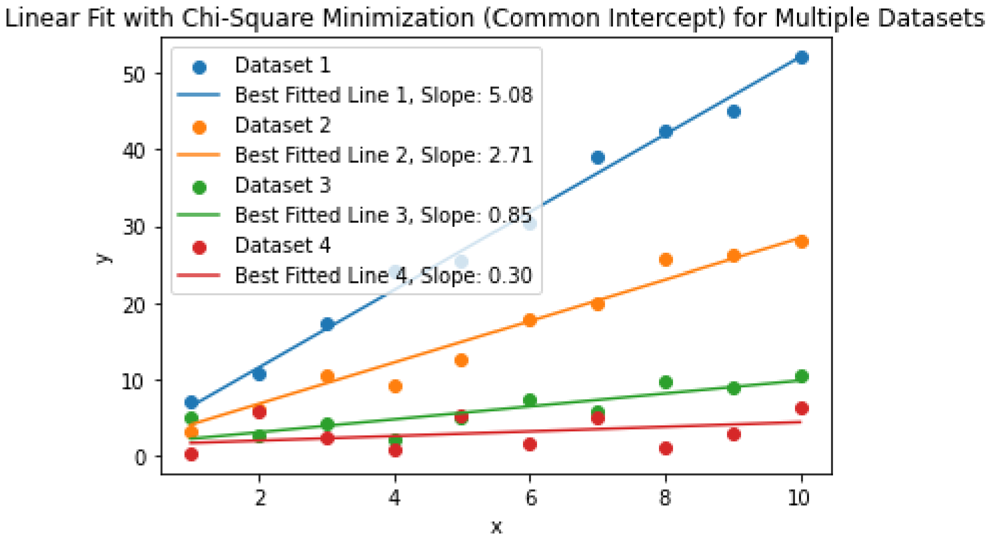
Solutions

1. Numerical solution to get both minimized slope and intercept of multiple datasets

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

# Given data
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
m_values = [5, 3, 1, 0.5]
c_true = 1
    avg_n = 0
    n_std = 2

    # Generate noisy data for each dataset with different noises
    np.random.seed(42)
    datasets = []

    for m in m_values:
        n = np.random.normal(avg_n, n_std, len(x))
        y_true = m * x + c_true + n
        datasets.append(y_true)

    # Define the chi-square function to minimize for each dataset, with common intercept
    def chi_square_fit(params, x, datasets, n_std):
        intercept, slopes = params[0], params[1:]
        total_chi_sq = 0

        for i, y_true in enumerate(datasets):
            y_model = slopes[i] * x + intercept
            chi_sq = np.sum(((y_true - y_model) / n_std) ** 2)
            total_chi_sq += chi_sq

        return total_chi_sq

    # Initial guess for intercept and slopes
    initial_guess = [c_true] + m_values

    # Minimize chi-square to find best-fitted slopes and common intercept
    result = minimize(chi_square_fit, initial_guess, args=(x, datasets, n_std), method='Nelder-Mead')
```

```python

    # Extract best-fitted intercept and slopes
    c_fitted = result.x[0]
    m_fitted_list = result.x[1:]

    # Plot the original data and the best-fitted lines for each dataset
    for i, y_true in enumerate(datasets):
        plt.scatter(x, y_true, label=f'Dataset {i + 1}')
        plt.plot(x, m_fitted_list[i] * x + c_fitted, label=f'Best Fitted Line {i + 1}, Slope: {m_fitted_list[i]:.2f}')

    # Display the plot
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Linear Fit with Chi-Square Minimization (Common Intercept) for Multiple Datasets')
    plt.legend()
    plt.show()

    # Display the results for each dataset
    for i, m_fitted in enumerate(m_fitted_list):
        print(f"\nDataset {i + 1}:")
        print("True Slope:", m_values[i])
        print("Common Fitted Intercept:", c_fitted)
        print("Fitted Slope:", m_fitted)

```

```
Dataset 1:
True Slope: 5
Common Fitted Intercept: 1.3464179718074534
Fitted Slope: 5.07557879162032

Dataset 2:
True Slope: 3
Common Fitted Intercept: 1.3464179718074534
Fitted Slope: 2.7053653974298286

Dataset 3:
True Slope: 1
Common Fitted Intercept: 1.3464179718074534
Fitted Slope: 0.8470491696971632

Dataset 4:
True Slope: 0.5
Common Fitted Intercept: 1.3464179718074534
      Fitted Slope: 0.3020330822000894
```



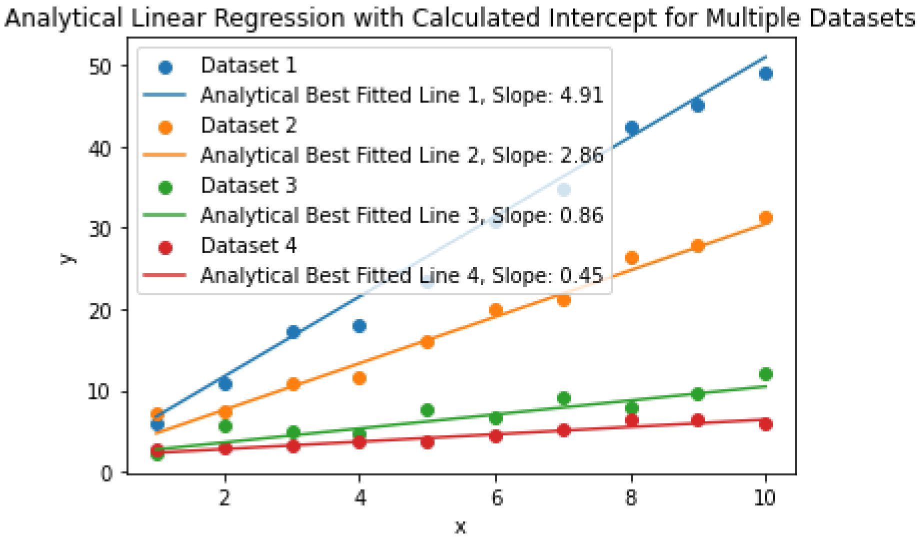
Single noise is added to every dataset.

2. Analytical solution of slope and common intercept of multiple datasets.

```
import numpy as np
import matplotlib.pyplot as plt

# Given data
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
m_values = [5, 3, 1, 0.5]
c_true = 1
avg_n = 0
n_std_values = [2, 1.5, 1, 0.5]

# Generate noisy data for each dataset with different noises
np.random.seed(42)
datasets = []

# Calculate the intercept for the first dataset
n_first_dataset = np.random.normal(avg_n, n_std_values[0], len(x))
y_first_dataset = m_values[0] * x + c_true + n_first_dataset
c_first_dataset = np.mean(y_first_dataset - m_values[0] * x)

for i, m in enumerate(m_values):
    n = np.random.normal(avg_n, n_std_values[i], len(x))
    y_true = m * x + c_first_dataset + n
    datasets.append(y_true)

# Analytical calculation for linear regression with a calculated
intercept
```

```python
def analytical_linear_regression(x, y):
    N = len(x)
    xy_sum = np.sum(x * y)
    x_sum = np.sum(x)
    y_sum = np.sum(y)
    x_squared_sum = np.sum(x ** 2)

    # Calculate the slope (m)
    m = (N * xy_sum - x_sum * y_sum) / (N * x_squared_sum - x_sum ** 2)

    # Calculate the intercept (c)
    c = (y_sum - m * x_sum) / N

    return m, c

# Initialize lists to store results
m_analytical_list = []

# Loop through each dataset and calculate analytical best-fitted slope
for i, y_values in enumerate(datasets):
    m_analytical, _ = analytical_linear_regression(x, y_values)

    # Append results to list
    m_analytical_list.append(m_analytical)

    # Plot the original data and the best-fitted line for each dataset
with a calculated intercept
    plt.scatter(x, y_values, label=f'Dataset {i + 1}')
    plt.plot(x, m_analytical * x + c_first_dataset, label=f'Analytical
Best Fitted Line {i + 1}, Slope: {m_analytical:.2f}')

# Display the plot
plt.xlabel('x')
plt.ylabel('y')
plt.title('Analytical Linear Regression with Calculated Intercept for
Multiple Datasets')
plt.legend()
plt.show()

# Display the analytical results for each dataset
for i, m_analytical in enumerate(m_analytical_list):
    print(f"\nDataset {i + 1} (Analytical Solution):")
    print("True Slope:", m_values[i])
    print("Calculated Intercept (c_first_dataset):", c_first_dataset)
    print("Analytical Fitted Slope:", m_analytical)
```

```
Dataset 1 (Analytical Solution):
True Slope: 5
Calculated Intercept (c_first_dataset): 1.8961222233975121
Analytical Fitted Slope: 4.910174165466983

Dataset 2 (Analytical Solution):
True Slope: 3
Calculated Intercept (c_first_dataset): 1.8961222233975121
Analytical Fitted Slope: 2.859793625585799

Dataset 3 (Analytical Solution):
True Slope: 1
Calculated Intercept (c_first_dataset): 1.8961222233975121
Analytical Fitted Slope: 0.8602989788674754

Dataset 4 (Analytical Solution):
True Slope: 0.5
Calculated Intercept (c_first_dataset): 1.8961222233975121
      Analytical Fitted Slope: 0.4540488319373424
```



Single noise is added to every dataset.

3. Error of slope and intercept derivation as well as solution from the code.

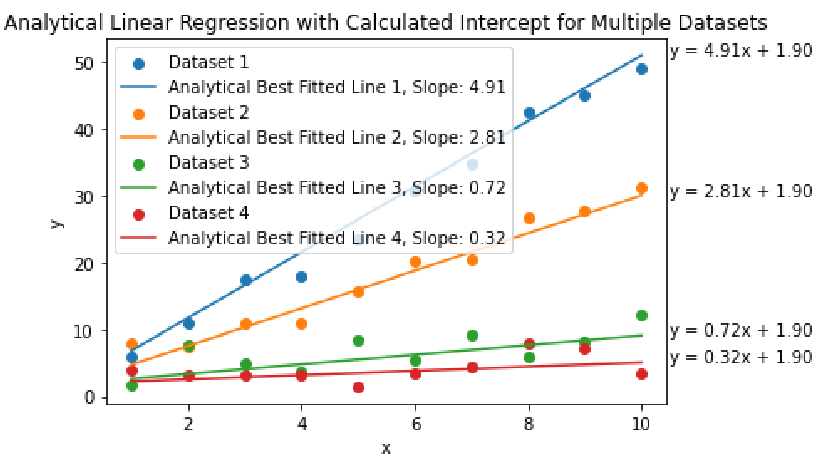
```python
import numpy as np
import matplotlib.pyplot as plt

# Given data
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

```
m_values = [5, 3, 1, 0.5]
c_true = 1
avg_n = 0
n_std_value = 2  # Use a single standard deviation value for all
datasets

# Generate noisy data for each dataset with the same standard deviation
np.random.seed(42)
datasets = []

# Calculate the intercept for the first dataset
n_first_dataset = np.random.normal(avg_n, n_std_value, len(x))
y_first_dataset = m_values[0] * x + c_true + n_first_dataset
c_first_dataset = np.mean(y_first_dataset - m_values[0] * x)

for i, m in enumerate(m_values):
    n = np.random.normal(avg_n, n_std_value, len(x))
    y_true = m * x + c_first_dataset + n
    datasets.append(y_true)

# Analytical calculation for linear regression with a calculated
intercept
def analytical_linear_regression(x, y, n_std):
    N = len(x)
    xy_sum = np.sum(x * y)
    x_sum = np.sum(x)
    y_sum = np.sum(y)
    x_squared_sum = np.sum(x ** 2)

    # Calculate the slope (m)
    m = (N * xy_sum - x_sum * y_sum) / (N * x_squared_sum - x_sum ** 2)

    # Calculate the intercept (c)
    c = (y_sum - m * x_sum) / N

    # Calculate the errors in slope and intercept using the partial
derivatives with respect to yi
    sigma_m_squared = np.sum(n_std**2 * ((N * x_sum - x_sum)/(N *
x_squared_sum - (x_sum)**2)**2))
    sigma_c_squared = np.sum(n_std**2/N**2)

    sigma_m = np.sqrt(sigma_m_squared)
    sigma_c = np.sqrt(sigma_c_squared)

    return m, c, sigma_m, sigma_c
```

```python
# Initialize lists to store results
m_analytical_list = []
c_analytical_list = []
sigma_m_list = []
sigma_c_list = []

# Loop through each dataset and calculate analytical best-fitted slope,
intercept, and errors
for i, y_values in enumerate(datasets):
    m_analytical, c_analytical, sigma_m, sigma_c =
analytical_linear_regression(x, y_values, n_std_value)

    # Append results to lists
    m_analytical_list.append(m_analytical)
    c_analytical_list.append(c_analytical)
    sigma_m_list.append(sigma_m)
    sigma_c_list.append(sigma_c)

    # Print the results for each dataset
    print(f"\nDataset {i + 1} (Analytical Solution):")
    print("True Slope:", m_values[i])
    print("Calculated Intercept (c_first_dataset):", c_first_dataset)
    print("Analytical Fitted Slope:", m_analytical)
    print("Error in Slope (sigma_m):", sigma_m)
    print("Error in Intercept (sigma_c):", sigma_c)

    # Plot the original data and the best-fitted line for each dataset
with a calculated intercept
    plt.scatter(x, y_values, label=f'Dataset {i + 1}')
    plt.plot(x, m_analytical * x + c_first_dataset, label=f'Analytical
Best Fitted Line {i + 1}, Slope: {m_analytical:.2f}')

    # Display the equation of the line on the plot
    plt.text(x[-1] + 0.5, m_analytical * x[-1] + c_first_dataset,
             f'y = {m_analytical:.2f}x + {c_first_dataset:.2f}',
color='black')

# Calculate common intercept and mean error in the intercept
common_intercept = np.mean(c_analytical_list)
mean_sigma_c = np.mean(sigma_c_list)

# Print common intercept and mean error in the intercept
print(f"\nCommon Fitted Intercept: {common_intercept}")
print(f"Mean Error in Intercept (sigma_c): {mean_sigma_c}")

# Display the plot with legend
```

```
plt.xlabel('x')
plt.ylabel('y')
plt.title('Analytical Linear Regression with Calculated Intercept for Multiple Datasets')
plt.legend()
plt.show()
```

```
Dataset 1 (Analytical Solution):
True Slope: 5
Calculated Intercept (c_first_dataset): 1.8961222233975121
Analytical Fitted Slope: 4.910174165466983
Error in Slope (sigma_m): 0.05393598899705936
Error in Intercept (sigma_c): 0.2

Dataset 2 (Analytical Solution):
True Slope: 3
Calculated Intercept (c_first_dataset): 1.8961222233975121
Analytical Fitted Slope: 2.813058167447728
Error in Slope (sigma_m): 0.05393598899705936
Error in Intercept (sigma_c): 0.2

Dataset 3 (Analytical Solution):
True Slope: 1
Calculated Intercept (c_first_dataset): 1.8961222233975121
Analytical Fitted Slope: 0.720597957734954
Error in Slope (sigma_m): 0.05393598899705936
Error in Intercept (sigma_c): 0.2

Dataset 4 (Analytical Solution):
True Slope: 0.5
Calculated Intercept (c_first_dataset): 1.8961222233975121
Analytical Fitted Slope: 0.3161953277493712
Error in Slope (sigma_m): 0.05393598899705936
      Error in Intercept (sigma_c): 0.2
```



Derivation:

Error in slope and intercept

$$m = \frac{N\sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i}{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2}$$

$$\sigma_m^2 = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial m}{\partial y_i}\right)^2$$

$$= \sigma^2 \sum_{i=1}^{N} \left(\frac{\partial m}{\partial y_i}\right)^2$$

Numerator

$$\frac{\partial m}{\partial y_i}$$

$$\frac{\partial}{\partial y_i}\left(N\sum xy - \sum x \sum y\right) = \frac{N\,\partial(\sum xy)}{\partial y_i} - \left(\sum x\right)\frac{\partial(\sum y)}{\partial y_i}$$

Expanding derivatives

$$N\sum x \frac{\partial y_i}{\partial y_i} - \sum x \frac{\partial y_i}{\partial y_i} = N\sum x - \left(\sum x\right)$$

$$\frac{\partial m}{\partial y_i} = \frac{N\sum x_i - \left(\sum x_i\right)}{N\sum x_i^2 - \left(\sum x_i\right)^2}$$

and so,

$$\sigma_m^2 = \sigma^2 \sum_{i=1}^{N} \left[\frac{N\left(\sum x_i\right) - \left(\sum x_i\right)}{N\sum x_i^2 - \left(\sum x_i\right)^2}\right]^2$$

Also,

$$c = \frac{\sum y_i - m\sum x_i}{N}$$

$$\frac{\partial c}{\partial y_i} = \frac{1}{N}$$

So, $$\sigma_c^2 = \frac{\sigma^2}{N}$$

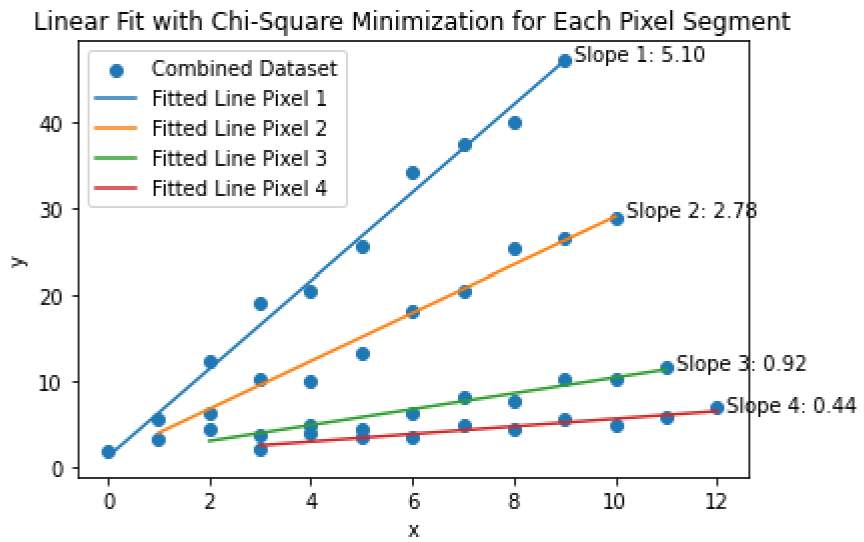
4. Problem given today as to take four pixels with different noises and slope, though the intercept should be common. We have plot the four pixelated data with a single array in a single plot. Numerical solution to this problem.

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

# Given data
num_pixels = 4
num_datapoints = 10
x_pixels = [np.arange(i, i + num_datapoints) for i in
range(num_pixels)]
m_values = [5, 3, 1, 0.5]
c_true = 1
n_avg = 0
n_std_values = [2, 1.5, 1, 0.5]

# Generate noisy data for each pixel
np.random.seed(42)
pixels_data = []

for i, x in enumerate(x_pixels):
    m = m_values[i]
```

```python
    n = np.random.normal(n_avg, n_std_values[i], len(x))
    y_true = m * x + c_true + n
    pixels_data.append((x, y_true))

# Combine all pixels into a single dataset
combined_x = np.concatenate([x for x, _ in pixels_data])
combined_y_true = np.concatenate([y_true for _, y_true in pixels_data])

# Define the chi-square function to minimize for the combined dataset
def chi_square_slope(params, x, y_true, n_std, num_datapoints):
    intercept = params[0]
    slopes = params[1:]
    chi_sq = 0

    # Split the combined dataset into segments based on the number of
datapoints in each pixel
    for i in range(0, len(x), num_datapoints):
        x_segment = x[i:i + num_datapoints]
        y_true_segment = y_true[i:i + num_datapoints]
        slope_segment = slopes[i // num_datapoints]
        y_model_segment = slope_segment * x_segment + intercept
        chi_sq += np.sum(((y_true_segment - y_model_segment) / n_std)
** 2)

    return chi_sq

# Initial guess for intercept and slopes
initial_guess = np.ones(num_pixels + 1)

# Minimize chi-square to find the best-fitted intercept and slopes for
the combined dataset
result = minimize(chi_square_slope, initial_guess, args=(combined_x,
combined_y_true, n_std_values[0], num_datapoints))

# Extract best-fitted intercept and slopes
intercept_fitted = result.x[0]
slopes_fitted = result.x[1:]

# Plot the combined dataset and the best-fitted line for each segment
plt.scatter(combined_x, combined_y_true, label='Combined Dataset')
for i, (x, _) in enumerate(pixels_data):
    plt.plot(x, slopes_fitted[i] * x + intercept_fitted, label=f'Fitted
Line Pixel {i + 1}')
    plt.text(x[-1] + 0.2, slopes_fitted[i] * x[-1] + intercept_fitted,
            f'Slope {i + 1}: {slopes_fitted[i]:.2f}', color='black')
```

```python
# Display the plot
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Fit with Chi-Square Minimization for Each Pixel Segment')
plt.legend()
plt.show()

# Display the results for the combined dataset
print("\nCombined Dataset:")
print("True Slopes:", m_values)
print("True Intercept (c_true):", c_true)
print("Fitted Slopes:", slopes_fitted)
print("Fitted Intercept:", intercept_fitted)
```

```
Combined Dataset:
True Slopes: [5, 3, 1, 0.5]
True Intercept (c_true): 1
Fitted Slopes: [5.0952298  2.77789425 0.92171961 0.44190841]
      Fitted Intercept: 1.2676987556569708
```



Here I got four different slopes of different segments of a single plot with a common intercept.