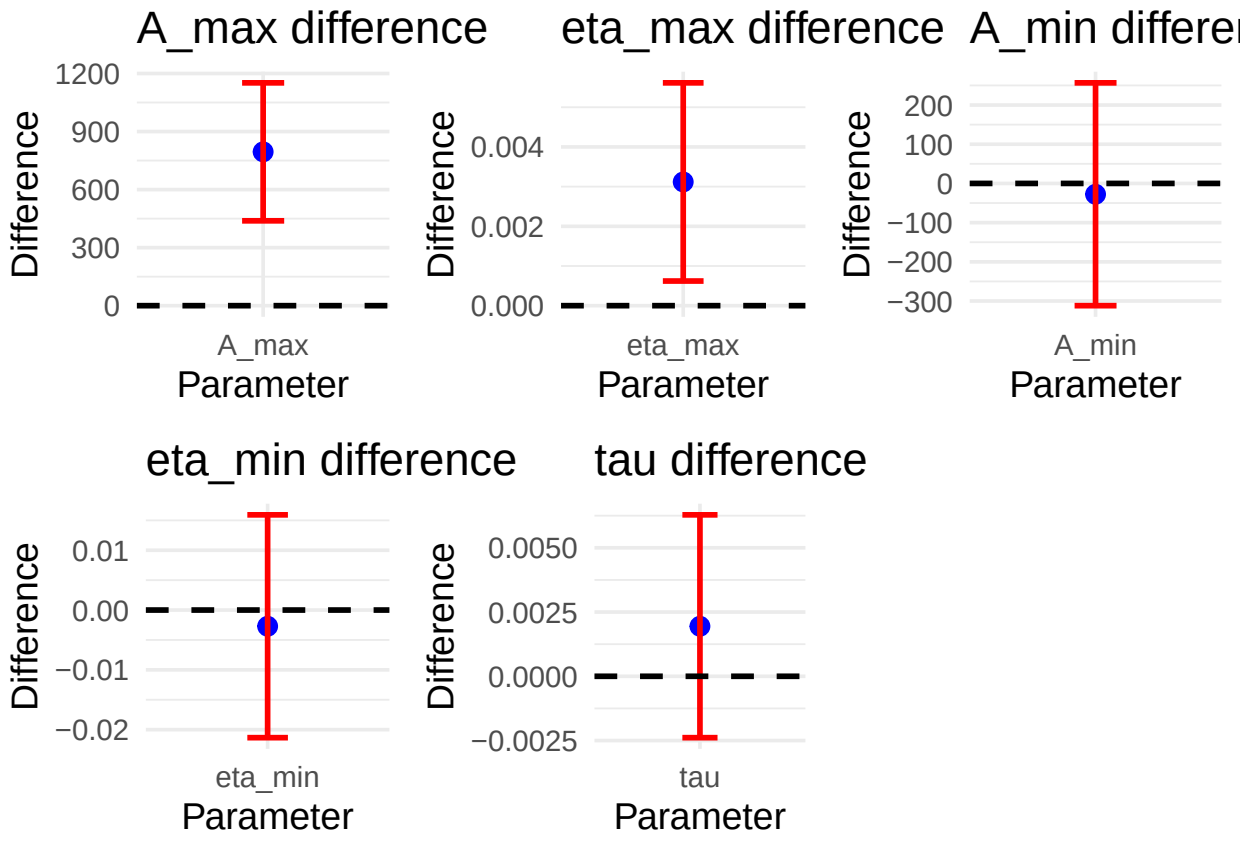
```
## Wald Test Statistic: 60.48885
```

```
cat("p-value:", wald_result_asym$p_value, "\n")
```

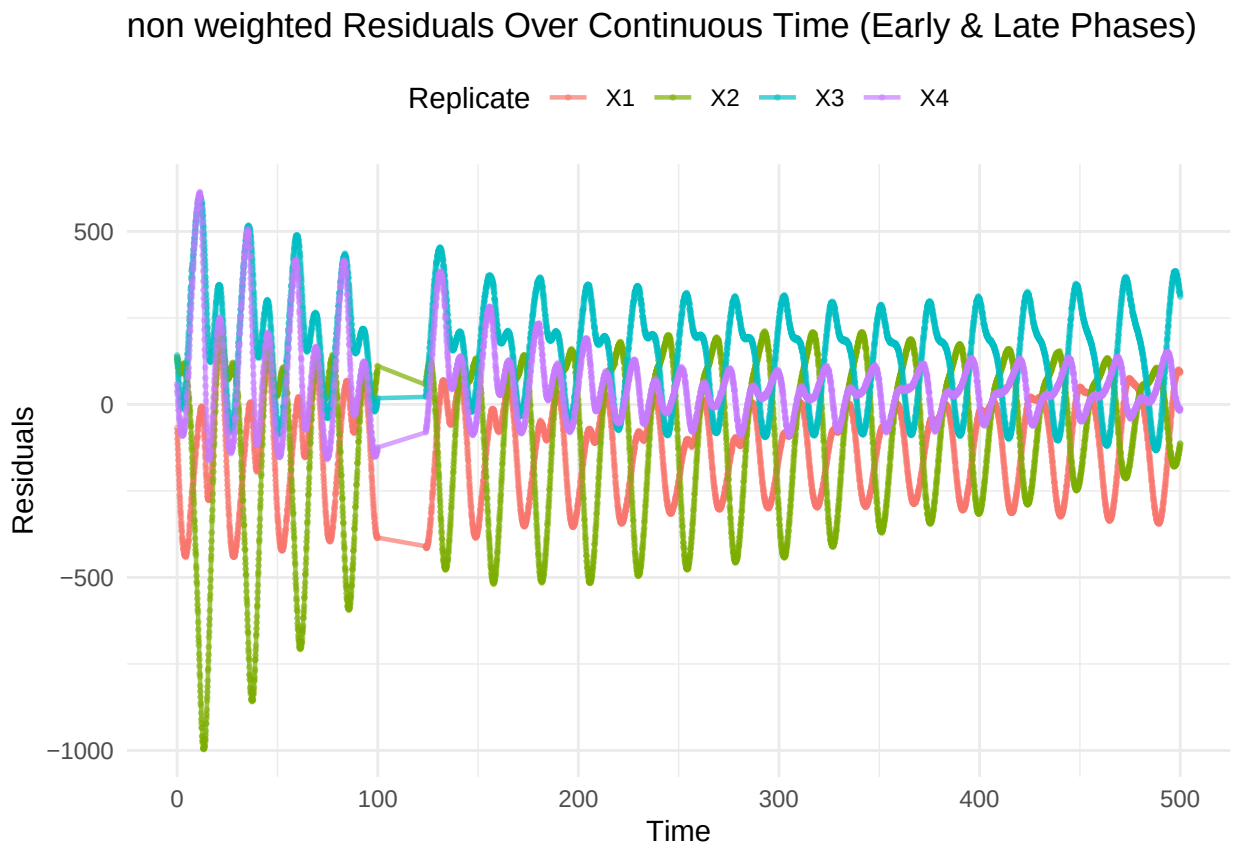
```
## p-value: 9.631629e-12
```

plot difference between time <100 and time>100 for each parameters.

```
param_diff <- early_filtered_asym$params - late_filtered_asym$params

covariance_early_fixed <- early_filtered_asym$covariance
covariance_late_fixed <- late_filtered_asym$covariance

# covariance from hessian
var_diff <- diag(covariance_early_fixed + covariance_early_fixed)
se_diff <- sqrt(var_diff)

# 99% CI
Z_99 <- 2.576  #
CI_lower <- param_diff - Z_99 * se_diff
CI_upper <- param_diff + Z_99 * se_diff

#
df_diff <- data.frame(
  Parameter = names(param_diff),
  Difference = param_diff,
  CI_Lower = CI_lower,
  CI_Upper = CI_upper
)

#
plots <- lapply(1:nrow(df_diff), function(i) {
  ggplot(df_diff[i, ], aes(x = Parameter, y = Difference)) +
    geom_point(size = 3, color = "blue") +  #
    geom_errorbar(aes(ymin = CI_Lower, ymax = CI_Upper), width = 0.2, size = 1, color = "red") +  # CI
    geom_hline(yintercept = 0, linetype = "dashed", color = "black", size = 1) +  # 0
    labs(title = paste0(df_diff$Parameter[i], " difference"),
         x = "Parameter", y = "Difference") +
    theme_minimal(base_size = 14)
})

#
grid.arrange(grobs = plots, ncol = 3)  # 3
```




```
residuals_early_asym$Time <- time_early
residuals_late_asym$Time <- time_late


# Convert residuals data to long format
residuals_early_long <- residuals_early_asym %>%
  pivot_longer(cols = -Time, names_to = "Replicate", values_to = "Residual")

residuals_late_long <- residuals_late_asym %>%
  pivot_longer(cols = -Time, names_to = "Replicate", values_to = "Residual")

# Combine early and late residuals (ensuring time remains continuous)
residuals_combined <- bind_rows(residuals_early_long, residuals_late_long) %>%
  arrange(Replicate, Time)  # Ensure time is ordered within each replicate

# Plot residuals over continuous time with lines connecting points
ggplot(residuals_combined, aes(x = Time, y = Residual, color = Replicate, group = Replicate)) +
  geom_line(alpha = 0.7, size = 0.8) +  # Connect points with lines
  geom_point(size = 0.5, alpha = 0.5) +  # Show individual points
  labs(title = "non weighted Residuals Over Continuous Time (Early & Late Phases)",
       x = "Time", y = "Residuals", color = "Replicate") +
  theme_minimal() +
  theme(legend.position = "top")
```




```
# weighted_fit_fft_model <- function(y_fft, t, N, init_params, weights) {
#   fit <- try(nlsLM(
#     y_fft ~ fft_extract(asym_model_function(t, A_max, eta_max, A_min, eta_min, tau, phi), N),
#     start = init_params,
#     weights = weights,  # Apply weights
#     data = data.frame(y_fft = y_fft)
#
#   ), silent = TRUE)
#
#   if (inherits(fit, "try-error")) {
#     return(NULL)
#   }
#   estimates <- coef(fit)
#   conf_intervals <- confint2(fit, level = 0.99, method = 'asymptotic')
#   return(list(estimates = estimates, conf_intervals = conf_intervals))
# }
#
# weighted_fft_fit_early <- weighted_fit_fft_model(y_mean_early, time_early, N_early,
#                                                  init_param_asymm_early, weights_early)
```

nlme with time dependent assumption(bayesian weights)

```
bayesian_weights_early_expanded <- rep(bayesian_weights_early, each = 4)
bayesian_weights_late_expanded <- rep(bayesian_weights_late, each = 4)
```

```
long_early$weights <- bayesian_weights_early_expanded


long_late$weights <- bayesian_weights_late_expanded

weighted_mixed_early <- nlme(y ~ asym_model_function(Time, A_max, eta_max, A_min, eta_min, tau, phi),
                random = A_max +  A_min +  tau ~ 1 | repl,  # Allow these parameters to vary across rep
                fixed = A_max + eta_max + A_min + eta_min + tau + phi ~ 1,

# Fixed effects shared across replicates
                data = long_early,  # Use the transformed long-format data
                weights = varExp(form = ~ weights),

                start = c(init_param_early$A_max, init_param_early$eta_max, init_param_early$A_min, ini

)
weighted_mixed_late <- nlme(y ~ asym_model_function(Time, A_max, eta_max, A_min, eta_min, tau, phi),
                random = A_max  ~ 1 | repl,  # Allow these parameters to vary across replicates
                fixed = A_max + eta_max + A_min + eta_min + tau + phi ~ 1,

# Fixed effects shared across replicates
                data = long_late,  # Use the transformed long-format data
                weights = varExp(form = ~ weights),

                start = c(init_param_late$A_max, init_param_late$eta_max, init_param_late$A_min, init_pa

)
summary(weighted_mixed_early)
```

```
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: y ~ asym_model_function(Time, A_max, eta_max, A_min, eta_min,      tau, phi)
##   Data: long_early
##        AIC      BIC    logLik
##   50971.05 51059.17 -25471.52
##
## Random effects:
##  Formula: list(A_max ~ 1, A_min ~ 1, tau ~ 1)
##  Level: repl
##  Structure: General positive-definite, Log-Cholesky parametrization
##          StdDev       Corr
## A_max    365.01097619 A_max  A_min
## A_min     63.21295405  0.734
## tau        0.00188235 -0.585 -0.051
## Residual 255.80137396
##
## Variance function:
##  Structure: Exponential of variance covariate
##  Formula: ~weights
##  Parameter estimates:
##     expon
## -10558.86
## Fixed effects:  A_max + eta_max + A_min + eta_min + tau + phi ~ 1
##             Value Std.Error   DF   t-value p-value
```

```
## A_max   2966.4328 183.14397 3991   16.1973       0
## eta_max    0.0055   0.00009 3991   59.6032       0
## A_min    455.2179  31.76953 3991   14.3288       0
## eta_min    0.0018   0.00009 3991   19.2546       0
## tau        0.2607   0.00095 3991  274.8958       0
## phi       -2.5633   0.00638 3991 -401.5913       0
##  Correlation:
##         A_max  eta_mx A_min  eta_mn tau
## eta_max  0.061
## A_min    0.728 -0.009
## eta_min -0.010 -0.136  0.085
## tau     -0.578  0.013 -0.049  0.014
## phi     -0.012 -0.129 -0.018 -0.152 -0.098
##
## Standardized Within-Group Residuals:
##        Min         Q1        Med         Q3        Max
## -3.1674225 -0.9881122 -0.2836992  0.7193555  2.0343769
##
## Number of Observations: 4000
## Number of Groups: 4
```

```
summary(weighted_mixed_late)
```

```
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: y ~ asym_model_function(Time, A_max, eta_max, A_min, eta_min,      tau, phi)
##   Data: long_late
##        AIC      BIC    logLik
##   185995.5 186064.1 -92988.74
##
## Random effects:
##  Formula: A_max ~ 1 | repl
##            A_max Residual
## StdDev: 349.2324 158.8953
##
## Variance function:
##  Structure: Exponential of variance covariate
##  Formula: ~weights
##  Parameter estimates:
##     expon
## -4698.539
## Fixed effects:  A_max + eta_max + A_min + eta_min + tau + phi ~ 1
##             Value Std.Error    DF  t-value p-value
## A_max   2203.0142 174.91510 15035   12.595       0
## eta_max    0.0022   0.00002 15035  141.333       0
## A_min    363.3851   3.75644 15035   96.737       0
## eta_min    0.0008   0.00003 15035   24.455       0
## tau        0.2590   0.00003 15035 8183.007       0
## phi        3.9989   0.00873 15035  457.952       0
##  Correlation:
##         A_max  eta_mx A_min  eta_mn tau
## eta_max  0.051
## A_min   -0.011 -0.182
## eta_min -0.011 -0.207  0.935
## tau      0.001  0.021  0.012  0.006
```

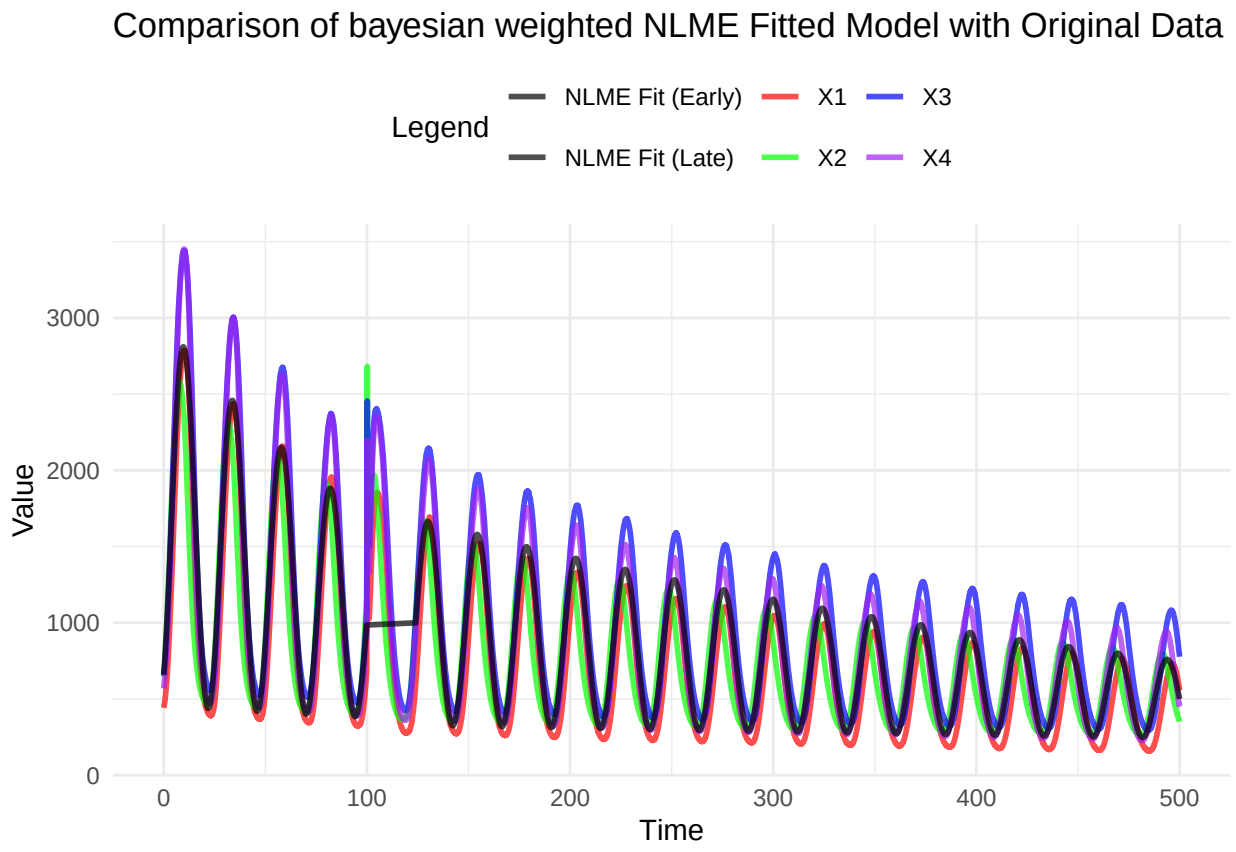
```
## phi     -0.002 -0.025 -0.013 -0.009 -0.931
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -2.36795867 -0.83188687 -0.06268363  0.62456376  2.83510766
##
## Number of Observations: 15044
## Number of Groups: 4
```

```
weighted_mixed_late
```

```
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: y ~ asym_model_function(Time, A_max, eta_max, A_min, eta_min,      tau, phi)
##   Data: long_late
##   Log-likelihood: -92988.74
##   Fixed: A_max + eta_max + A_min + eta_min + tau + phi ~ 1
##        A_max      eta_max        A_min      eta_min          tau          phi
## 2.203014e+03 2.156172e-03 3.633851e+02 7.982067e-04 2.590102e-01 3.998892e+00
##
## Random effects:
##  Formula: A_max ~ 1 | repl
##            A_max Residual
## StdDev: 349.2324 158.8953
##
## Variance function:
##  Structure: Exponential of variance covariate
##  Formula: ~weights
##  Parameter estimates:
##     expon
## -4698.539
## Number of Observations: 15044
## Number of Groups: 4
```

```
#get estimated parameters
weighted_params_early_nlme <- fixef(weighted_mixed_early)
weighted_params_late_nlme <- fixef(weighted_mixed_late)

# plot fitted curve vs oringinal curve
weighted_y_fit_early_nlme <- asym_model_function(time_early,
                                        weighted_params_early_nlme["A_max"], weighted_params_early_nlme
                                        weighted_params_early_nlme["A_min"], weighted_params_early_nlme
                                        weighted_params_early_nlme["tau"], weighted_params_early_nlme["

weighted_y_fit_late_nlme <- asym_model_function(time_late,
                                       weighted_params_late_nlme["A_max"], weighted_params_late_nlme["e
                                       weighted_params_late_nlme["A_min"], weighted_params_late_nlme["e
                                       weighted_params_late_nlme["tau"], weighted_params_late_nlme["phi



df_weighted_nlme_early <- data.frame(Time = time_early, Value = weighted_y_fit_early_nlme, Type = "NLME
df_weighted_nlme_late <- data.frame(Time = time_late, Value = weighted_y_fit_late_nlme, Type = "NLME Fi
```

```r
df_plot <- bind_rows(df_MC_long %>% mutate(Type = Replicate), df_weighted_nlme_early, df_weighted_nlme_l


ggplot(df_plot, aes(x = Time, y = Value, color = Type, group = interaction(Replicate, Type))) +
  geom_line(size = 1, alpha = 0.7) +
  scale_color_manual(values = c("X1" = "red", "X2" = "green", "X3" = "blue", "X4" = "purple",
                                "NLME Fit (Early)" = "black", "NLME Fit (Late)" = "black")) +
  labs(title = "Comparison of bayesian weighted NLME Fitted Model with Original Data",
       x = "Time", y = "Value", color = "Legend") +
  theme_minimal() +
  theme(legend.position = "top")
```



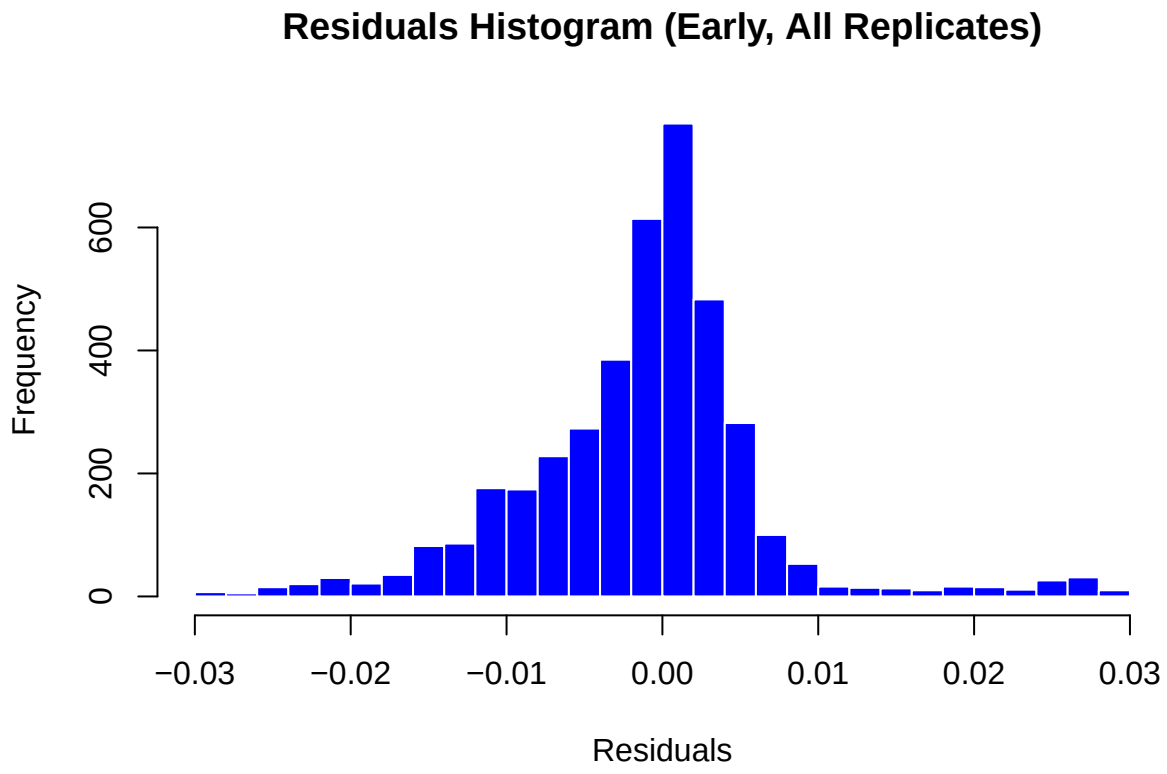

```r
# Compute sigma² for Early & Late
weighted_sigma2_early_asym <- estimate_sigma2_asym(df_MC_early, time_early, weighted_params_early_nlme)
weighted_sigma2_late_asym <- estimate_sigma2_asym(df_MC_late, time_late, weighted_params_late_nlme)

cat("Sigma² for early:", sigma2_early_asym, "\n")
```

```
## Sigma² for early: 2547861
```

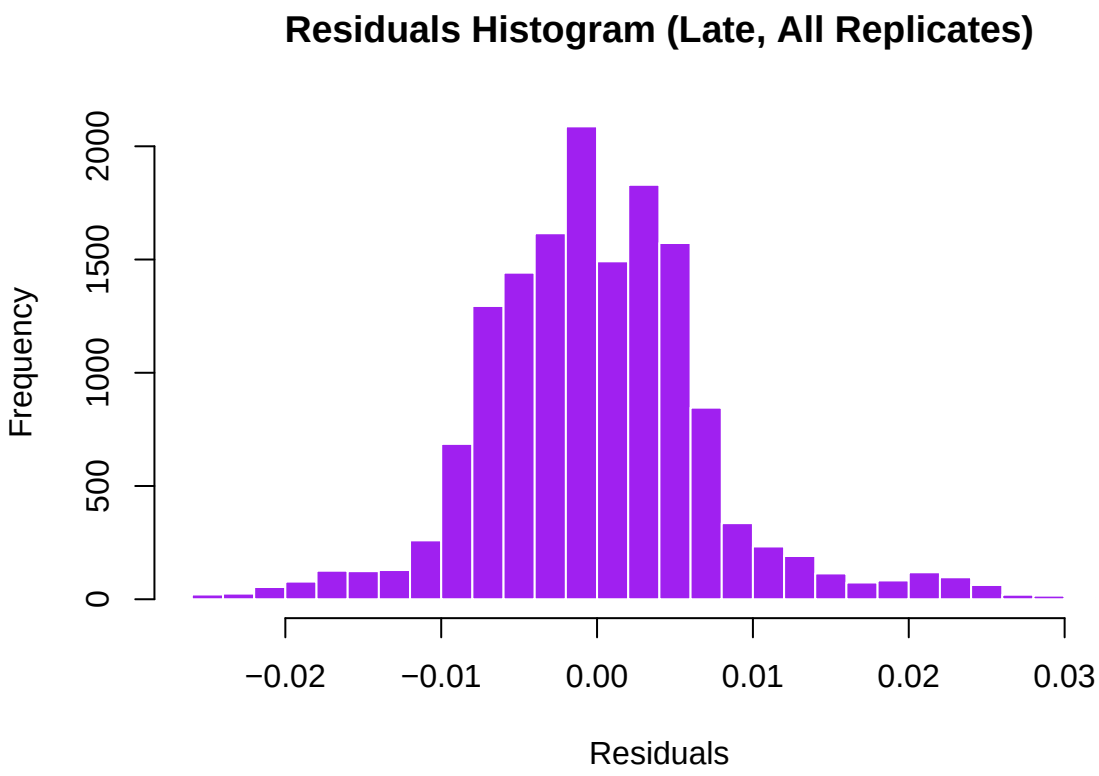
```r
cat("Sigma² for late:", sigma2_late_asym, "\n")
```

```
## Sigma² for late: 479839
```

```r
# Compute residuals
weighted_residuals_early_asym <- compute_residuals_asym(df_MC_early[, -1], time_early,weighted_params_ea
weighted_residuals_late_asym <- compute_residuals_asym(df_MC_late[, -1], time_late, weighted_params_late

# Plot residual histograms
 # Arrange in one row
hist(unlist(weighted_residuals_early_asym), breaks = 30, col = "blue", border = "white",
     main = "Residuals Histogram (Early, All Replicates)", xlab = "Residuals", ylab = "Frequency")
```




```r
hist(unlist(weighted_residuals_late_asym), breaks = 30, col = "purple", border = "white",
     main = "Residuals Histogram (Late, All Replicates)", xlab = "Residuals", ylab = "Frequency")
```



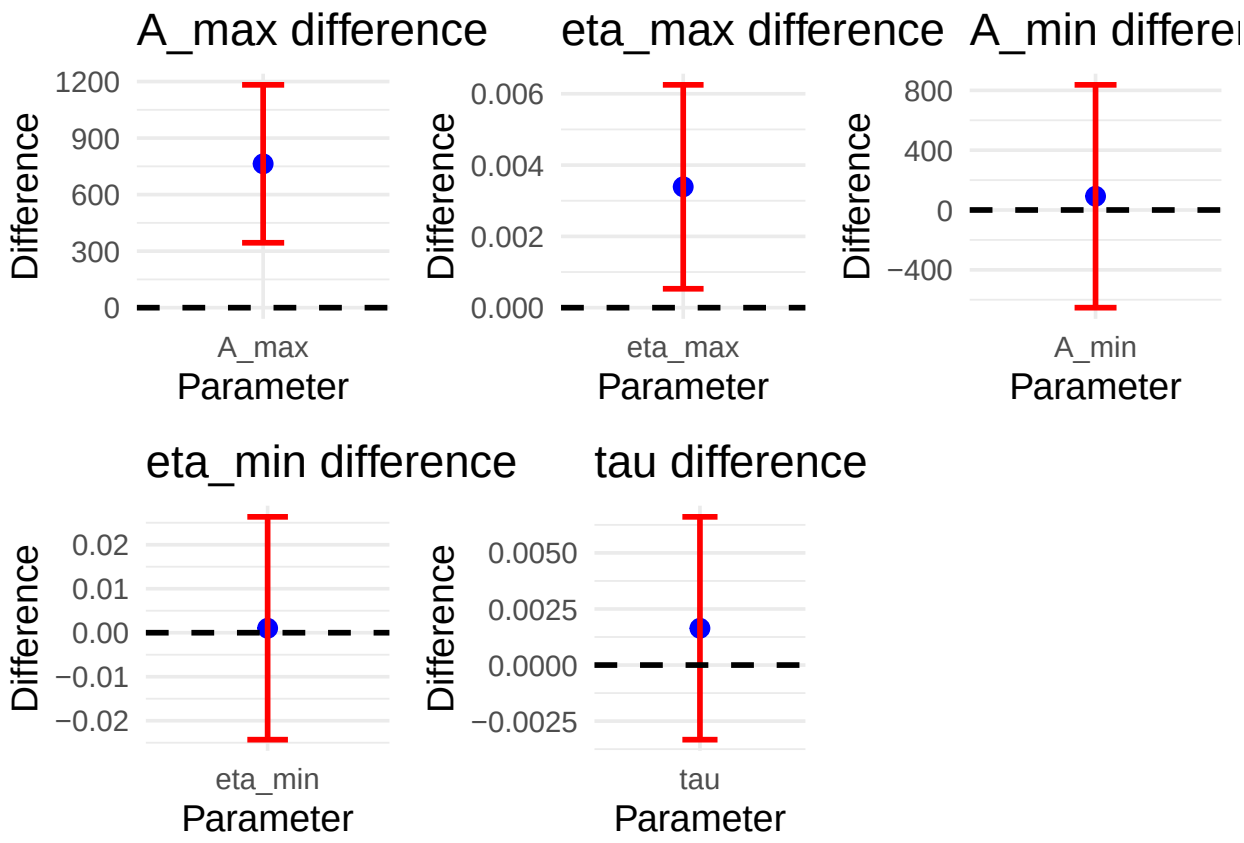

```
# Compute covariance matrices
covariance_early_asym <- compute_covariance_asym(weighted_params_early_nlme, time_early, df_MC_early[, -1],
covariance_late_asym <- compute_covariance_asym(weighted_params_late_nlme, time_late, df_MC_late[, -1],


# Remove phi from early & late estimates
early_filtered_asym <- remove_phi_asym(weighted_params_early_nlme, covariance_early_asym)
late_filtered_asym <- remove_phi_asym(weighted_params_late_nlme, covariance_late_asym)

# Perform Wald Test
weighted_wald_result_asym <- wald_test(early_filtered_asym$params, late_filtered_asym$params,
                            early_filtered_asym$covariance, late_filtered_asym$covariance)


# Output Wald Test results
cat("Wald Test Statistic:", weighted_wald_result_asym$T_Wald, "\n")
```

```
## Wald Test Statistic: 63.27964
```

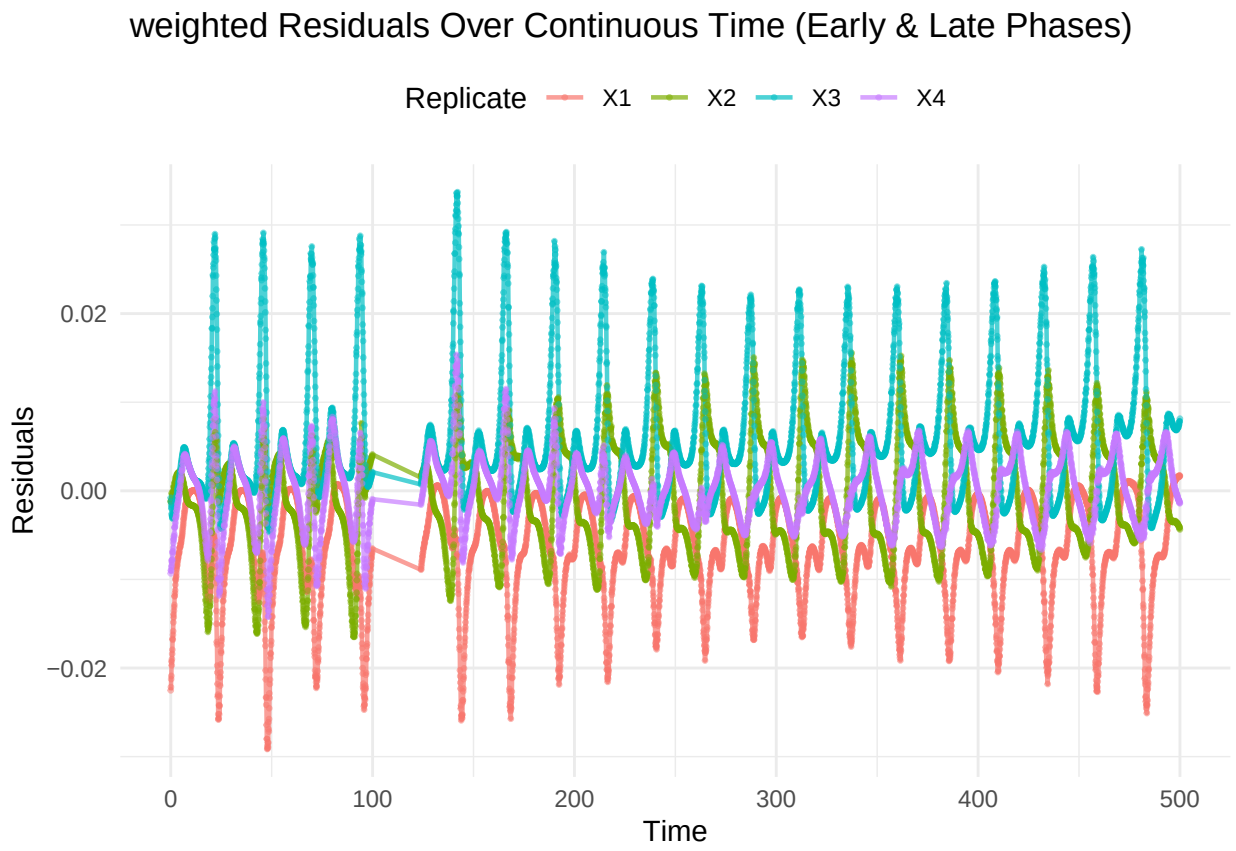
```
cat("p-value:", weighted_wald_result_asym$p_value, "\n")
```

```
## p-value: 2.547629e-12
```

```r
param_diff <- early_filtered_asym$params - late_filtered_asym$params

covariance_early_fixed <- early_filtered_asym$covariance
covariance_late_fixed <- late_filtered_asym$covariance

# covariance from hessian
var_diff <- diag(covariance_early_fixed + covariance_early_fixed)
se_diff <- sqrt(var_diff)

# 99% CI
Z_99 <- 2.576  #
CI_lower <- param_diff - Z_99 * se_diff
CI_upper <- param_diff + Z_99 * se_diff

#
df_diff <- data.frame(
  Parameter = names(param_diff),
  Difference = param_diff,
  CI_Lower = CI_lower,
  CI_Upper = CI_upper
)

#
plots <- lapply(1:nrow(df_diff), function(i) {
  ggplot(df_diff[i, ], aes(x = Parameter, y = Difference)) +
    geom_point(size = 3, color = "blue") +  #
    geom_errorbar(aes(ymin = CI_Lower, ymax = CI_Upper), width = 0.2, size = 1, color = "red") +  # CI
    geom_hline(yintercept = 0, linetype = "dashed", color = "black", size = 1) +
    labs(title = paste0(df_diff$Parameter[i], " difference"),
         x = "Parameter", y = "Difference") +
    theme_minimal(base_size = 14)
})

#
grid.arrange(grobs = plots, ncol = 3)  # 3
```




```r
weighted_residuals_early_asym$Time <- time_early
weighted_residuals_late_asym$Time <- time_late


# Convert residuals data to long format
residuals_early_long <- weighted_residuals_early_asym %>%
  pivot_longer(cols = -Time, names_to = "Replicate", values_to = "weighted Residual")

residuals_late_long <- weighted_residuals_late_asym %>%
  pivot_longer(cols = -Time, names_to = "Replicate", values_to = "weighted Residual")

# Combine early and late residuals (ensuring time remains continuous)
residuals_combined <- bind_rows(residuals_early_long, residuals_late_long) %>%
  arrange(Replicate, Time)  # Ensure time is ordered within each replicate

# Plot residuals over continuous time with lines connecting points
ggplot(residuals_combined, aes(x = Time, y = `weighted Residual`, color = Replicate, group = Replicate)
  geom_line(alpha = 0.7, size = 0.8) +  # Connect points with lines
  geom_point(size = 0.5, alpha = 0.5) +  # Show individual points
  labs(title = " weighted Residuals Over Continuous Time (Early & Late Phases)",
       x = "Time", y = "Residuals", color = "Replicate") +
  theme_minimal() +
  theme(legend.position = "top")
```



nlme on frequance domain

generate Error Error in chol.default((value + t(value))/2) : the leading minor of order 1 is not positive definite

```
# N_early =20
# #long chart fft
# df_early_fft <- data.frame(
#   replicate = rep(c("X1","X2","X3","X4"), times = (2*N_early+1)),
#   freqIndex = rep(seq_len(2*N_early+1), each=4),
#   FFT_Obs   = y_fft_stacked_early
# )
#
#
# df_late_fft <- data.frame(
#   replicate = rep(c("X1","X2","X3","X4"), times = (2*N_late+1)),
#   freqIndex = rep(seq_len(2*N_late+1), each=4),
#   FFT_Obs   = y_fft_stacked_late
# )
#
# fft_extract <- function(y, N) {
#   y_fft <- fft(y)
#   features <- c(Re(y_fft)[1:(N + 1)], Im(y_fft)[2:(N + 1)])
#   return(features)
# }
```

```
# N_early <- 20
# N_late <- 50
#
# fft_predicted_early <- fft_extract(asym_model_function(time_early, init_param_early$A_max, init_param
#                                                        init_param_early$A_min, init_param_early$eta_m
#                                                        init_param_early$tau, init_param_early$phi), N
#
# df_early_fft <- data.frame(
#   replicate = rep(c("X1","X2","X3","X4"), times = (2*N_early+1)),
#
#   FFT_Obs   = y_fft_stacked_early,
#   fft_predicted = rep(fft_predicted_early, each = 4)
# )
# fft_replicates_early <- apply(y_early, 2, function(y) fft_extract(y, N_early))
# #
# observed_variances_early <- calculate_variance(fft_replicates_early)
# weights_early <- 1 / sqrt(observed_variances_early)
#
# library(nlme)
# mixed_early_fft <- nlme(
#   FFT_Obs ~ fft_predicted,  #
#   random = A_max ~ 1 | replicate,  #
#   fixed = A_max + eta_max + A_min + eta_min + tau + phi ~ 1,  #
#   data = df_early_fft,
#   weights = varExp(form =~ weights_early_stacked),
#   start = c(
#     A_max   = init_param_early$A_max,
#     eta_max = init_param_early$eta_max,
#     A_min   = init_param_early$A_min,
#     eta_min = init_param_early$eta_min,
#     tau   = init_param_early$tau,
#     phi     = init_param_early$phi
#   ),
#   control = nlmeControl(
#     maxIter   = 500,
#     msMaxIter = 500,
#     opt       = "nlm",
#     pnlsTol   = 1e-8
#   )
# )
#
# summary(mixed_early_fft)
```

```
# fft_predicted_late <- fft_extract(asym_model_function(time_late, init_param_late$A_max, init_param_la
#                                                       init_param_late$A_min, init_param_late$eta_min
#                                                       init_param_late$tau, init_param_late$phi), N_l
#
# df_late_fft <- data.frame(
#   replicate = rep(c("X1","X2","X3","X4"), times = (2*N_late+1)),
#
#   FFT_Obs   = y_fft_stacked_late,
#   fft_predicted = rep(fft_predicted_late, each = 4)
# )
```

```
#
# library(nlme)
# mixed_late_fft <- nlme(
#   FFT_Obs ~ fft_predicted,  #
#   random = A_max + A_min + tau  ~ 1 | replicate,  #
#   fixed = A_max + eta_max + A_min + eta_min + tau + phi ~ 1,  #
#   data = df_late_fft,
#   start = c(
#     A_max   = init_param_late$A_max,
#     eta_max = init_param_late$eta_max,
#     A_min   = init_param_late$A_min,
#     eta_min = init_param_late$eta_min,
#     tau   = init_param_late$tau,
#     phi     = init_param_late$phi
#   ),
#   control = nlmeControl(
#     maxIter   = 500,
#     msMaxIter = 500,
#     opt       = "nlm",
#     pnlsTol   = 1e-8
#   )
# )
```