INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

Department of Mathematics and Statistics

COURSE PROJECT

MTH686: Non-Linear Regression

Author:
Shivansh Yadav
Roll No: 230977
shivanshy23@iitk.ac.in

Supervisor:
Prof. Debasis Kundu
kundu@iitk.ac.in

1. Introduction and Model Definitions

The analysis is based on the dataset $\{(t_i, y_i)\}_{i=1}^{75}$, assuming the error terms $\epsilon(t)$ are independent and identically distributed (i.i.d.) $\mathcal{N}(0, \sigma^2)$. We fit the following three models:

- **Model 1 (M1: Sum of Exponentials, $P = 5$):** $y(t) = \alpha_0 + \alpha_1 e^{\beta_1 t} + \alpha_2 e^{\beta_2 t} + \epsilon(t)$.
- **Model 2 (M2: Rational Function, $P = 4$):** $y(t) = \frac{\alpha_0 + \alpha_1 t}{\beta_0 + \beta_1 t} + \epsilon(t)$.
- **Model 3 (M3: 4th Degree Polynomial, $P = 5$):** $y(t) = \beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3 + \beta_4 t^4 + \epsilon(t)$.

The objective of the Least Squares Estimation is to minimize the Residual Sum of Squares:

$$RSS = \sum_{t=1}^{N} (y_t - f_t(\hat{\theta}))^2.$$

Here, N is the total number of observations, y_t is the observed response, and $f_t(\hat{\boldsymbol{\theta}})$ is the value predicted by the model with estimated parameters $\hat{\boldsymbol{\theta}}$.

2. Least Squares Estimation Methodology (Q1, Q2)

Estimation Methods and Initial Guesses (Q2)

(i) **Model 1 (Nonlinear; Sum of Exponentials):** The parameters were estimated using the `curve_fit` routine from the `scipy.optimize` library, which performs nonlinear least squares via a damped modification of the Gauss–Newton method and iteratively updates the parameter vector based on the Jacobian of the model. Suitable initial values were chosen to ensure convergence.

(ii) **Model 2 (Nonlinear; Rational Function):** This model was fitted using an **explicit Gauss–Newton iterative Least Squares procedure**, implemented manually. Let $f(t, \theta)$ denote the model function and $\theta^{(k)}$ denote the parameter estimate at iteration k. At each iteration, the update step is:

$$\theta^{(k+1)} = \theta^{(k)} + \left(J(\theta^{(k)})^\top J(\theta^{(k)}) \right)^{-1} J(\theta^{(k)})^\top (Y - f(\theta^{(k)})),$$

where $J(\theta)$ is the Jacobian matrix with entries $J_{ij} = \frac{\partial f(t_i, \theta)}{\partial \theta_j}$. The iterations were continued until convergence based on the norm $\|\theta^{(k+1)} - \theta^{(k)}\| < 10^{-6}$. This ensures that the parameter estimates are obtained without using any black-box optimizer.

(iii) **Linear Model (M3):** The LSEs were found using the **closed-form solution** (Normal Equations):

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}.$$

The terms are defined as:

- $\hat{\boldsymbol{\beta}}$ is the $P \times 1$ vector of estimated parameters for Model 3.
- $\mathbf{Y}$ is the $N \times 1$ vector of observed responses, $\mathbf{Y} = \begin{bmatrix} y_1 & y_2 & \dots & y_N \end{bmatrix}^T$.
- $\mathbf{X}$ is the $N \times P$ *Design Matrix* (where $P = 5$), structured as:

$$\mathbf{X} = \begin{bmatrix} 1 & t_1 & t_1^2 & t_1^3 & t_1^4 \\ 1 & t_2 & t_2^2 & t_2^3 & t_2^4 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_N & t_N^2 & t_N^3 & t_N^4 \end{bmatrix}.$$

Parameter Estimates (Q1)

The final LSEs obtained for all three models are summarized below:

Table 1: Parameter Estimates ($\hat{\boldsymbol{\theta}}$) for All Models (Q1)

Model	Parameter	Estimate
M1: $y = \alpha_0 + \alpha_1 e^{\beta_1 t} + \alpha_2 e^{\beta_2 t}$	$\hat{\alpha}_0$	1.5601
	$\hat{\alpha}_1$	−90.2479
	$\hat{\beta}_1$	−0.05832
	$\hat{\alpha}_2$	88.7897
	$\hat{\beta}_2$	−0.05760
M2: $y = \frac{\alpha_0+\alpha_1 t}{\beta_0+\beta_1 t}$	$\hat{\alpha}_0$	−0.144587
	$\hat{\alpha}_1$	0.495948
	$\hat{\beta}_0$	1.528560
	$\hat{\beta}_1$	0.266332
M3: $y = \beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3 + \beta_4 t^4$	$\hat{\beta}_0$	0.145249
	$\hat{\beta}_1$	0.127578
	$\hat{\beta}_2$	−0.00375066
	$\hat{\beta}_3$	0.00004542
	$\hat{\beta}_4$	−0.000000193

Choice of Initial Guesses

For nonlinear least squares estimation, the convergence and stability of the iterative method depend strongly on the choice of initial parameter values. Therefore, care was taken to select starting values that reflect the observed scale and qualitative behavior of the data.

Model 1 (Sum of Exponentials):

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

The data exhibit a smooth monotonic decay pattern. Therefore:

- α_0 was initialized at $\min(y)$, representing the asymptotic long-term baseline level.
- The amplitudes α_1 and α_2 were chosen as approximately half of the observed dynamic range:
$$\frac{\max(y) - \min(y)}{2},$$
ensuring that the initial scale of the model matches the data magnitude.
- The decay rates β_1 and β_2 were initialized as small negative values (-0.05 and -0.01), since the observed curves decrease smoothly and slowly.

Thus, the initial parameter vector was:

$$p0\_1 = [\min(y),\ (\max(y) - \min(y))/2,\ -0.05,\ (\max(y) - \min(y))/2,\ -0.01].$$

Model 2 (Rational Function):

$$y(t) = \frac{\alpha_0 + \alpha_1 t}{\beta_0 + \beta_1 t}$$

This form represents a ratio of two linear expressions, which varies gradually with t:

- The intercepts α_0 and β_0 were initialized to 1.0, reflecting a moderate baseline scale.

- The slopes α_1 and β_1 were set to small positive values (0.1 and 0.01), which prevents numerical instability or division by values close to zero in the denominator.

The initial guess used for the Gauss–Newton iteration was:

$$p0\_2 = [1.0,\ 0.1,\ 1.0,\ 0.01].$$

Model 3 (4th Degree Polynomial):

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

Since this model is linear in its parameters, the Least Squares Estimators are obtained directly from the normal equation:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y},$$

and therefore **no initial guesses are required**.

3. Model Selection and Estimated Variance (Q3, Q4)

Best Fitted Model (Q3)

Model selection is performed by comparing the **Estimated Error Variance**

$$\hat{\sigma}^2 = \frac{\text{RSS}}{N-P},$$

which penalizes models with a larger number of parameters. The model with the smallest value of $\hat{\sigma}^2$ is considered the best fit. Here,

- N is the total number of observed data points,
- P is the number of parameters in the model, and
- $(N-P)$ is the **degrees of freedom** of the model.

Table 2: Model Comparison using RSS and Estimated Error Variance

Model	P	**RSS**	$\hat{\sigma}^2$
M1	5	2.67846	0.038263
M2	4	3.24739	0.045738
M3	5	2.58705	0.036958

Conclusion (Q3): Model **3** is selected as the best fitted model, as it has the lowest estimated error variance $\hat{\sigma}^2$ and the lowest RSS value.

Estimate of σ^2 (Q4)

The estimated variance for the best model (Model 3) is:

$$\boxed{\hat{\sigma}^2 = 0.036958}$$

4. Statistical Inference and Diagnostics (Q5, Q6, Q7, Q8)

Confidence Intervals (Q5)

The approximate 95% confidence intervals for the parameters are obtained using

$$\hat{\Sigma}_{\hat{\boldsymbol{\theta}}} = \hat{\sigma}^2(\mathbf{J}^\top\mathbf{J})^{-1},$$

where $\mathbf{J}$ is the Jacobian matrix of partial derivatives evaluated at $\hat{\boldsymbol{\theta}}$. The standard error of each parameter is given by the square root of the corresponding diagonal entry of $\hat{\Sigma}_{\hat{\boldsymbol{\theta}}}$.

Table 3: 95% Confidence Intervals for Model 3

Parameter	**Estimate Interval**
β_0	$[-0.09092020467046463,\ 0.3814186484645672]$
β_1	$[0.08512644571312367,\ 0.17002864828386688]$
β_2	$[-0.006000910567014501,\ -0.0015004173025636378]$
β_3	$[1.0721433438486716e-06,\ 8.976724191853944e-05]$
β_4	$[-4.822285887568381e-07,\ 9.684591205528406e-08]$

Residual Plot and Fitted Curve (Q6, Q8)

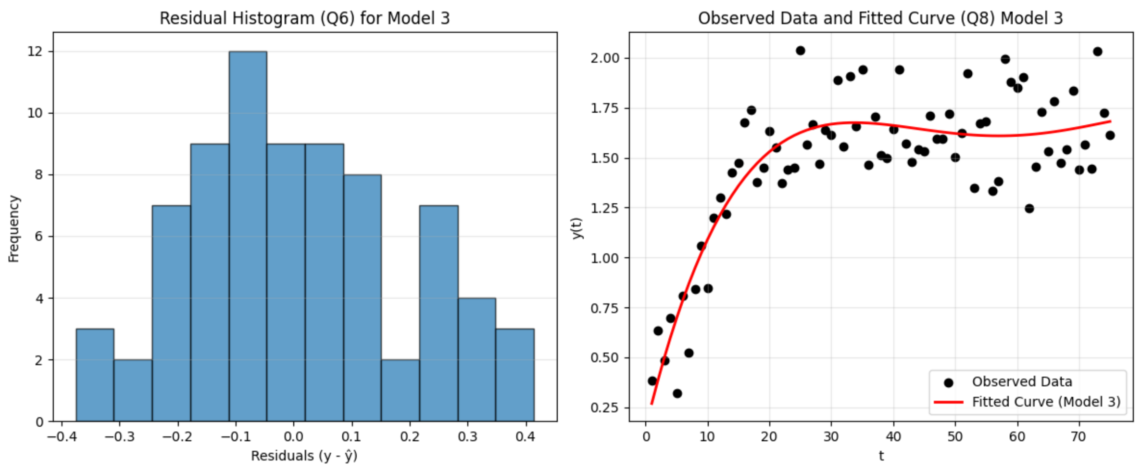

Figure 1: Observed Data and Fitted Curve (Right, Q8) with Residual Histogram Plot (Left, Q6) for **Model 3**

Normality Test for Residuals (Q7)

A Chi-Square Goodness-of-Fit test was performed to assess whether the residuals follow a normal distribution.

Table 4: Chi-Square Test for Normality of Residuals (Model 3)

Statistic	**Value**	**Result**
Chi-Square Statistic	5.2883	
P-value	0.2590	

The null hypothesis H_0 is that the residuals follow a Normal distribution with mean 0 and variance $\hat{\sigma}^2$. The alternative hypothesis H_1 is that they do not follow a Normal distribution.

Since the p-value is greater than 0.05, we **do not reject** the null hypothesis. Thus, the residuals are consistent with the i.i.d. normal error assumption required for regression inference. The residual histogram also shows a bell-shaped pattern consistent with a Normal distribution.

5. Conclusion

Model 3 (4th Degree Polynomial) was identified as the best fitted model, based on the minimum estimated error variance. Models 1 and 2 required iterative Least Squares estimation and were fitted using the **curve_fit** routine and Gauss-Newton algorithm with analytically derived Jacobians respectively. Model 3 was fitted using the closed-form Linear Least Squares solution. Residual diagnostics confirmed that the assumptions of i.i.d. normal errors were satisfied, validating the reliability of the parameter estimates and confidence intervals.

Appendix A: Python Code for Model Estimation

```
# ===================
# 1. Import Libraries
# ===================
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import chi2, norm
import scipy.stats as stats
from scipy.optimize import curve_fit

# ===============
# 2. Data Loading
# ===============
try:
    df = pd.read_csv('set-34.dat', sep=r'\s+', header=None, names=['t', 'y'])
    t = df['t'].values
    y = df['y'].values.astype(np.float64)
    N_global = len(y)
    print(f"Loaded {N_global} data points.")
except FileNotFoundError:
    print("ERROR: Data file 'set-34.dat' not found.")
    exit()

# ==============================
# 3. General Gauss-Newton Solver
# ==============================
def general_gauss_newton(t, y, model_func, jacobian_func, p0, max_iter=100, tol=1e-8):
    """
    Gauss-Newton algorithm for Non-Linear Least Squares.
    Returns estimated parameters, covariance, RSS, and iteration log.
    """
    theta_current = np.array(p0, dtype=float)
    P = len(p0)
    N = len(y)
    log_data = []

    initial_rss = np.sum((y - model_func(t, theta_current))**2)
    print(f"\nStarting Gauss-Newton (P={P}). Initial RSS: {initial_rss:.4f}")

    for iteration in range(max_iter):
        y_pred = model_func(t, theta_current)
        r = y - y_pred
        J = jacobian_func(t, theta_current)

        JT_J = J.T @ J
        JT_r = J.T @ r

        try:
            d_theta, _, _, _ = np.linalg.lstsq(JT_J, JT_r, rcond=None)
        except np.linalg.LinAlgError:
            print(f"Warning: Singularity at iteration {iteration}.")
            break

        theta_next = theta_current + d_theta
        step_size = np.linalg.norm(d_theta)

        log_data.append([iteration] + list(theta_current) + [np.sum(r**2)])

        if step_size < tol * (np.linalg.norm(theta_current) + tol):
            log_data.append([iteration+1] + list(theta_next) + [np.sum((y - model_func(t, theta_next))**2)])
            print(f"Gauss-Newton converged in {iteration+1} iterations.")
            theta_current = theta_next
            break

        theta_current = theta_next
    else:
        log_data.append([max_iter] + list(theta_current) + [np.sum((y - model_func(t, theta_current))**2)])
        print(f"Reached max iterations ({max_iter}).")

    # Covariance and RSS
    r_final = y - model_func(t, theta_current)
    RSS_final = np.sum(r_final**2)
    sigma2_hat = RSS_final / (N - P)
    try:
        J_final = jacobian_func(t, theta_current)
        JT_J_inv = np.linalg.inv(J_final.T @ J_final)
        pcov = sigma2_hat * JT_J_inv
    except np.linalg.LinAlgError:
        print("Warning: Covariance estimation failed.")
        pcov = np.full((P, P), np.nan)

    column_names = [f' {i}' for i in range(P)]
    log_df = pd.DataFrame(log_data, columns=['Iteration'] + column_names + ['RSS'])
    print(f"Final RSS: {RSS_final:.6f}")
    return theta_current, pcov, RSS_final, log_df

# ====================
# 4. Model Definitions
# ====================
def model1_func(t, a0, a1, b1, a2, b2):
    return a0 + a1*np.exp(b1*t) + a2*np.exp(b2*t)

def model2_func(t, theta):
    a0, a1, b0, b1 = theta
    denominator = b0 + b1 * t
    return np.divide((a0 + a1 * t), denominator,
                     out=np.full_like(t, 1e10, dtype=np.float64),
                     where=np.abs(denominator) > 1e-12)

def model3_func(t, theta):
    b0, b1, b2, b3, b4 = theta
    return b0 + b1*t + b2*t**2 + b3*t**3 + b4*t**4

# =======================================
# 5. Jacobian / Design Matrix Definitions
# =======================================
def jacobian_model1(t, theta):
    a0, a1, b1, a2, b2 = theta
    J = np.zeros((len(t), 5))
    J[:, 0] = 1
    J[:, 1] = np.exp(b1*t)
    J[:, 2] = a1 * t * np.exp(b1*t)
    J[:, 3] = np.exp(b2*t)
    J[:, 4] = a2 * t * np.exp(b2*t)
    return J

def jacobian_model2(t, theta):
    a0, a1, b0, b1 = theta
    J = np.zeros((len(t), 4))
    denom = b0 + b1 * t
    safe_denom = np.where(np.abs(denom) > 1e-12, denom, 1e10)
    safe_denom2 = safe_denom**2
    numerator = a0 + a1 * t
    J[:, 0] = 1.0 / safe_denom
    J[:, 1] = t / safe_denom
    J[:, 2] = - numerator / safe_denom2
    J[:, 3] = - (t * numerator) / safe_denom2
    return J

def design_matrix_model3(t):
    return np.vstack([np.ones_like(t), t, t**2, t**3, t**4]).T

# ==================
# 6. Initial Guesses
# ==================
p0_1 = [np.min(y), (np.max(y)-np.min(y)) / 2, -0.05, (np.max(y)-np.min(y)) / 2, -0.01]
p0_2 = [1.0, 0.1, 1.0, 0.01]

# =============
# 7. Estimation
# =============
print("=== Model 1 Estimation ===")

popt1, pcov1 = curve_fit(model1_func, t, y, p0_1, maxfev=50000)
y_pred1 = model1_func(t, *popt1)
RSS1 = np.sum((y - y_pred1)**2)
print("Model 1 Parameters (a0, a1, b1, a2, b2):")
print(popt1)
print("Model 1 RSS:", RSS1)
```

```python

print("=== Model 2 Estimation ===")
popt2, pcov2, RSS2, log_df2 = general_gauss_newton(t, y, model2_func, jacobian_model2, p0_2)
print("\n--- Model 2 Iteration History (Q2 Table) ---")
print(log_df2.round(6).to_markdown(index=False))

print("=== Model 3 Estimation ===")
X3 = design_matrix_model3(t)
XtX_inv = np.linalg.inv(X3.T @ X3)
popt3 = XtX_inv @ X3.T @ y
y_pred3 = model3_func(t, popt3)
RSS3 = np.sum((y - y_pred3)**2)
sigma2_3 = RSS3 / (N_global - X3.shape[1])
pcov3 = sigma2_3 * XtX_inv
print("Model 3 Polynomial Coefficients (highest degree first):")
print(popt3)
print("Model 3 RSS:", RSS3)

# ===================
# 8. Model Selection
# ===================
sigma2_1 = RSS1 / (N_global - len(popt1))
sigma2_2 = RSS2 / (N_global - len(popt2))
comparison = pd.DataFrame({
    'Model': [1, 2, 3],
    'P': [len(popt1), len(popt2), X3.shape[1]],
    'RSS': [RSS1, RSS2, RSS3],
    'sigma^2': [sigma2_1, sigma2_2, sigma2_3],
})
best_model_idx = comparison['sigma^2'].astype(float).idxmin()
best_model_num = comparison.loc[best_model_idx, 'Model']

# ==============================================
# 9. Confidence Intervals, Residuals, and Plots
# ==============================================
print("--- CONFIDENCE INTERVALS (95%) (Q5) ---")
alpha = 0.05
Z_score = stats.norm.ppf(1 - alpha / 2)

if best_model_num == 1:
    best_popt, best_pcov, param_names = popt1, pcov1, ['a0', 'a1', 'b1', 'a2', 'b2']
    best_y_pred = y_pred1
elif best_model_num == 2:
    best_popt, best_pcov, param_names = popt2, pcov2, ['a0', 'a1', 'b0', 'b1']
    best_y_pred = y_pred2
else:
    best_popt, best_pcov, param_names = popt3, pcov3, ['b0', 'b1', 'b2', 'b3', 'b4']
    best_y_pred = y_pred3

try:
    best_se = np.sqrt(np.diag(best_pcov))
    CI_best_model = np.array([best_popt - Z_score * best_se, best_popt + Z_score * best_se]).T

    print(f"Confidence Intervals for Model {best_model_num}:")
    for i, (low, high) in enumerate(CI_best_model):
        print(f"  {param_names[i]}: [{low}, {high}]")
except Exception:
    print(f"Could not calculate CIs for Model {best_model_num}. Check for singularity in Fisher Information Matrix.")

# --- Chi-Square Goodness-of-Fit Test Function ---
def chi_square_normality_test(residuals, N_obs, df_loss=2, num_bins=7):
    """Chi-Square Goodness-of-Fit test for normality of residuals."""
    if N_obs < 30:
        return np.nan, np.nan, "Sample size too small."

    mu, sigma = np.mean(residuals), np.std(residuals, ddof=0)
    bins = np.linspace(residuals.min(), residuals.max(), num_bins+1)
    O_i, _ = np.histogram(residuals, bins=bins)

    # Expected frequencies
    E_i = np.array([(norm.cdf(bins[i+1], mu, sigma) - norm.cdf(bins[i], mu, sigma)) * N_obs
                    for i in range(num_bins)])
    E_i[0] = norm.cdf(bins[1], mu, sigma) * N_obs      # lower tail
    E_i[-1] = (1 - norm.cdf(bins[-2], mu, sigma)) * N_obs  # upper tail

    valid = E_i > 0
    chi2_stat = np.sum((O_i[valid] - E_i[valid])**2 / E_i[valid])
    df = np.sum(valid) - 1 - df_loss
    if df <= 0:
        return chi2_stat, np.nan, "Insufficient degrees of freedom."

    p_value = 1 - chi2.cdf(chi2_stat, df)
    return chi2_stat, p_value, f"df={df}, bins_used={valid.sum()}"

best_res = y - best_y_pred
chi2_stat, p_value, details = chi_square_normality_test(best_res, N_global, df_loss=2, num_bins=7)

print(f"--- Normality Test (Q7) for Model {best_model_num} (² Goodness-of-Fit) ---")
print(f"Chi-Square Statistic: {chi2_stat:.4f}")
if not np.isnan(p_value):
    print(f"P-value: {p_value:.4f}")
    conclusion = "DO NOT reject" if p_value > 0.05 else "REJECT"
    print(f"Conclusion: {conclusion} the null hypothesis (residuals ~ normal).")
else:
    print(f"P-value: {p_value}, Test Warning: {details}")


# Q6, Q8: Plots
plt.figure(figsize=(12, 5))

# Plot 1: Residual historgram
plt.subplot(1, 2, 1)
plt.hist(best_res, bins=12, edgecolor='black', alpha=0.7)
plt.title(f"Residual Histogram (Q6) for Model {best_model_num}")
plt.xlabel("Residuals (y - ŷ)")
plt.ylabel("Frequency")
plt.grid(axis='y', alpha=0.3)

# Plot 2: Data and Fitted Curve (Q8)
plt.subplot(1, 2, 2)
plt.scatter(t, y, label="Observed Data", color='black', marker='o')
plt.plot(t, best_y_pred, label=f"Fitted Curve (Model {best_model_num})", color='red', linewidth=2)
plt.title(f"Observed Data and Fitted Curve (Q8) Model {best_model_num}")
plt.xlabel("t")
plt.ylabel("y(t)")
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

Listing 1: Complete Python Code