# 實驗物理學（二）
# 實驗結報

# Fundamental Python
# Curve Fit

Group 2

洪　瑜 B125090009
黃巧涵 B122030003
洪懌平 B102030019

2025/04/22

## 摘要

This week, we did an exercise to fit a random dataset with an exponential model using `curve_fit`, probing for the best-fit parameters and their uncertainties estimated by the covariance matrix. Moreover, we reanalyze the previous dataset of a simple pendulum using the method from the exercise. These practices enhance our ability in programming and data analysis.


# 1 前言

In this week's coding practice, we performed an exercise in which we fitted a random dataset with an exponential model using `scipy.optimize.curve_fit`. Also, we estimated the uncertainty of each parameter by analyzing the covariance matrix given by the `scipy.optimize.curve_fit`, investigating the difference between `absolute_sigma = True` and `absolute_sigma = False`.

The following subsections briefly introduce the usage of `scipy.optimize.curve_fit` and the concept of the covariance matrix.

## 1.1 Curve fitting with `scipy.optimize.curve_fit`

`scipy.optimize.curve_fit` is a function from the SciPy library used for fitting a curve to data using non-linear least squares. It allows you to fit a user-defined model to a set of data points by optimizing the model parameters to minimize the error between the model predictions and the observed data. The fitting process involves iteratively adjusting the model parameters to reduce the sum of squared residuals, which is the difference between the model and actual data.

## 1.2 Covariance matrix and uncertainty

In data analysis, particularly in the context of parameter estimation and model fitting, the covariance matrix and uncertainty of each parameter are essential concepts used to assess the reliability and relationship of the parameters being estimated.

The covariance matrix is a square matrix that provides a measure of the covariance (i.e., how two variables change together) between each pair of parameters in a multi-dimensional dataset. It is advantageous when you have multiple parameters that are estimated simultaneously.

For example, if we consider a two-parameter model with parameters $a$ and $b$, the covariance matrix is

$$\mathrm{pcov} = \begin{pmatrix} \sigma_a^2 & \mathrm{cov}(b,a) \\ \mathrm{cov}(a,b) & \sigma_b^2 \end{pmatrix} \tag{1}$$

In the covariance matrix, the diagonal elements represent the variance of each parameter, and the off-diagonal elements are the covariance between each pair of parameters. Hence, the uncertainty of each parameter can be determined by the square root of the diagonal elements of the covariance matrix.

To be mentioned, the `pcov` given by the `scipy.optimize.curve_fit` can be different with `absolute_sigma = True` and `absolute_sigma = False` since `absolute_sigma = False`, the default choice, self-adjusts the covariance matrix by multiplying the calculated covariance matrix (`absolute_sigma = True`) with the reduced chi-squared $\chi^2_\nu$, i.e.,

$$\mathrm{pcov}_{False} = \sqrt{\chi^2_\nu}\,\mathrm{pcov}_{True} \tag{2}$$

The reason for this design is to force the reported uncertainties to match the observed variance in the residuals, so that the estimated $\chi^2_\nu$ effectively becomes unity.

## 2 實驗步驟

使用講義之參數、條件：

- 設定隨機種子爲42
- 產生50個0-10之間的等間距之$x$值
- 定義用來擬合的模型，透過`curve_fit`找到最適合的$A$和$k$
- 建立模型：$y = 2.5e^{-1.3x}$（$A = 2.5$，$k = 1.3$），並在每一個$y$值上加入一高斯隨機誤差（平均爲0，標準差爲1）
- 每個點的誤差設爲$\pm 0.1$

程式碼詳情見Fig.1、Fig.2之註解。

```python
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

#隨機種子&資料
np.random.seed(42)
x = np.linspace(0, 10, 50)
#y誤差相同，固定為0.1
y = 2.5 * np.exp(-1.3*x)+np.random.normal(0, 0.1, x.size)
sigma = 0.1 * np.ones_like(y)

#model
def model(x, A, k):
    return A * np.exp(-k*x)

#曲線的兩種設定
popt_true,pcov_true = curve_fit(model, x, y, sigma=sigma, absolute_sigma=True)
popt_false,pcov_false = curve_fit(model, x, y, sigma=sigma, absolute_sigma=False)

# 殘差計算
residuals_true = y - model(x, *popt_true)
residuals_false = y - model(x, *popt_false)

# 擬合曲線
x_fit = np.linspace(0, 10, 200)
y_fit_true = model(x_fit, *popt_true)
y_fit_false = model(x_fit, *popt_false)

```

Figure 1: 程式碼(1)

```python
# 繪圖：資料與擬合結果
plt.figure(figsize=(10, 8))

plt.subplot(2, 1, 1)
plt.errorbar(x, y, yerr=sigma, fmt='o', capsize=3, label='Data with error bars')
plt.plot(x_fit, y_fit_true, 'r-', label='Fit (absolute_sigma=True)')
plt.plot(x_fit, y_fit_false, 'g--', label='Fit (absolute_sigma=False)')
plt.title('Curve Fitting with Fixed Sigma = 0.1')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()

# 繪圖：殘差比較
plt.subplot(2, 1, 2)
plt.plot(x, residuals_true, 'ro-', label='Residuals (True)')
plt.plot(x, residuals_false, 'go--', label='Residuals (False)')
plt.axhline(0, color='gray', linestyle='--')
plt.xlabel('x')
plt.ylabel('Residuals')
plt.legend()

plt.tight_layout()
plt.show()

# 列印擬合參數與不確定度
perr_true = np.sqrt(np.diag(pcov_true))
perr_false = np.sqrt(np.diag(pcov_false))

print("=== absolute_sigma=True ===")
print(f"A = {popt_true[0]:.4f} ± {perr_true[0]:.4f}")
print(f"k = {popt_true[1]:.4f} ± {perr_true[1]:.4f}")

print("\n=== absolute_sigma=False ===")
print(f"A = {popt_false[0]:.4f} ± {perr_false[0]:.4f}")
print(f"k = {popt_false[1]:.4f} ± {perr_false[1]:.4f}")
```

Figure 2: 程式碼(2)

1. 設定講義之參數、建立給定模型。
2. 透過`curve_fit`擬合。
3. 繪出擬合曲線及殘差圖。

# 3 實驗數據與分析

```
=== absolute_sigma=True ===
A = 2.5626 ± 0.0808
k = 1.2977 ± 0.0657

=== absolute_sigma=False ===
A = 2.5626 ± 0.0776
k = 1.2977 ± 0.0631
```

Figure 3: 最佳擬合參數

輸出：
`absolute_sigma=True`

- $A = 2.5626 \pm 0.0808$
- $k = 1.2977 \pm 0.0657$

`absolute_sigma=False`

- $A = 2.5626 \pm 0.0776$
- $k = 1.2977 \pm 0.0631$

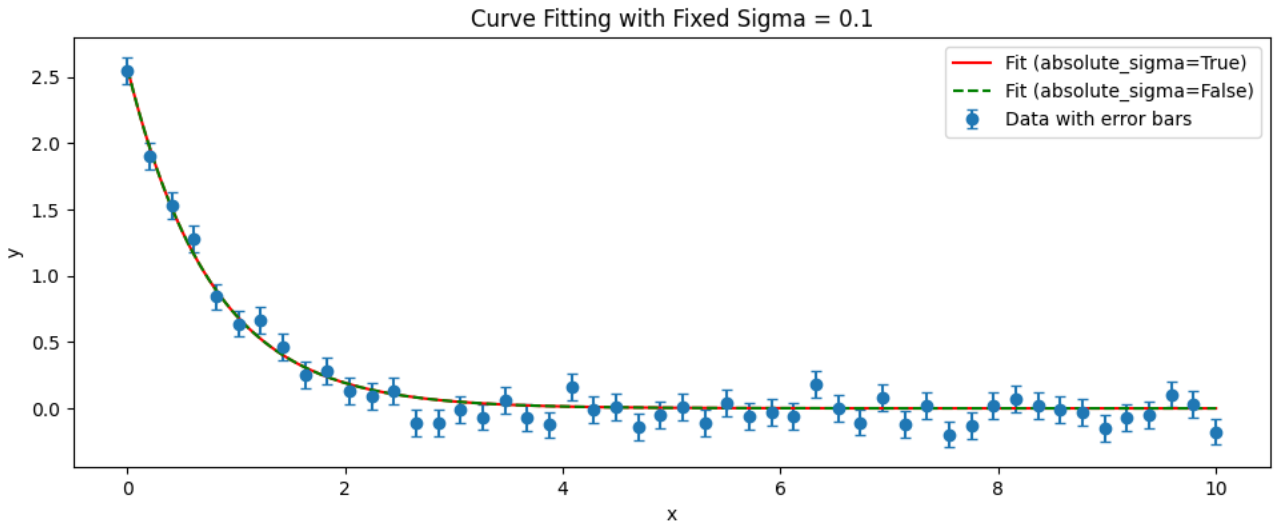


Figure 4: original data with error bars

透過Fig.4可以確認擬合出的曲線是否貼近資料點。

同時也可看出，`absolute_sigma=True`與`absolute_sigma=False`的曲線高度重疊，這是因爲 `absolute_sigma` 並不會影響擬合曲線，而只會影響參數的誤差估計。

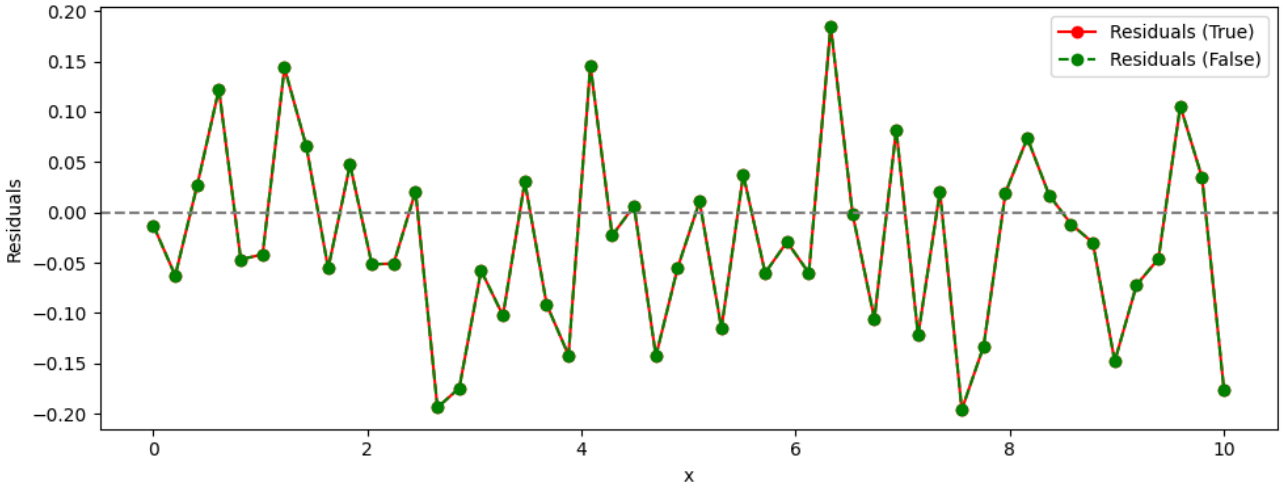


Figure 5: residuals for each of the two cases separately

殘差圖可以比較這兩種擬合之下資料與模型之間的偏差。可以看出二者的殘差分布皆沒有明顯趨勢，且分布於0附近，顯示出模型與資料吻合程度相當高，可以應證absolute_sigma只會影響參數的誤差估計而不影響擬合結果與殘差本身。

# 4 問題討論

## 4.1 First, average the five period measurements for each pendulum length and obtain the error bar (standard deviation), then perform the fitting.

回顧上週內容，我們所做的$\chi^2$ fitting的程式碼（如下Fig.6到Fig.10）

```
# Define the chi-squared function
def chi_sq_optimization(params, x_data, y_data, sigma):
    model = linear_function(x_data, *params)  # Use your model function
    return chi_sq(y_data, model, sigma)

# Initial guess for the parameters
initial_guess = [0.5, 0.5]  # Adjust based on your model

# Perform the optimization
result = minimize(chi_sq_optimization, initial_guess,
                  args=(np.log(pendulum_length_matrix), np.log(period_1swing.flatten()), np.log(np.tile(std_1period, 5))))

# Extract the optimized parameters
optimized_params = result.x
m_chisq = optimized_params[0]
b_chisq = optimized_params[1]
print("Optimized parameters:", optimized_params)
```

Figure 6: $\chi^2$ fitting程式碼(1)

```
plt.scatter(np.log(pendulum_length_matrix), np.log(period_1swing.flatten()), s=50, marker='x', color='silver', label='Experiment')
plt.errorbar(np.log(pendulum_length), np.log(avg_1period), yerr=std_1period_log, fmt='o', label='Avg T', color='k', capsize=5)
# plt.plot(np.log(l_axis), linear_function(np.log(l_axis), *propt), color='orangered',linestyle='dashed',
#          label=f'curve_fit')
plt.plot(np.log(l_axis), np.log(t_ideal), label='Theoretical T', color='teal', linestyle='--')
# plt.plot(np.log(l_axis), np.dot(np.vander(np.log(l_axis), 2), w), "--k", label="LS")
plt.plot(np.log(l_axis), linear_function(np.log(l_axis), *optimized_params), linestyle="-", color='goldenrod', label="Chi-squared")
plt.xlabel('log(Length) [m]')
plt.ylabel('log(Duration of a swing) [s]')
plt.title('Duration of a swing vs Length (Log-Log)')
plt.xlim(left=np.log(0.05), right=np.log(0.65))
plt.ylim(bottom=np.log(0.5), top=np.log(1.7))
plt.legend()
plt.savefig('./figures/chisq_log.pdf', transparent=True)
plt.show()
```

Figure 7: $\chi^2$ fitting程式碼(2)

Fig.6是先定義$\chi^2$ function，並定義：

- params爲我們要最佳化的參數（直線之斜率或截距）
- $x_data$和$y_data$ 爲輸入數據
- sigma爲測量不確定性
- linear_function($x\_data$, params) 爲一條直線（$y = ax + b$）
- chi_sq(y_data, model, sigma) 爲計算$\chi^2$

接著設定初始參數，先猜測m=0.5、b=0.5；並進行最小化$\chi^2$。
而因爲實驗要求擬合擺動週期與擺長的對數關係，所以需取對數，使其變線性關係（詳情見預習問題1、2）即可求得a、b。最後把最佳化後的m、b，和$\chi^2$值顯示出來。
最終輸出結果：

- Optimized parameters: [0.48023142 0.79271392]
- Chi-square value: 0.15251356436613928
- *reduced* $\chi^2 \approx 0.0019$

```
plt.scatter(pendulum_length_matrix, period_1swing.flatten(), s=50, marker='x', color='silver', label='Experiment')
plt.errorbar(pendulum_length, avg_1period, yerr=std_1period, fmt='o', label='Avg T', color='k', capsize=5)
# plt.plot(l_axis, np.e**linear_function(np.log(l_axis), *propt), color='orangered',linestyle='dashed',
#          label=f'curve_fit')
plt.plot(l_axis, t_ideal, label='Theoretical T', color='teal', linestyle='--')
# plt.plot(l_axis, np.e**np.dot(np.vander(np.log(l_axis), 2), w), "--k", label="LS")
plt.plot(l_axis, np.e**linear_function(np.log(l_axis), *optimized_params), linestyle="-", color='goldenrod', label="Chi-squared")
plt.xlabel('Length [m]')
plt.ylabel('Duration of a swing [s]')
plt.title('Duration of a swing vs Length')
plt.xlim(left=0.07, right=0.6)
plt.ylim(bottom=0.5, top=1.6)
plt.legend()
plt.savefig('./figures/chisq.pdf', transparent=True)
plt.show()
```

Figure 8: $\chi^2$ fitting程式碼(3)

Fig.8使用`plt.scatter()`繪製實驗數據點；x軸爲擺長取對數；y軸爲單次擺動週期取對數。利用交叉符號點出個別數據點。

接著繪出理論曲線和$\chi^2$擬合的曲線，並設定座標軸與標題及範圍；結果爲Fig.11右圖。

```
g_fitted = np.e**(2*(np.log(2*np.pi) - b_chisq))
print(f'Ideal g = {g0.value:.3f} m/s^2')
print(f'Fitted g = {g_fitted:.3f} m/s^2')
print(f'Error = {np.abs(g_fitted - g0.value) / g0.value * 100:.3f}%')
```

Figure 9: $\chi^2$ fitting程式碼(4)

Fig.9繪製擺長與擺動週期的關係圖，但和上部分的對數-對數圖不同，這次是線性尺度。

令x軸爲擺長、y軸爲擺動週期，用銀色交叉符號標出實驗數據，並繪出error bar（使用黑色圓點，表測量之不確定性）；最後劃出理論曲線和$\chi^2$理論曲線與我們擬合出的結果（Fig.11左圖）。

```
chi_sq_val = chi_sq_optimization(optimized_params, L, np.log(periods), sigma)
print("Chi-square value:", chi_sq_val)
```

Figure 10: $\chi^2$ fitting程式碼(5)

Fig.10透過$\chi^2$擬合的結果估算重力加速度$g$，並和理論值（$g_0$）比較。

結果輸出爲：

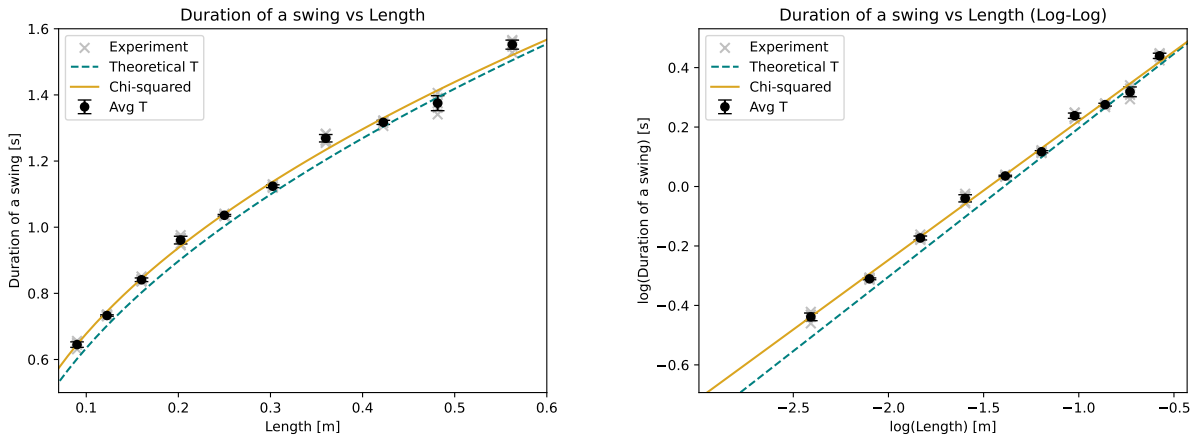


Figure 11: 數據平均後$\chi^2$擬合結果（左圖爲線性擬合；右圖爲對數擬合）

```
方法一輸出：
Ideal g = 9.807 m/s^2
Fitted g = 9.969 m/s^2
Error = 1.656%
```

Figure 12: 使用數據平均後擬合推測之重力加速度

殘差圖如下：

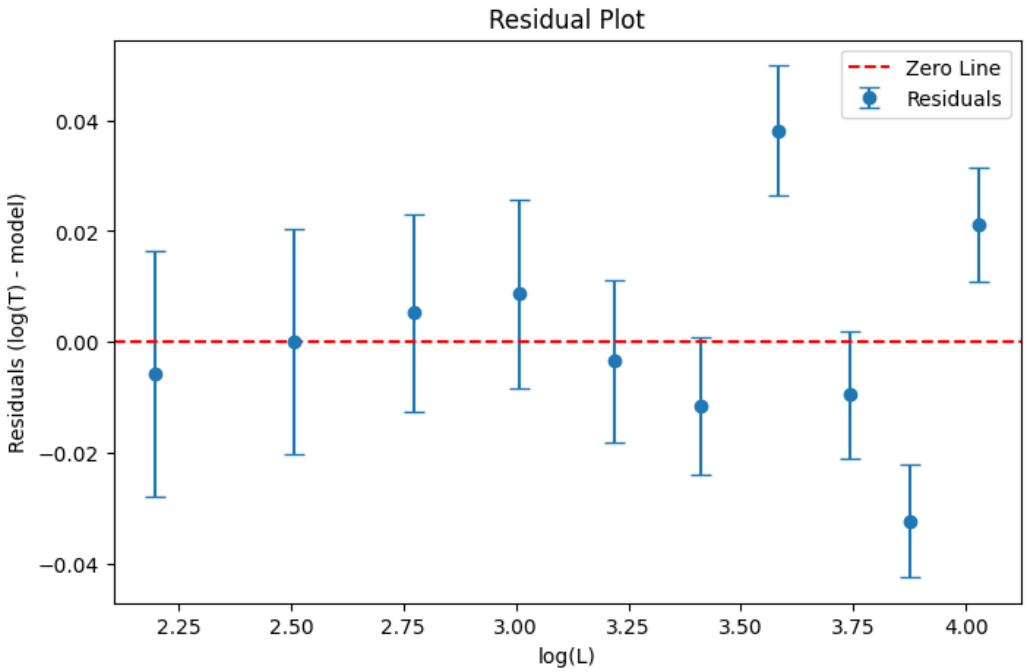


Figure 13: 數據平均後擬合之殘差圖

## 4.2 Fit all 25 data points using the errors obtained by the method mentioned above for $\chi^2$ fitting.

我們藉由第一題的基礎下再加上以下的程式碼（見Fig.14）

```
#初始猜測
initial_guess = [0.5, 0.5]

#最小化
result_all = minimize(chi_sq_optimization, initial_guess, args=(log_L_all, log_T_all, sigma_log_T_all))
optimized_params_all = result_all.x
m_all, b_all = optimized_params_all

#計算法二卡方值
chisq_all = chi_sq(log_T_all, linear_function(log_L_all, m_all, b_all), sigma_log_T_all)
dof_all = len(log_L_all) - 2
print(f"最佳擬合參數: m = {m_all:.4f}, b = {b_all:.4f}")
print(f"卡方值: {chisq_all:.2f}")
print(f"reduced chi-squared: {chisq_all/dof_all:.2f}")
```

Figure 14: 將25個數據點做擬合-程式碼(1)

得到輸出如下：

- 最佳擬合參數:$m = 0.4631, b = 0.6802$
- 卡方值: 1890.42
- reduced $\chi^2$: 39.38

將結果會製成圖：

```
l_axis = np.linspace(min(L), max(L), 200)
log_l_axis = np.log(l_axis)

plt.figure(figsize=(8,6))
plt.errorbar(log_L_all, log_T_all, yerr=sigma_log_T_all, fmt='x', color='silver', label='Raw Data', capsize=5)
plt.plot(log_l_axis, linear_function(log_l_axis, *optimized_params_all), label='Chi-squared Fit (All)', color='darkorange')
plt.xlabel('log(Length) [log(m)]')
plt.ylabel('log(Period) [log(s)]')
plt.title('Chi-squared Fit (All Data Points)')
plt.legend()
#plt.grid(True)
#plt.savefig('./figures/chisq_all_loglog.pdf', transparent=True)
plt.show()
```

Figure 15: 將25個數據點做擬合-程式碼(2)

使用對數模型繪出對數-對數圖，輸出見Fig.18右圖。

```python
import matplotlib.pyplot as plt

# 畫圖(linear mode)
plt.figure(figsize=(8,6))

plt.errorbar(L_all, T_all, yerr=sigma_T_all, fmt='x', color='darkblue', label='Raw Data', capsize=5)

L_axis = np.linspace(min(L_all), max(L_all), 200)
T_fit = np.exp(optimized_params_all[1]) * L_axis**optimized_params_all[0]

plt.plot(L_axis, T_fit, label='Chi-squared Fit (All)', color='darkorange')

plt.xlabel('Length [m]')
plt.ylabel('Period [s]')
plt.title('Chi-squared Fit (All Data Points)')
plt.legend()
#plt.grid(True)
#plt.savefig('./figures/chisq_all_linear.pdf', transparent=True)
plt.show()
```

Figure 16: 將25個數據點做擬合-程式碼(3)

使用線性模型繪出成圖，輸出見Fig.18左圖。

```python
T_model = np.exp(optimized_params_all[1]) * L_all**optimized_params_all[0]
residuals = T_all - T_model

plt.figure(figsize=(8,6))
plt.axhline(0, color='black', linestyle='--')
plt.errorbar(L_all, residuals, yerr=sigma_T_all, fmt='o', color='teal', capsize=5, label='Residuals')
plt.xlabel('Length [m]')
plt.ylabel('Residual (Observed - Fitted) [s]')
plt.title('Residual Plot (All Data Points)')
plt.legend()
#plt.grid(True)
#plt.savefig('./figures/residuals_all.pdf', transparent=True)
plt.show()
```

Figure 17: 將25個數據點做擬合-程式碼(4)

繪出25個數據點的殘差圖，輸出見Fig.20

結果爲：

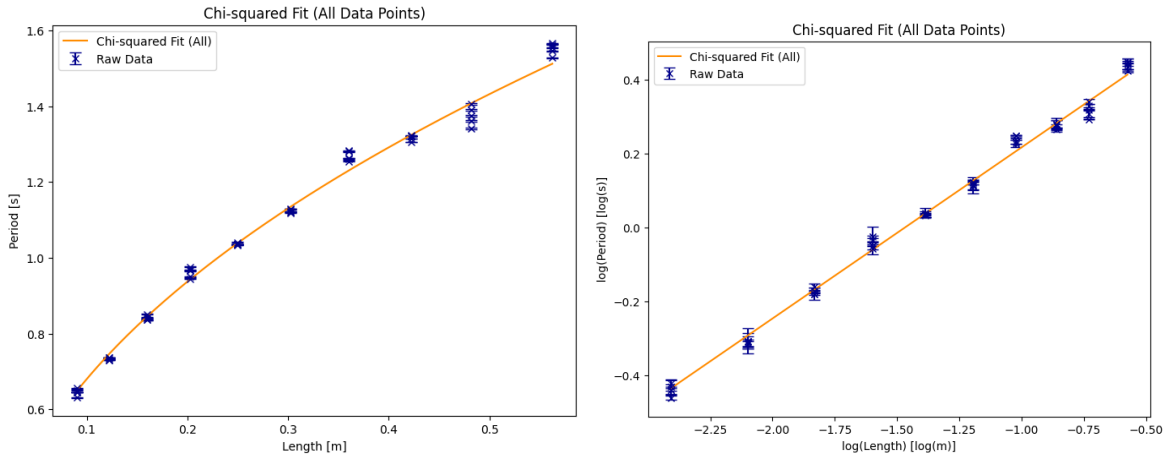


Figure 18: 25個數據$\chi^2$擬合結果（左圖爲線性擬合；右圖爲對數擬合）

```
方法二輸出：
Ideal g = 9.807 m/s^2
Fitted g = 10.129 m/s^2
Error = 3.283%
```

Figure 19: 使用25個數據點擬合後推測之重力加速度

殘差圖如下：

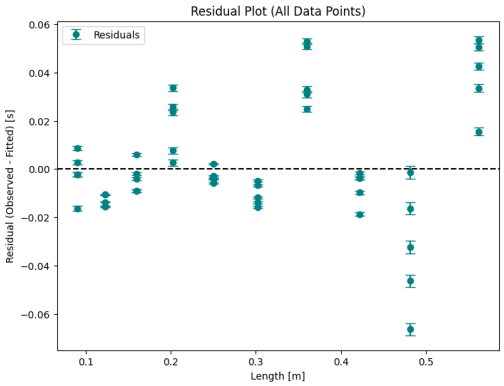


Figure 20: 25個數據之殘差圖

### 4.3 分析問題一、二之差距

比較「平均後擬合」與「直接用25筆資料擬合」的結果，可以發現「直接用25筆資料擬合」所得到的 reduced $\chi^2$ 明顯較大，且透過斜率推算出的重力加速度與理論值之誤差也更大，其原因在於每筆原始資料本身皆有測量誤差，且殘差分布較爲分散，造成擬合品質下降。

相較之下，「平均後擬合」藉由對每組資料取平均，有效降低了隨機誤差，使擬合結果更爲穩定且接近理論預期；而「直接用25筆資料擬合」雖然資料量較多，但誤差的累積與異常值的影響使得擬合結果偏離理論。

## 5 總結

本次實驗中，第一部分使用`scipy.optimize.curve_fit`函數擬合指數衰減模型，並分別設定`absolute_sigma=True`和`absolute_sigma=False`兩種情況來比較。

結果顯示兩種設定下擬合曲線高度重疊，代表`absolute_sigma`的設定不會影響擬合曲線本身，而是影響參數誤差的估計範圍。

從殘差圖可觀察出，兩種情況的殘差均無明顯的系統性偏差，且皆分布於0附近，表示擬合品質佳。本次模型物理上描述了一個指數衰減的過程，符合資料特性，而主要誤差來源爲每個資料點的高斯隨機誤差與統計波動。由於本次實驗中每筆資料的誤差設定一致，且權重相同，因此不論`absolute_sigma`設定爲True或False，最佳擬合參數（如A和k）幾乎一致，進而導致擬合曲線相同。差異僅體現在擬合後參數誤差（即±數值）的計算方式上。

關於`absolute_sigma`的影響，整理如下：

- `absolute_sigma=False`（預設值）：
  假設資料誤差可能存在低估或高估的情況，`curve_fit` 會根據殘差自動調整誤差大小，再以此計算參數的不確定性。

- `absolute_sigma=True`：
  假設提供的error bars已準確反映眞實誤差，因此擬合時不進行額外調整，直接使用輸入的誤差計算參數不確定性。

而第二部分的延伸實驗則有兩種$\chi^2$擬合結果比較：平均資料後再擬合以及直接使用單次測量資料擬合。結果發現前者的 reduced $\chi^2$ 值更接近理想值，且重力加速度的值也更接近理論值。

此次實驗延續前面實驗，繼續了解與學習誤差處理的方式，並透過了解不同的計算方式，學習分析並找出對於不同類型的實驗數據較適合的計算方式，日後能運用於眞實實驗數據的分析。

## 6 分工

- 洪瑜： $\chi^2$ fitting、誤差討論及問題討論
- 黃巧涵： 數據分析
- 洪懌平： 摘要、前言