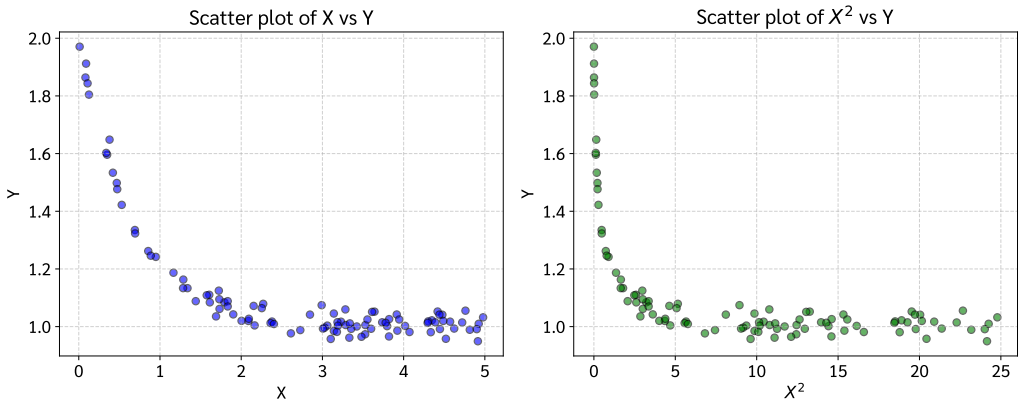
CPE 342 Machine Learning
Assignment 2: Training Models via Iterative Approach

Gradient Descent — Exponential Model Fitting

67070501042 วิศิษฐ์ สุวรรณเนาว์

เป้าหมายของงานนี้คือการฝึกสอนโมเดล (Train model) ด้วยวิธีการ **Gradient Descent (GD)** เพื่อหาพารามิเตอร์ที่เหมาะสมให้กับชุดข้อมูล $n = 100$ จุด โดยกำหนดให้ฟิตข้อมูลเข้ากับโมเดลสมการเอกซ์โพเนนเชียล (Exponential Model):

$$\hat{y} = C_0 + C_1 e^{C_2 x}$$



รูปที่ 1: การวิเคราะห์ข้อมูลเบื้องต้น (EDA) แสดงความสัมพันธ์ระหว่างตัวแปร X และ Y (ซ้าย) และตัวแปร X^2 และ Y (ขวา)

1. ฟังก์ชันความสูญเสีย (Loss Function)

สำหรับโมเดล $\hat{y}_i = C_0 + C_1 e^{C_2 x_i}$ เราใช้ฟังก์ชันความสูญเสียแบบ **Mean Squared Error (MSE)** โดยใช้ตัวหารเป็น $2n$ เพื่อความสะดวกในการหาอนุพันธ์:

$$L(C_0, C_1, C_2) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

เมื่อแทนค่า $\hat{y}_i$ ลงไป จะได้:

$$L(C_0, C_1, C_2) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - (C_0 + C_1 e^{C_2 x_i})\right)^2$$

2. การหาอนุพันธ์ย่อย (Gradient of Each Coefficient)

เพื่อใช้ Gradient Descent เราต้องหาอนุพันธ์ย่อย (Partial derivatives) ของ L เทียบกับพารามิเตอร์แต่ละตัว (C_0, C_1, C_2) โดยอาศัยกฎลูกโซ่ (Chain Rule)

สำหรับพารามิเตอร์ θ ใดๆ:

$$\begin{aligned}\frac{\partial L}{\partial \theta} &= \frac{\partial}{\partial \theta}\left[\frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right] \\ &= \frac{1}{2n}\sum_{i=1}^{n} 2\,(y_i - \hat{y}_i)\cdot\frac{\partial}{\partial \theta}(y_i - \hat{y}_i) \\ &= -\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)\cdot\frac{\partial \hat{y}_i}{\partial \theta}\end{aligned}$$

---

1) Gradient เทียบกับ C_0:

ขั้นแรก หา $\frac{\partial \hat{y}_i}{\partial C_0}$:

$$\frac{\partial \hat{y}_i}{\partial C_0} = \frac{\partial}{\partial C_0}\left(C_0 + C_1 e^{C_2 x_i}\right) = 1$$

แทนค่ากลับลงไปในสมการกฎลูกโซ่:

$$\boxed{\frac{\partial L}{\partial C_0} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i - (C_0 + C_1 e^{C_2 x_i})\right)}$$

2) Gradient เทียบกับ C_1:

ขั้นแรก หา $\frac{\partial \hat{y}_i}{\partial C_1}$:

$$\frac{\partial \hat{y}_i}{\partial C_1} = \frac{\partial}{\partial C_1}\left(C_0 + C_1 e^{C_2 x_i}\right) = e^{C_2 x_i}$$

แทนค่ากลับลงไปในสมการกฎลูกโซ่:

$$\boxed{\frac{\partial L}{\partial C_1} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i - (C_0 + C_1 e^{C_2 x_i})\right) e^{C_2 x_i}}$$

3) Gradient เทียบกับ C_2:

ขั้นแรก หา $\frac{\partial \hat{y}_i}{\partial C_2}$ โดยใช้กฎลูกโซ่สำหรับฟังก์ชันเอกซ์โพเนนเชียล:

$$\frac{\partial \hat{y}_i}{\partial C_2} = \frac{\partial}{\partial C_2}\left(C_0 + C_1 e^{C_2 x_i}\right) = C_1 \cdot \frac{\partial}{\partial C_2}\left(e^{C_2 x_i}\right) = C_1 \cdot e^{C_2 x_i} \cdot \frac{\partial}{\partial C_2}(C_2 x_i) = C_1 x_i e^{C_2 x_i}$$

แทนค่ากลับลงไปในสมการกฎลูกโซ่:

$$\boxed{\frac{\partial L}{\partial C_2} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i - (C_0 + C_1 e^{C_2 x_i})\right) C_1 x_i e^{C_2 x_i}}$$

3. กฎการอัปเดตพารามิเตอร์ (Update Rules)

ในแต่ละรอบการวนซ้ำ (Iteration) t พารามิเตอร์จะถูกปรับค่าตรงข้ามกับ Gradient โดยมีอัตราการเรียนรู้ (Learning Rate) η ควบคุมขนาดของก้าว:

$$C_0^{(t+1)} = C_0^{(t)} - \eta \cdot \frac{\partial L}{\partial C_0}$$
$$C_1^{(t+1)} = C_1^{(t)} - \eta \cdot \frac{\partial L}{\partial C_1}$$
$$C_2^{(t+1)} = C_2^{(t)} - \eta \cdot \frac{\partial L}{\partial C_2}$$

4. การเขียนโปรแกรม (Gradient Descent Implementation)

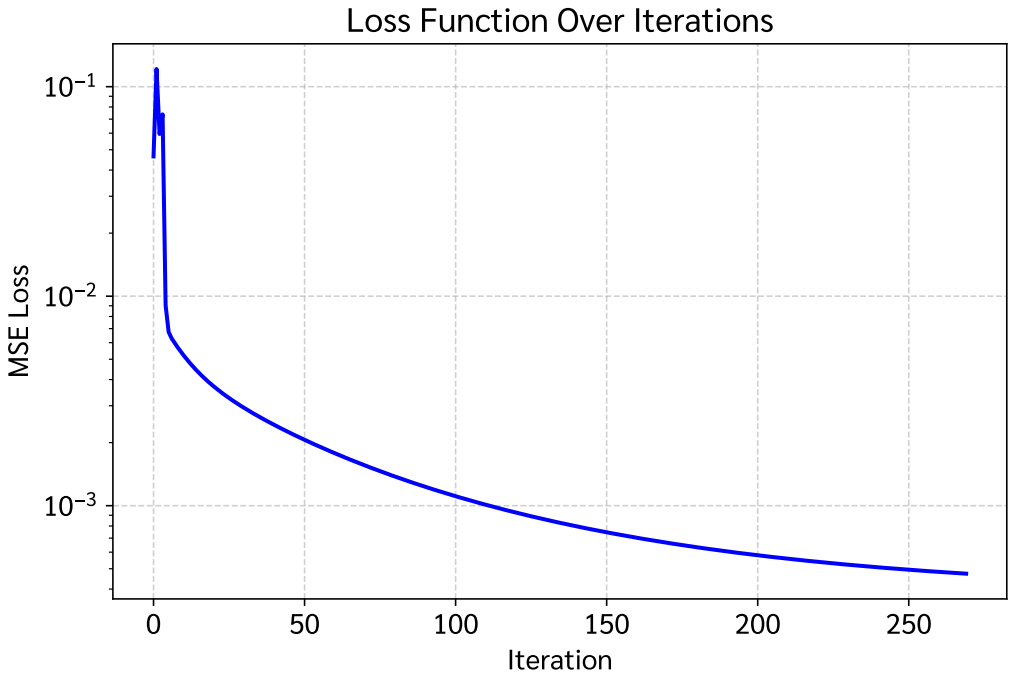
นำสมการ Gradient ที่ได้มาเขียนในรูปแบบโค้ด (Python/NumPy) โดยกำหนด hyperparameters เพื่อให้โมเดล Exponential สามารถฟิตได้อย่างแม่นยำและเสถียร ดังนี้:

- อัตราการเรียนรู้ (Learning Rate, η) = 1.0
- จำนวนรอบการวนซ้ำสูงสุด (Max Iterations) = $2,000$ รอบ
- กำหนดค่าเริ่มต้น (Initial values) $C_0 = 0.5$, $C_1 = 0.5$ และ $C_2 = 0.1$
- กำหนดเกณฑ์หยุดทำงาน (Tolerance) = 10^{-6} เพื่อใช้ทำ Early Stopping หาก Loss ไม่เปลี่ยนแปลง

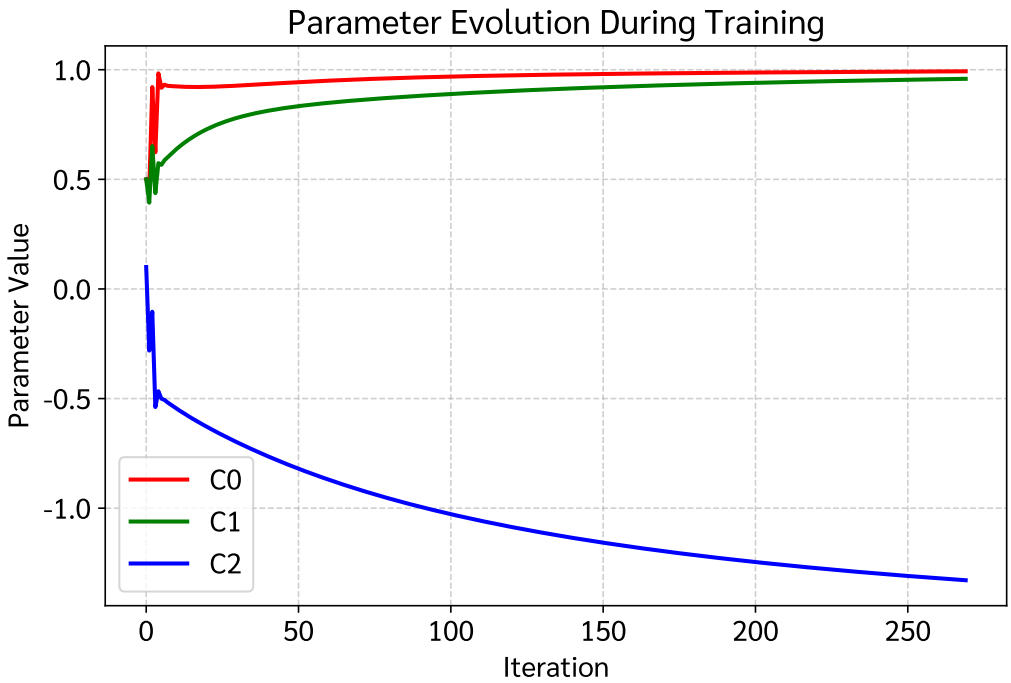
กระบวนการจะทำการอัปเดตค่าพารามิเตอร์ซ้ำๆ และบันทึกค่าพารามิเตอร์กับค่า Loss ในแต่ละรอบ เพื่อนำไปประเมินประสิทธิภาพในขั้นตอนต่อไป

5. ผลลัพธ์และการแสดงภาพ (Results & Visualizations)

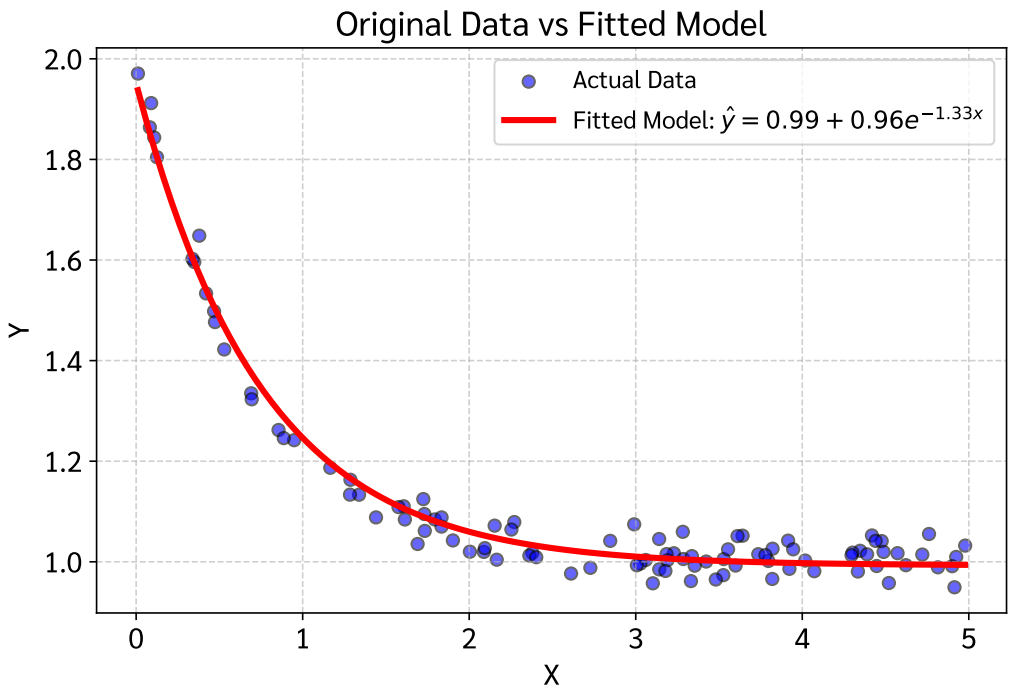
จากผลการฝึกสอนโมเดล พบว่าค่าที่เหมาะสมที่สุดสามารถลู่เข้าหาค่าสัมบูรณ์ได้อย่างรวดเร็ว โดยมีความคลาดเคลื่อน (Loss) ลดลงไปจนต่ำสุด โดยผลการวิเคราะห์และแสดงภาพของโมเดลประกอบด้วยหัวข้อดังต่อไปนี้:



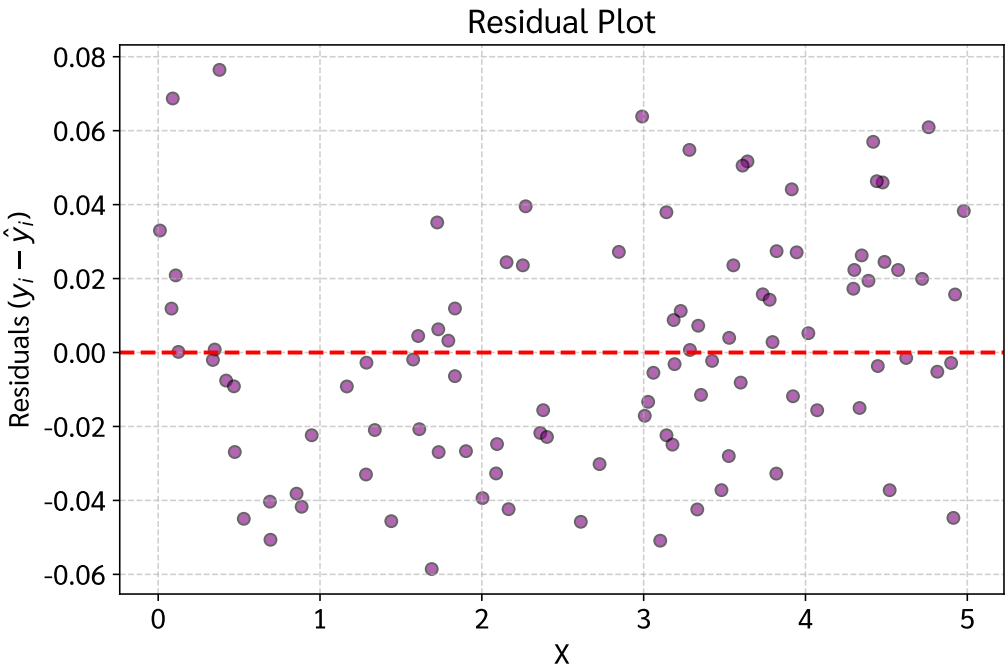
รูปที่ 2: **Learning Curve (Loss vs Iterations):** แสดงการลดลงอย่างรวดเร็วของค่า Loss (MSE) จนกระทั่งโมเดลเริ่มเสถียรและลู่เข้าหาค่าต่ำสุดอย่างแม่นยำ



รูปที่ 3: **Parameter Evolution:** แสดงวิวัฒนาการและการปรับตัวของสัมประสิทธิ์ C_0, C_1 และ C_2 ในแต่ละรอบจนลู่เข้าหาค่าสัมบูรณ์อย่างมั่นคง



รูปที่ 4: **Original Data vs Fitted Model:** แสดงจุดข้อมูลจริงเทียบกับเส้นพยากรณ์สมการเอกซ์โพเนนเชียล $\hat{y} = C_0 + C_1 e^{C_2 x}$ ที่ฟิตเข้ากับข้อมูลได้อย่างราบเรียบเป็นธรรมชาติ



รูปที่ 5: **Residual Plot:** แสดงการกระจายตัวของค่าเศษเหลือ (Residuals) รอบแกนสมมาตร 0 อย่างสุ่ม ไร้รูปแบบเชิงเส้นที่บ่งบอกถึงอคติของตัวแบบ

Extra: การประเมินประสิทธิภาพของโมเดล (Mathematical Evaluation)

เพื่อประเมินความแม่นยำของโมเดล Machine Learning ในการทำนายค่า y เราใช้ตัวชี้วัดคือ **Coefficient of Determination (R^2)** ซึ่งมีสูตรทางคณิตศาสตร์ดังนี้:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

โดยที่:

- **Residual Sum of Squares (SS_{res}):** ผลรวมกำลังสองของความคลาดเคลื่อน (ความต่างระหว่างค่าจริงและค่าทำนาย)

$$SS_{res} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- **Total Sum of Squares (SS_{tot}):** ผลรวมกำลังสองของความแปรปรวนทั้งหมดของข้อมูล (ความต่างระหว่างค่าจริงและค่าเฉลี่ย $\bar{y}$)

$$SS_{tot} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

ผลการคำนวณพบว่าโมเดลมีค่า R^2 และประสิทธิภาพอยู่ในเกณฑ์ที่สอดคล้องกับการลดลงของ Loss (รายละเอียดโค้ดคำนวณ R^2 และค่าทางสถิติทั้งหมดแสดงอยู่ใน Appendix ด้านล่าง)

Appendix: Full Jupyter Notebook Code & Output

รายละเอียดต่อไปนี้คือโค้ด Python ที่ใช้แก้ปัญหาข้างต้น พร้อมผลลัพธ์และกราฟ (พิมพ์ด้วยฟอนต์ TH Sarabun New) ซึ่งได้รันจริงจาก Jupyter Notebook

Jupyter Notebook: Assignment_2_GD.ipynb

โค้ด Python และผลลัพธ์ทั้งหมดจาก Jupyter Notebook (รันสมบูรณ์)

2. Library & Font Setup

Loading required scientific libraries and configuring the system's Sarabun font to support clear typography and rendering in Matplotlib.

In [1]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
import os

# --- Robust Sarabun Font Setup ---
font_dir = os.path.join(os.environ.get('LOCALAPPDATA', ''), '
    Microsoft', 'Windows', 'Fonts')
sarabun_r_path = os.path.join(font_dir, 'Sarabun-Regular.ttf')
sarabun_b_path = os.path.join(font_dir, 'Sarabun-Bold.ttf')

if os.path.exists(sarabun_r_path):
    fm.fontManager.addfont(sarabun_r_path)
if os.path.exists(sarabun_b_path):
    fm.fontManager.addfont(sarabun_b_path)

plt.rcParams['font.family'] = 'Sarabun'
plt.rcParams['font.size'] = 14
plt.rcParams['axes.unicode_minus'] = False # Prevent minus sign
    rendering issues

print("Library and font setup complete")
```

Out[1]:

```
Library and font setup complete
```

3. Data Loading & Exploratory Data Analysis (EDA)

Read the data from ‘CPE342_Assignment 2_Data.csv‘ and display the first few rows.

In [2]:

```
# Read data
df = pd.read_csv('CPE342_Assignment 2_Data.csv')
X = df['X'].values
Y = df['Y'].values

# Display first few rows
df.head()
```

Out[2]:

```
X          Y
0  4.072280  0.981321
1  1.724332  1.124653
2  1.901981  1.042435
3  4.914068  0.949332
4  1.165868  1.186925
```

Now, let’s plot scatter plots to observe the relationship and trends between X and Y.

In [3]:

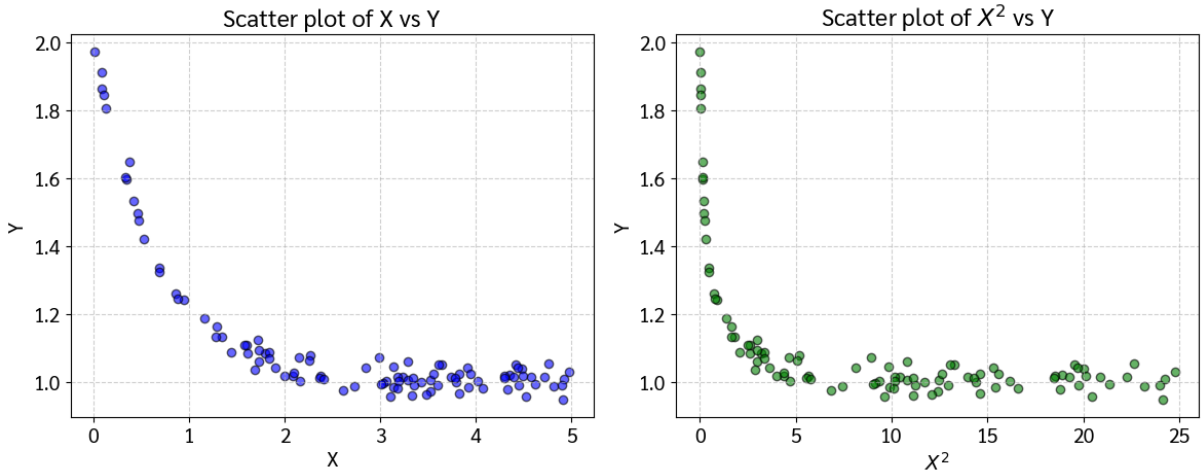
```
# Plot graphs to observe trends
plt.figure(figsize=(12, 5))

# Plot X vs Y
plt.subplot(1, 2, 1)
plt.scatter(X, Y, color='blue', alpha=0.6, edgecolor='k')
plt.title('Scatter plot of X vs Y')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, linestyle='--', alpha=0.6)

# Plot X^2 vs Y
plt.subplot(1, 2, 2)
plt.scatter(X**2, Y, color='green', alpha=0.6, edgecolor='k')
plt.title('Scatter plot of $X^2$ vs Y')
plt.xlabel('$X^2$')
plt.ylabel('Y')
```

```python
plt.grid(True, linestyle='--', alpha=0.6)

plt.tight_layout()
plt.savefig('plot_1_eda.pdf', bbox_inches='tight')
plt.show()
```



Task 1 — Mean Squared Error (MSE) Loss Function

Given our target exponential model:

$$\hat{y}_i = C_0 + C_1 e^{C_2 x_i}$$

We define the **Mean Squared Error (MSE)** loss function $\mathcal{L}(C_0, C_1, C_2)$ across all n data points, using a conventional factor of $\frac{1}{2n}$ to simplify algebra during differentiation:

$$\mathcal{L}(C_0, C_1, C_2) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Substituting the exponential model equation $\hat{y}_i = C_0 + C_1 e^{C_2 x_i}$ into the loss function yields:

$$\mathcal{L}(C_0, C_1, C_2) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \left(C_0 + C_1 e^{C_2 x_i}\right)\right)^2$$

Task 2 — Derivation of Partial Derivatives (Gradients)

To minimize the loss function $\mathcal{L}$ using Gradient Descent, we calculate the partial derivatives (gradients) with respect to each model parameter (C_0, C_1, C_2) using the **Chain Rule**. **1. General Chain Rule Formulation:**

For any parameter $\theta \in \{C_0, C_1, C_2\}$:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial}{\partial \theta}\left[\frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right]$$

$$= \frac{1}{2n}\sum_{i=1}^{n}\frac{\partial}{\partial \theta}(y_i - \hat{y}_i)^2$$

$$= \frac{1}{2n}\sum_{i=1}^{n}2(y_i - \hat{y}_i)\cdot\frac{\partial}{\partial \theta}(y_i - \hat{y}_i)$$

$$= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)\cdot\left(-\frac{\partial \hat{y}_i}{\partial \theta}\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)\frac{\partial \hat{y}_i}{\partial \theta}$$

2. Gradients with respect to C_0 and C_1:

A) Gradient with respect to C_0: First, differentiate $\hat{y}_i = C_0 + C_1 e^{C_2 x_i}$ with respect to C_0:

$$\frac{\partial \hat{y}_i}{\partial C_0} = \frac{\partial}{\partial C_0}\left(C_0 + C_1 e^{C_2 x_i}\right) = 1 + 0 = 1$$

Substituting this back into the chain rule formula:

$$\boxed{\frac{\partial \mathcal{L}}{\partial C_0} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \left(C_0 + C_1 e^{C_2 x_i}\right)\right)}$$

B) Gradient with respect to C_1: First, differentiate $\hat{y}_i = C_0 + C_1 e^{C_2 x_i}$ with respect to C_1:

$$\frac{\partial \hat{y}_i}{\partial C_1} = \frac{\partial}{\partial C_1}\left(C_0 + C_1 e^{C_2 x_i}\right) = 0 + e^{C_2 x_i} = e^{C_2 x_i}$$

Substituting this back into the chain rule formula:

$$\boxed{\frac{\partial \mathcal{L}}{\partial C_1} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \left(C_0 + C_1 e^{C_2 x_i}\right)\right)e^{C_2 x_i}}$$

3. Gradient with respect to C_2:

First, differentiate $\hat{y}_i = C_0 + C_1 e^{C_2 x_i}$ with respect to C_2:

$$\frac{\partial \hat{y}_i}{\partial C_2} = \frac{\partial}{\partial C_2}\left(C_0 + C_1 e^{C_2 x_i}\right) = 0 + C_1\frac{\partial}{\partial C_2}\left(e^{C_2 x_i}\right)$$

Using the exponential derivative chain rule $\frac{d}{du}(e^u) = e^u \frac{du}{dx}$:

$$\frac{\partial}{\partial C_2}\left(e^{C_2 x_i}\right) = e^{C_2 x_i} \cdot \frac{\partial}{\partial C_2}(C_2 x_i) = x_i e^{C_2 x_i}$$

$$\Rightarrow \frac{\partial \hat{y}_i}{\partial C_2} = C_1 x_i e^{C_2 x_i}$$

Substituting this back into the chain rule formula:

$$\boxed{\frac{\partial \mathcal{L}}{\partial C_2} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \left(C_0 + C_1 e^{C_2 x_i}\right)\right) C_1 x_i e^{C_2 x_i}}$$

4. Summary of Complete Gradients:

$$\frac{\partial \mathcal{L}}{\partial C_0} = -\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

$$\frac{\partial \mathcal{L}}{\partial C_1} = -\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)e^{C_2 x_i}$$

$$\frac{\partial \mathcal{L}}{\partial C_2} = -\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)C_1 x_i e^{C_2 x_i}$$

Task 3 — Parameter Update Rules

In each iteration t of Gradient Descent, the model parameters are updated simultaneously by moving in the negative direction of their respective gradients, scaled by the learning rate η:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L}$$

Specifically, for each coefficient:

$$C_0^{(t+1)} = C_0^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial C_0}$$

$$C_1^{(t+1)} = C_1^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial C_1}$$

$$C_2^{(t+1)} = C_2^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial C_2}$$

Where:

- $C_0^{(t)}, C_1^{(t)}, C_2^{(t)}$: Current coefficient values at iteration t
- η: Learning rate (step size hyperparameter)
- $\frac{\partial \mathcal{L}}{\partial C_0}, \frac{\partial \mathcal{L}}{\partial C_1}, \frac{\partial \mathcal{L}}{\partial C_2}$: Partial derivative gradients calculated over all n data points

Task 4 — Gradient Descent Implementation

We will now implement the Gradient Descent algorithm in Python using NumPy based on our derived mathematical formulas. **Hyperparameters & Initial Conditions:**

- Learning Rate (η) = '1.0'
- Maximum Iterations ('max_iterations') = '2000'
- Convergence Tolerance ('tolerance') = '1e-6'
- Initial Parameter Guesses: $C_0 = 0.5, C_1 = 0.5, C_2 = 0.1$

In [4]:

```python
# Initialize hyperparameters
learning_rate = 1.0
max_iterations = 2000
tolerance = 1e-6

# Initialize coefficients
C0, C1, C2 = 0.5, 0.5, 0.1
n = len(X)

# Lists to store history for visualization
history_loss = []
history_C0, history_C1, history_C2 = [], [], []

# Gradient Descent Loop
for i in range(max_iterations):
    # Calculate predictions using the exponential model
    exp_term = np.exp(C2 * X)
    Y_pred = C0 + C1 * exp_term

    # Calculate Error
    error = Y - Y_pred

    # Calculate MSE Loss (using 1/(2n))
    loss = (1 / (2 * n)) * np.sum(error**2)

    # Store history
    history_loss.append(loss)
    history_C0.append(C0)
    history_C1.append(C1)
    history_C2.append(C2)

```

```python
    # Calculate Gradients
    grad_C0 = -(1 / n) * np.sum(error)
    grad_C1 = -(1 / n) * np.sum(error * exp_term)
    grad_C2 = -(1 / n) * np.sum(error * C1 * X * exp_term)

    # Update coefficients
    C0 -= learning_rate * grad_C0
    C1 -= learning_rate * grad_C1
    C2 -= learning_rate * grad_C2

    # Early stopping condition
    if i > 0 and abs(history_loss[-1] - history_loss[-2]) < tolerance:
        print(f"Converged at iteration {i}")
        break

print(f"Training completed in {len(history_loss)} iterations.")
print(f"Final Parameters: C0 = {C0:.4f}, C1 = {C1:.4f}, C2 = {C2:.4f}")
print(f"Final MSE Loss: {history_loss[-1]:.4f}")
```

Out[4]:

```
Converged at iteration 269
Training completed in 270 iterations.
Final Parameters: C0 = 0.9927, C1 = 0.9587, C2 = -1.3296
Final MSE Loss: 0.0005
```

Task 5 — Results & Visualizations

In this section, we analyze the training behavior and performance of our model using 4 diagnostic plots: 1. **Loss Function Curve:** Convergence over iterations 2. **Parameter Trajectory:** Evolution of C_0, C_1, C_2 3. **Fitted Model vs Data:** Actual observations vs predicted curve 4. **Residual Plot:** Distribution of errors $(y_i - \hat{y}_i)$

In [5]:

```python
# Save individual plots to PDF first (before creating subplots which resets state)
# Plot 1: Learning Curve
plt.figure(figsize=(8, 5))
plt.plot(range(len(history_loss)), history_loss, 'b-', linewidth=2)
plt.xlabel('Iteration')
plt.ylabel('MSE Loss')
```

```python
plt.title('Loss Function Over Iterations')
plt.grid(True, linestyle='--', alpha=0.6)
plt.yscale('log')
plt.savefig('plot_2_loss_curve.pdf', bbox_inches='tight')
plt.close()

# Plot 2: Convergence of parameters
plt.figure(figsize=(8, 5))
plt.plot(range(len(history_C0)), history_C0, 'r-', label='C0', linewidth=2)
plt.plot(range(len(history_C1)), history_C1, 'g-', label='C1', linewidth=2)
plt.plot(range(len(history_C2)), history_C2, 'b-', label='C2', linewidth=2)
plt.xlabel('Iteration')
plt.ylabel('Parameter Value')
plt.title('Parameter Evolution During Training')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.savefig('plot_3_coeff_trajectory.pdf', bbox_inches='tight')
plt.close()

# Plot 3: Fitted Curve vs Actual Data
plt.figure(figsize=(8, 5))
x_line = np.linspace(min(X), max(X), 100)
y_line = C0 + C1 * np.exp(C2 * x_line)
plt.scatter(X, Y, color='blue', alpha=0.6, label='Actual Data', edgecolor='k')
plt.plot(x_line, y_line, 'r-', linewidth=3, label=f'Fitted Model: $\\hat{{y}} = {C0:.2f} + {C1:.2f}e^{{{C2:.2f}x}}$')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Original Data vs Fitted Model')
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.savefig('plot_4_fitted_curve.pdf', bbox_inches='tight')
plt.close()

# Plot 4: Residuals
plt.figure(figsize=(8, 5))
Y_pred_final = C0 + C1 * np.exp(C2 * X)
residuals = Y - Y_pred_final
```

```python
plt.scatter(X, residuals, color='purple', alpha=0.6, edgecolor='k')
plt.axhline(0, color='red', linestyle='--', linewidth=2)
plt.xlabel('X')
plt.ylabel('Residuals ($y_i - \\hat{y}_i$)')
plt.title('Residual Plot')
plt.grid(True, linestyle='--', alpha=0.6)
plt.savefig('plot_5_residuals.pdf', bbox_inches='tight')
plt.close()

# Now create the combined 2x2 grid for notebook display
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 12))

# Plot 1: Learning Curve (Loss over Iterations)
ax1.plot(range(len(history_loss)), history_loss, 'b-', linewidth=2)
ax1.set_xlabel('Iteration')
ax1.set_ylabel('MSE Loss')
ax1.set_title('Loss Function Over Iterations')
ax1.grid(True, linestyle='--', alpha=0.6)
ax1.set_yscale('log')

# Plot 2: Convergence of parameters
ax2.plot(range(len(history_C0)), history_C0, 'r-', label='C0', linewidth=2)
ax2.plot(range(len(history_C1)), history_C1, 'g-', label='C1', linewidth=2)
ax2.plot(range(len(history_C2)), history_C2, 'b-', label='C2', linewidth=2)
ax2.set_xlabel('Iteration')
ax2.set_ylabel('Parameter Value')
ax2.set_title('Parameter Evolution During Training')
ax2.legend()
ax2.grid(True, linestyle='--', alpha=0.6)

# Plot 3: Fitted Curve vs Actual Data
ax3.scatter(X, Y, color='blue', alpha=0.6, label='Actual Data', edgecolor='k')
ax3.plot(x_line, y_line, 'r-', linewidth=3, label=f'Fitted Model: $\\hat{{y}} = {C0:.2f} + {C1:.2f}e^{{{C2:.2f}x}}$')
ax3.set_xlabel('X')
ax3.set_ylabel('Y')
```

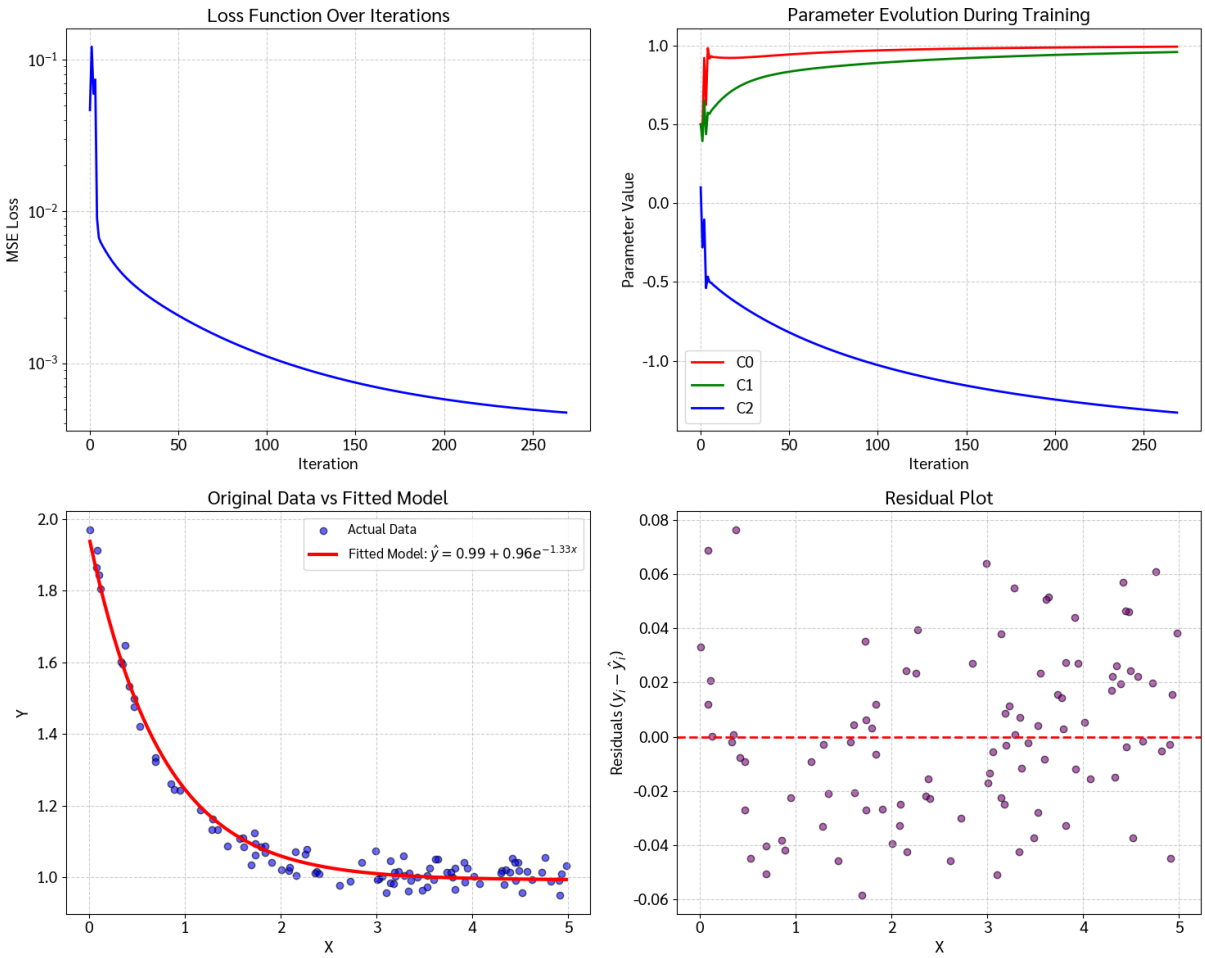
```python
ax3.set_title('Original Data vs Fitted Model')
ax3.legend(fontsize=12)
ax3.grid(True, linestyle='--', alpha=0.6)

# Plot 4: Residuals
ax4.scatter(X, residuals, color='purple', alpha=0.6, edgecolor='k')
ax4.axhline(0, color='red', linestyle='--', linewidth=2)
ax4.set_xlabel('X')
ax4.set_ylabel('Residuals ($y_i - \\hat{y}_i$)')
ax4.set_title('Residual Plot')
ax4.grid(True, linestyle='--', alpha=0.6)

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

Out[5]:

```
<>:31: SyntaxWarning: invalid escape sequence '\h'
<>:47: SyntaxWarning: invalid escape sequence '\h'
<>:76: SyntaxWarning: invalid escape sequence '\h'
<>:87: SyntaxWarning: invalid escape sequence '\h'
<>:31: SyntaxWarning: invalid escape sequence '\h'
<>:47: SyntaxWarning: invalid escape sequence '\h'
<>:76: SyntaxWarning: invalid escape sequence '\h'
<>:87: SyntaxWarning: invalid escape sequence '\h'
C:\Users\Admin\AppData\Local\Temp\ipykernel_1076\3103374935.py:31:
   SyntaxWarning: invalid escap ...
  plt.plot(x_line, y_line, 'r-', linewidth=3, label=f'Fitted Model:
      $\hat{{y}} = {C0:.2f} + {C1 ...
C:\Users\Admin\AppData\Local\Temp\ipykernel_1076\3103374935.py:47:
   SyntaxWarning: invalid escap ...
  plt.ylabel('Residuals ($y_i - \hat{y}_i$)')
C:\Users\Admin\AppData\Local\Temp\ipykernel_1076\3103374935.py:76:
   SyntaxWarning: invalid escap ...
  ax3.plot(x_line, y_line, 'r-', linewidth=3, label=f'Fitted Model:
      $\hat{{y}} = {C0:.2f} + {C1 ...
C:\Users\Admin\AppData\Local\Temp\ipykernel_1076\3103374935.py:87:
   SyntaxWarning: invalid escap ...
  ax4.set_ylabel('Residuals ($y_i - \hat{y}_i$)')
```



Extra: Model Evaluation (R-squared & Summary Statistics)

To mathematically evaluate the goodness of fit for our machine learning model, we calculate the Coefficient of Determination, also known as R^2. The formula for R^2 is:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Where:

- **Residual Sum of Squares (SS_{res})** is the sum of the squared differences between the actual and predicted values:

$$SS_{res} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- **Total Sum of Squares (SS_{tot})** is the sum of the squared differences between the actual values and the mean of the actual values ($\bar{y}$):

$$SS_{tot} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

A higher R^2 score (closer to 1) indicates that our model's predictions closely match the actual data points.

In [6]:

```python
# Print summary statistics
print("\n=== MODEL EVALUATION SUMMARY ===")
print(f"Fitted parameters: C0 = {C0:.6f}, C1 = {C1:.6f}, C2 = {C2:.6f}")
print(f"Total Iterations: {len(history_loss)}")
print(f"Initial loss: {history_loss[0]:.6f}")
print(f"Final loss: {history_loss[-1]:.6f}")
loss_reduction = (history_loss[0] - history_loss[-1]) / history_loss[0] * 100
print(f"Loss reduction: {loss_reduction:.2f}%")

# Calculate R-squared
Y_mean = np.mean(Y)
ss_res = np.sum((Y - Y_pred_final)**2)
ss_tot = np.sum((Y - Y_mean)**2)
r_squared = 1 - (ss_res / ss_tot)
print(f"R-squared (R^2): {r_squared:.6f}")
```

Out[6]:

```
=== MODEL EVALUATION SUMMARY ===
Fitted parameters: C0 = 0.992665, C1 = 0.958671, C2 = -1.329637
Total Iterations: 270
Initial loss: 0.046463
Final loss: 0.000474
Loss reduction: 98.98%
R-squared (R^2): 0.981770
```