EchoModel

Siqi

2025-03-05

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: # Symmetric Model(removed Y shift)

```
# 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(stats)
library(nlstools)
```

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

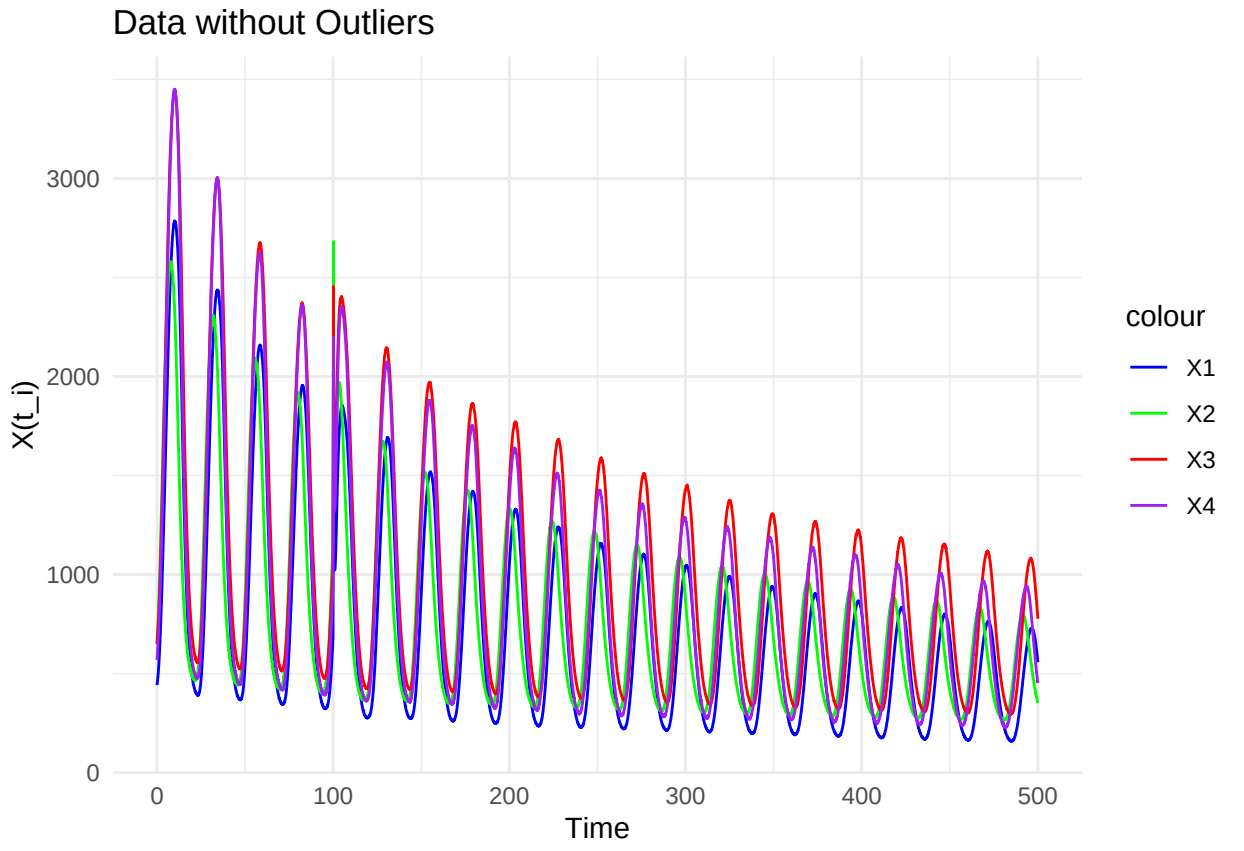
```
## IMPORTANT NOTICE: Most nonlinear regression models and data set examples
```

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

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

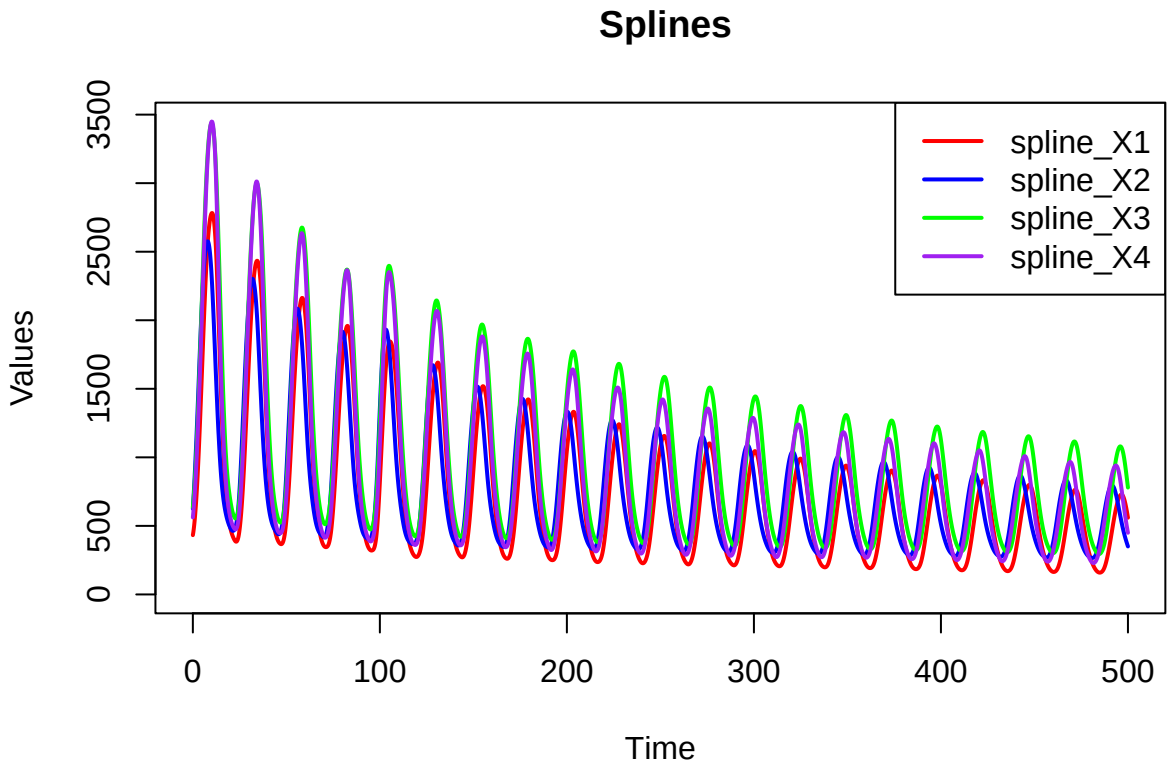
```
## New names:
## * `` -> `...1`
```

```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()
```

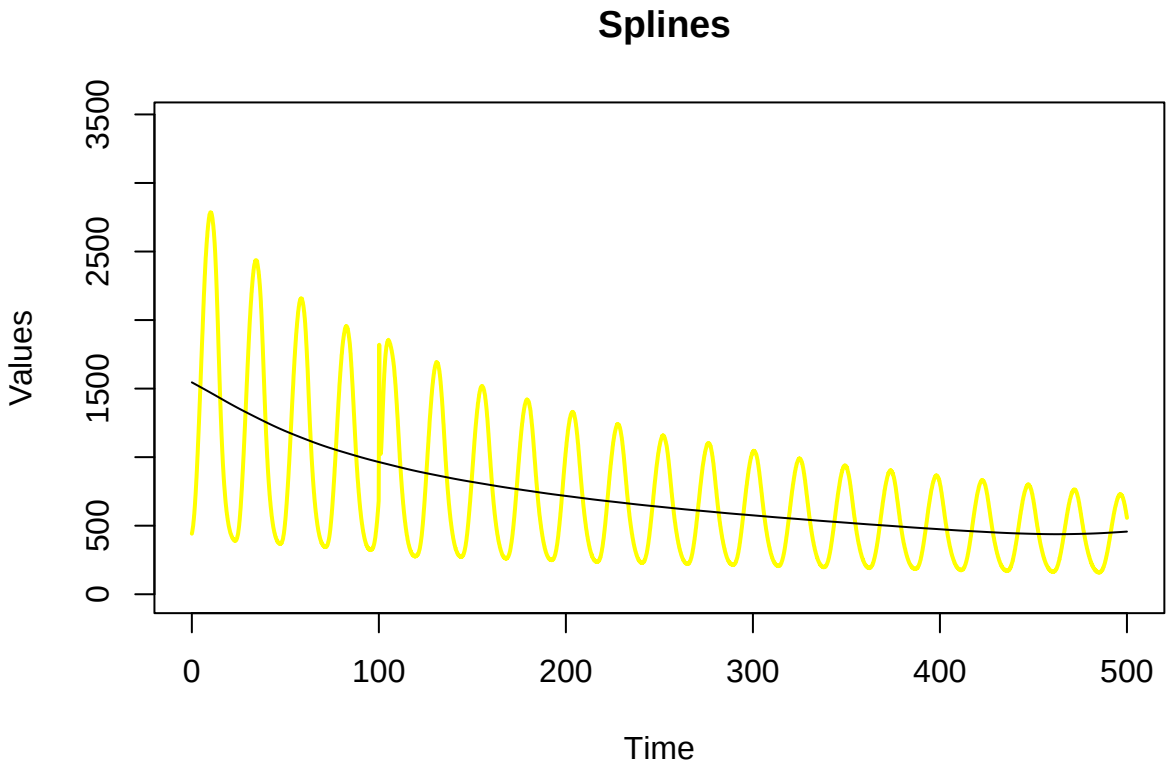



```r
spline_X1 <- predict(smooth.spline(df_MC$Time, df_MC$X1),    spar =)
spline_X2 <- predict(smooth.spline(df_MC$Time, df_MC$X2))
spline_X3 <- predict(smooth.spline(df_MC$Time, df_MC$X3))
spline_X4 <- predict(smooth.spline(df_MC$Time,df_MC$X4))

# Plot the spline-smoothed data
plot(df_MC$Time, spline_X1$y, type = "l", col = "red", lwd = 2,
```

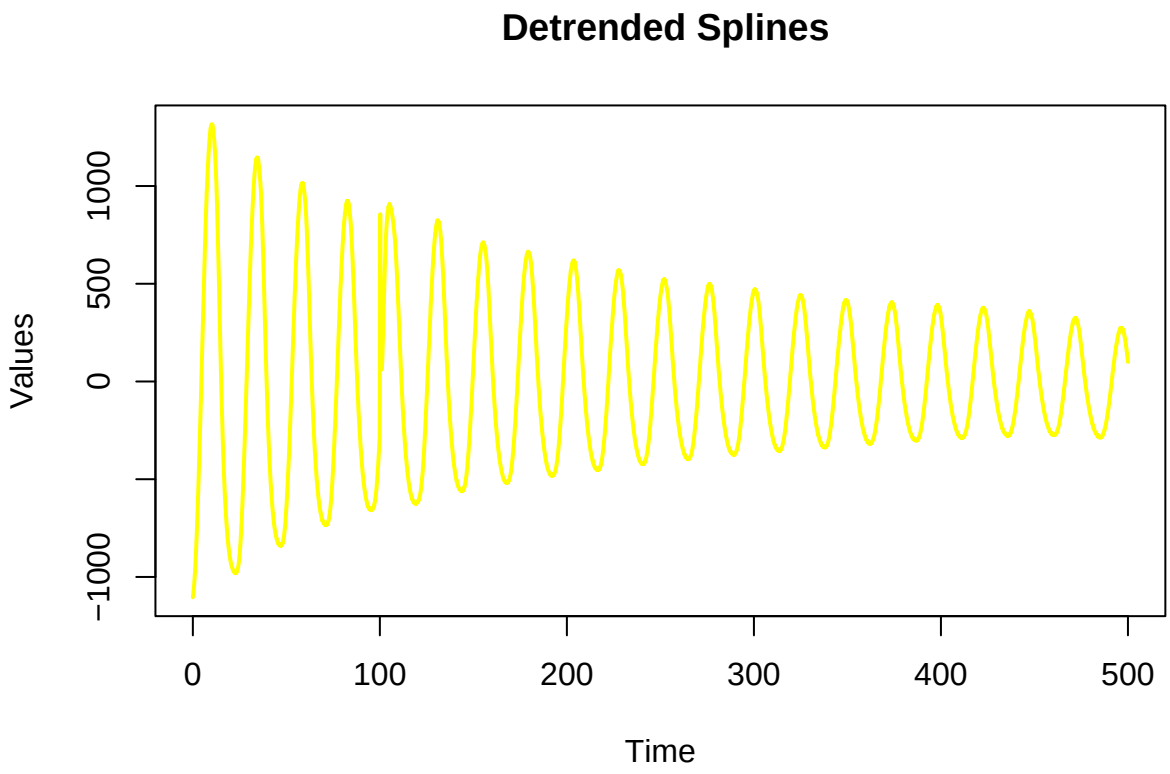
```
     xlab = "Time", ylab = "Values", main = " Splines",
     ylim = range(c(spline_X1$y, spline_X2$y, spline_X3$y, spline_X4)))
lines(df_MC$Time, spline_X2$y, col = "blue", lwd = 2)
lines(df_MC$Time, spline_X3$y, col = "green", lwd = 2)
lines(df_MC$Time, spline_X4$y, col = "purple", lwd = 2)

legend("topright", legend = c("spline_X1", "spline_X2", "spline_X3", "spline_X4"),
       col = c("red", "blue", "green", "purple"), lwd = 2)
```




```
# Plot the spline-smoothed data
spline_X1 <- predict(smooth.spline(df_MC$Time, df_MC$X1, spar=1)) #black line in 2nd #plot average for X

plot(df_MC$Time, df_MC$X1, type = "l", col = "yellow", lwd = 2,
     xlab = "Time", ylab = "Values", main = " Splines",
     ylim = range(c(spline_X1$y, spline_X2$y, spline_X3$y, spline_X4)))
lines(df_MC$Time, spline_X1$y)
```

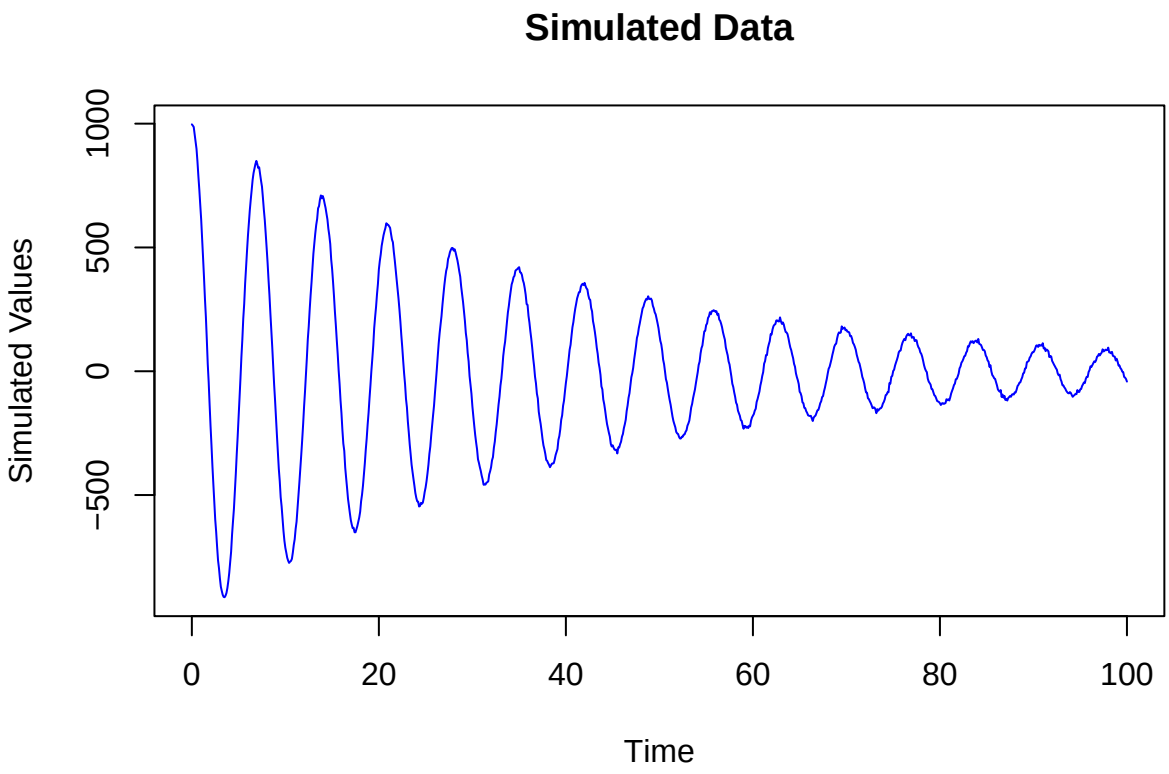



```
plot(df_MC$Time, df_MC$X1-spline_X1$y, type = "l", col = "yellow", lwd = 2,
     xlab = "Time", ylab = "Values", main = " Detrended Splines")
```



Simulate data

```r
t <- seq(0, 100, by = 0.1)

model_function <- function(t, A, gamma, tau, phi) {
  A * exp(-gamma * t ) * cos(tau * t + phi)
}

# using estimated parameters to def simulating function
simulate_data <- function(t, params, sigma) {
  predicted <- model_function(t, params[1], params[2], params[3], params[4])
  noise <- rnorm(length(t), mean = 0, sd = sigma)  #using sigma0 to produce noise
  simulated_data <- predicted + noise
  return(simulated_data)
}
sigma0 = 5
param0=c(1000, 0.025, 0.9, 0)
simulated_data <- simulate_data(t, param0, sigma0 )
plot(t, simulated_data, type = "l", col = "blue", main = "Simulated Data", xlab = "Time", ylab = "Simula
```




```r
library(nlsMicrobio)
library(pracma)  #   findpeaks

find_initial_params <- function(t_data, y_data) {
  # 1 fin dpeaks
  peaks <- findpeaks(y_data, minpeakheight = 100, minpeakdistance = 15)
  if (is.null(peaks) || nrow(peaks) < 2) {
    stop("Not enough peaks detected to compute parameters.")
  }

  t_peaks <- t_data[peaks[, 2]]
  y_peaks <- peaks[, 1]

  # 2 caiculate  A, gamma
  log_y_peaks <- log(y_peaks)
  fit_linear <- lm(log_y_peaks ~ t_peaks)
  log_A <- coef(fit_linear)[1]
  init_gamma <- -coef(fit_linear)[2]
  init_A <- exp(log_A)

  # 3  tau
  delta_values <- diff(t_peaks[1:4])  #
  Delta <- mean(delta_values)         #
  init_tau <- 2 * pi / Delta

  # random_t <- sample(t_data, size = 5)  # choosing 5 random points
```

```r
  # phi_values <- numeric(5)  #
  #
  # for (i in 1:5) {
  #   t0 <- random_t[i]
  #   y_t0 <- y_data[which.min(abs(t_data - t0))]  #
  #
  #   cos_input <- y_t0 / (init_A * exp(-init_gamma * t0))
  #   cos_input <- pmin(pmax(cos_input, -1), 1)  # constrain between -1 and 1
  #   phi_values[i] <- (acos(cos_input) - init_tau * t0 + pi) %% (2 * pi) -  pi #  phi
  # }
  # init_phi <- mean(phi_values)
  t0 <- t_peaks[1]
  y_t0 <- y_peaks[1]
  cos_input <- y_t0 / (init_A * exp(-init_gamma * t0))
  cos_input <- pmin(pmax(cos_input, -1), 1)  #    [-1,1]
  #init_phi <- (acos(cos_input) - init_tau * t0 + 2*pi) %% (4 * pi) - 2 * pi
  init_phi <- (acos(cos_input) - init_tau * t0) %% (2 * pi)  #     2
  init_phi <- ifelse(init_phi > pi, init_phi - 2 * pi, init_phi)  #    -


  return(list(A = init_A, gamma = init_gamma, tau = init_tau, phi = init_phi))
}
simulate_and_estimate <- function(true_params, sigma0 = 10) {
  #simulate data
  t <- seq(0, 100, by = 0.1)

  y_sim <- model_function(t, true_params[1], true_params[2], true_params[3], true_params[4]) +
    rnorm(length(t), mean = 0, sd = sigma0)

  #find initial value
  init_simulated_params = find_initial_params(t, y_sim)
  fit_nls <- try(nlsLM(
    y_sim ~ model_function(t, A, gamma, tau, phi),
    start = init_simulated_params,
    lower = c(500, 0.005, 0.85, -pi),
    upper = c(1500, 0.06, 0.95, pi),
    data = data.frame(t = t, y_sim = y_sim)
  ), silent = TRUE)

  if (class(fit_nls) == "try-error") {
    return(NULL)
  }

  # find estmate parameters and standard errors
  param_estimates <- coef(fit_nls)
  # Calculate standard errors and confidence intervals
  conf_intervals <- confint2(fit_nls, level = 0.99, method = 'asymptotic')
  return(list(estimates = param_estimates, conf_intervals = conf_intervals))
}
# test function
true_params <- c(A = 1000, gamma = 0.025, tau = 0.9, phi = 0)
result <- simulate_and_estimate(true_params)
result
```

```
## $estimates
##   A.(Intercept)   gamma.t_peaks             tau phi.(Intercept)
##    9.998091e+02    2.501079e-02    9.000943e-01   -7.503928e-04
##
## $conf_intervals
##                         0.5 %       99.5 %
## A.(Intercept)   996.01720376 1.003601e+03
## gamma.t_peaks     0.02486747 2.515411e-02
## tau               0.89995064 9.002380e-01
## phi.(Intercept)  -0.00457290 3.072115e-03
```

```
# simulate 100
set.seed(37)
n_simulations <- 100
results <- vector("list", n_simulations)
#
for (i in 1:n_simulations) {
  results[[i]] <- simulate_and_estimate(true_params)
}
results <- Filter(Negate(is.null), results)
null_count <- sum(sapply(results, is.null))
cat("Number of unsuccessful fits (NULL results):", null_count, "\n")
```

```
## Number of unsuccessful fits (NULL results): 0
```

```
# Initialize coverage counters, one counter for each parameter
coverage_counts <- rep(0, length(true_params))

# Iterate through each simulation result to check coverage.
for (i in 1:length(results)) {
  ci <- results[[i]]$conf_intervals
  for (j in 1:length(true_params)) {
    # Check if it falls within the confidence interval.
    if (true_params[j] >= ci[j, 1] && true_params[j] <= ci[j, 2]) {
      coverage_counts[j] <- coverage_counts[j] + 1
    }
  }
}
#calculate coverage
coverage_proportions <- coverage_counts / n_simulations
# print unsuccessful fits

cat("Coverage for A:", coverage_proportions[1], "\n")
```

```
## Coverage for A: 0.96
```

```
cat("Coverage for gamma:", coverage_proportions[2], "\n")
```

```
## Coverage for gamma: 0.98
```

```r
cat("Coverage for tau:", coverage_proportions[3], "\n")
```

```
## Coverage for tau: 0.99
```

```r
cat("Coverage for phi:", coverage_proportions[4], "\n")
```

```
## Coverage for phi: 0.98
```

boxplot of estimated parames for 100 simulations

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x purrr::cross()  masks pracma::cross()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

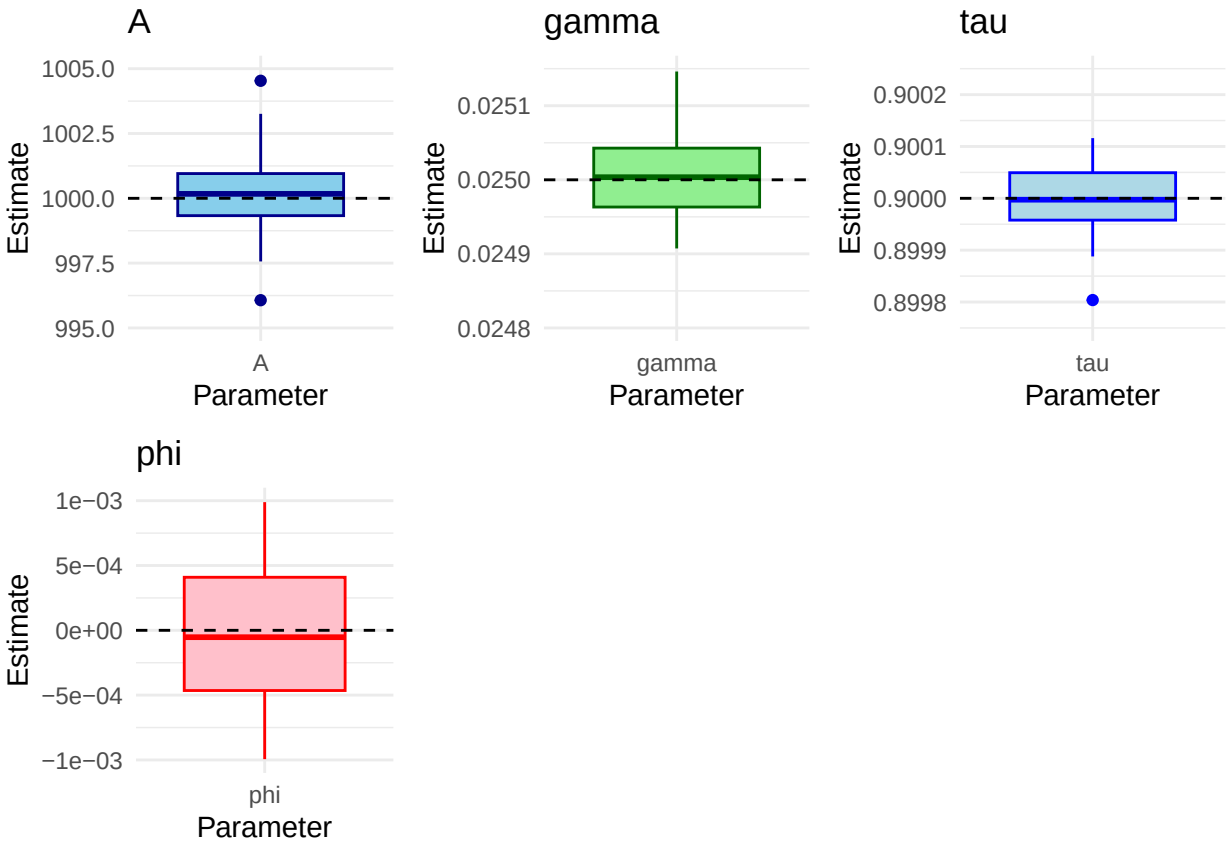
```r
library(dplyr)
library(tidyr)

# Extract valid results (remove NULL values);
valid_results <- Filter(Negate(is.null), results)

# Extract parameter estimates into a data frame
estimate_data <- do.call(rbind, lapply(valid_results, function(res) {
  as.data.frame(t(res$estimates))
}))


colnames(estimate_data) <- c("A", "gamma", "tau", "phi")

estimate_long <- estimate_data %>%
  pivot_longer(cols = everything(), names_to = "Parameter", values_to = "Estimate")

#install.packages("gridExtra")
library(gridExtra) # arange multiple plots
```

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

```r
# Create separate boxplots for each parameter with suitable y-axis ranges;
plot_A <- ggplot(estimate_long %>% filter(Parameter == "A"),
                 aes(x = Parameter, y = Estimate)) +
  geom_boxplot(fill = "skyblue", color = "darkblue") +
  geom_hline(yintercept = true_params["A"], linetype = "dashed") +
  labs(title = "A", y = "Estimate") +
  theme_minimal() +
  ylim(995, 1005)

plot_gamma <- ggplot(estimate_long %>% filter(Parameter == "gamma"),
                     aes(x = Parameter, y = Estimate)) +
  geom_boxplot(fill = "lightgreen", color = "darkgreen") +
  geom_hline(yintercept = true_params["gamma"], linetype = "dashed") +
  labs(title = "gamma", y = "Estimate") +
  theme_minimal() +
  ylim(0.0248, 0.02515)

plot_tau <- ggplot(estimate_long %>% filter(Parameter == "tau"),
                     aes(x = Parameter, y = Estimate)) +
  geom_boxplot(fill = "lightblue", color = "blue") +
  geom_hline(yintercept = true_params["tau"], linetype = "dashed") +
  labs(title = "tau", y = "Estimate") +
  theme_minimal() +
  ylim(0.89975, 0.90025)

plot_phi <- ggplot(estimate_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(-0.001, 0.001)



# Arrange all subplots into a single large plot.
grid.arrange(plot_A, plot_gamma, plot_tau, plot_phi, ncol = 3)
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
## Removed 2 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```

```
## Warning: Removed 45 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```




```
# split detrended X1 (original data) into 2 parts : early and late
time_early <- df_MC$Time[df_MC$Time < 100]
y_early <- df_MC$X1[df_MC$Time < 100] - spline_X1$y[df_MC$Time < 100]

time_late <- df_MC$Time[df_MC$Time >= 124]
y_late <- df_MC$X1[df_MC$Time >= 124] - spline_X1$y[df_MC$Time >= 124]

Ini_params_early <- find_initial_params(time_early, y_early)
Ini_params_late <- find_initial_params(time_late, y_late)



library(tibble)
initial_params_tibble <- tibble::tibble(
  Parameter = c("A", "gamma", "tau", "phi"),
  Early = c(Ini_params_early$A, Ini_params_early$gamma, Ini_params_early$tau, Ini_params_early$phi),
  Late = c(Ini_params_late$A, Ini_params_late$gamma, Ini_params_late$tau, Ini_params_late$phi)
)

#
print(initial_params_tibble)
```

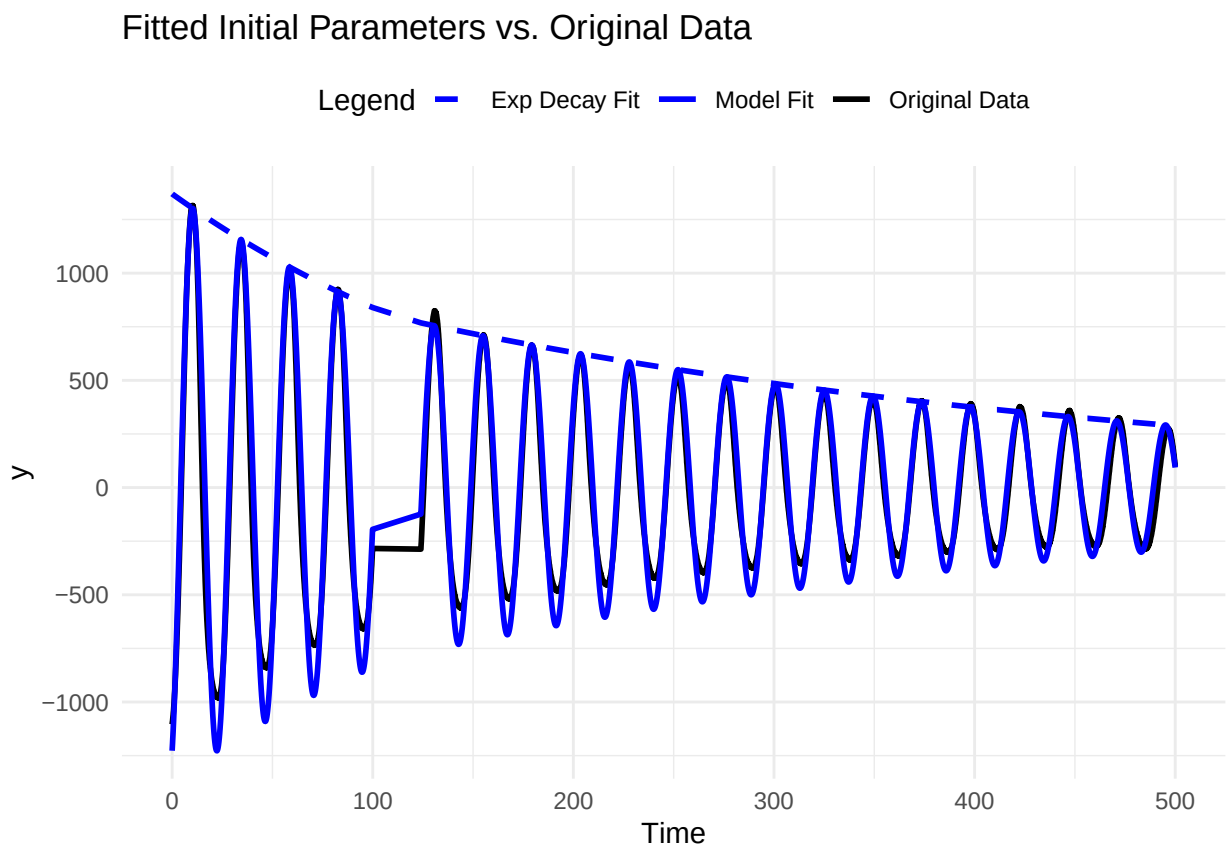
```
## # A tibble: 4 x 3
##   Parameter      Early       Late
##   <chr>          <dbl>      <dbl>
```

```
## 1 A         1370.      1061.
## 2 gamma        0.00490    0.00261
## 3 tau          0.260      0.259
## 4 phi         -2.68      -2.38
```

```r
library(ggplot2)

# **Step 1: calculate time_early and time_late initial parameters**
params_early <- find_initial_params(time_early, y_early)
params_late  <- find_initial_params(time_late, y_late)

# **Step 2: fitted curve by intial parameters**
y_model_early <- model_function(time_early,
                                params_early$A,
                                params_early$gamma,
                                params_early$tau,
                                params_early$phi
                                )

y_model_late <- model_function(time_late,
                               params_late$A,
                               params_late$gamma,
                               params_late$tau,
                               params_late$phi
                               )

#  cos
y_exp_decay_early <- params_early$A * exp(-params_early$gamma * time_early)
y_exp_decay_late  <- params_late$A * exp(-params_late$gamma * time_late)

# **Step 3: plot to compare**
df_plot_early <- data.frame(Time = time_early,
                            Original = y_early,
                            ModelFit = y_model_early,
                            ExpDecay = y_exp_decay_early,
                            Group = "Early")

df_plot_late <- data.frame(Time = time_late,
                           Original = y_late,
                           ModelFit = y_model_late,
                           ExpDecay = y_exp_decay_late,
                           Group = "Late")

df_plot <- rbind(df_plot_early, df_plot_late)


library(ggplot2)

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) +
```

```r
  geom_line(aes(y = ExpDecay, color = "Exp Decay Fit", linetype = "Exp Decay Fit"), size = 1) +


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

  scale_linetype_manual(values = c("Original Data" = "solid",
                                   "Model Fit" = "solid",
                                   "Exp Decay Fit" = "dashed")) +

  #
  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.
```



```r
fit_early <- nlsLM(
  y ~ model_function(t, A, gamma, tau, phi),
  start = list(
    A = as.numeric(Ini_params_early$A),
    gamma = as.numeric(Ini_params_early$gamma),
    tau = as.numeric(Ini_params_early$tau),
    phi = as.numeric(Ini_params_early$phi)

  ),
  data = data.frame(t = time_early, y = y_early),
  lower = c(500, 0.001, 0.1, -pi),
  upper = c(2000, 0.05, 1.5, pi)
)

fit_late <- nlsLM(
  y ~ model_function(t, A, gamma, tau, phi),
  start = list(
    A = as.numeric(Ini_params_late$A),
    gamma = as.numeric(Ini_params_late$gamma),
    tau = as.numeric(Ini_params_late$tau),
    phi = as.numeric(Ini_params_late$phi)

  ),
  data = data.frame(t = time_late, y = y_late),
  lower = c(500, 0.001, 0.1, -pi),
  upper = c(2000, 0.05, 1.5, pi)
)

#
estimates_early <- coef(fit_early)
estimates_late <- coef(fit_late)

#
print("Estimated Parameters for Early Time Period:")
```

```
## [1] "Estimated Parameters for Early Time Period:"
```

```r
print(estimates_early)
```

```
##             A         gamma           tau           phi
##  1.307728e+03  5.893703e-03  2.603823e-01 -2.631845e+00
```

```r
print("Estimated Parameters for Late Time Period:")
```

```
## [1] "Estimated Parameters for Late Time Period:"
```

```r
print(estimates_late)
```

```
##             A         gamma           tau           phi
## 943.872908777   0.002609503   0.258331639  -2.337729068
```

```r
#install.packages("gt")


#   99%
conf_intervals_early <- confint2(fit_early, level = 0.99, method = "asymptotic")
conf_intervals_late <- confint2(fit_late, level = 0.99, method = "asymptotic")

#
print("99% Confidence Intervals for Early Time Period:")
```

```
## [1] "99% Confidence Intervals for Early Time Period:"
```

```r
print(conf_intervals_early)
```

```
##               0.5 %        99.5 %
## A      1.274891e+03  1.340566e+03
## gamma  5.390845e-03  6.396561e-03
## tau    2.598989e-01  2.608658e-01
## phi   -2.655117e+00 -2.608574e+00
```

```r
print("99% Confidence Intervals for Late Time Period:")
```

```
## [1] "99% Confidence Intervals for Late Time Period:"
```

```r
print(conf_intervals_late)
```

```
##               0.5 %        99.5 %
## A     924.957221534 962.788596019
## gamma   0.002536251   0.002682755
## tau     0.258258393   0.258404885
## phi    -2.357650176  -2.317807961
```

```r
#
conf_table <- data.frame(
  Parameter = names(estimates_early),
  Estimate_Early = estimates_early,
  Lower_CI_Early = conf_intervals_early[, 1],
  Upper_CI_Early = conf_intervals_early[, 2],
  Estimate_Late = estimates_late,
  Lower_CI_Late = conf_intervals_late[, 1],
  Upper_CI_Late = conf_intervals_late[, 2]
)

#
print(conf_table)
```

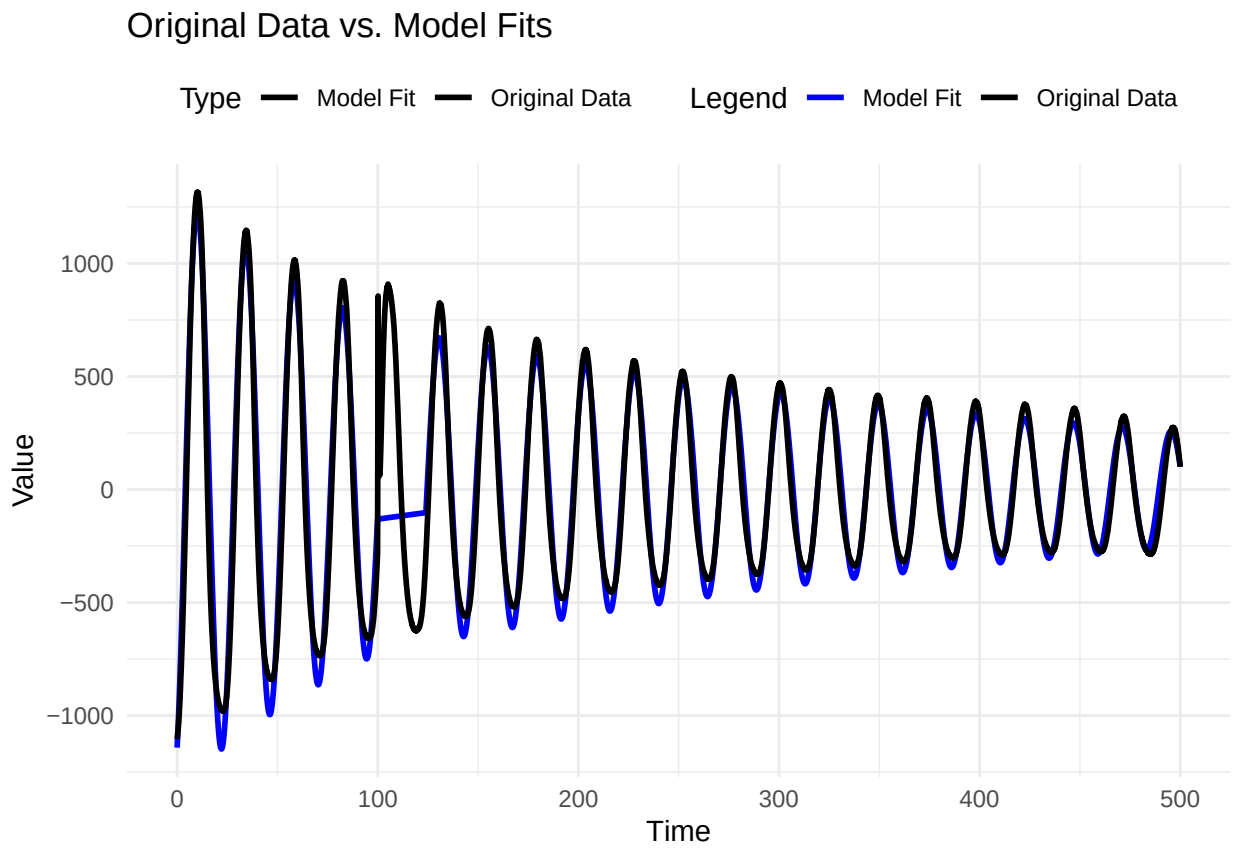
```
##       Parameter Estimate_Early Lower_CI_Early Upper_CI_Early Estimate_Late
## A             A   1.307728e+03   1.274891e+03   1.340566e+03 943.872908777
## gamma     gamma   5.893703e-03   5.390845e-03   6.396561e-03   0.002609503
## tau         tau   2.603823e-01   2.598989e-01   2.608658e-01   0.258331639
```

```
## phi         phi  -2.631845e+00  -2.655117e+00  -2.608574e+00  -2.337729068
##       Lower_CI_Late Upper_CI_Late
## A     924.957221534 962.788596019
## gamma   0.002536251   0.002682755
## tau     0.258258393   0.258404885
## phi    -2.357650176  -2.317807961
```

```
#    kable()   gt()
library(knitr)
kable(conf_table, format = "markdown")
```

	Parameter	Estimate_Early	Lower_CI_Early	Upper_CI_Early	Estimate_Late	Lower_CI_Late	Upper_CI_Late
A	A	1307.7282617	1274.8907368	1340.5657866	943.8729088	924.9572215	962.7885960
gamma	gamma	0.0058937	0.0053908	0.0063966	0.0026095	0.0025363	0.0026828
tau	tau	0.2603823	0.2598989	0.2608658	0.2583316	0.2582584	0.2584049
phi	phi	-2.6318455	-2.6551171	-2.6085739	-2.3377291	-2.3576502	-2.3178080

```
y_fit_early <- model_function(time_early,
                              estimates_early["A"],
                              estimates_early["gamma"],
                              estimates_early["tau"],
                              estimates_early["phi"]
                              )

y_fit_late <- model_function(time_late,
                             estimates_late["A"],
                             estimates_late["gamma"],
                             estimates_late["tau"],
                             estimates_late["phi"])

#  to plot
df_plot_early <- data.frame(Time = time_early, Value = y_fit_early, Type = "Model Fit")
df_plot_late <- data.frame(Time = time_late, Value = y_fit_late, Type = "Model Fit")
df_plot_original <- data.frame(Time = df_MC$Time, Value = df_MC$X1 - spline_X1$y, Type = "Original Data

#
df_plot <- rbind(df_plot_early, df_plot_late, df_plot_original)

# compare
ggplot(df_plot, aes(x = Time, y = Value, color = Type, linetype = Type)) +
  geom_line(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 = "Original Data vs. Model Fits",
       x = "Time", y = "Value", color = "Legend") +
  theme_minimal() +
  theme(legend.position = "top")
```



T wald test

```
#
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
```

```
#
estimate_sigma2 <- function(y, t, params) {
  A <- params["A"]
```

```
  gamma <- params["gamma"]
  tau <- params["tau"]
  phi <- params["phi"]
  y_shift <- 0

  f_values <- model_function(t, A, gamma, tau, phi)
  residuals <- y - f_values
  SSR <- sum(residuals^2)
  sigma2 <- SSR / length(t)
  return(sigma2)
}

#
log_likelihood <- function(params, t, y, sigma2) {
  A <- params["A"]
  gamma <- params["gamma"]
  tau <- params["tau"]
  phi <- params["phi"]
  y_shift <-0

  f_values <- model_function(t, A, gamma, tau, phi)
  residuals <- y - f_values
  n <- length(t)
  log_likelihood_value <- -n / 2 * log(2 * pi * sigma2) - sum(residuals^2) / (2 * sigma2)

  return(log_likelihood_value)
}

# covariance from hessian
compute_covariance <- function(params, t, y, sigma2) {
  hessian_matrix <- hessian(func = log_likelihood, x = params, t = t, y = y, sigma2 = sigma2)
  fisher_information <- -hessian_matrix
  covariance_matrix <- solve(fisher_information)
  return(covariance_matrix)
}

# Wald test
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))
}



#

#   sigma^2
sigma2_early <- estimate_sigma2(y_early, time_early, estimates_early)
sigma2_late <- estimate_sigma2(y_late, time_late, estimates_late)
```

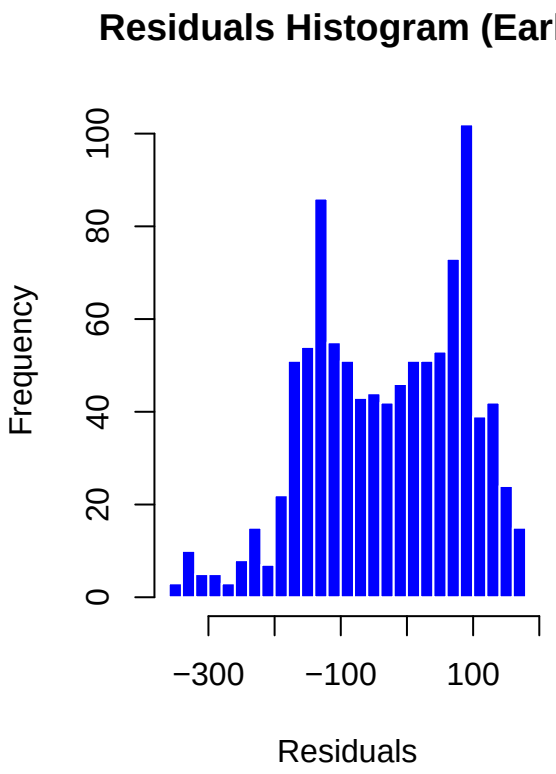
```r
cat("Sigma² for early:", sigma2_early, "\n")
```

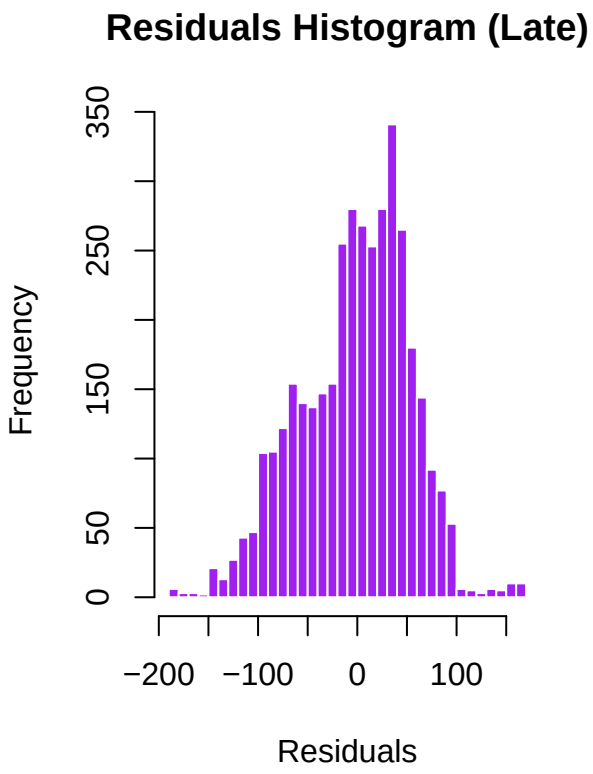
```
## Sigma² for early: 14072.79
```

```r
cat("Sigma² for late:", sigma2_late, "\n")
```

```
## Sigma² for late: 3043.673
```

```r
# compute residuals
compute_residuals <- function(y, t, params) {
  A <- params["A"]
  gamma <- params["gamma"]
  tau <- params["tau"]
  phi <- params["phi"]
  y_shift <- params["y_shift"]

  f_values <- model_function(t, A, gamma, tau, phi)
  residuals <- y - f_values
  return(residuals)
}
residuals_early <- compute_residuals(y_early, time_early, estimates_early)
residuals_late <- compute_residuals(y_late, time_late, estimates_late)

# plot residual
par(mfrow = c(1, 2))  #


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


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






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

```r
covariance_early <- compute_covariance(estimates_early, time_early, y_early, sigma2_early)
covariance_late <- compute_covariance(estimates_late, time_late, y_late, sigma2_late)

remove_phi <- function(params, covariance_matrix) {
  #find phi in parameter list
  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))
}


early_filtered <- remove_phi(estimates_early, covariance_early)
late_filtered <- remove_phi(estimates_late, covariance_late)

#  Wald Test (exclude phi)
wald_result <- wald_test(early_filtered$params, late_filtered$params,
                         early_filtered$covariance, late_filtered$covariance)
```

```r
# **output Wald Test **
cat("Wald Test Statistic:", wald_result$T_Wald, "\n")
```

```
## Wald Test Statistic: 763.3974
```

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

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

```r
print(estimates_early)
```

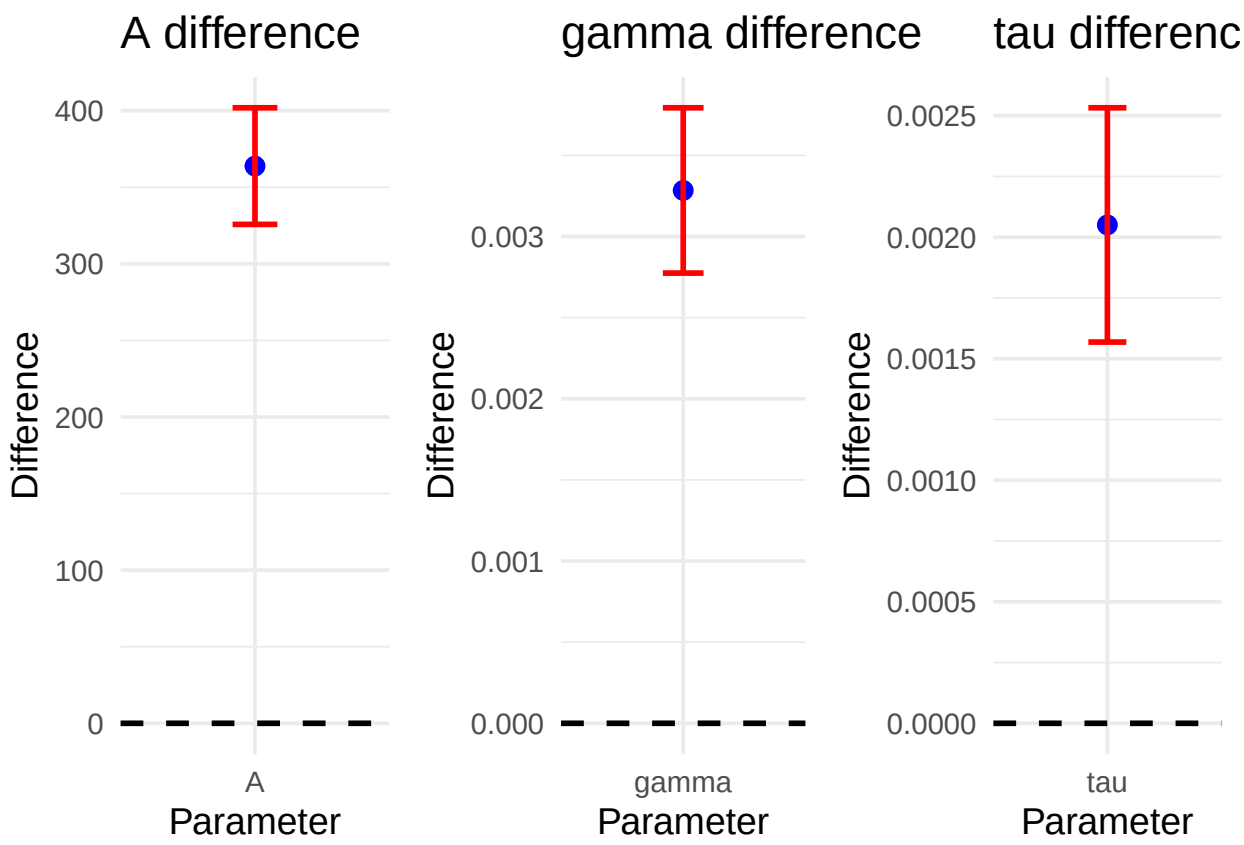
```
##              A         gamma           tau           phi
##  1.307728e+03  5.893703e-03  2.603823e-01 -2.631845e+00
```

```r
print(estimates_late)
```

```
##              A         gamma           tau           phi
## 943.872908777   0.002609503   0.258331639  -2.337729068
```

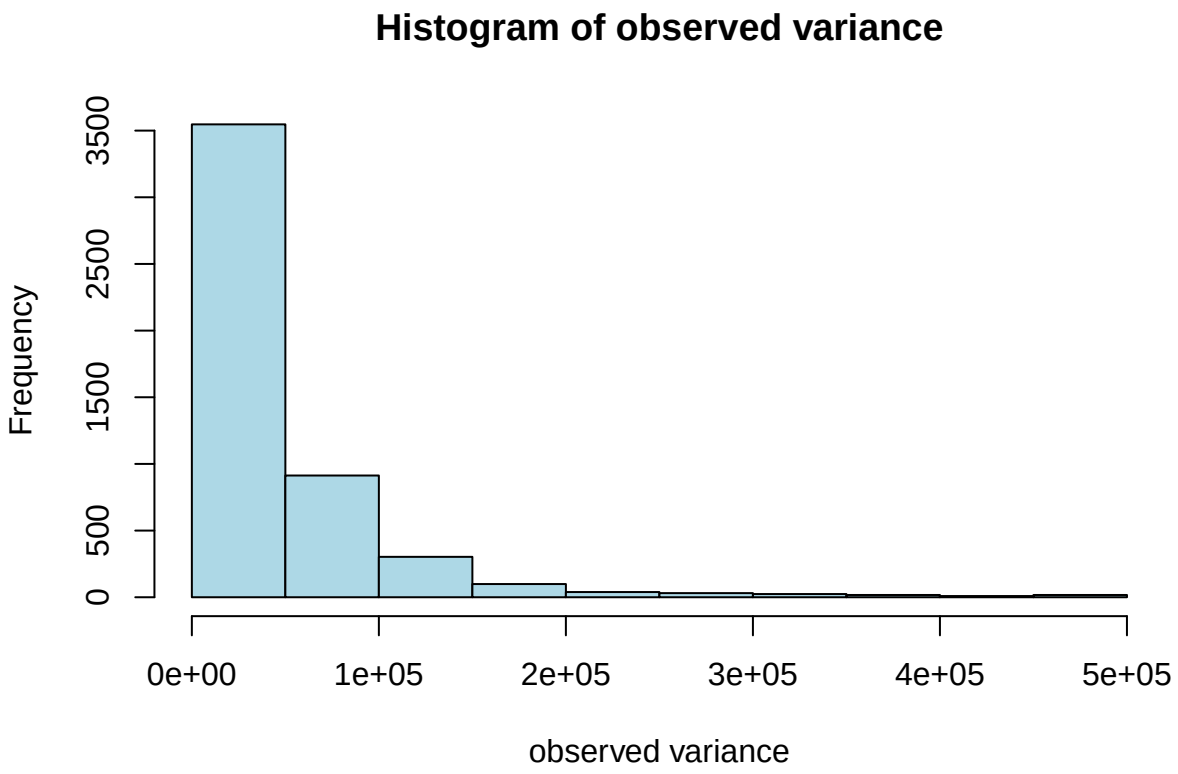
```r
library(dplyr)

param_diff <- early_filtered$params - late_filtered$params

# covariance from hessian
var_diff <- diag(early_filtered$covariance + late_filtered$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
```



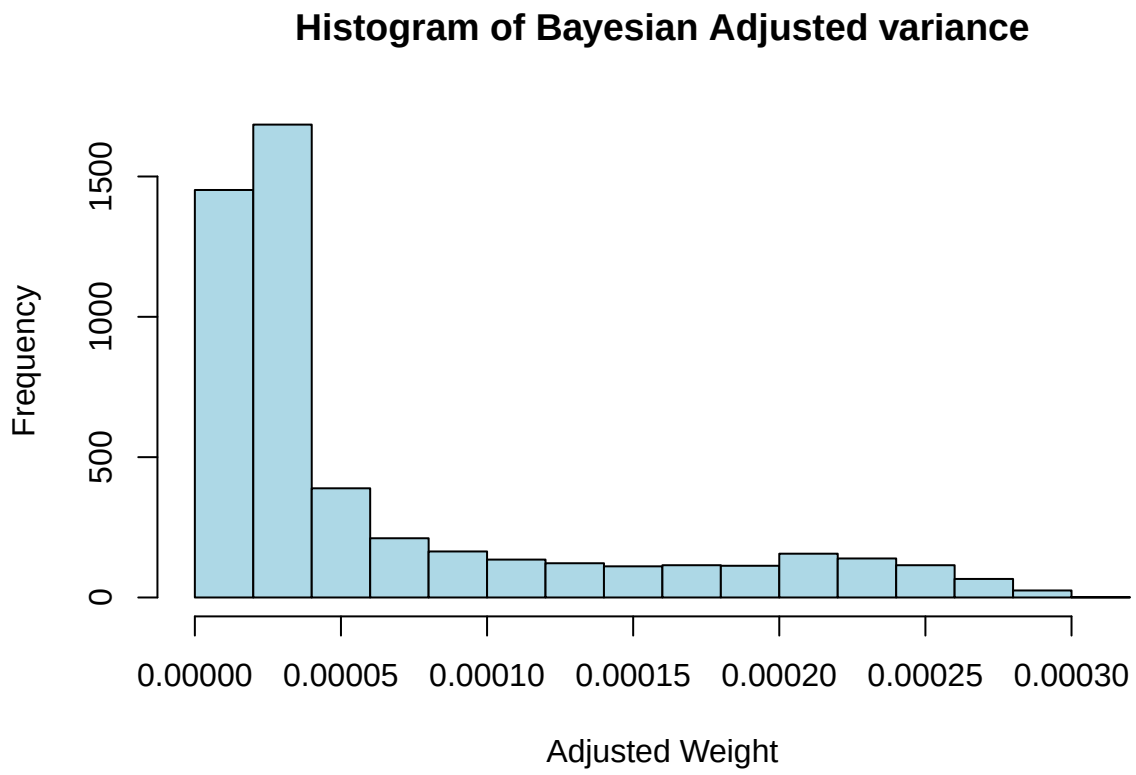
Symmetric Model with Time dependent Variable

```
# Function to calculate variance at each time point
calculate_variance <- function(replicates) {
  n <- 4
  mean_vals <- rowMeans(replicates)  # Mean for each time point
  variance <- rowSums((replicates - mean_vals)^2) / (4 - 1)  # Variance formula
  return(variance)
}
replicates <- df_MC[, c("X1", "X2", "X3", "X4")]
# Calculate variance
observed_variances <- calculate_variance(replicates)
hist(observed_variances, main = "Histogram of observed variance",
     xlab = "observed variance",
     ylab = "Frequency",
     col = "lightblue",
     border = "black")
```

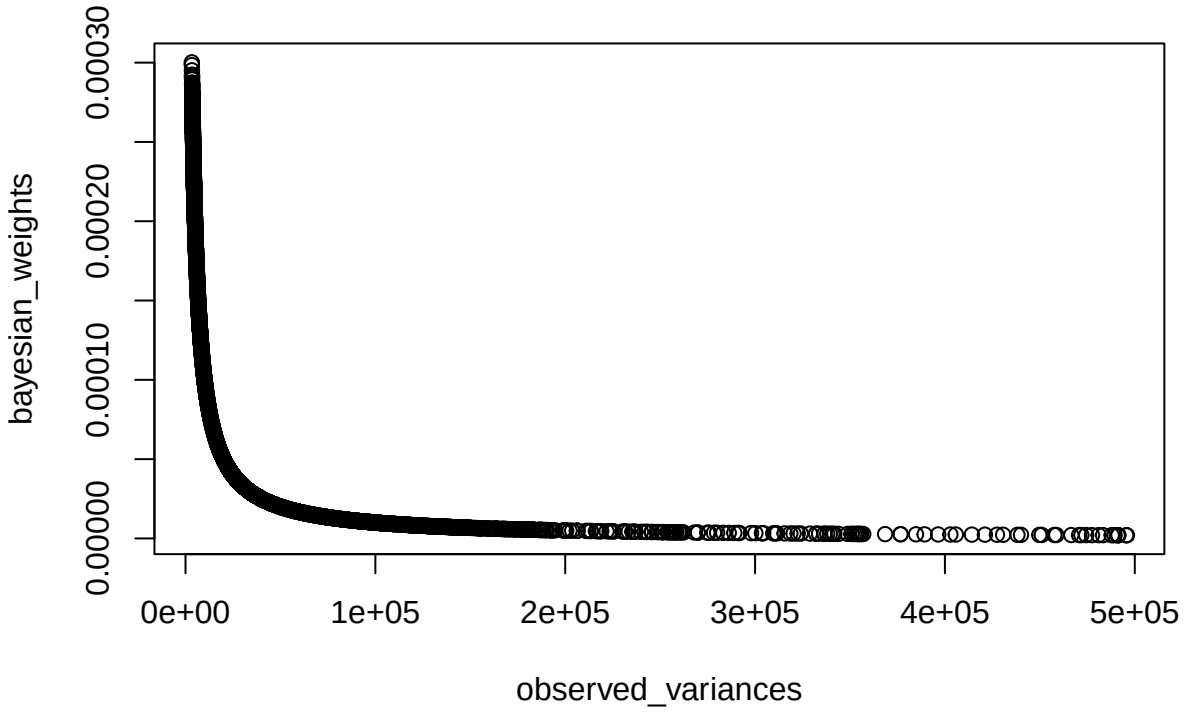



```r
bayesian_variance_estimation <- function(observed_variances, n, alpha_prior, beta_prior) {
  weights <- numeric(length(observed_variances))
  for (i in 1:length(observed_variances)) {
    S2_i <- observed_variances[i]
    alpha_post <- alpha_prior + (n - 1) / 2
    beta_post <- beta_prior + S2_i * (n - 1) / 2
    posterior_mean <- alpha_post / beta_post # expectation for sigma^(-2)
    weights[i] <- posterior_mean # so weight = sigma^(-1)
  }
  return(weights)
}

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

  # Calculate alpha_prior and beta_prior
  beta_prior <- sample_mean / sample_variance
  alpha_prior <- sample_mean^2 / sample_variance

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

```r
# Step 2: Perform Bayesian estimation using priors from Method of Moments

# Get priors from Method of Moments
prior <- calculate_moments(observed_variances)
alpha_prior <- prior$alpha_prior
beta_prior <- prior$beta_prior

# Step 3: Compute Bayesian weights
n <- nrow(replicates)  # Number of samples per group
bayesian_weights <- bayesian_variance_estimation(observed_variances, n,
                                                 alpha_prior, beta_prior)

# Output the Bayesian weights

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




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




```
print(alpha_prior)
```

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

```
print(beta_prior)
```

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

```
summary(bayesian_weights)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 2.017e-06 1.808e-05 2.952e-05 6.409e-05 8.109e-05 3.002e-04
```

```
# Prepare x and y data
# x represents the time points, y represents the detrended data
x <- 1:length(df_MC$Time)  # Time points
y <- df_MC$X1 - spline_X1$y  # Detrended data

# Use Bayesian weights obtained from posterior estimation
weights <- bayesian_weights  # w_i = 1 / sigma_i^2
```

```r
#       y_shift
Ini_params_early <- find_initial_params(time_early, y_early)
Ini_params_late <- find_initial_params(time_late, y_late)

# Bayesian
weights_early <- bayesian_weights[1:length(time_early)]
weights_late  <- bayesian_weights[1241:5001]


# **nlsLM     y_shift **
fit_early_ST <- nlsLM(
  y_early ~ model_function(time_early, A, gamma, tau, phi),
  start = list(A = Ini_params_early$A, gamma = Ini_params_early$gamma,
               tau = Ini_params_early$tau, phi = Ini_params_early$phi),
  weights = weights_early,
  data = data.frame(time_early, y_early)
)

fit_late_ST <- nlsLM(
  y_late ~ model_function(time_late, A, gamma, tau, phi),
  start = list(A = Ini_params_late$A, gamma = Ini_params_late$gamma,
               tau = Ini_params_late$tau, phi = Ini_params_late$phi),
  weights = weights_late,
  data = data.frame(time_late, y_late)
)

# **    **
estimates_early_ST <- coef(fit_early_ST)
estimates_late_ST <- coef(fit_late_ST)

# **  99%   **
conf_early_ST <- confint2(fit_early_ST, level = 0.99)
conf_late_ST <- confint2(fit_late_ST, level = 0.99)

names(estimates_early_ST) <- c("A", "gamma", "tau", "phi")

#   estimates_late_ST
names(estimates_late_ST) <- c("A", "gamma", "tau", "phi")
# **   **
results_table <- data.frame(
  Parameter = c("A_ST", "gamma_ST", "tau_ST", "phi_ST"),
  Estimate_Early = estimates_early_ST,
  CI_Lower_Early = conf_early_ST[, 1],
  CI_Upper_Early = conf_early_ST[, 2],
  Estimate_Late = estimates_late_ST,
  CI_Lower_Late = conf_late_ST[, 1],
  CI_Upper_Late = conf_late_ST[, 2]
)
```

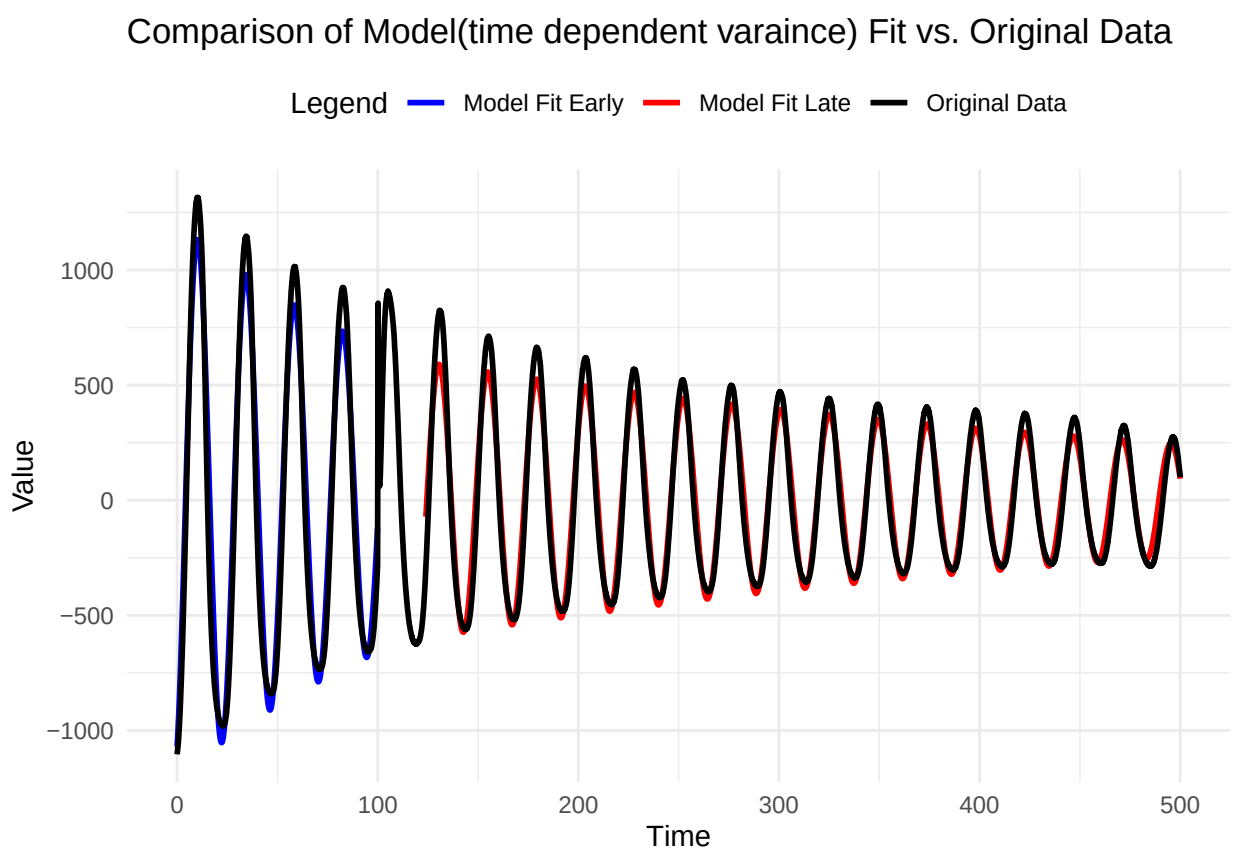
```r
results_table
```

```
##       Parameter Estimate_Early CI_Lower_Early CI_Upper_Early Estimate_Late
## A          A_ST   1.203141e+03   1.179924e+03   1.226359e+03 805.196637233
```

```
## gamma  gamma_ST   5.990269e-03   5.650847e-03   6.329691e-03   0.002379595
## tau      tau_ST   2.607200e-01   2.602271e-01   2.612128e-01   0.258291089
## phi      phi_ST  -2.666469e+00  -2.693138e+00  -2.639800e+00  -2.302549663
##       CI_Lower_Late CI_Upper_Late
## A     793.257336329 817.135938137
## gamma   0.002325874   0.002433317
## tau     0.258219439   0.258362740
## phi    -2.323278968  -2.281820359
```

```r
library(ggplot2)

# 1
y_fit_early_ST <- model_function(time_early,
                                 estimates_early_ST["A"],
                                 estimates_early_ST["gamma"],
                                 estimates_early_ST["tau"],
                                 estimates_early_ST["phi"])

y_fit_late_ST <- model_function(time_late,
                                estimates_late_ST["A"],
                                estimates_late_ST["gamma"],
                                estimates_late_ST["tau"],
                                estimates_late_ST["phi"])

# 2   DataFrame   ggplot
df_plot_original <- data.frame(Time = df_MC$Time,
                               Value = df_MC$X1 - spline_X1$y,
                               Type = "Original Data")

df_plot_early_ST <- data.frame(Time = time_early,
                               Value = y_fit_early_ST,
                               Type = "Model Fit Early")

df_plot_late_ST <- data.frame(Time = time_late,
                              Value = y_fit_late_ST,
                              Type = "Model Fit Late")

#
df_plot <- rbind(df_plot_original, df_plot_early_ST, df_plot_late_ST)

#
ggplot(df_plot, aes(x = Time, y = Value, color = Type, linetype = Type)) +
  geom_line(size = 1) +

  #
  scale_color_manual(values = c("Original Data" = "black",
                                "Model Fit Early" = "blue",
                                "Model Fit Late" = "red")) +

  #
  scale_linetype_manual(values = c("Original Data" = "solid",
                                   "Model Fit Early" = "solid",
                                   "Model Fit Late" = "solid")) +
```

```
  #
  labs(title = "Comparison of Model(time dependent varaince) Fit vs. Original Data",
       x = "Time", y = "Value", color = "Legend", linetype = "Legend") +
  theme_minimal() +
  theme(legend.position = "top")
```



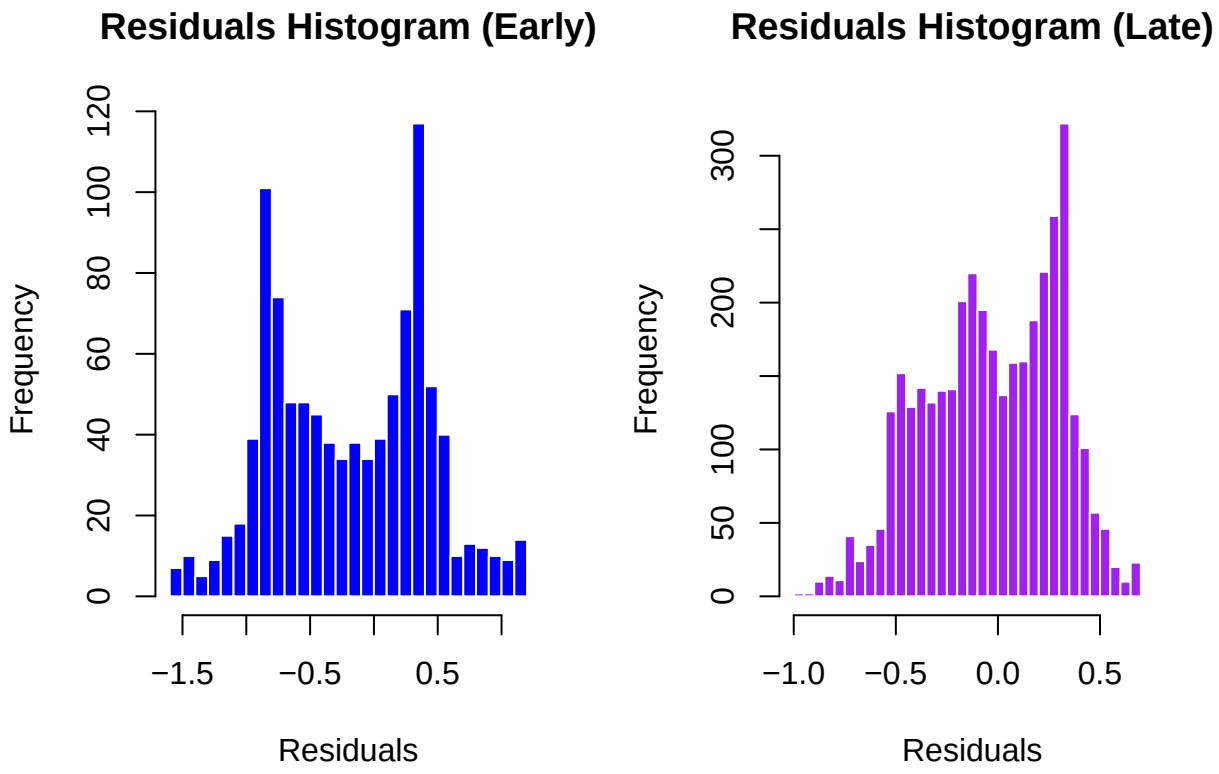

```
sigma2_early_ST <- estimate_sigma2(y_early, time_early, estimates_early_ST)
sigma2_late_ST  <- estimate_sigma2(y_late, time_late, estimates_late_ST)
covariance_early_ST <- compute_covariance(estimates_early_ST, time_early, y_early, sigma2_early_ST)
covariance_late_ST  <- compute_covariance(estimates_late_ST, time_late, y_late, sigma2_late_ST)
early_filtered_ST <- remove_phi(estimates_early_ST, covariance_early_ST)
late_filtered_ST  <- remove_phi(estimates_late_ST, covariance_late_ST)
wald_result_ST <- wald_test(early_filtered_ST$params, late_filtered_ST$params,
                            early_filtered_ST$covariance, late_filtered_ST$covariance)

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

```
## Wald Test Statistic: 973.3046
```

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

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

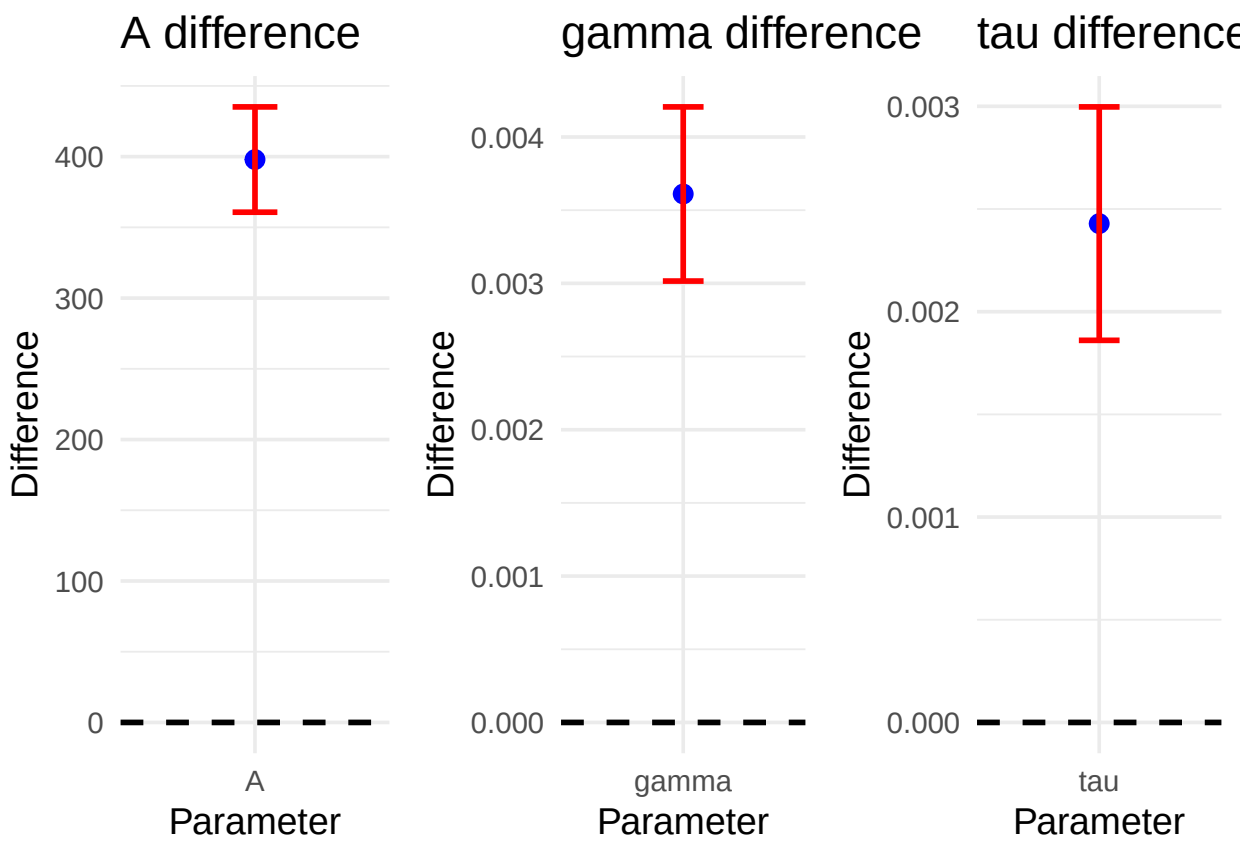
```r
residuals_early_ST <- compute_residuals(y_early, time_early, estimates_early_ST) * sqrt(weights_early)
residuals_late_ST <- compute_residuals(y_late, time_late, estimates_late_ST) *sqrt(weights_late)

#
par(mfrow = c(1, 2))  #

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

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




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

```r
param_diff <- early_filtered_ST$params - late_filtered_ST$params

# covariance from hessian
var_diff <- diag(early_filtered_ST$covariance + late_filtered_ST$covariance)
se_diff <- sqrt(var_diff)

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

```r
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
```

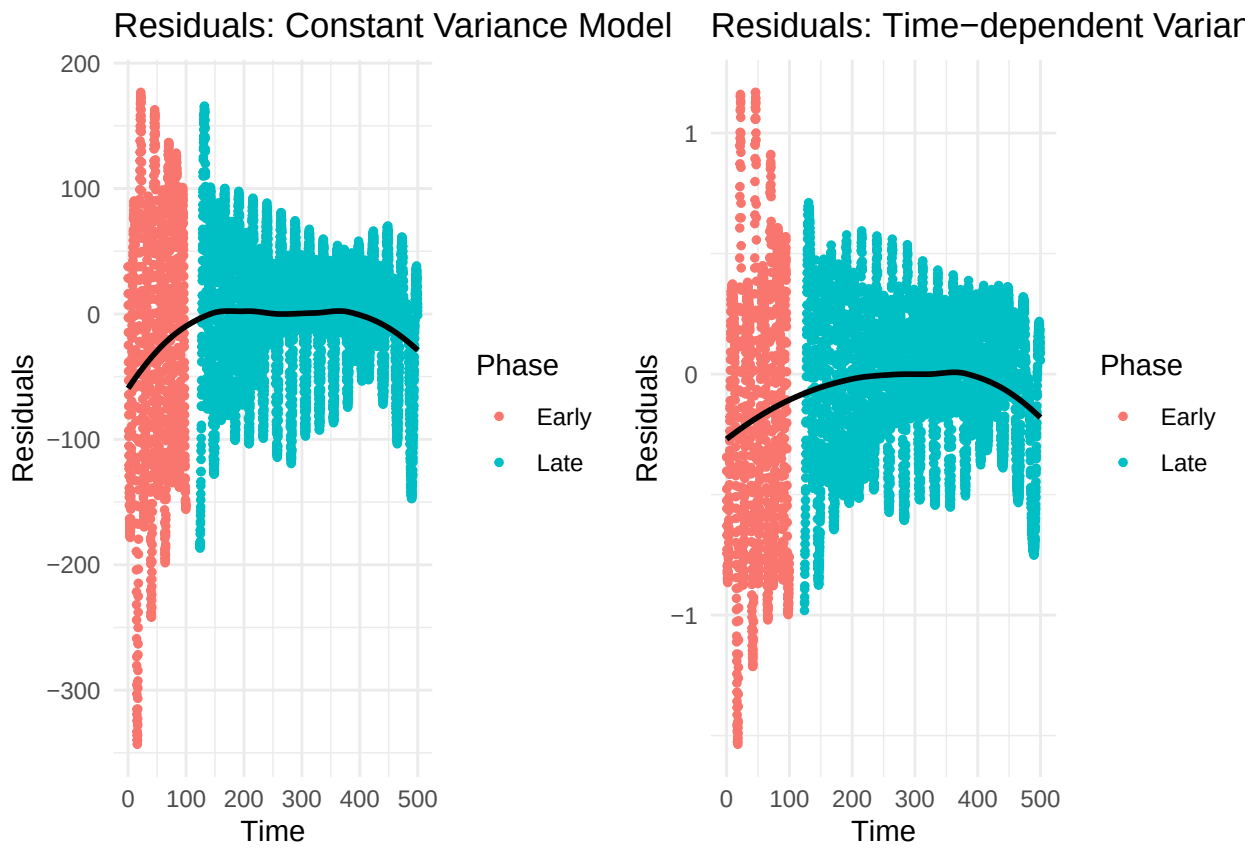


```r
library(ggplot2)
library(gridExtra)
detrend_curve = df_MC$X1-spline_X1$y

#
df_residuals <- data.frame(
  Time = c(time_early, time_late, time_early, time_late),
  Residuals = c(residuals_early, residuals_late, residuals_early_ST, residuals_late_ST),
  Model = rep(c("Constant Variance", "Constant Variance", "Time-dependent Variance", "Time-dependent Va
              times = c(length(time_early), length(time_late), length(time_early), length(time_late))),
  Phase = rep(c("Early", "Late", "Early", "Late"),
              times = c(length(time_early), length(time_late), length(time_early), length(time_late)))
)

# Constant Variance residual vs time
p1 <- ggplot(df_residuals[df_residuals$Model == "Constant Variance", ], aes(x = Time, y = Residuals, co
  geom_point(size = 1) +
  geom_smooth(method = "loess", se = FALSE, color = "black") +
  labs(title = "Residuals: Constant Variance Model", x = "Time", y = "Residuals") +
  theme_minimal()

# Time-dependent Variance ressidual vs time
p2 <- ggplot(df_residuals[df_residuals$Model == "Time-dependent Variance", ], aes(x = Time, y = Residual
  geom_point(size = 1) +
  geom_smooth(method = "loess", se = FALSE, color = "black") +
  labs(title = "Residuals: Time-dependent Variance Model", x = "Time", y = "Residuals") +
  theme_minimal()

#
grid.arrange(p1, p2, ncol = 2)
```

```
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_smooth()`).
```

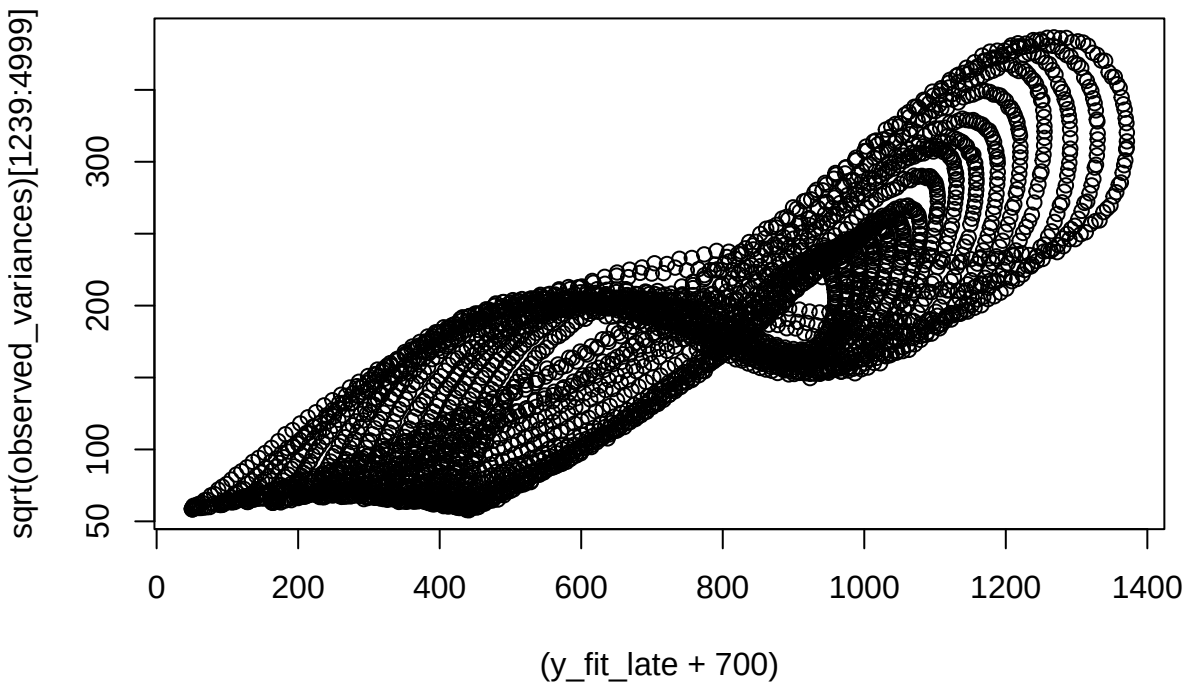
```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_point()`).
```




```
length(y_fit_late)
```

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

```
plot((y_fit_late+700),sqrt(observed_variances)[1239:4999])
```

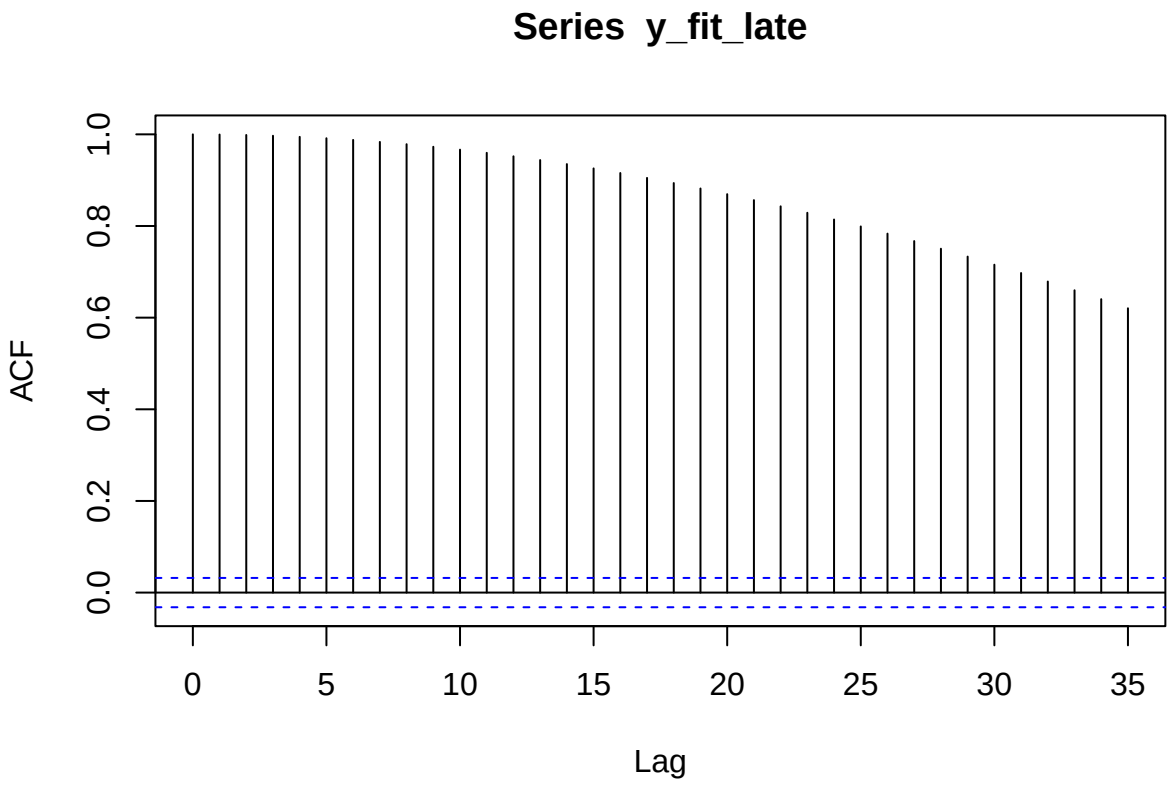



```
xxx=y_fit_late+700
yyy=sqrt(observed_variances)[1239:4999]
lm(yyy~xxx)
```

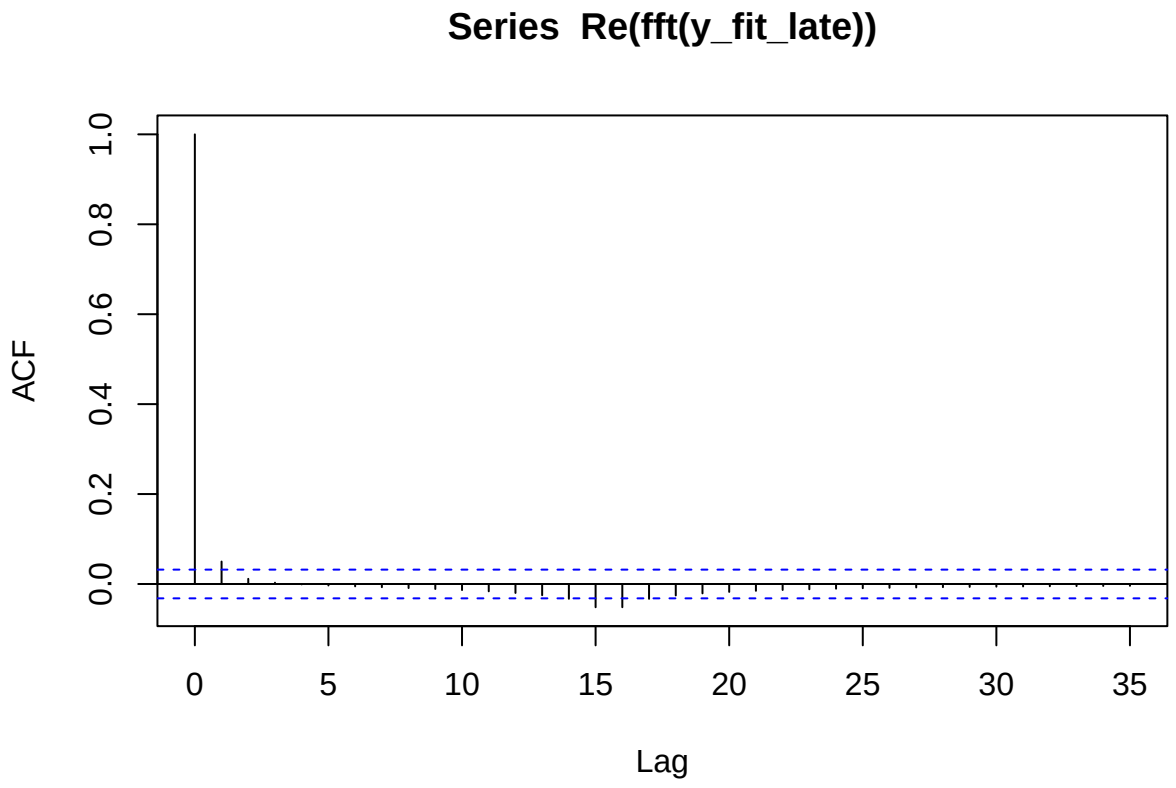
```
##
## Call:
## lm(formula = yyy ~ xxx)
##
## Coefficients:
## (Intercept)          xxx
##     32.8819       0.2024
```

FFT

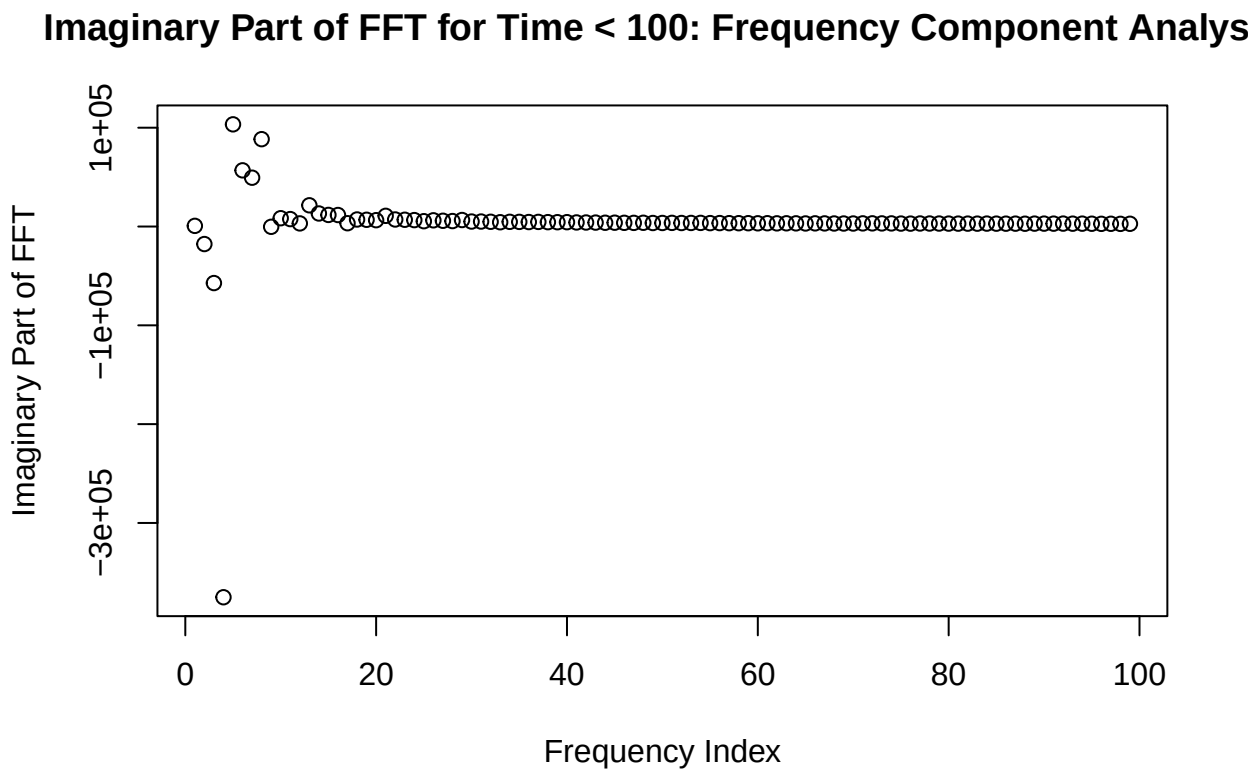
```
acf(y_fit_late)
```



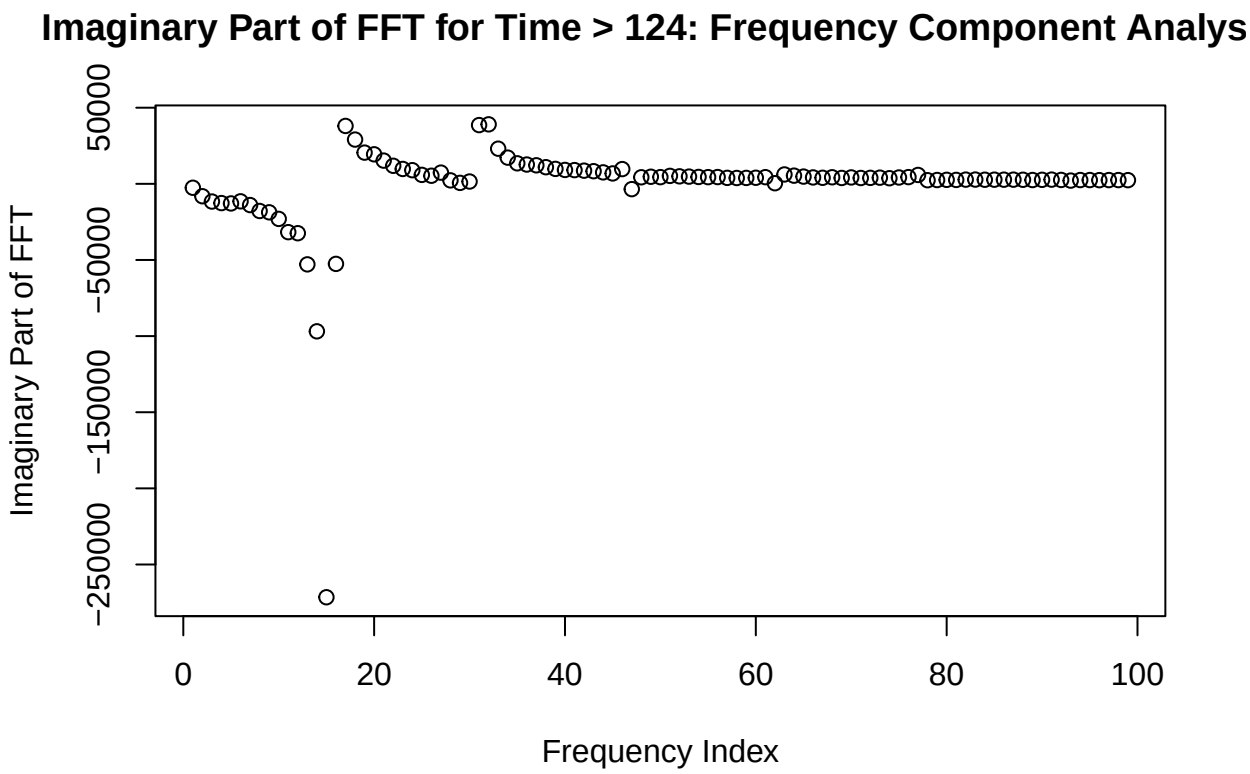

```
acf(Re(fft(y_fit_late)))
```




```
detrend_curve = df_MC$X1-spline_X1$y
plot(Im(fft(detrend_curve[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")
```




```
plot(Im(fft(detrend_curve[1239:4999]))[2:100],type = "p",
     main = "Imaginary Part of FFT for Time > 124: Frequency Component Analysis",
     xlab = "Frequency Index",
     ylab = "Imaginary Part of FFT")
```




```
# find_initial_params <- function(t, y) {
#   peaks <- findpeaks(y, minpeakheight = 100, minpeakdistance = 15)
#   if (is.null(peaks) || nrow(peaks) < 2) {
#     return(NULL)  #         NULL
#   }
#
#   t_peaks <- t[peaks[, 2]]
#   y_peaks <- peaks[, 1]
#
#   #   A   gamma
#   log_y_peaks <- log(y_peaks)
#
#   fit_linear <- lm(log_y_peaks ~ t_peaks)
#   log_A <- coef(fit_linear)[1]
#   init_gamma <- -coef(fit_linear)[2]
#   init_A <- exp(log_A)
#
#   #   tau
#   delta_values <- diff(t_peaks[1:min(5, length(t_peaks))])  #   8
#   Delta <- mean(delta_values)
#   init_tau <- 2 * pi / Delta
#
#     #
#   random_t <- sample(t, 5)
#   phi_values <- sapply(random_t, function(t0) {
#     y_t0 <- model_function(t0, init_A, init_gamma, init_tau, 0)  #   y_t0
```

```
#
#     cos_input <- y_t0 / (init_A * exp(-init_gamma * t0))
#     cos_input <- pmin(pmax(cos_input, -1), 1)  #    [-1,1]
#     phi_t <- acos(cos_input) - init_tau * t0
#     return(phi_t)
#   })
#
#   # ** phi   **
#   init_phi <- mean(phi_values)
#
#   # **  phi   [-2 , 2]  **
#   init_phi <- (init_phi + 2 * pi) %% (4 * pi) - 2 * pi
#   return(c(A = init_A, gamma = init_gamma, tau = init_tau, phi = init_phi))
# }
```

```
# Function to extract first N FFT 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)
}

# Function to fit FFT-based symmetric model
fit_fft_model <- function(y_fft, t, N, init_params) {
  fit <- try(nlsLM(
    y_fft ~ fft_extract(model_function(t, A, gamma, tau, phi), N),
    start = init_params,
    data = data.frame(y_fft = y_fft)
  ), silent = TRUE)

  if (inherits(fit, "try-error")) {
    return(NULL)
  }
  estimates <- coef(fit)
  names(estimates) <- c("A", "gamma", "tau", "phi")  # Standardize names
  conf_intervals <- confint2(fit, level = 0.99, method = 'asymptotic')
  return(list(estimates = estimates, conf_intervals = conf_intervals))
}

# Run simulations for FFT coverage
run_fft_simulations <- function(n_simulations = 100, sigma = 10, N = 20) {
  true_params <- c(A = 1000, gamma = 0.025, tau = 0.9, phi = 0)
  t <- seq(0, 100, by = 0.1)


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

    names(init_params) =  c("A", "gamma", "tau", "phi")
    y_fft <- fft_extract(simulated_data, N)
    fit_result <- fit_fft_model(y_fft, t, N, init_params)
```

```
    results[[i]] <- fit_result
  }

  results <- Filter(Negate(is.null), results)

  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)
}
fft_coverage_results <- run_fft_simulations(n_simulations = 100, sigma = 10, N = 20)
# Print results
print(fft_coverage_results)
```

```
## [1] 0.99 0.97 0.99 0.97
```

```
# Apply to Real Data
N_early <- 20
N_late <- 50
# time_early <- df_MC$Time[df_MC$Time < 100]
# y_early <- df_MC$X1[df_MC$Time < 100] - spline_X1$y[df_MC$Time < 100]
#
# time_late <- df_MC$Time[df_MC$Time >= 124]
# y_late <- df_MC$X1[df_MC$Time >= 124] - spline_X1$y[df_MC$Time >= 124]
y_fft_early <- fft_extract(y_early, N_early)
y_fft_late <- fft_extract(y_late, N_late)

# Ini_params_early <- find_initial_params(time_early, y_early)
# Ini_params_late <- find_initial_params(time_late, y_late)

fit_early_fft <- fit_fft_model(y_fft_early, time_early, N_early, Ini_params_early)
fit_late_fft <- fit_fft_model(y_fft_late, time_late, N_late, Ini_params_late)

# Hypothesis Testing
sigma2_early_fft <- estimate_sigma2(y_early, time_early, fit_early_fft$estimates)
sigma2_late_fft <- estimate_sigma2(y_late, time_late, fit_late_fft$estimates)

covariance_early_fft <- compute_covariance(fit_early_fft$estimates, time_early, y_early, sigma2_early_fft
covariance_late_fft <- compute_covariance(fit_late_fft$estimates, time_late, y_late, sigma2_late_fft)

early_filtered_fft <- remove_phi(fit_early_fft$estimates, covariance_early_fft)
late_filtered_fft <- remove_phi(fit_late_fft$estimates, covariance_late_fft)

wald_result_fft <- wald_test(
  early_filtered_fft$params, late_filtered_fft$params,
  early_filtered_fft$covariance, late_filtered_fft$covariance
```

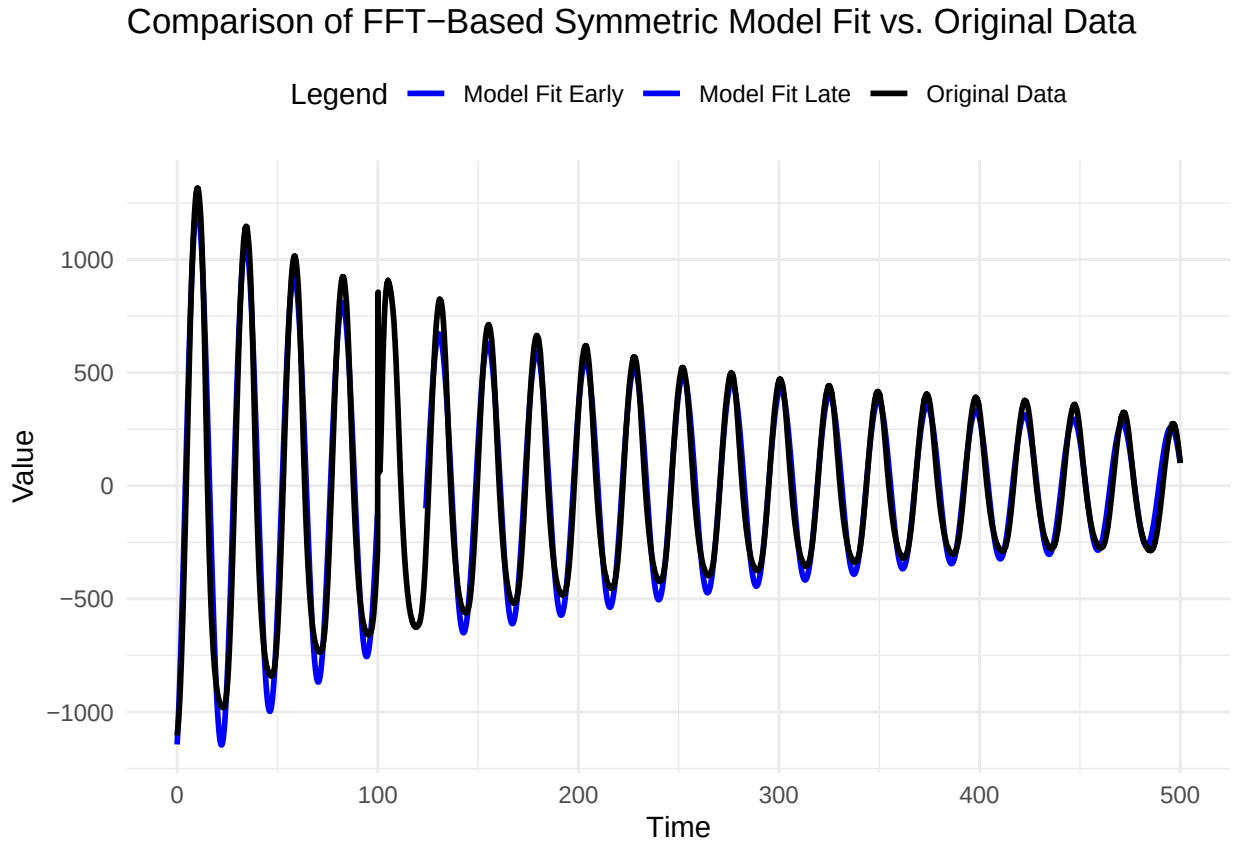
```
)

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

```
## fft Wald Test Statistic : 786.6955
```

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

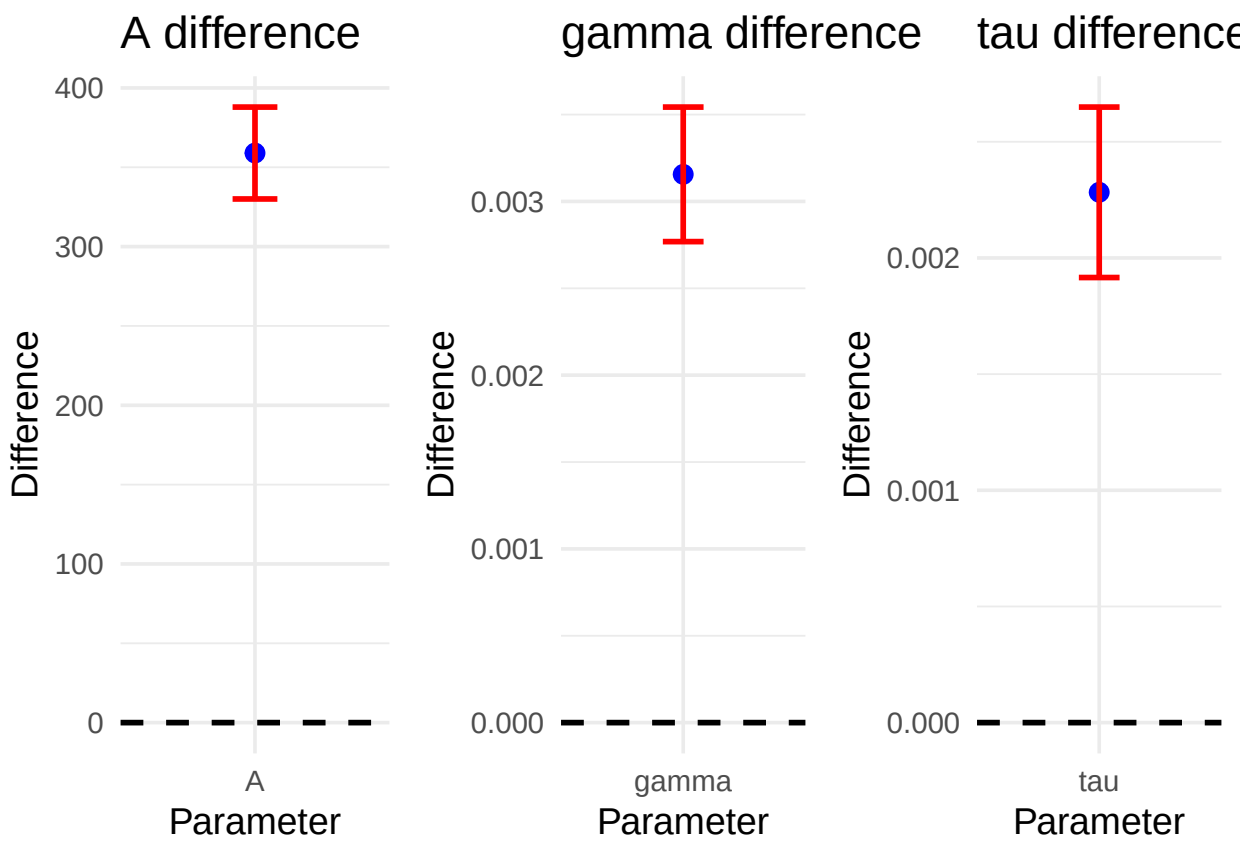
```
## p-value : 0
```

```
# Visualization
y_fit_early_fft <- model_function(time_early,
                                          fit_early_fft$estimates["A"],
                                          fit_early_fft$estimates["gamma"],
                                          fit_early_fft$estimates["tau"],
                                          fit_early_fft$estimates["phi"])

y_fit_late_fft <- model_function(time_late,
                                         fit_late_fft$estimates["A"],
                                         fit_late_fft$estimates["gamma"],
                                         fit_late_fft$estimates["tau"],
                                         fit_late_fft$estimates["phi"])

df_plot_early_fft <- data.frame(Time = time_early, Value = y_fit_early_fft, Type = "Model Fit Early")
df_plot_late_fft <- data.frame(Time = time_late, Value = y_fit_late_fft, Type = "Model Fit Late")
df_plot_original <- data.frame(Time = df_MC$Time, Value = df_MC$X1 - spline_X1$y, Type = "Original Data

df_plot <- rbind(df_plot_original, df_plot_early_fft, df_plot_late_fft)

ggplot(df_plot, aes(x = Time, y = Value, color = Type, linetype = Type)) +
  geom_line(size = 1) +
  scale_color_manual(values = c("Original Data" = "black", "Model Fit Early" = "blue", "Model Fit Late"
  scale_linetype_manual(values = c("solid", "solid", "solid")) +
  labs(title = "Comparison of FFT-Based Symmetric Model Fit vs. Original Data",
       x = "Time", y = "Value", color = "Legend", linetype = "Legend") +
  theme_minimal() +
  theme(legend.position = "top")
```




```
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 - 1.96 * se_diff
CI_upper <- fft_param_diff + 1.96 * 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) +
    labs(title = paste0(df_diff_fft$Parameter[i], " difference"),
         x = "Parameter", y = "Difference") +
    theme_minimal(base_size = 14)
```

```
})


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




```
# Create a data frame to display estimates in a tabular format
fft_estimates_table <- data.frame(
  Parameter = c("A", "gamma", "tau", "phi"),
  Estimate_Early = fit_early_fft$estimates,
  Estimate_Late = fit_late_fft$estimates
)

# Print table using kable for better formatting
library(knitr)
kable(fft_estimates_table, format = "markdown", caption = "FFT Model Parameter Estimates for Early and
```

Table 2: FFT Model Parameter Estimates for Early and Late Time Periods

	Parameter	Estimate_Early	Estimate_Late
A	A	1302.3137595	943.3343747
gamma	gamma	0.0057660	0.0026100
tau	tau	0.2606079	0.2583253
phi	phi	-2.6427786	-2.3342559

Parameter	Estimate_Early	Estimate_Late

```
# Display as a tibble for better readability in console
library(tibble)
as_tibble(fft_estimates_table)
```

```
## # A tibble: 4 x 3
##   Parameter Estimate_Early Estimate_Late
##   <chr>              <dbl>         <dbl>
## 1 A             1302.          943.
## 2 gamma            0.00577       0.00261
## 3 tau              0.261         0.258
## 4 phi             -2.64         -2.33
```

fit FFT-based model with weights

```
# Function to fit FFT-based model with weights
weighted_fit_fft_model <- function(y_fft, t, N, init_params, weights) {
  if (any(is.na(y_fft))) {
    stop("Error: FFT extracted values contain NA!")
  }
  fit <- try(nlsLM(
    y_fft ~ fft_extract(model_function(t, A, gamma, tau, phi), N),
    start = init_params,
    weights = weights,
    control = nls.lm.control(maxiter = 1000),  # Increase iterations
    data = data.frame(y_fft = y_fft)
), silent = TRUE)

  if (inherits(fit, "try-error")) {
    return(NULL)
  }
  estimates <- coef(fit)
  names(estimates) <- c("A", "gamma", "tau", "phi")
  conf_intervals <- confint2(fit, level = 0.99, method = 'asymptotic')
  return(list(estimates = estimates, conf_intervals = conf_intervals))
}

set.seed(347)
# Function to simulate data and compute coverage for Asymmetric Model with Time Dependent Variance
run_time_dep_simulations <- function(n_simulations = 100, sigma_base = 10, scale_factor = 0.2,
                                     N = 20, n_replicates = 4) {
  true_params <- c(A = 1000, gamma = 0.025, tau = 0.9, phi = 0)
  t <- seq(0, 100, by = 0.1)
  f_true <- model_function(t, true_params["A"], true_params["gamma"],
                           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 + 1000)  # _t depends on f_true

    if (any(is.na(sigma_t))) {
      stop("Error: sigma_t contains NA values!")
    }
    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)
    }
    if (any(is.na(replicates))) {
      stop("Error: replicates contain NA values!")
    }
    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
time_dep_coverage_results <- run_time_dep_simulations(n_simulations = 100, sigma_base = 10, scale_factor
                                                      N = 20, n_replicates = 4)
# Print results
print(time_dep_coverage_results)
```

```
## [1] 0.82 1.00 0.98 0.80
```

```r
spline_X1 <- predict(smooth.spline(df_MC$Time, df_MC$X1, spar=1))$y
spline_X2 <- predict(smooth.spline(df_MC$Time, df_MC$X2, spar=1))$y
spline_X3 <- predict(smooth.spline(df_MC$Time, df_MC$X3, spar=1))$y
spline_X4 <- predict(smooth.spline(df_MC$Time, df_MC$X4, spar=1))$y

replicates <- df_MC[, c("X1", "X2", "X3", "X4")]
replicates_detrend <- as.data.frame(sapply(1:4, function(i) replicates[, i] - get(paste0("spline_X", i))


colnames(replicates_detrend) <- colnames(replicates)
head(replicates_detrend)
```

```
##          X1        X2        X3        X4
## 1 -1103.732 -878.1452 -1349.750 -1331.915
## 2 -1095.366 -861.4066 -1335.491 -1315.269
## 3 -1089.260 -841.8980 -1321.251 -1305.253
## 4 -1079.305 -823.6294 -1305.522 -1286.726
## 5 -1069.469 -791.4008 -1289.662 -1268.870
## 6 -1060.623 -780.0022 -1276.763 -1257.684
```

```r
stacked_fft_pred <- function(t, A, gamma, tau, phi,N, n_reps = 4) {
  # 1)  time-domain
  pred_time_domain <- model_function(t, A, gamma, 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)
}
```

apply method on real data(full dataset)

```r
y_early <- replicates_detrend[df_MC$Time < 100, ]
y_late <- replicates_detrend[df_MC$Time >= 124, ]
init_param_early <- find_initial_params(time_early, rowMeans(y_early))
init_param_late <- find_initial_params(time_late, rowMeans(y_late))
print(init_param_early)
```

```
## $A
## (Intercept)
##    1420.214
##
## $gamma
##     t_peaks
## 0.005031248
##
## $tau
```

```
## [1] 0.2599939
##
## $phi
## (Intercept)
##   -2.443942
```

```
print(init_param_late)
```

```
## $A
## (Intercept)
##    1138.477
##
## $gamma
##     t_peaks
## 0.002919151
##
## $tau
## [1] 0.2599939
##
## $phi
## (Intercept)
##   -2.409277
```

FFT with frequency dependent

```
# Define FFT components for early and late phases
N_early <- 20
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 / 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))  #

#
fit_late_fft_stacked <- try(
  nlsLM(
    formula = y_fft_stacked_late ~ stacked_fft_pred(
      t = time_late,
      A, gamma, tau, phi,
      N = N_late,
      n_reps = 4
    ),
    start = init_param_late,
```

```
    weights = weights_late_stacked,
    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)
  names(params_late_fft_stacked) <- c("A", "gamma", "tau", "phi")
  confint_late_fft_stacked <- confint2(fit_late_fft_stacked, level = 0.99, method = 'asymptotic')
    #  adjust phi
  rownames(confint_late_fft_stacked) <- c("A", "gamma", "tau", "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, late):")
  print(params_late_fft_stacked)

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

}
```

```
## [1] "Estimated parameters (stacked, late):"
##             A         gamma           tau           phi
##  1.428122e+03  4.206432e-03  2.595412e-01 -2.527091e+00
## [1] "\n99% confidence intervals (phi adjusted):"
##               0.5 %        99.5 %
## A     768.993323383  2.087251e+03
## gamma   0.001776079  6.636785e-03
## tau     0.257189910  2.618924e-01
## phi    -2.977692916 -2.076489e+00
```

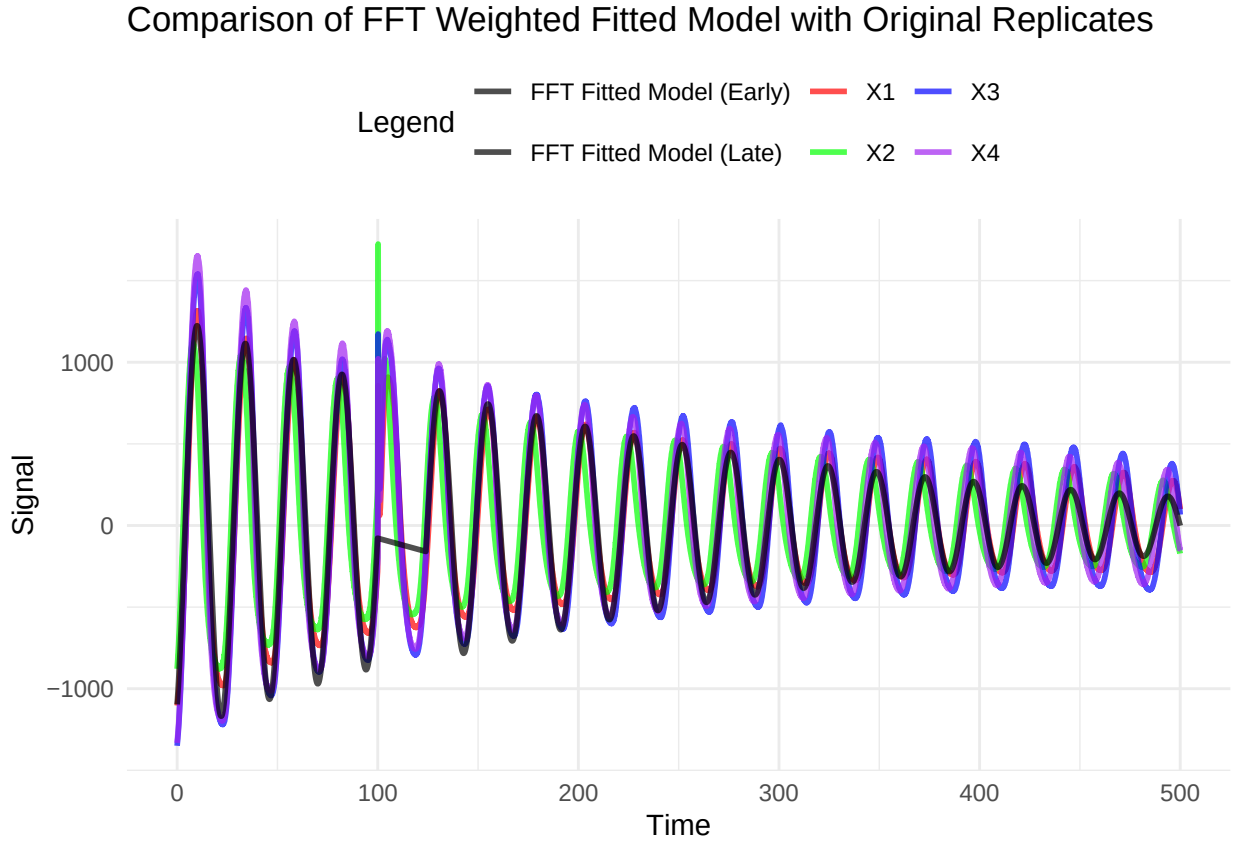
```
# fft_replicates_early : (2*N_early+1)  × 4
fft_replicates_early <- apply(y_early, 2, function(y) fft_extract(y, N_early))
#
observed_variances_early <- calculate_variance(fft_replicates_early)
weights_early <- 1 / observed_variances_early
#
weights_early[is.infinite(weights_early)] <- max(weights_early[!is.infinite(weights_early)])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


#  nlsLM fit
fit_early_fft_stacked <- try(
```

```
  nlsLM(
    formula = y_fft_stacked_early ~ stacked_fft_pred(
      t = time_early,
      A,gamma, tau, phi,
      N = N_early,
      n_reps = 4
    ),
    start = init_param_early,
    weights = weights_early_stacked,

    control = nls.lm.control(maxiter = 1000)  #
  ),
  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)
  names(params_early_fft_stacked) <- c("A", "gamma", "tau", "phi")
  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
  rownames(confint_early_fft_stacked) <- c("A", "gamma", "tau", "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)

}
```

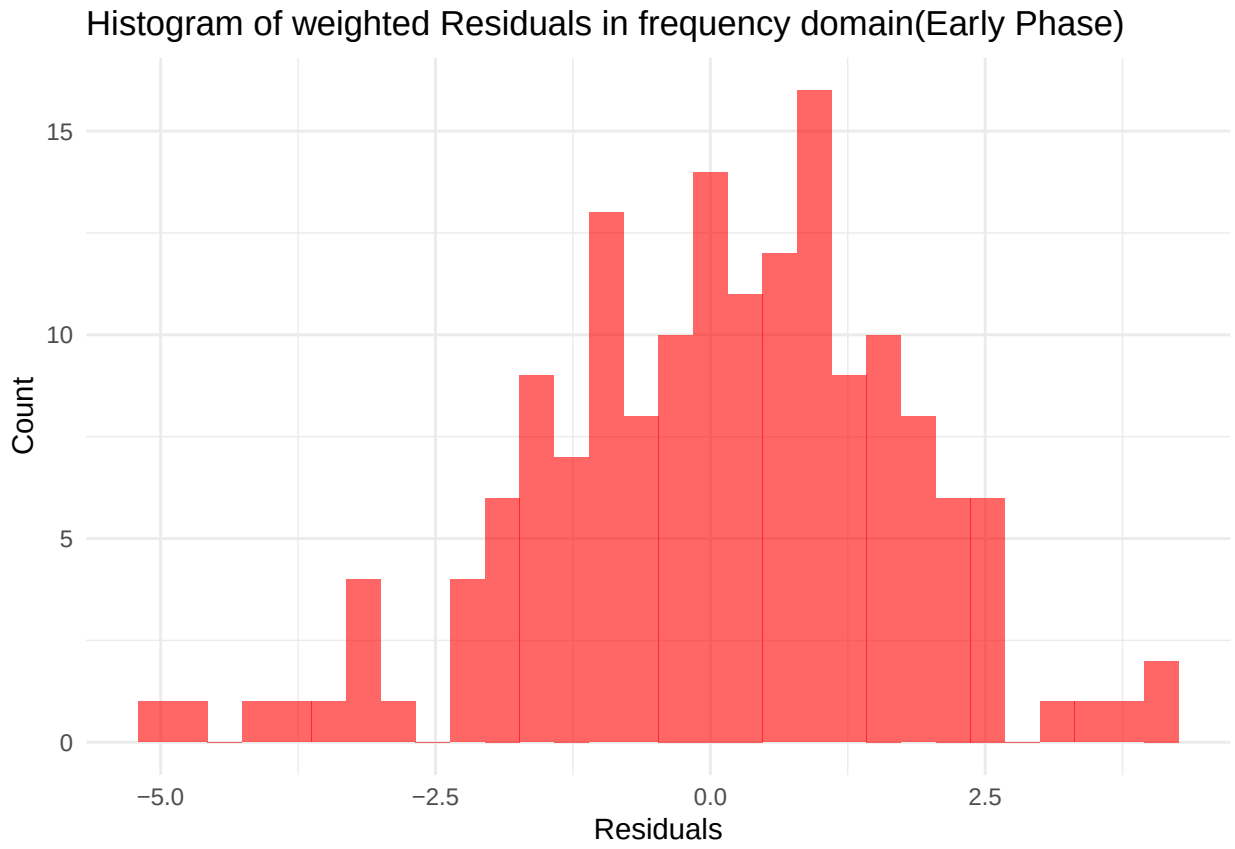
```
## [1] "Estimated parameters (stacked, early):"
##             A         gamma           tau           phi
##  1.272088e+03  3.865935e-03  2.610718e-01 -2.609126e+00
## [1] "\n99% confidence intervals (phi adjusted):"
##               0.5 %        99.5 %
## A      7.370627e+02  1.807113e+03
## gamma  9.748425e-05  7.634386e-03
## tau    2.561627e-01  2.659809e-01
## phi   -2.865723e+00 -2.352529e+00
```

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

# 1) **fitted curve vs original curves**
```

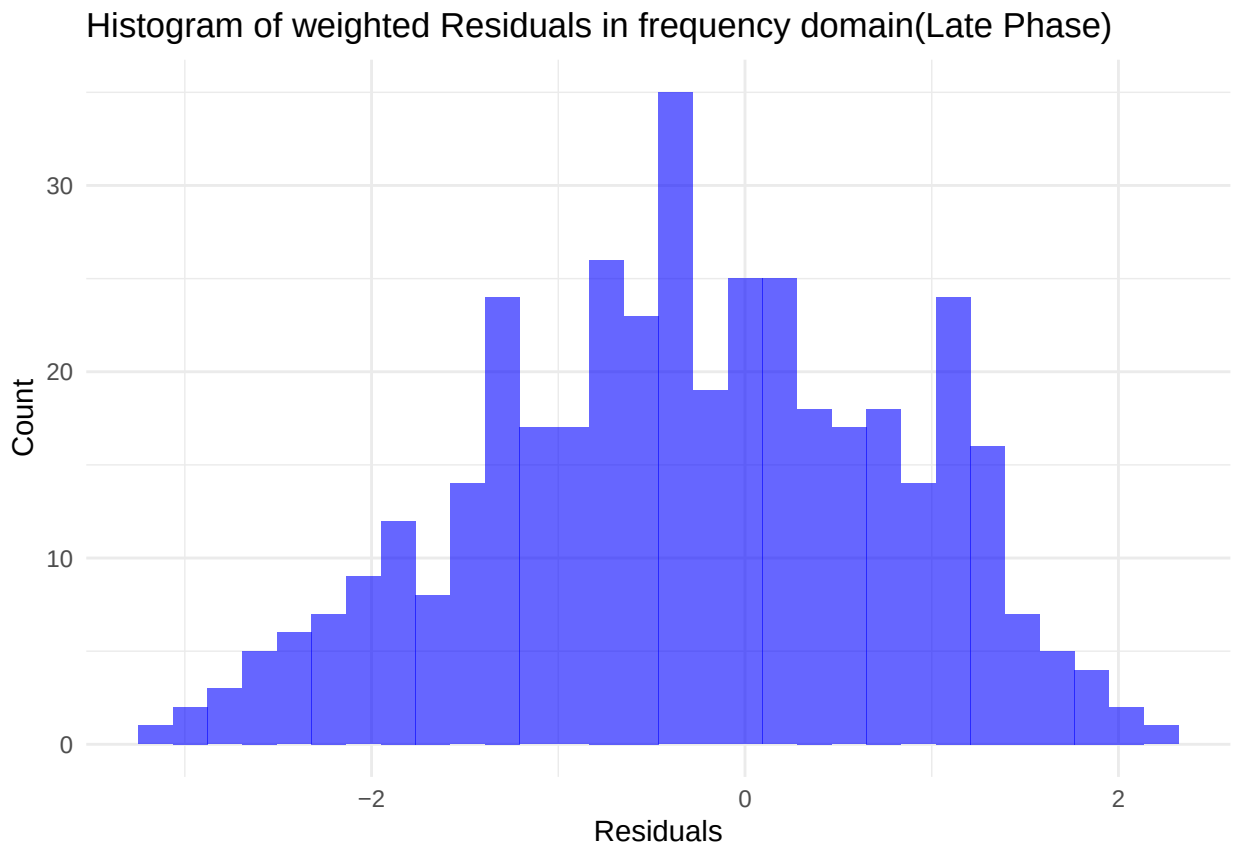
```r
y_fit_early <- model_function(time_early, params_early_fft_stacked["A"],
                                  params_early_fft_stacked["gamma"],

                                  params_early_fft_stacked["tau"],
                                  params_early_fft_stacked["phi"])

y_fit_late <- model_function(time_late, params_late_fft_stacked["A"],

                                 params_late_fft_stacked["gamma"],
                                 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 <- replicates_detrend %>%mutate(Time = df_MC$Time) %>%  pivot_longer(cols = c("X1", "X2", "

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")
```



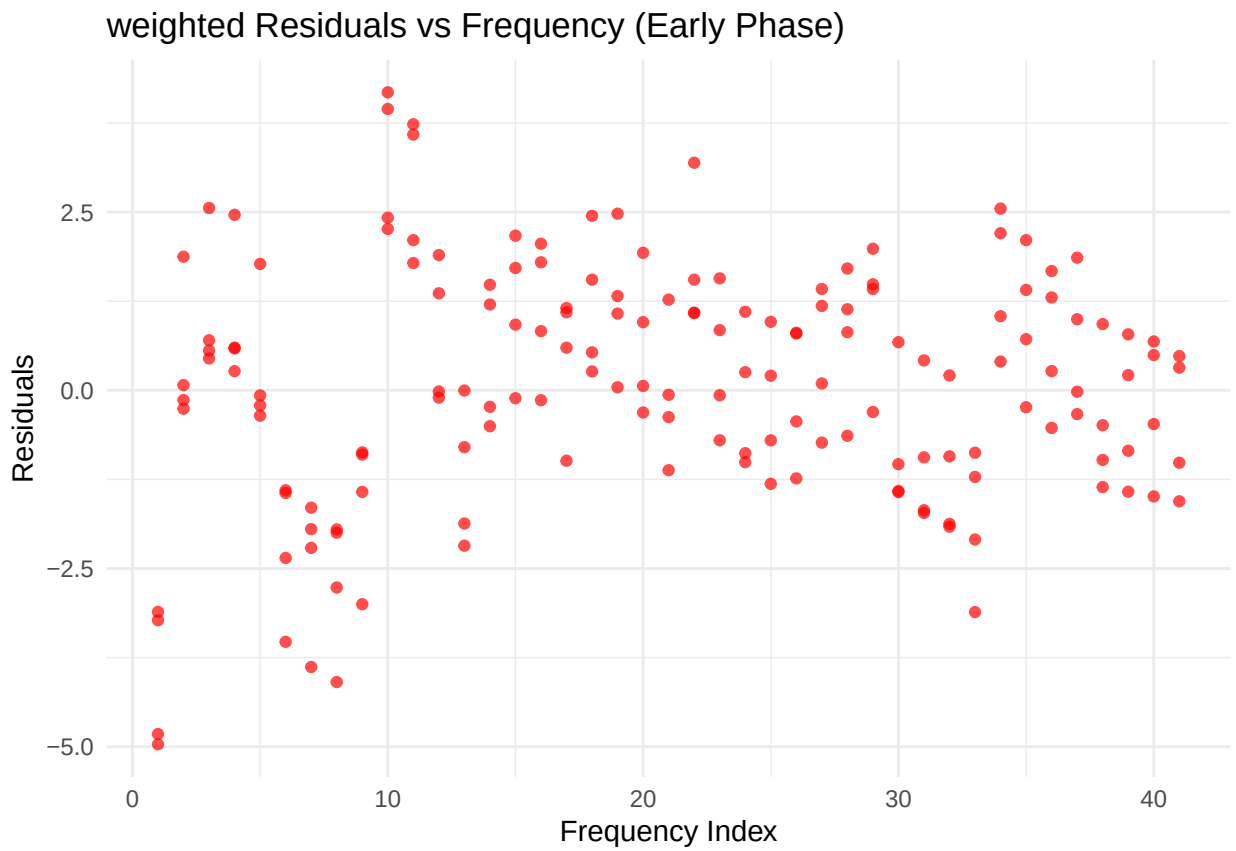

```r
# 2) **computes residuals**
residuals_early <- (fft_replicates_early - fft_extract(y_fit_early,N_early)) * sqrt(weights_early) #41*
residuals_late <- (fft_replicates_late - fft_extract(y_fit_late,N_late)) * sqrt(weights_late) # 101*4

# **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()
```



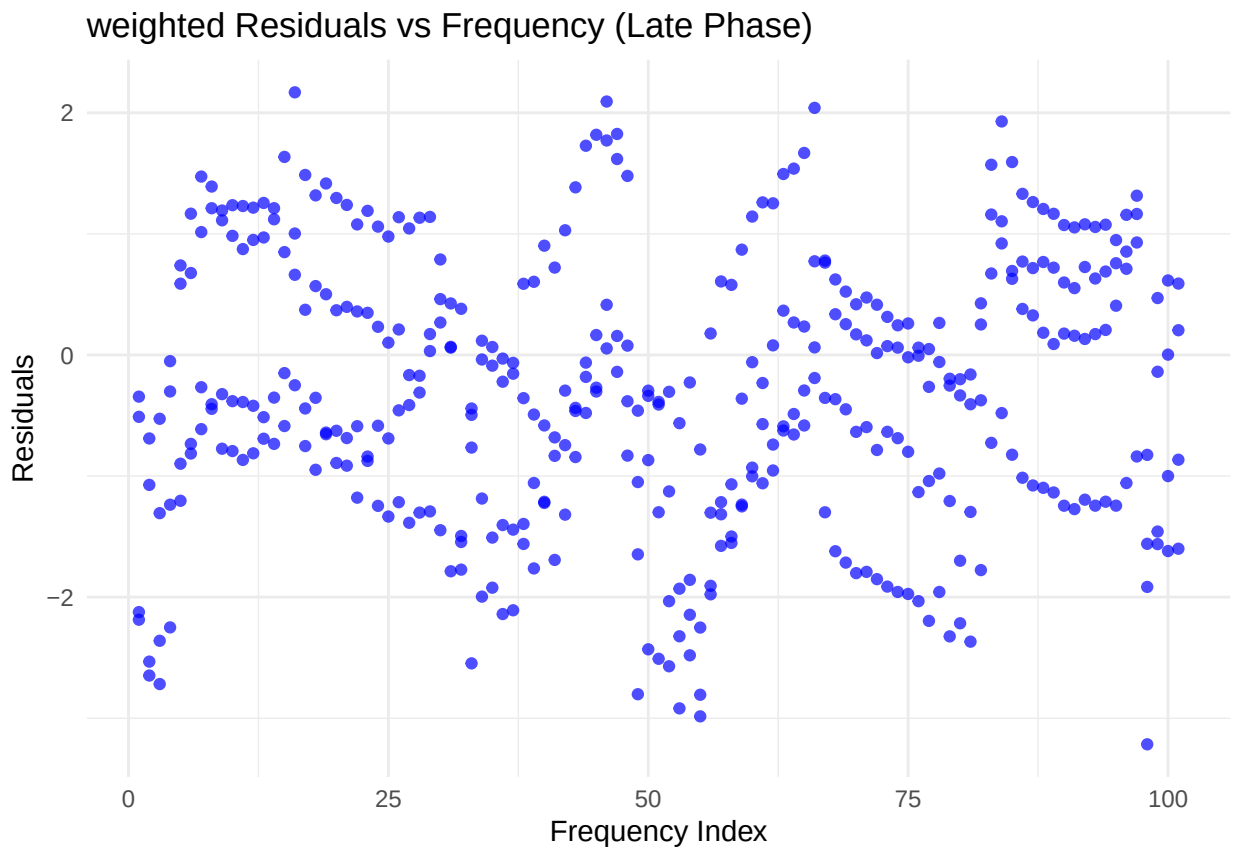

```r
# * plot Late res hist**
df_residuals_late <- data.frame(Residuals = as.vector(residuals_late))
ggplot(df_residuals_late, aes(x = Residuals)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.6) +
  labs(title = "Histogram of weighted Residuals in frequency domain(Late Phase)",
       x = "Residuals",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




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


# **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()
```



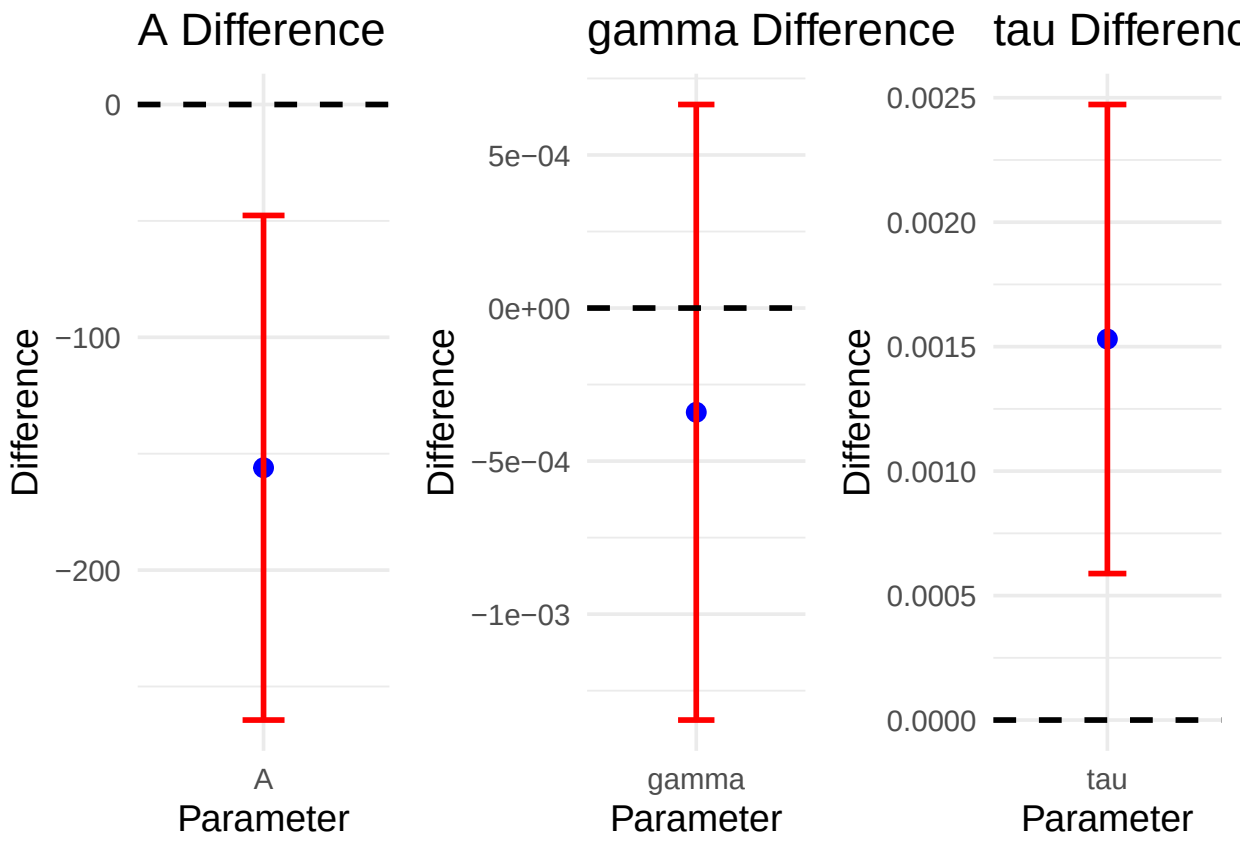

```
# 3) **T Wald hypothesis test**
sigma2_early_fft_weighted <- estimate_sigma2(y_early, time_early, params_early_fft_stacked)
sigma2_late_fft_weighted <- estimate_sigma2(y_late, time_late, params_late_fft_stacked)

covariance_early_fft_weighted <- compute_covariance(params_early_fft_stacked, time_early, y_early, sigma
covariance_late_fft_weighted <- compute_covariance(params_late_fft_stacked, time_late, y_late, sigma2_la

early_filtered_fft_weighted <- remove_phi(params_early_fft_stacked, covariance_early_fft_weighted)
late_filtered_fft_weighted <- remove_phi(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): 37.81257
```

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

```
## p-value: 3.096884e-08
```

```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)
```



FFT with constant frequency

```
fit_early_constant_fft_stacked <- try(
  nlsLM(
    formula = y_fft_stacked_early ~ stacked_fft_pred(
      t = time_early,
      A,gamma, tau, phi,
      N = N_early,
      n_reps = 4
    ),
    start = init_param_early,
    control = nls.lm.control(maxiter = 1000)  #
  ),
  silent = TRUE
)

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

  params_early_constant_fft_stacked <- coef(fit_early_constant_fft_stacked)
  names(params_early_constant_fft_stacked) <- c("A", "gamma", "tau", "phi")
  confint_early_constant_fft_stacked <- confint2(fit_early_constant_fft_stacked, level = 0.99, method =
    # adjust phi (control between -2pi and 2pi)
  params_early_constant_fft_stacked["phi"] <- ((params_early_constant_fft_stacked["phi"] + 2 * pi) %% (4
```

```r
  # adjust phi
  rownames(confint_early_constant_fft_stacked) <- c("A", "gamma", "tau", "phi")
  confint_early_constant_fft_stacked["phi", ] <- ((confint_early_constant_fft_stacked["phi", ] + 2 * pi

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

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

}
```

```
## [1] "Estimated parameters (stacked, early):"
##              A         gamma           tau           phi
##  1.357752e+03  5.710627e-03  2.611915e-01 -2.529480e+00
## [1] "\n99% confidence intervals (phi adjusted):"
##                0.5 %        99.5 %
## A      1.196492e+03  1.519012e+03
## gamma  3.348713e-03  8.072541e-03
## tau    2.589399e-01  2.634432e-01
## phi   -2.638159e+00 -2.420801e+00
```

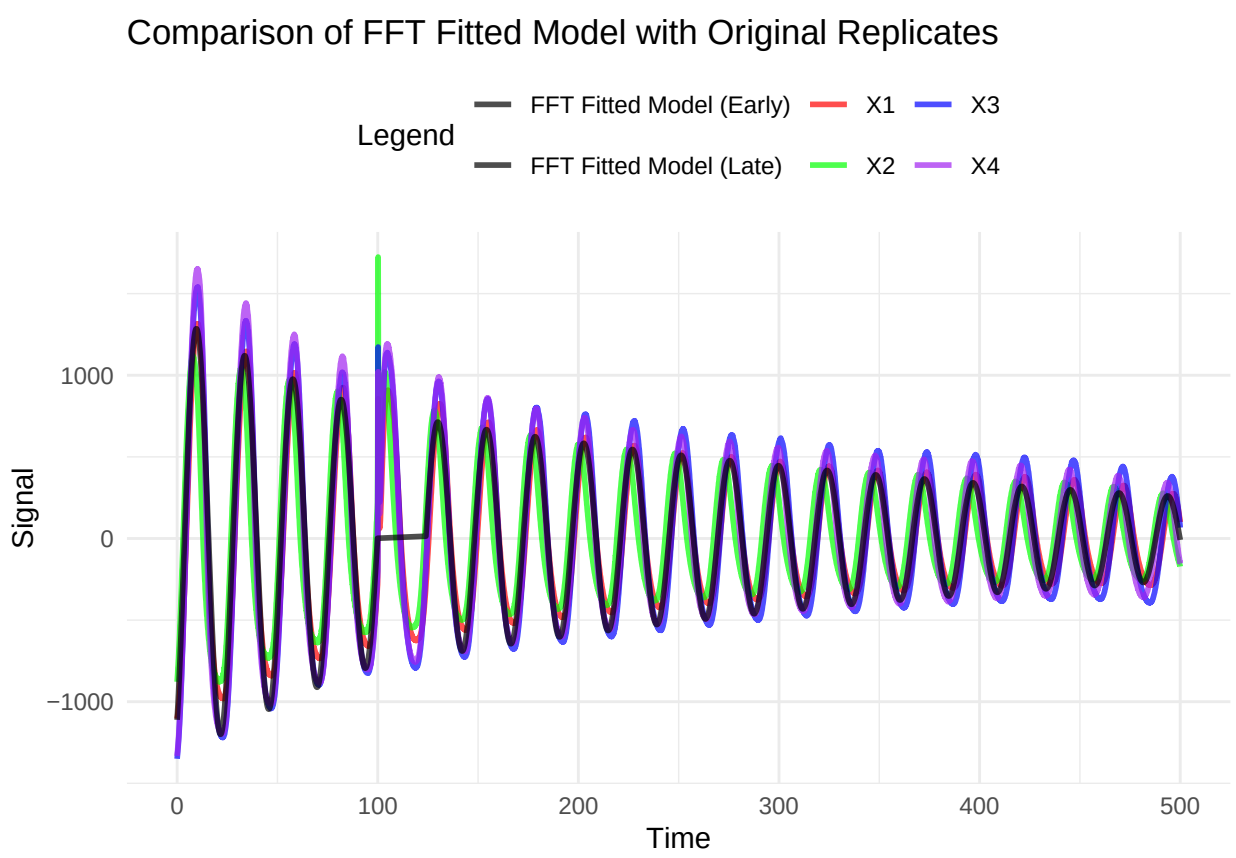
```r
fit_late_constant_fft_stacked <- try(
  nlsLM(
    formula = y_fft_stacked_late ~ stacked_fft_pred(
      t = time_late,
      A, gamma, tau, phi,
      N = N_late,
      n_reps = 4
    ),
    start = init_param_late,

    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_constant_fft_stacked)
  #
  params_late_constant_fft_stacked <- coef(fit_late_constant_fft_stacked)
  names(params_late_constant_fft_stacked) <- c("A", "gamma", "tau", "phi")
  confint_late_constant_fft_stacked <- confint2(fit_late_constant_fft_stacked, level = 0.99, method = '
    #  adjust phi
  rownames(confint_late_constant_fft_stacked) <- c("A", "gamma", "tau", "phi")

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


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

  print("Estimated parameters (stacked, late):")
```

```
  print(params_late_constant_fft_stacked)

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

}
```

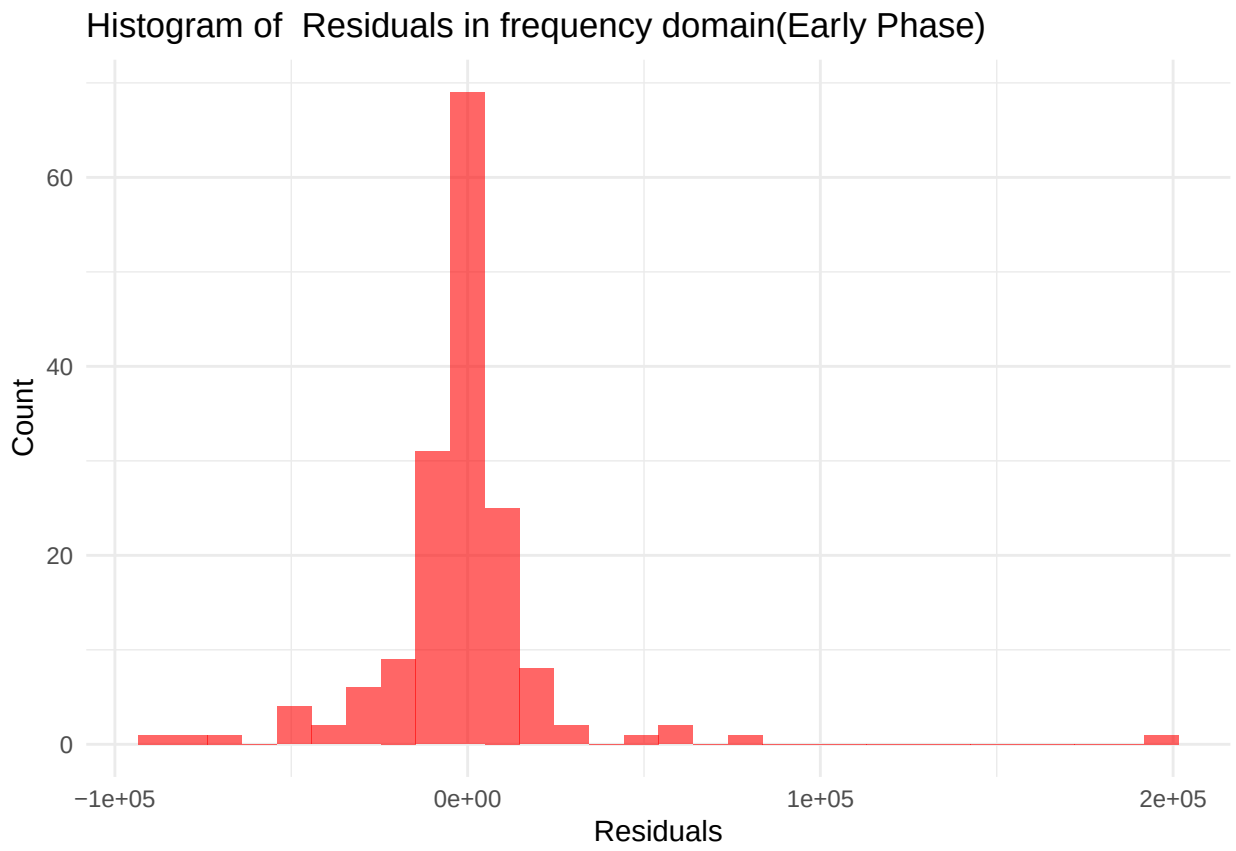
```
## [1] "Estimated parameters (stacked, late):"
##              A         gamma           tau           phi
##  1.021884e+03  2.767808e-03  2.590565e-01 -2.258430e+00
## [1] "\n99% confidence intervals (phi adjusted):"
##                0.5 %        99.5 %
## A     873.700785398  1.170067e+03
## gamma   0.002229668  3.305948e-03
## tau     0.258518969  2.595940e-01
## phi    -2.403551979 -2.113309e+00
```

plot and results

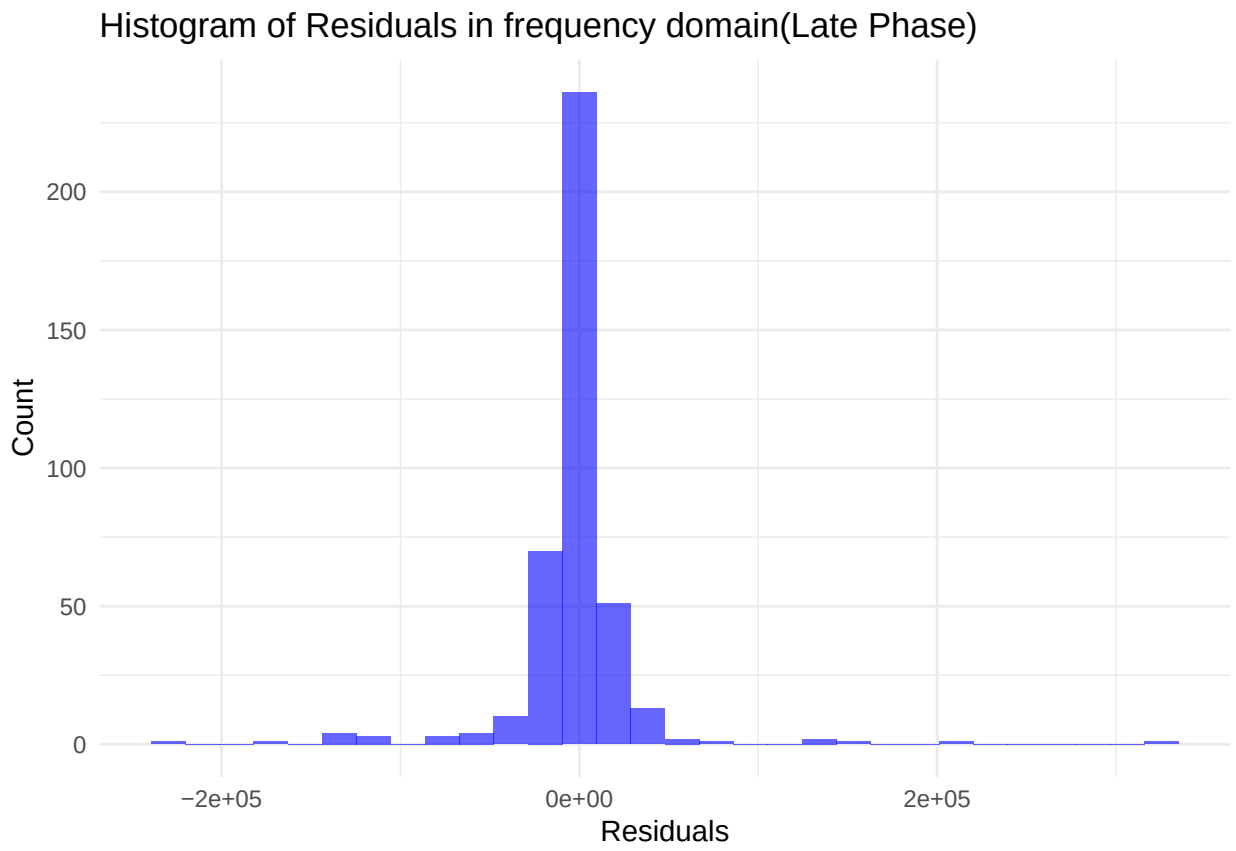
```
# 1) **fitted curve vs original curves**
y_fit_early <- model_function(time_early, params_early_constant_fft_stacked["A"],
                                  params_early_constant_fft_stacked["gamma"],

                                  params_early_constant_fft_stacked["tau"],
                                  params_early_constant_fft_stacked["phi"])

y_fit_late <- model_function(time_late, params_late_constant_fft_stacked["A"],

                                 params_late_constant_fft_stacked["gamma"],
                                 params_late_constant_fft_stacked["tau"],
                                 params_late_constant_fft_stacked["phi"])

df_fitted_early_constant <- data.frame(Time = time_early, Value = y_fit_early, Type = "FFT Fitted Model
df_fitted_late_constant <- data.frame(Time = time_late, Value = y_fit_late, Type = "FFT Fitted Model (La

df_plot_constant <- bind_rows(df_original %>% mutate(Type = Replicate), df_fitted_early_constant, df_fi

ggplot(df_plot_constant, 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)" = "black
  labs(title = "Comparison of FFT Fitted Model with Original Replicates",
       x = "Time", y = "Signal", color = "Legend") +
  theme_minimal() +
  theme(legend.position = "top")
```



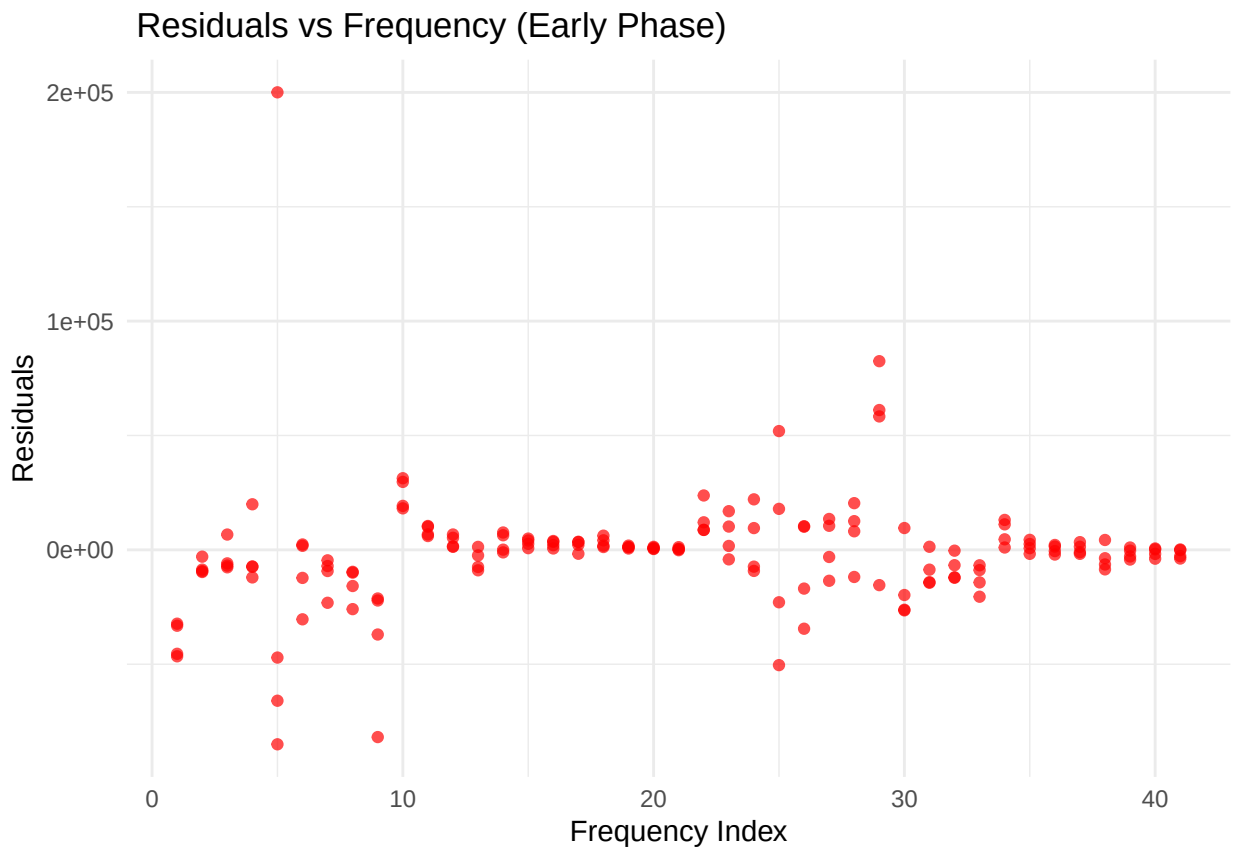

```r
# 2) **computes residuals**
residuals_early <- fft_replicates_early - fft_extract(y_fit_early,N_early)
residuals_late <- fft_replicates_late - fft_extract(y_fit_late,N_late)

# **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  Residuals in frequency domain(Early Phase)",
       x = "Residuals", y = "Count") +
  theme_minimal()
```



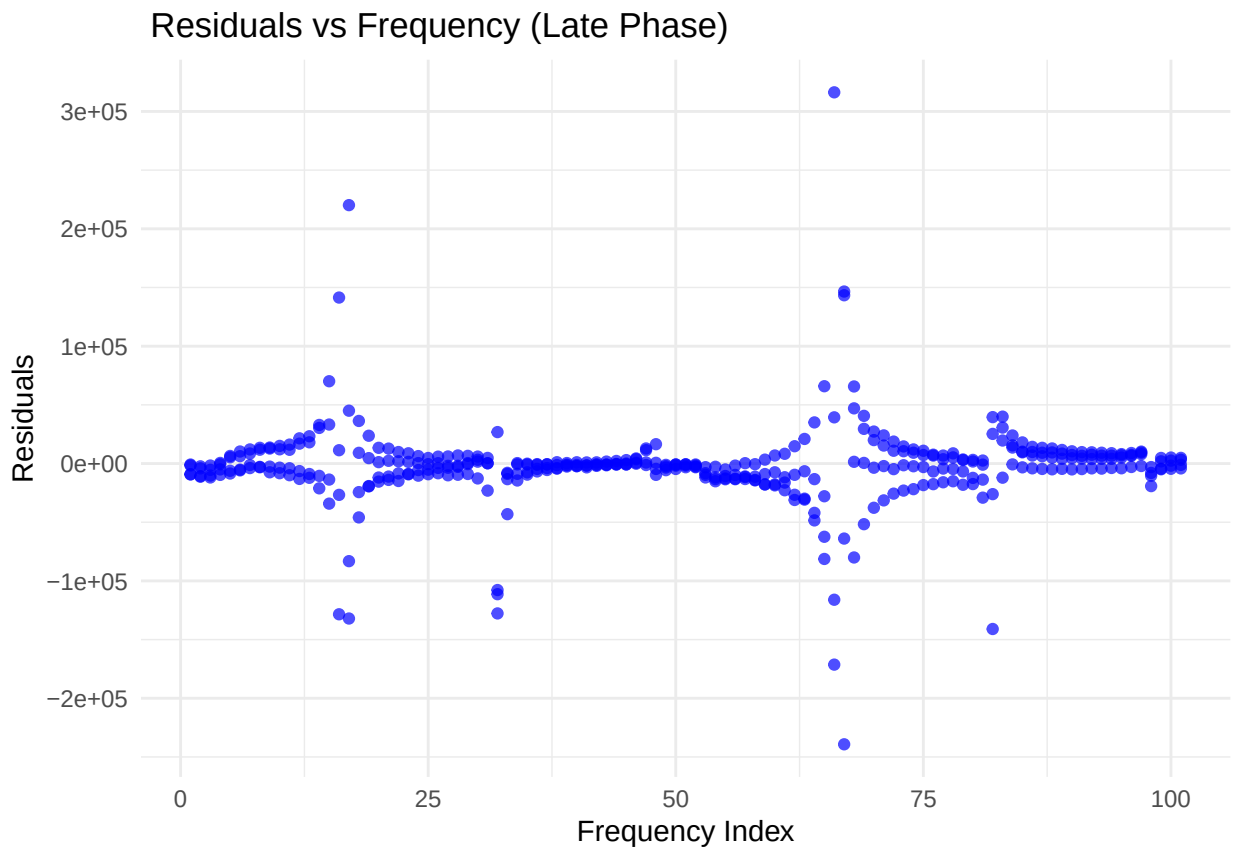

```
# * plot Late res hist**
df_residuals_late <- data.frame(Residuals = as.vector(residuals_late))
ggplot(df_residuals_late, aes(x = Residuals)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.6) +
  labs(title = "Histogram of Residuals in frequency domain(Late Phase)",
       x = "Residuals",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



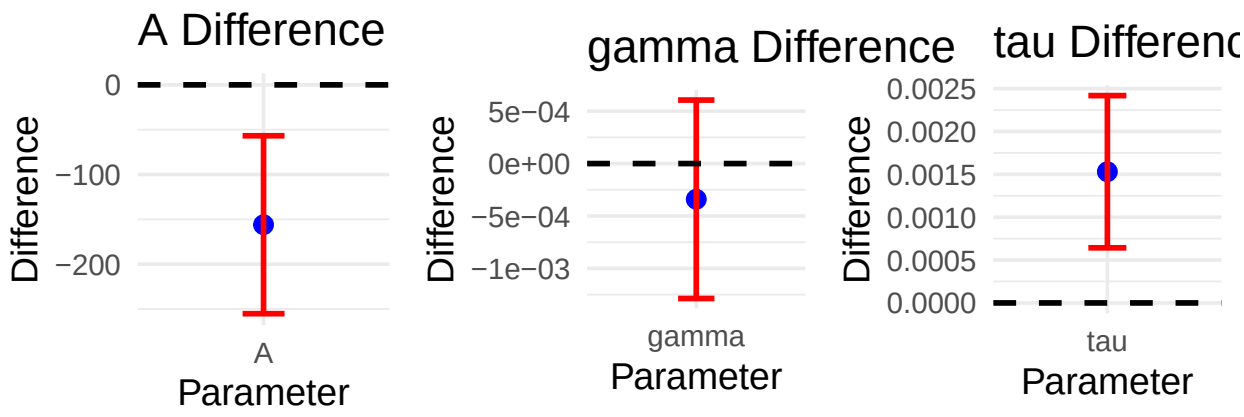

```
# 3) **T Wald hypothesis test**
sigma2_early_fft <- estimate_sigma2(y_early, time_early, params_early_constant_fft_stacked)
sigma2_late_fft <- estimate_sigma2(y_late, time_late, params_late_constant_fft_stacked)

covariance_early_fft <- compute_covariance(params_early_fft_stacked, time_early, y_early, sigma2_early_
covariance_late_fft <- compute_covariance(params_late_fft_stacked, time_late, y_late, sigma2_late_fft)

early_filtered_fft <- remove_phi(params_early_fft_stacked, covariance_early_fft)
late_filtered_fft <- remove_phi(params_late_fft_stacked, covariance_late_fft)

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

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

```
## Wald Test Statistic (FFT Weighted): 44.41118
```

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

```
## p-value: 1.234208e-09
```

```r
# 4) **99% CI**
covariance_early_fft <- early_filtered_fft$covariance
covariance_late_fft <- late_filtered_fft$covariance

#
fft_param_names <- names(early_filtered_fft$params)
fft_param_diff <- early_filtered_fft$params - late_filtered_fft$params
var_diff <- diag(covariance_early_fft + covariance_late_fft)  #
se_diff <- sqrt(var_diff)
Z_99 <- qnorm(0.995)
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
)

#  99% CI plot
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") +  # CI
    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)+
    theme(aspect.ratio = 1)

})

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



constant variance in time domain

```
y_early <- replicates_detrend[df_MC$Time < 100, ]
y_late <- replicates_detrend[df_MC$Time >= 124, ]
df_original <- replicates_detrend %>%mutate(Time = df_MC$Time) %>%  pivot_longer(cols = c("X1", "X2", "
long_early = df_original[df_original$Time< 100,]
long_late = df_original[df_original$Time>= 124,]

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_early <- nlsLM(
  Value ~ model_function(Time, A, gamma, tau, phi),
  start = list(
    A = as.numeric(init_param_early$A),
    gamma = as.numeric(init_param_early$gamma),
    tau = as.numeric(init_param_early$tau),
    phi = as.numeric(init_param_early$phi)

  ),
  data = long_early,
  lower = c(500, 0.001, 0.1, -pi),
  upper = c(2000, 0.05, 1.5, pi)
)

fit_late <- nlsLM(
  Value ~ model_function(Time, A, gamma, tau, phi),
  start = list(
```

```
    A = as.numeric(init_param_late$A),
    gamma = as.numeric(init_param_late$gamma),
    tau = as.numeric(init_param_late$tau),
    phi = as.numeric(init_param_late$phi)

  ),
  data = long_late,
  lower = c(500, 0.001, 0.1, -pi),
  upper = c(2000, 0.05, 1.5, pi)
)


# Extract estimated parameters for early and late phases
params_early <- coef(fit_early)
params_late <- coef(fit_late)



summary(fit_early)
```

```
##
## Formula: Value ~ model_function(Time, A, gamma, tau, phi)
##
## Parameters:
##         Estimate Std. Error t value Pr(>|t|)
## A      1.372e+03  1.225e+01  111.99   <2e-16 ***
## gamma  5.972e-03  1.799e-04   33.19   <2e-16 ***
## tau    2.610e-01  1.688e-04 1546.09   <2e-16 ***
## phi   -2.519e+00  8.136e-03 -309.67   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 226.7 on 3996 degrees of freedom
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 1.49e-08
```

```
summary(fit_late)
```

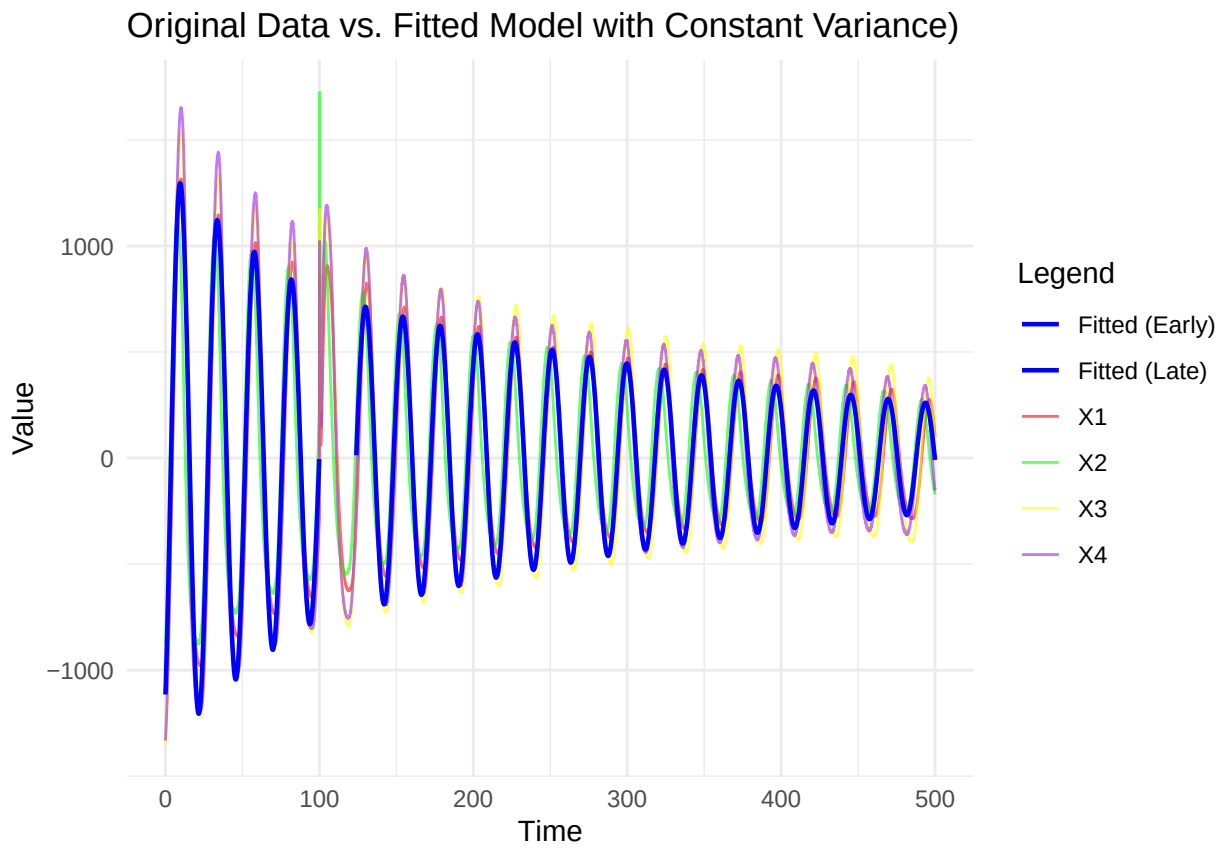
```
##
## Formula: Value ~ model_function(Time, A, gamma, tau, phi)
##
## Parameters:
##         Estimate Std. Error t value Pr(>|t|)
## A      1.022e+03  9.369e+00  109.13   <2e-16 ***
## gamma  2.769e-03  3.401e-05   81.43   <2e-16 ***
## tau    2.591e-01  3.395e-05 7630.35   <2e-16 ***
## phi   -2.260e+00  9.151e-03 -246.98   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 135.8 on 15040 degrees of freedom
```

```
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 1.49e-08
```

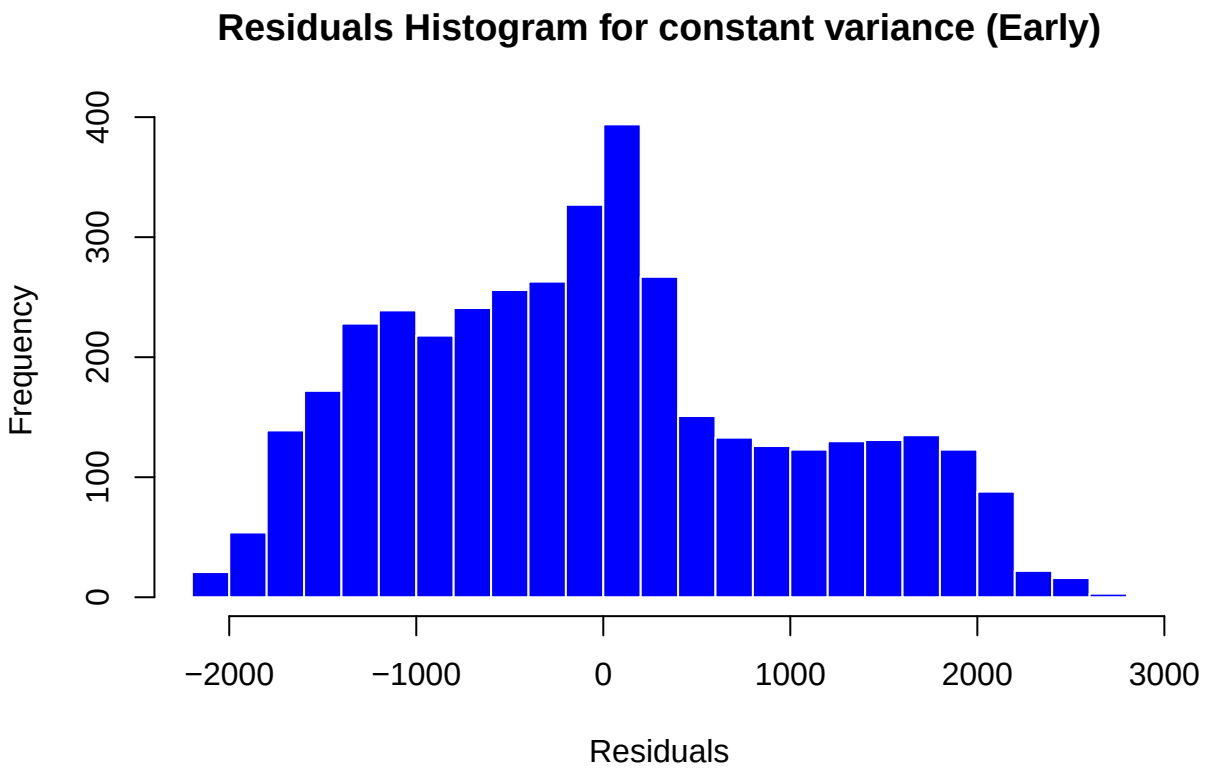
```
params_early
```

```
##             A         gamma           tau           phi
## 1372.17972060    0.00597157    0.26100972   -2.51936866
```

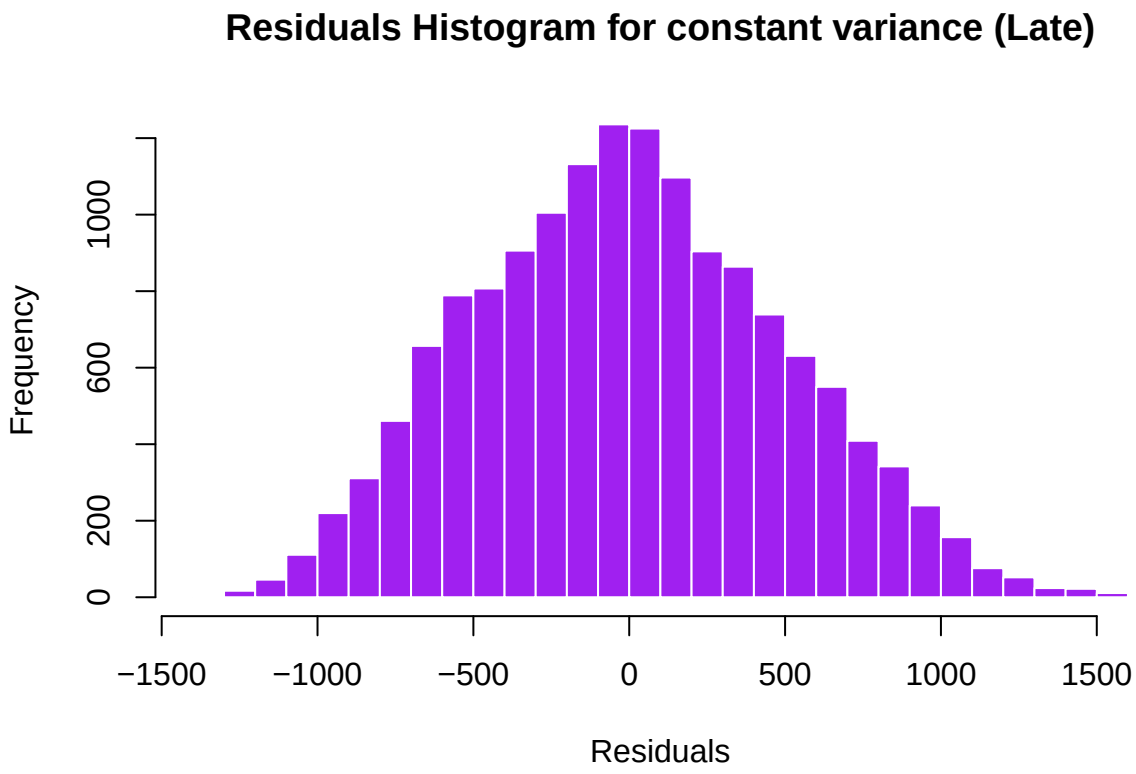
```
# Compute fitted values for early phase
y_fit_early <- model_function(time_early, params_early["A"],
                                  params_early["gamma"], params_early["tau"],
                                   params_early["phi"])

# Compute fitted values for late phase
y_fit_late <- model_function(time_late, params_late["A"], params_late["gamma"],

                                 params_late["tau"], params_late["phi"])

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

# Plot original data vs fitted curves
ggplot(df_original, 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 with Constant Variance)",
       x = "Time", y = "Value", color = "Legend") +
  theme_minimal()
```



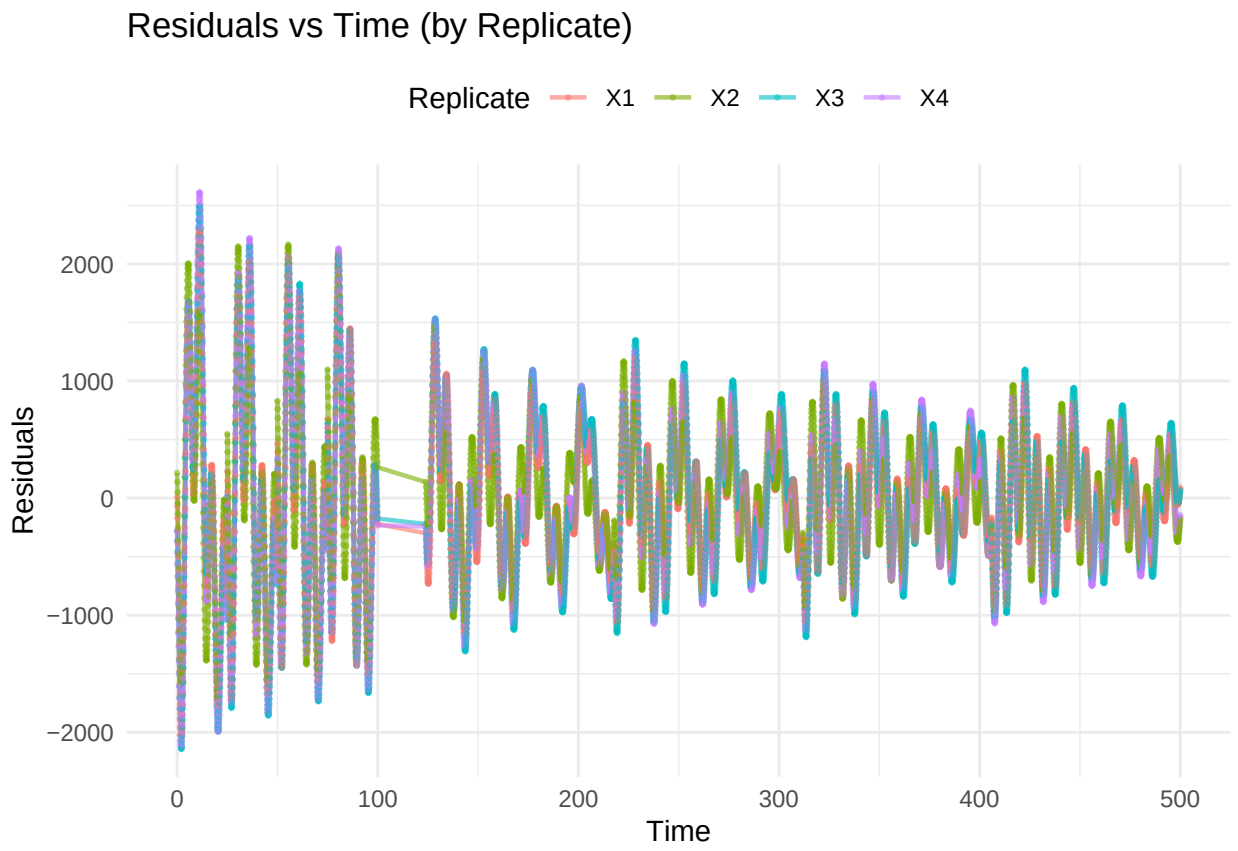

```r
# Compute residuals
residuals_early <- long_early$Value - y_fit_early
residuals_late <- long_late$Value - y_fit_late


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




```r
hist(residuals_late, breaks = 30, col = "purple", border = "white",
     main = "Residuals Histogram for constant variance (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_early$Time, Residual = residuals_early, Phase = "Early", Replicate = long_early
  data.frame(Time = long_late$Time, Residual = residuals_late, Phase = "Late", Replicate = long_late$Rep
)
# 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")
```



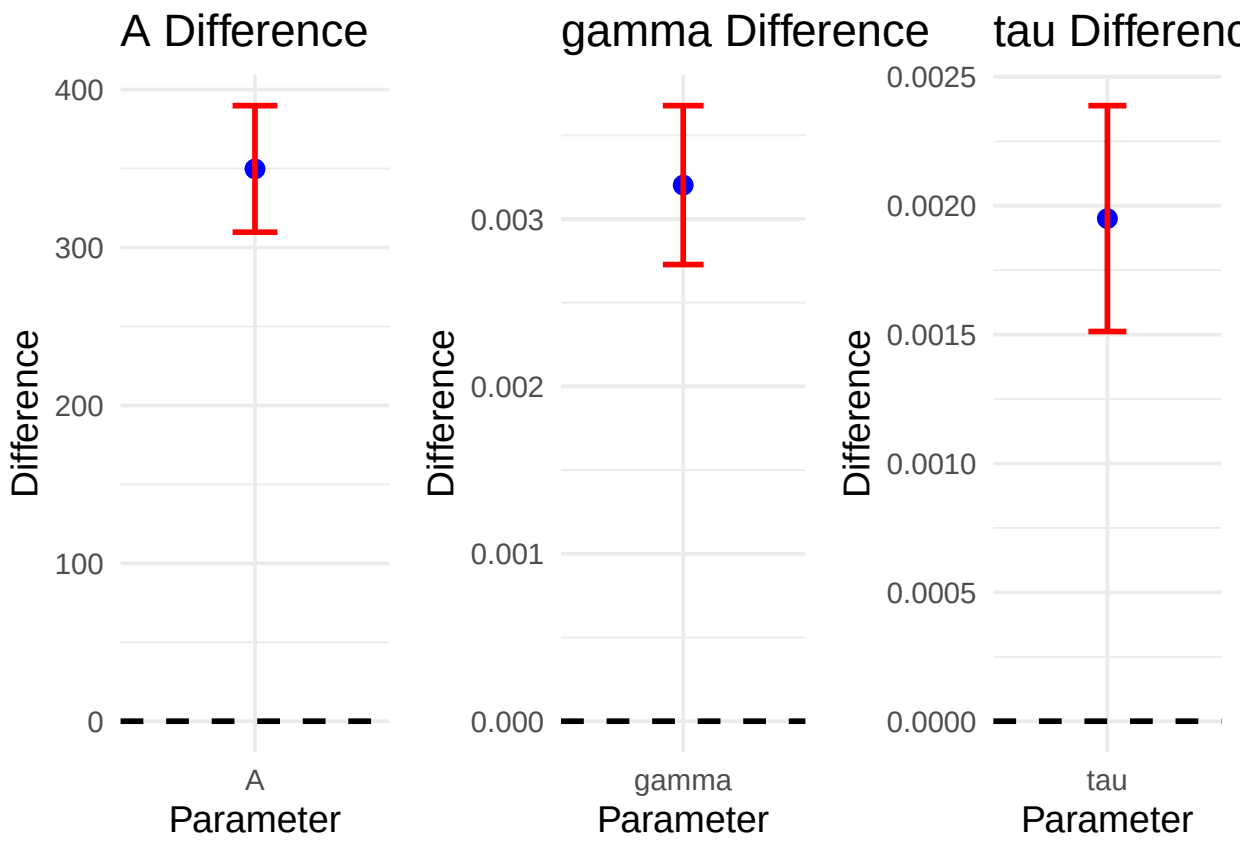

```
# Compute sigma² for early and late phases
sigma2_early <- estimate_sigma2(long_early$Value, long_early$Time, params_early)
sigma2_late <- estimate_sigma2(long_late$Value, long_late$Time, params_late)

# Compute covariance matrices
covariance_early <- compute_covariance(params_early, long_early$Time, long_early$Value, sigma2_early)
covariance_late <- compute_covariance(params_late, long_late$Time, long_late$Value, sigma2_late)  #from

# Remove phi parameter from estimated parameters and covariance matrix
early_filtered <- remove_phi(params_early, covariance_early)
late_filtered <- remove_phi(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: 649.6444
```

```
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)
```



time dependent variance

```
weights_early <- bayesian_weights[1:length(time_early)]
weights_late  <- bayesian_weights[(length(bayesian_weights)-length(time_late)):(length(bayesian_weights)

weights_early_expanded <- rep(weights_early, each = 4)
weights_late_expanded <- rep(weights_late, each = 4)
fit_early <- nlsLM(
  Value ~ model_function(Time, A, gamma, tau, phi),
  start = list(
    A = as.numeric(init_param_early$A),
    gamma = as.numeric(init_param_early$gamma),
    tau = as.numeric(init_param_early$tau),
    phi = as.numeric(init_param_early$phi)
  ),
  weights = weights_early_expanded,
  data = long_early,
  lower = c(500, 0.001, 0.1, -pi),
  upper = c(2000, 0.05, 1.5, pi)
)

fit_late <- nlsLM(
  Value ~ model_function(Time, A, gamma, tau, phi),
  start = list(
    A = as.numeric(init_param_late$A),
    gamma = as.numeric(init_param_late$gamma),
    tau = as.numeric(init_param_late$tau),
    phi = as.numeric(init_param_late$phi)

  ),
  weights = weights_late_expanded,
  data = long_late,
  lower = c(500, 0.001, 0.1, -pi),
  upper = c(2000, 0.05, 1.5, pi)
)


# Extract estimated parameters for early and late phases
params_early <- coef(fit_early)
params_late <- coef(fit_late)



summary(fit_early)
```

```
##
## Formula: Value ~ model_function(Time, A, gamma, tau, phi)
##
## Parameters:
##         Estimate Std. Error t value Pr(>|t|)
## A      1.294e+03  8.601e+00  150.45   <2e-16 ***
## gamma  6.040e-03  1.167e-04   51.77   <2e-16 ***
## tau    2.611e-01  1.678e-04 1556.01   <2e-16 ***
```

```
## phi   -2.580e+00  9.050e-03 -285.05   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.199 on 3996 degrees of freedom
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 1.49e-08
```

```
summary(fit_late)
```

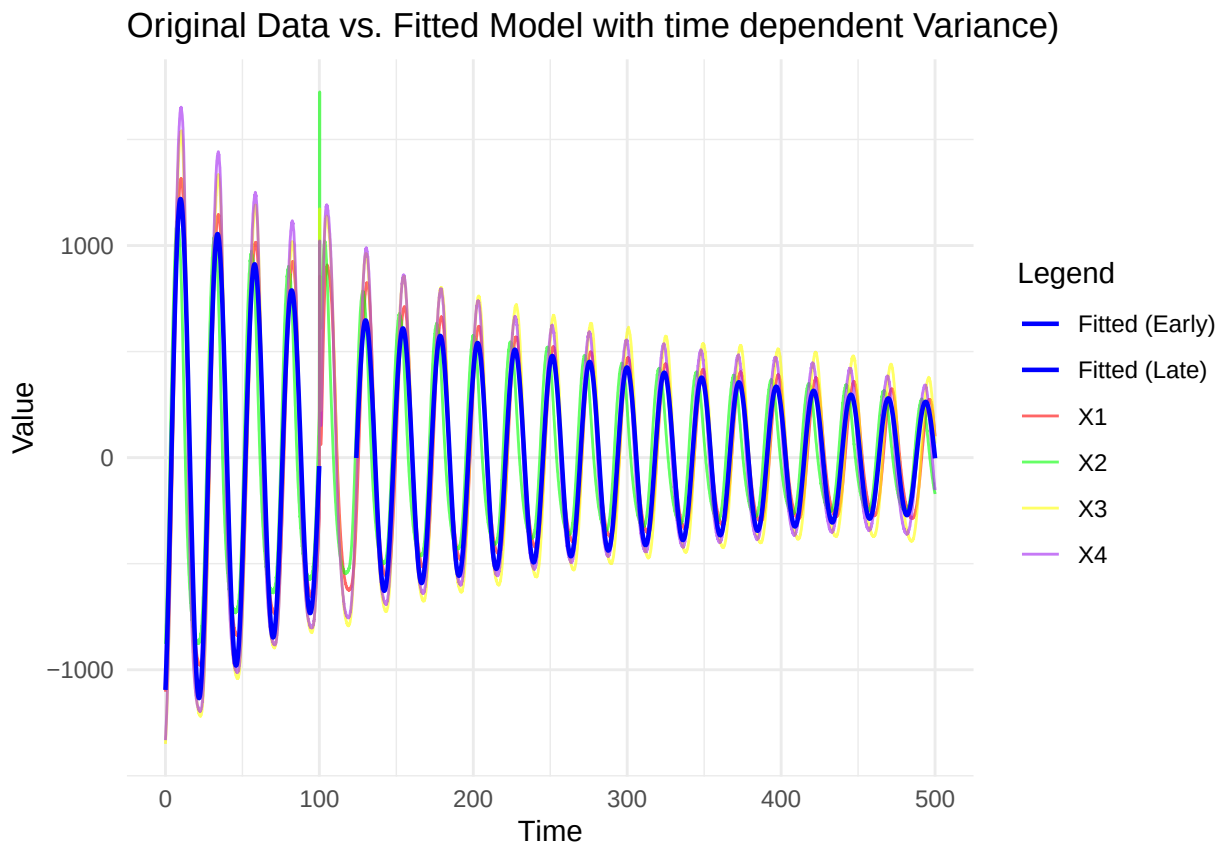
```
##
## Formula: Value ~ model_function(Time, A, gamma, tau, phi)
##
## Parameters:
##         Estimate Std. Error t value Pr(>|t|)
## A      8.925e+02  5.494e+00   162.5   <2e-16 ***
## gamma  2.473e-03  2.199e-05   112.5   <2e-16 ***
## tau    2.591e-01  3.336e-05  7766.6   <2e-16 ***
## phi   -2.281e+00  9.478e-03  -240.6   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8078 on 15040 degrees of freedom
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 1.49e-08
```

```
params_early
```

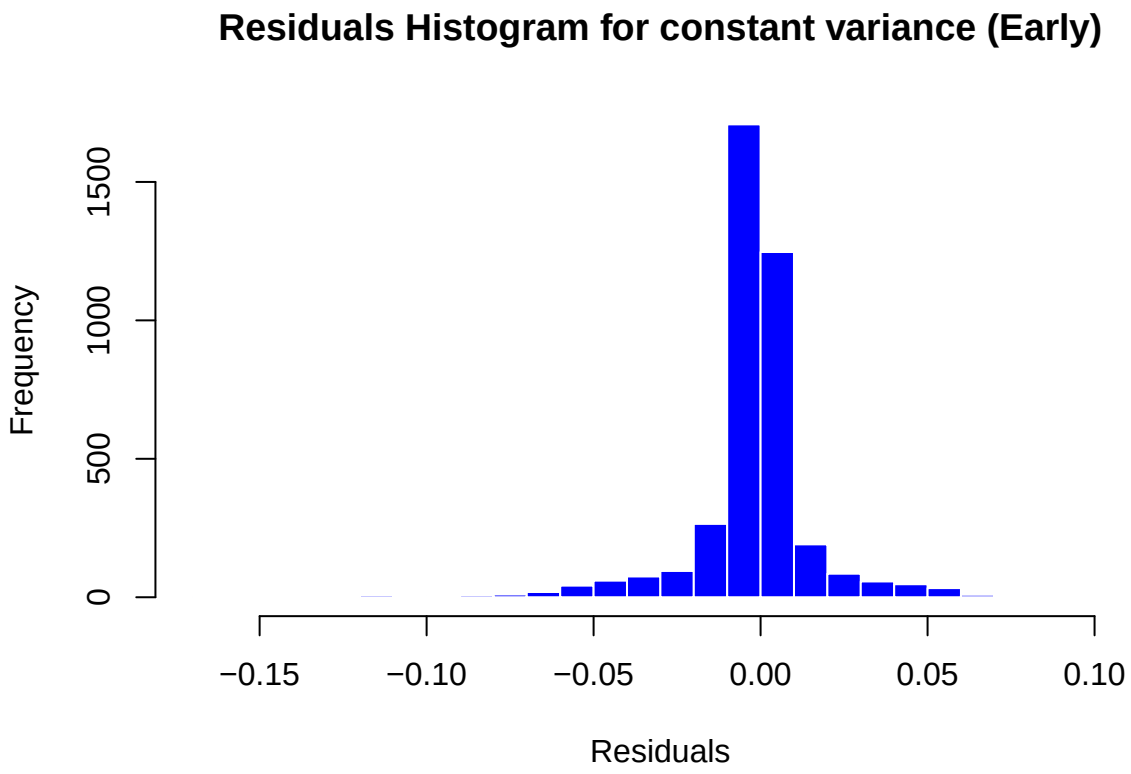
```
##             A         gamma           tau           phi
##  1.294092e+03  6.040278e-03  2.611164e-01 -2.579793e+00
```

```
# Compute fitted values for early phase
y_fit_early <- model_function(time_early, params_early["A"],
                              params_early["gamma"], params_early["tau"],
                               params_early["phi"])

# Compute fitted values for late phase
y_fit_late <- model_function(time_late, params_late["A"], params_late["gamma"],

                             params_late["tau"], params_late["phi"])

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

# Plot original data vs fitted curves
ggplot(df_original, 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) +
```

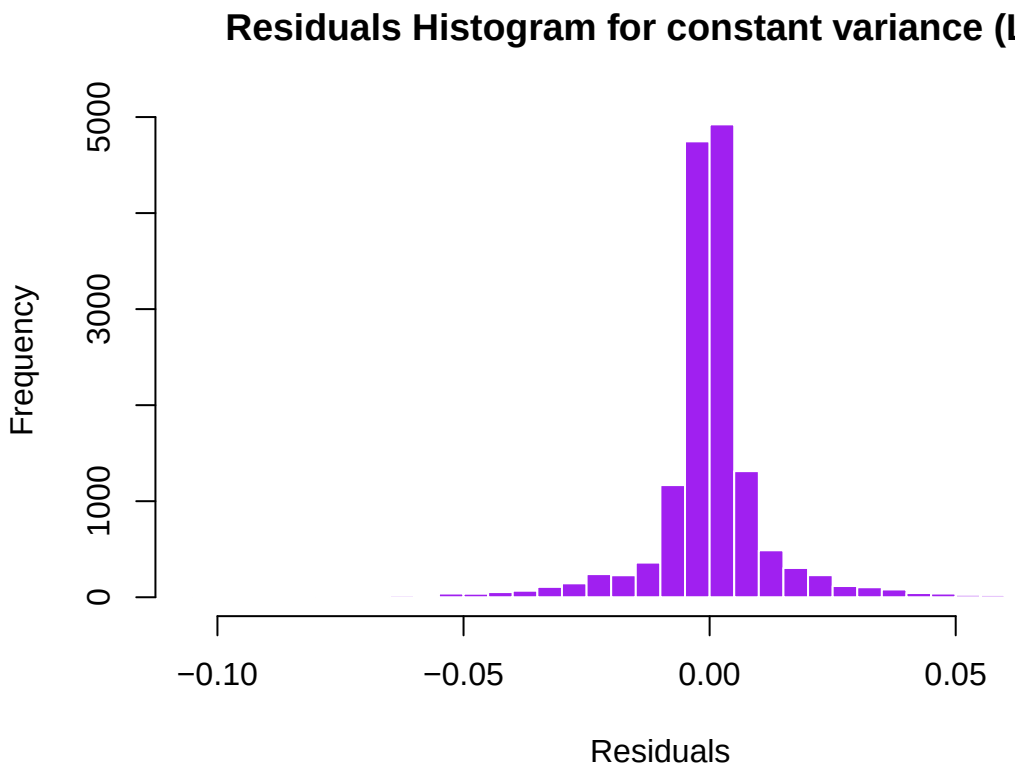
```
  scale_color_manual(values = c("X1" = "red", "X2" = "green", "X3" = "yellow", "X4" = "purple",
                                "Fitted (Early)" = "blue", "Fitted (Late)" = "blue")) +
  labs(title = "Original Data vs. Fitted Model with time dependent Variance)",
       x = "Time", y = "Value", color = "Legend") +
  theme_minimal()
```



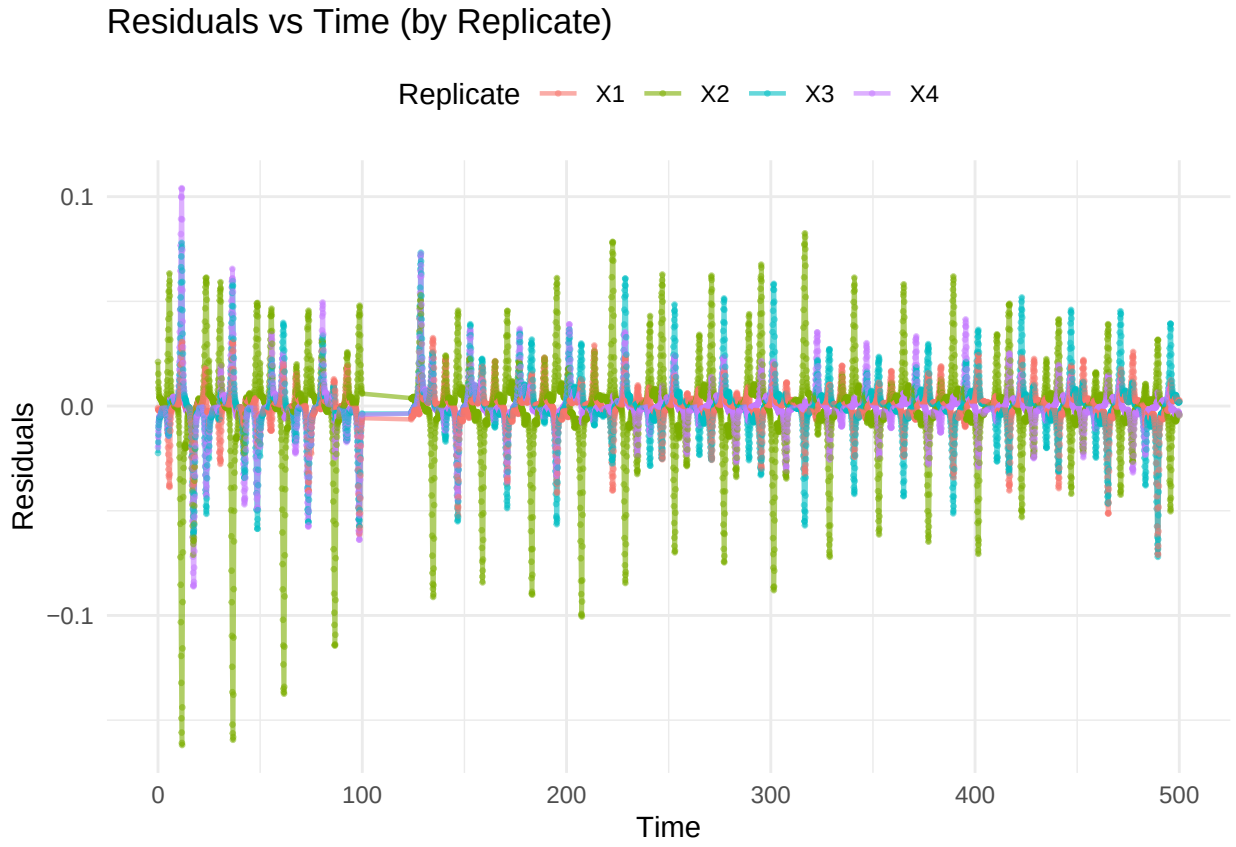

```
# Compute residuals for early and late phase using the function
residuals_early <- compute_residuals(long_early$Value, long_early$Time, params_early) *weights_early
residuals_late <- compute_residuals(long_late$Value, long_late$Time, params_late)*weights_late

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




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




```r
# Create a combined dataframe for plotting
df_residuals <- bind_rows(
  data.frame(Time = long_early$Time, Residual = residuals_early, Phase = "Early", Replicate = long_early
  data.frame(Time = long_late$Time, Residual = residuals_late, Phase = "Late", Replicate = long_late$Rep
)
# 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")
```



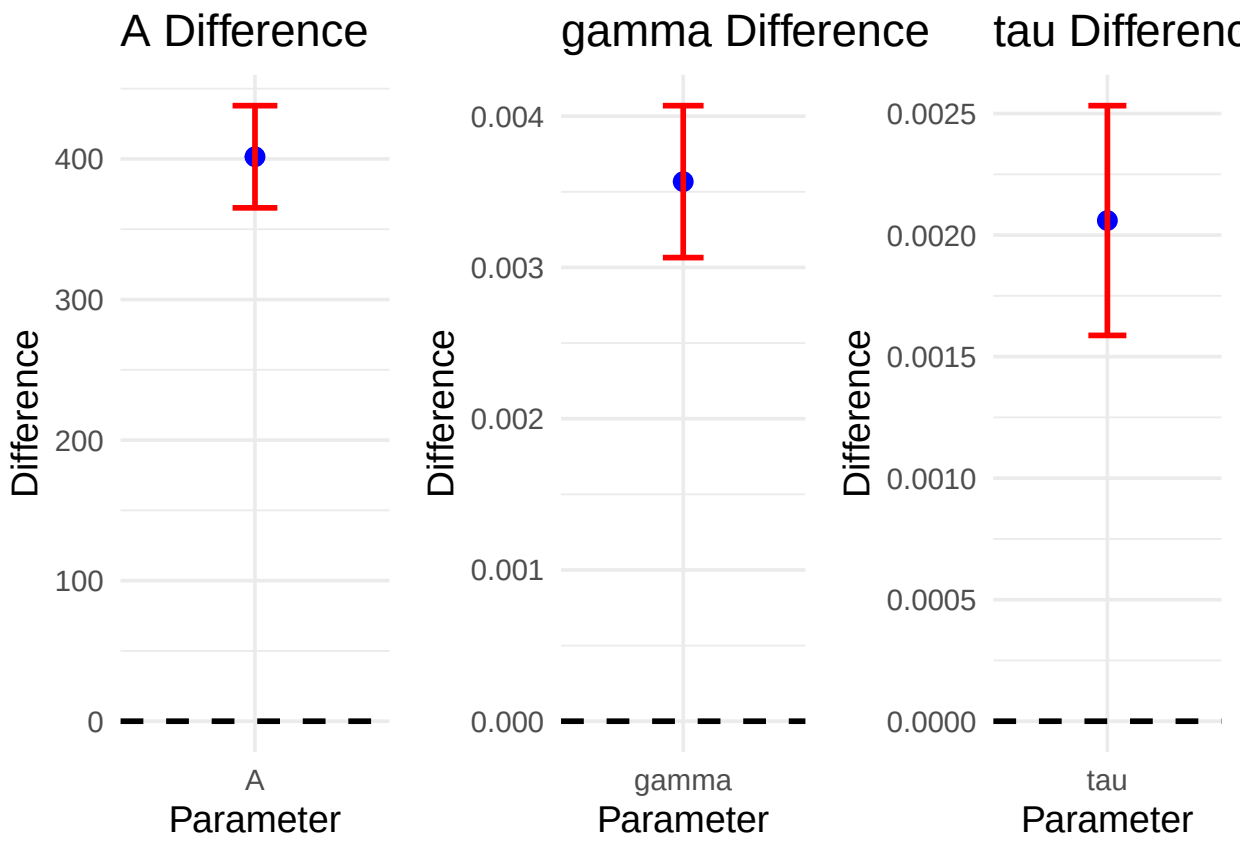

```
# Compute sigma² for early and late phases
sigma2_early <- estimate_sigma2(long_early$Value, long_early$Time, params_early)
sigma2_late <- estimate_sigma2(long_late$Value, long_late$Time, params_late)

# Compute covariance matrices
covariance_early <- compute_covariance(params_early, long_early$Time, long_early$Value, sigma2_early)
covariance_late <- compute_covariance(params_late, long_late$Time, long_late$Value, sigma2_late)  #from

# Remove phi parameter from estimated parameters and covariance matrix
early_filtered <- remove_phi(params_early, covariance_early)
late_filtered <- remove_phi(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: 999.1333
```

```
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)
```



“‘