Appendix: Non Linear Regression Project (Corrected for t_orig)

Mainak Sarkar | 230619

Loading the Data Set

```
# Loading data
data <- read.table("set-49.dat")
head(data)

# Extract time and observed values
t_orig <- as.numeric(unlist(data[1]))  # Original time (1-60)
y1 <- as.numeric(unlist(data[2]))      # Observed y(t) values
n <- length(y1)                         # Get n from y1
```

Model 1: $y(t) = \alpha_0 + \alpha_1 e^{\beta_1 t} + \alpha_2 e^{\beta_2 t} + \epsilon(t)$

1.1 Fitting

Method changed to Non-Linear Least Squares (NLS) using `optim` for stability with original t values

```
# --- NLS implementation for Model 1 ---

# Use original time values
tt_m1 <- t_orig
yy_m1 <- y1

# Model 1 function: p = c(a0, a1, b1, a2, b2)
f_model_1 <- function(p, t) {
  a0 <- p[1]; a1 <- p[2]; b1 <- p[3]; a2 <- p[4]; b2 <- p[5]
  return(a0 + a1 * exp(b1 * t) + a2 * exp(b2 * t))
}

# SSE cost function for Model 1
sse_1 <- function(p) {
  yhat <- f_model_1(p, tt_m1)
  if (any(!is.finite(yhat))) return(1e10) # Penalize explosions
  sum((yy_m1 - yhat)^2)
}

# Initial guesses
init_1 <- c(1.0, 0.2, 0.001, 2.0, -0.3)

# Run optim
fit_1 <- optim(init_1, sse_1, method = "L-BFGS-B",
               control = list(maxit = 2000, fnscale = 1e-1))

params_1 <- fit_1$par
parameter_estimates_1 <- params_1

alpha0_1 <- params_1[1]
alpha1_1 <- params_1[2]
beta1_1 <- params_1[3]
alpha2_1 <- params_1[4]
beta2_1 <- params_1[5]

fitted_values_m1 <- f_model_1(params_1, tt_m1)
residuals_m1 <- yy_m1 - fitted_values_m1
```

1.2 Checking the Model Accuracy for Model 1

```
# Calculate model accuracy metrics
rmse_1 <- sqrt(mean((residuals_m1)^2, na.rm = TRUE))
rss_1 <- sum((residuals_m1)^2)
tss_1 <- sum((y1 - mean(y1))^2)
r_squared_1 <- 1 - (rss_1 / tss_1)
k_1 <- 5 # Number of parameters (alpha0, alpha1, beta1, alpha2, beta2)
adjusted_r_squared_1 <- 1 - ((1 - r_squared_1) * (n - 1) / (n - k_1 - 1))
```

1.3 Calculating the Jacobian Matrix, FIM and CI

```
# Define the Jacobian of the model (partial derivatives)
jacobian_1 <- function(t, alpha0, alpha1, beta1, alpha2, beta2) {
  d_alpha0 <- 1
  d_alpha1 <- exp(beta1 * t)
  d_beta1 <- alpha1 * t * exp(beta1 * t)
  d_alpha2 <- exp(beta2 * t)
  d_beta2 <- alpha2 * t * exp(beta2 * t)

  return(c(d_alpha0, d_alpha1, d_beta1, d_alpha2, d_beta2))
}

# Fisher Information matrix calculation
fisher_information <- function(t_vec, params, jac_func, sigma2) {
  n_obs <- length(t_vec)
  n_params <- length(params)
  J <- matrix(0, nrow = n_obs, ncol = n_params)

  for (i in 1:n_obs) {
    J[i, ] <- jac_func(t_vec[i], params[1], params[2], params[3], params[4], params[5])
  }

  # FIM is (1/sigma2) * (J'J)
  fisher_matrix <- (1 / (n_obs * sigma2)) * (t(J) %*% J)

  # Regularization (if required)
  lambda <- 1e-6
  fisher_matrix_regularized <- fisher_matrix + lambda * diag(n_params)

  return(fisher_matrix_regularized)
}

# Confidence Intervals calculation
confidence_intervals <- function(fisher_matrix, parameter_estimates, alpha = 0.05) {
  # Inverse Fisher Information Matrix (for variance)
  fisher_inv <- ginv(fisher_matrix)

  # Standard errors
  se <- sqrt(diag(fisher_inv))

  # Z value for 95% confidence
  z_value <- qnorm(1 - alpha / 2)

  # Confidence intervals
  ci_lower <- parameter_estimates - z_value * se
  ci_upper <- parameter_estimates + z_value * se
```

```
  return(list(lower = ci_lower, upper = ci_upper, se = se))
}

sigma2_1 <- rss_1 / (n - k_1)

# Calculate Fisher Information Matrix for the model parameters
fisher_matrix_1 <- fisher_information(tt_m1, parameter_estimates_1, jacobian_1, sigma2_1)

# Calculate Confidence Intervals
ci_1 <- confidence_intervals(fisher_matrix_1, parameter_estimates_1)

# Print the results
cat("--- Model 1 Results ---\n")
cat("Parameters (alpha0, alpha1, beta1, alpha2, beta2):\n")
print(parameter_estimates_1)
cat("\nFisher Information Matrix:\n")
print(fisher_matrix_1)
cat("\nConfidence Intervals:\n")
print(ci_1)
```

1.4 Plots

```
# Set up the plotting area to have two rows
par(mfrow = c(2, 2), mar = c(4, 4, 2, 1))

# Residuals plot
plot(t_orig, residuals_m1, main = "1.6: Residuals vs. Time (Model 1)",
     xlab = "Time (Original)", ylab = "Residuals", col = "black", pch = 19)
abline(h = 0, col = "red")

# Q-Q plot of residuals
qqnorm(residuals_m1, main = "1.7(a): Q-Q Plot of Residuals (Model 1)")
qqline(residuals_m1, col = "red")

# Histogram of residuals
hist(residuals_m1, breaks = 10, main = "1.7(b): Histogram of Residuals (Model 1)",
     xlab = "Residuals", col = "lightblue", border = "black")

# Fitted values plot
plot(t_orig, y1, type = "l", col = "blue", main = "1.8: Observed vs. Fitted Data (Model 1)",
     xlab = "Time (Original)", ylab = "Values", lwd = 2)
lines(t_orig, fitted_values_m1, col = "red", lwd = 2)
legend("topright", legend = c("Observed Data", "Model-1 Fit"),
       col = c("blue", "red"), lty = 1, lwd = 2)

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

1.5 Applying KS Test

```
# Standardize residuals
standardized_residuals_1 <- scale(residuals_m1)

# Perform the Kolmogorov-Smirnov test
ks_test_1 <- ks.test(standardized_residuals_1, "pnorm", mean = 0, sd = 1)
```

```r
# Display KS test results
cat("--- Model 1 KS Test ---\n")
cat("KS Test Statistic:", ks_test_1$statistic, "\n")
cat("KS Test p-value:", ks_test_1$p.value, "\n")
```

Model 2: $y(t) = \frac{\beta_0+\beta_1 t}{\alpha_0+\alpha_1 t} + \epsilon(t)$

No changes needed, this model already used `t_orig`

2.1 Fitting

```r
# Load and align t and y again
tt <- t_orig
yy <- y1
stopifnot(length(tt) == length(yy))

# Define model [a0, a1, b0, b1]
f_model_2 <- function(p, t) {
  a0 <- p[1]; a1 <- p[2]; b0 <- p[3]; b1 <- p[4]
  denom <- b0 + b1 * t
  # Avoid division by zero
  if (any(abs(denom) < 1e-8)) return(rep(Inf, length(t)))
  (a0 + a1 * t) / denom
}

# Define SSE cost function
sse_2 <- function(p) {
  yhat <- f_model_2(p, tt)
  if (any(!is.finite(yhat))) return(1e10) # Penalize bad parameters
  sum((yy - yhat)^2)
}

# Initial guess from simple linear fit
fit_poly_lin <- lm(yy ~ tt)
a0_init <- coef(fit_poly_lin)[1] # Intercept
a1_init <- coef(fit_poly_lin)[2] # Slope
b0_init <- 1.0 # Guess
b1_init <- 0.1 # Guess
init_2 <- c(a0_init, a1_init, b0_init, b1_init)

# Use L-BFGS-B with bounds
fit_2 <- optim(init_2, sse_2, method = "L-BFGS-B",
               lower = c(-10, -10, 0.01, -10), # Keep b0 positive
               upper = c(10, 10, 20, 20),
               control = list(maxit = 2000))

params_2 <- fit_2$par
a0_2 <- params_2[1]
a1_2 <- params_2[2]
b0_2 <- params_2[3]
b1_2 <- params_2[4]

# Final predictions and residuals
fitted_values_2 <- f_model_2(params_2, tt)
```

```
residuals_2 <- yy - fitted_values_2
```

2.2 Checking the Model Accuracy for Model 2

```
# Calculate model accuracy metrics
rmse_2 <- sqrt(mean(residuals_2^2, na.rm = TRUE))
rss_2 <- sum(residuals_2^2)
tss_2 <- sum((yy - mean(yy))^2)
r_squared_2 <- 1 - (rss_2 / tss_2)
k_2 <- 4 # Number of parameters (a0, a1, b0, b1)
adjusted_r_squared_2 <- 1 - ((1 - r_squared_2) * (n - 1) / (n - k_2 - 1))
```

2.3 Calculating the Jacobian Matrix, FIM and CI

```
# Define the Jacobian for Model 2
jacobian_2 <- function(t, a0, a1, b0, b1) {
  denom <- b0 + b1 * t
  d_a0 <- 1 / denom
  d_a1 <- t / denom
  d_b0 <- -(a0 + a1 * t) / (denom^2)
  d_b1 <- -t * (a0 + a1 * t) / (denom^2)
  return(c(d_a0, d_a1, d_b0, d_b1))
}

# Fisher Information matrix calculation (4-param version)
fisher_information_4param <- function(t_vec, params, jac_func, sigma2) {
  n_obs <- length(t_vec)
  n_params <- length(params)
  J <- matrix(0, nrow = n_obs, ncol = n_params)

  for (i in 1:n_obs) {
    J[i, ] <- jac_func(t_vec[i], params[1], params[2], params[3], params[4])
  }

  fisher_matrix <- (1 / (n_obs * sigma2)) * (t(J) %*% J)

  lambda <- 1e-6
  fisher_matrix_regularized <- fisher_matrix + lambda * diag(n_params)

  return(fisher_matrix_regularized)
}

# Confidence Intervals calculation (re-defined for clarity, same as M1)
confidence_intervals_4param <- function(fisher_matrix, parameter_estimates, alpha = 0.05) {
  fisher_inv <- ginv(fisher_matrix)
  se <- sqrt(diag(fisher_inv))
  z_value <- qnorm(1 - alpha / 2)
  ci_lower <- parameter_estimates - z_value * se
  ci_upper <- parameter_estimates + z_value * se
  return(list(lower = ci_lower, upper = ci_upper, se = se))
}

# Calculate sigma^2 estimate
sigma2_2 <- rss_2 / (n - k_2)

# Calculate Fisher Information Matrix
```

```r
fisher_matrix_2 <- fisher_information_4param(tt, params_2, jacobian_2, sigma2_2)

# Calculate Confidence Intervals
ci_2 <- confidence_intervals_4param(fisher_matrix_2, params_2)

# Print the results
cat("--- Model 2 Results ---\n")
cat("Parameters (a0, a1, b0, b1):\n")
print(params_2)
cat("\nFisher Information Matrix:\n")
print(fisher_matrix_2)
cat("\nConfidence Intervals:\n")
print(ci_2)
```

2.4 Plots

```r
par(mfrow = c(2, 2), mar = c(4, 4, 2, 1))

# Residuals plot
plot(tt, residuals_2, main = "2.6: Residuals vs. Time (Model 2)",
     xlab = "Time (Original)", ylab = "Residuals", col = "black", pch = 19)
abline(h = 0, col = "red")

# Q-Q plot of residuals
qqnorm(residuals_2, main = "2.7(a): Q-Q Plot of Residuals (Model 2)")
qqline(residuals_2, col = "red")

# Histogram of residuals
hist(residuals_2, breaks = 10, main = "2.7(b): Histogram of Residuals (Model 2)",
     xlab = "Residuals", col = "lightblue", border = "black")

# Fitted values plot
tt_ord <- order(tt)
plot(tt[tt_ord], yy[tt_ord], type = "l", col = "blue",
     main = "2.8: Observed vs. Fitted Data (Model 2)",
     xlab = "Time (Original)", ylab = "Values", lwd = 2)
lines(tt[tt_ord], fitted_values_2[tt_ord], col = "red", lwd = 2)
legend("topright", legend = c("Observed Data", "Model-2 Fit"),
       col = c("blue", "red"), lty = 1, lwd = 2)

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

2.5 Applying KS Test

```r
# Standardize residuals
standardized_residuals_2 <- scale(residuals_2)

# Perform the Kolmogorov-Smirnov test
ks_test_2 <- ks.test(standardized_residuals_2, "pnorm", mean = 0, sd = 1)

# Display KS test results
cat("--- Model 2 KS Test ---\n")
cat("KS Test Statistic:", ks_test_2$statistic, "\n")
cat("KS Test p-value:", ks_test_2$p.value, "\n")
```

Model 3: $y(t) = \beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3 + \beta_4 t^4 + \epsilon(t)$

Updated to use `t_orig` instead of rescaled t

3.1 Fitting

```r
# Fit a 4th-degree polynomial regression model using ORIGINAL t
model3 <- lm(y1 ~ poly(t_orig, 4, raw = TRUE))

# Calculate fitted values and residuals
fitted_values_3 <- fitted(model3)
residuals_3 <- resid(model3)
```

3.2 Checking the Model Accuracy for Model 3

```r
# Calculate model accuracy metrics
rmse_3 <- sqrt(mean(residuals_3^2, na.rm = TRUE))
rss_3 <- sum(residuals_3^2)
tss_3 <- sum((y1 - mean(y1))^2)
r_squared_3 <- 1 - (rss_3 / tss_3)
k_3 <- 5 # Number of parameters (beta0, beta1, beta2, beta3, beta4)
adjusted_r_squared_3 <- 1 - ((1 - r_squared_3) * (n - 1) / (n - k_3 - 1))
```

3.3 Calculating the Jacobian Matrix, FIM and CI

```r
# Define the Jacobian of the model
jacobian_3 <- function(t, beta0, beta1, beta2, beta3, beta4) {
  d_beta0 <- 1
  d_beta1 <- t
  d_beta2 <- t^2
  d_beta3 <- t^3
  d_beta4 <- t^4
  return(c(d_beta0, d_beta1, d_beta2, d_beta3, d_beta4))
}

# Fisher Information matrix calculation (using 5-param jacobian_3)
fisher_information_5param <- function(t_vec, params, jac_func, sigma2) {
  n_obs <- length(t_vec)
  n_params <- length(params)
  J <- matrix(0, nrow = n_obs, ncol = n_params)

  for (i in 1:n_obs) {
    J[i, ] <- jac_func(t_vec[i], params[1], params[2], params[3], params[4], params[5])
  }

  fisher_matrix <- (1 / (n_obs * sigma2)) * (t(J) %*% J)

  lambda <- 1e-6
  fisher_matrix_regularized <- fisher_matrix + lambda * diag(n_params)

  return(fisher_matrix_regularized)
}

# Confidence Intervals calculation (using 5-param)
confidence_intervals_5param <- function(fisher_matrix, parameter_estimates, alpha = 0.05) {
  fisher_inv <- ginv(fisher_matrix)
```

```
  se <- sqrt(diag(fisher_inv))
  z_value <- qnorm(1 - alpha / 2)
  ci_lower <- parameter_estimates - z_value * se
  ci_upper <- parameter_estimates + z_value * se
  return(list(lower = ci_lower, upper = ci_upper, se = se))
}

# Get parameters from model
parameter_estimates_3 <- coef(model3)
beta0 <- parameter_estimates_3[1]
beta1 <- parameter_estimates_3[2]
beta2 <- parameter_estimates_3[3]
beta3 <- parameter_estimates_3[4]
beta4 <- parameter_estimates_3[5]

# Calculate sigma^2 estimate
sigma2_3 <- rss_3 / (n - k_3)

# Calculate Fisher Information Matrix using t_orig
fisher_matrix_3 <- fisher_information_5param(t_orig, parameter_estimates_3, jacobian_3, sigma2_3)

# Calculate Confidence Intervals
ci_3 <- confidence_intervals_5param(fisher_matrix_3, parameter_estimates_3)

# Print the results
cat("--- Model 3 Results ---\n")
cat("Parameters (beta0, beta1, beta2, beta3, beta4):\n")
print(parameter_estimates_3)
cat("\nFisher Information Matrix:\n")
print(fisher_matrix_3)
cat("\nConfidence Intervals:\n")
print(ci_3)
```

3.4 Plots

```
# Set up the plotting area
par(mfrow = c(2, 2), mar = c(4, 4, 2, 1))

# Residuals Plot for Model 3
plot(t_orig, residuals_3, main = "3.6: Residuals vs. Time (Model 3)",
     xlab = "Time (Original)", ylab = "Residuals", col = "black", pch = 19)
abline(h = 0, col = "red")

# Q-Q Plot for Model 3 Residuals
qqnorm(residuals_3, main = "3.7(a): Q-Q Plot of Residuals (Model 3)")
qqline(residuals_3, col = "red")

# Histogram of Residuals for Model 3
hist(residuals_3, breaks = 10, main = "3.7(b): Histogram of Residuals (Model 3)",
     xlab = "Residuals", col = "lightblue", border = "black")

# Observed vs. Fitted Data Plot for Model 3
plot(t_orig, y1, type = "l", col = "blue", main = "3.8: Observed vs. Fitted Data (Model 3)",
     xlab = "Time (Original)", ylab = "Values", lwd = 2)
lines(t_orig, fitted_values_3, col = "red", lwd = 2)
legend("topright", legend = c("Observed Data", "Model-3 Fit"),
```

```
       col = c("blue", "red"), lty = 1, lwd = 2)

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

3.5 Applying KS Test

```
# Standardize residuals
standardized_residuals_3 <- scale(residuals_3)

# Perform the Kolmogorov-Smirnov test
ks_test_3 <- ks.test(standardized_residuals_3, "pnorm", mean = 0, sd = 1)

# Display KS test results
cat("--- Model 3 KS Test ---\n")
cat("KS Test Statistic:", ks_test_3$statistic, "\n")
cat("KS Test p-value:", ks_test_3$p.value, "\n")
```

4.0 Model Comparison

This section provides a summary of all three models to help determine the 'best' fit.

```
# Create a comparison table
model_names <- c("Model 1 (Exponential)", "Model 2 (Rational [1/1])", "Model 3 (Poly 4th deg)")
rss_values <- c(rss_1, rss_2, rss_3)
r_squared_values <- c(r_squared_1, r_squared_2, r_squared_3)
adj_r_squared_values <- c(adjusted_r_squared_1, adjusted_r_squared_2, adjusted_r_squared_3)
ks_p_values <- c(ks_test_1$p.value, ks_test_2$p.value, ks_test_3$p.value)

comparison_df <- data.frame(
  Model = model_names,
  RSS = rss_values,
  R_Squared = r_squared_values,
  Adjusted_R_Squared = adj_r_squared_values,
  KS_Test_p_value = ks_p_values
)

cat("## Model Fit Comparison\n\n")
```

Model Fit Comparison

```
knitr::kable(comparison_df, digits = 6, caption = "Comparison of Model Fit Statistics")
```

Table 1: Comparison of Model Fit Statistics

Model	RSS	R_Squared	Adjusted_R_Squared	KS_Test_p_value
Model 1 (Exponential)	3.283324	0.810364	0.792806	0.713978
Model 2 (Rational [1/1])	3.628816	0.790410	0.775167	0.663296
Model 3 (Poly 4th deg)	3.777512	0.781821	0.761620	0.968311

```
cat("\n\n### Interpretation Guide\n\n")
```

Interpretation Guide

```
cat("* **RSS (Residual Sum of Squares):** Lower is better.\n")
```

- **RSS (Residual Sum of Squares):** Lower is better.

```
cat("* **Adjusted R-Squared:** Higher is better (adjusts for the number of parameters).\n")
```

- **Adjusted R-Squared:** Higher is better (adjusts for the number of parameters).

```
cat("* **KS Test p-value:** A value < 0.05 suggests the residuals are *not* normally distributed, which
```

- **KS Test p-value:** A value < 0.05 suggests the residuals are *not* normally distributed, which violates a model assumption.