Linear Regression Report

Samuel Allpass

Experimental data collected will always contain a margin of error. Whether a result of measuring equipment, environmental factors or human limitation, it is important to appropriately propagate uncertainty into the final conclusions of an experiment. It is for this reason that a linear regression software was developed in order to streamline the process.

Playing to my strengths in Java, it was decided that the software would consist of a java project in charge of analysing the data, which was then translated through a .dat file to be visualised on MATLAB's graphing resources. The java file "Linear.java" consists of a main() method in charge of collecting relevant user input as seen in figure 1, and then, using a series of methods correlating to the equations for linear regression, calculates a linear regression plot with accurate uncertainties, this includes the option of a weighted or unweighted regression. In addition to this, the file uses a saveDataToDat() method to write relevant information into the .dat file. See appendix 1 for full java code.

Proceeding this process, a MATLAB file, 'LinearRun.m' , reads the .dat file and plots a graph appropriately. This graph includes relevant general and axis titles and error bars. Additionally, a key indicates the symbols seen on the graph. See appendix 2 for full MATLAB code.

In order to test the validity of the software, a test case (table 1) was conducted such that the data included two tests, one with constant uncertainty in the y (liquid) values, and one with varying uncertainty. The data was designed to mimic the fluid loss through drool of a camel over time.

Time (hrs)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.1	1.2	1.3	1.4
Fluid (mL)	7	14	20	26	33	40	46	52	59	72	78	85	91

(a) 5 ml
(b) 5% of the measurement

Test case ‘a’ was first conducted, the java output of which is shown in figure 1. It is relevant to point out that the regression is weighted, however, in this case, an unweighted test only showed a 0.02 decrease to the uncertainty in the y intercept. The java file concluded that the time-fluid trend followed the equation with appropriate significant figures:

$$Fluid\ (mL) = (64.7 \pm 0.2)(time\ (hrs)) + (0.7 \pm 0.2)$$

The MATLAB file was then run, such to plot the graph seen in figure 2. This same process was conducted for case ‘b’ where the uncertainty is 5% of a given fluid value. As shown in figure 3 and 4, this resulted in an equation:

$$Fluid\ (mL) = (64.8\ \pm\ 0.3)(time(hrs))\ +\ (0.6\ \pm\ 0.1)$$

```
What is the name of the x data:
Time (hrs)
What is the name of the y data:
Fluid (mL)
Enter the first (x) data set:
0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.1,1.2,1.3,1.4
Enter the second (y) data set:
7,14,20,26,33,40,46,52,59,72,78,85,91
Enter the uncertainty of the second (y) data set:
5,5,5,5,5,5,5,5,5,5,5,5,5
Is the regression Weighted (true or false):
true
y = (64.65 ± 0.22)x + (0.68 ± 0.18)
PS C:\Users\samal\OneDrive\Semester 2 2024\PHYS2941\Linear
```

Figure 1: Java output for case ‘a’

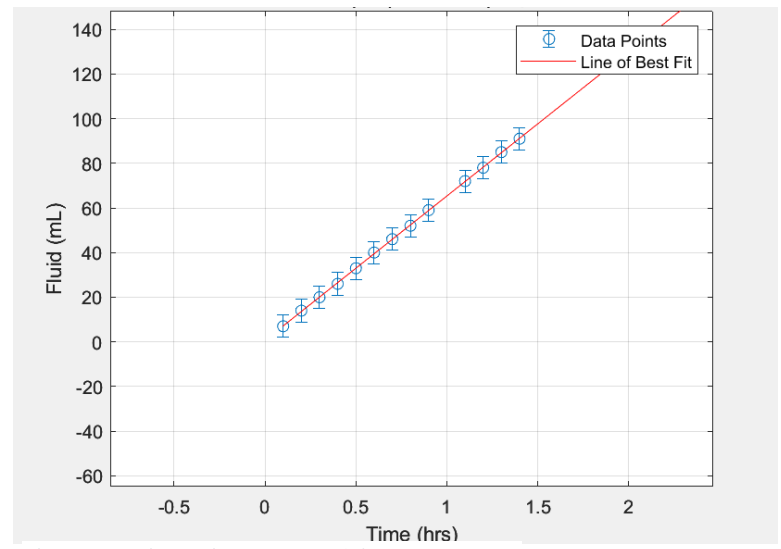

Figure 3: Java output for case 'b'

Figure 2: Fluid vs Time for case ‘a’

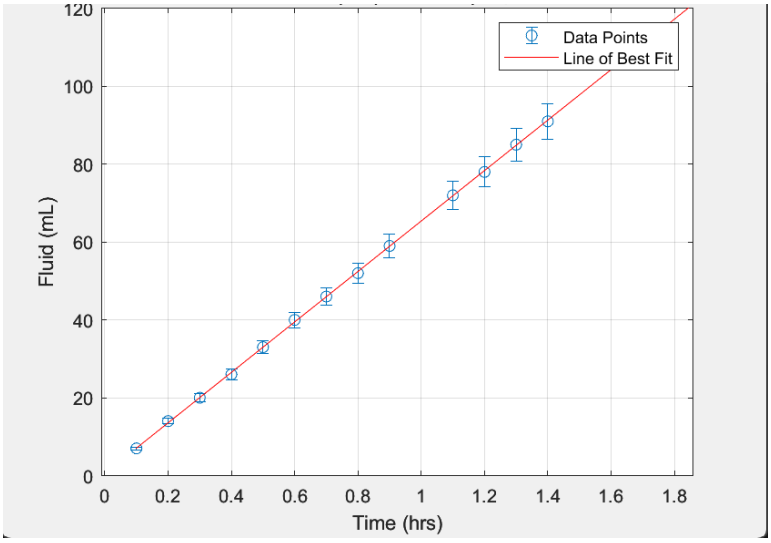

Figure 4: Fluid vs Time for case ‘b’

A final task was then conducted to test the general logic of the software. It was given that the total fluid loss rate of a camel was equivalent to the sum of the drool rate and the perspiration rate (p). The task was to calculate the amount of drool a camel loses in an average day. Independent measurements concluded that:

$$total\ loss\ rate = d + p$$

$$p = (25 \pm 2)\ mL/hour$$

And given the gradient of the equation in case 'a':

$$d = (64.7 \pm 0.2)\ mL/hour$$

Thus:

$$total\ loss\ rate = \big((25 \pm 2) + (64.7 \pm 0.2)\big)\ mL/hour$$

Given the uncertainty propagation of the sum of two variables is calculated using:

$\Delta p = \sqrt{(\Delta x)^2 + (\Delta y)^2}$, the total loss rate must be (with appropriate significant figures):

$$(90 \pm 2)\ mL/hour$$

In order to calculate the drool lost in a day, propagation of uncertainty must be analysed again given the average length of a day in the Sahara is $(12 \pm 2) hours$.

$$Total\ loss\ in\ a\ day = total\ loss\ rate * t(hours) = (89 \pm 2) * (12 \pm 2)$$

Given uncertainty propagates through the multiplication of two variables according to:

$$\frac{\Delta p}{|p|} = \sqrt{\left(\frac{\Delta x}{x}\right)^2 + \left(\frac{\Delta y}{y}\right)^2}$$

$$\Delta p = (89 * 12) * \sqrt{\left(\frac{2}{89}\right)^2 + \left(\frac{2}{12}\right)^2}$$

Such that with appropriate significant figures:

$$Total\ loss\ in\ a\ \ day = (1100 \pm 200)\ mL$$

Appendix 1: java code

```
//  Script  to  calculate the Linear regression plot
//  Written by  Samuel Allpass
//  Version 1
//  4th August 2024

import java.util.Arrays;
import java.util.List;
import java.util.Scanner;
import java.util.stream.Collectors;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class Linear {
  private List<Float> x;
  private List<Float> y;
  private List<Float> uncerty;
  private boolean isWeighted;
  private Float m;
  private Float c;
  private static String xname;
  private static String yname;

  public Linear(List<Float> x, List<Float> y, List<Float> uncerty, boolean isWeighted) {
    this.x = x;
    this.y = y;
    this.uncerty = uncerty;
    this.isWeighted = isWeighted;
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    try {
      System.out.println("What is the name of the x data:");
      xname = scanner.nextLine();
      System.out.println("What is the name of the y data:");
      yname = scanner.nextLine();

      System.out.println("Enter the first (x) data set:");
      List<Float> x = Arrays.stream(scanner.nextLine().split(","))
                  .map(Float::parseFloat)
                  .collect(Collectors.toList());

      System.out.println("Enter the second (y) data set:");
      List<Float> y = Arrays.stream(scanner.nextLine().split(","))
                  .map(Float::parseFloat)
                  .collect(Collectors.toList());

      System.out.println("Enter the uncertainty of the second (y) data set:");
      List<Float> uncerty = Arrays.stream(scanner.nextLine().split(","))
                    .map(Float::parseFloat)
                    .collect(Collectors.toList());

      System.out.println("Is the regression Weighted (true or false):");
      boolean isWeighted = Boolean.parseBoolean(scanner.nextLine());

      scanner.close();

      Linear linear = new Linear(x, y, uncerty, isWeighted);
      linear.performRegression();
    } catch (Exception e) {
      System.err.println("Invalid input. Please enter valid numbers.");
    }
  }

  private void performRegression() {
    float deltam = 0, deltac = 0;

    if (isWeighted) {
      m = weightedGradientCalc();
      deltam = weightedGradientUncertCalc();
      c = weightedYintCalc();
      deltac = weightedYintUncertCalc();
    } else {
      m = gradientCalc();
      deltam = gradientUncertCalc();
```

```
        c = yintCalc();
        deltac = yintUncertCalc();
    }

    System.out.printf("y = (%.2f ± %.2f)x + (%.2f ± %.2f)%n", m, deltam, c, deltac);

    saveDataToDat();
}

private void saveDataToDat() {
    try (PrintWriter writer = new PrintWriter(new FileWriter("src/data.dat"))) {
        for (int i = 0; i < x.size(); i++) {
            writer.printf("%f, %f, %f%n", x.get(i), y.get(i), uncerty.get(i));
        }
        writer.printf("%s\n", xname);
        writer.printf("%s\n", yname);
        writer.printf("%.2f\n", m);
        writer.printf("%.2f", c);

    } catch (IOException e) {
        e.printStackTrace();
    }
}


private float weightedMeanCalc(List<Float> f, List<Float> uncert) {
    float numerator = 0f;
    float denominator = 0f;
    for (int i = 0; i < f.size(); i++) {
        float weight = 1 / (uncert.get(i) * uncert.get(i));
        numerator += f.get(i) * weight;
        denominator += weight;
    }
    return numerator / denominator;
}

private float weightedGradientCalc() {
    float sum = 0f;
    float meanX = weightedMeanCalc(x, uncerty);
    for (int i = 0; i < x.size(); i++) {
        float weight = 1 / (uncerty.get(i) * uncerty.get(i));
        sum += weight * (x.get(i) - meanX) * y.get(i);
    }
    return sum / weightedDcalc();
}

private float weightedDcalc() {
    float sum = 0f;
    float meanX = weightedMeanCalc(x, uncerty);
    for (int i = 0; i < x.size(); i++) {
        float weight = 1 / (uncerty.get(i) * uncerty.get(i));
        sum += weight * (x.get(i) - meanX) * (x.get(i) - meanX);
    }
    return sum;
}

private float weightedYintCalc() {
    return weightedMeanCalc(y, uncerty) - weightedGradientCalc() * weightedMeanCalc(x, uncerty);
}

private float weightedGradientUncertCalc() {
    float sum = 0f;
    float gradient = weightedGradientCalc();
    float intercept = weightedYintCalc();

    for (int i = 0; i < x.size(); i++) {
        float weight = 1 / (uncerty.get(i) * uncerty.get(i));
        float residual = y.get(i) - gradient * x.get(i) - intercept;
        sum += weight * residual * residual;
    }

    float deltamsq = (1 / weightedDcalc()) * (sum / (x.size() - 2));
    return (float) Math.sqrt(deltamsq);
}

private float weightedYintUncertCalc() {
    float sum = 0f;
    float sumw = 0f;
    float gradient = weightedGradientCalc();
    float intercept = weightedYintCalc();
    float meanX = weightedMeanCalc(x, uncerty);
```

```
        for (int i = 0; i < x.size(); i++) {
            sumw += 1 / (uncerty.get(i) * uncerty.get(i));
        }

        for (int i = 0; i < x.size(); i++) {
            float weight = 1 / (uncerty.get(i) * uncerty.get(i));
            float residual = y.get(i) - gradient * x.get(i) - intercept;
            sum += weight * residual * residual;
        }

        float deltacsq = ((1 / sumw) + ((meanX * meanX) / weightedDcalc())) * (sum / (x.size() - 2));
        return (float) Math.sqrt(deltacsq);
    }

    private float meanCalc(List<Float> f) {
        float sum = 0f;
        for (Float value : f) {
            sum += value;
        }
        return sum / f.size();
    }

    private float gradientCalc() {
        float sum = 0f;
        float meanX = meanCalc(x);
        for (int i = 0; i < x.size(); i++) {
            sum += (x.get(i) - meanX) * y.get(i);
        }
        return sum / dcalc();
    }

    private float dcalc() {
        float sum = 0f;
        float meanX = meanCalc(x);
        for (int i = 0; i < x.size(); i++) {
            sum += (x.get(i) - meanX) * (x.get(i) - meanX);
        }
        return sum;
    }

    private float yintCalc() {
        return meanCalc(y) - gradientCalc() * meanCalc(x);
    }

    private float gradientUncertCalc() {
        float sum = 0f;
        float gradient = gradientCalc();
        float intercept = yintCalc();

        for (int i = 0; i < x.size(); i++) {
            float residual = y.get(i) - gradient * x.get(i) - intercept;
            sum += residual * residual;
        }

        float deltamsq = (1 / dcalc()) * (sum / (x.size() - 2));
        return (float) Math.sqrt(deltamsq);
    }

    private float yintUncertCalc() {
        float sum = 0f;
        float gradient = gradientCalc();
        float intercept = yintCalc();
        float meanX = meanCalc(x);

        for (int i = 0; i < x.size(); i++) {
            float residual = y.get(i) - gradient * x.get(i) - intercept;
            sum += residual * residual;
        }

        float deltacsq = ((1 / x.size()) + ((meanX * meanX) / dcalc())) * (sum / (x.size() - 2));
        return (float) Math.sqrt(deltacsq);
    }
}
```

Appendix 2: MATLAB code

```
%  Script  to  display Linear regression plot
%  Written by  Samuel Allpass
%  Version 1
%  4th August 2024

data = readtable('data.dat');

x = data.Var1;
y = data.Var2;
uncerty = data.Var3;

fid = fopen('data.dat', 'rt');
lines = textscan(fid, '%s', 'Delimiter', '\n');
fclose(fid);

xtitle = lines{1}{end-3};
ytitle = lines{1}{end-2};

m = str2double(lines{1}{end-1});
c = str2double(lines{1}{end});

y_fit = m * x + c;

figure;
errorbar(x, y, uncerty, 'o');
hold on;
plot(x, y_fit, '-r');
xlabel(xtitle);
ylabel(ytitle);
title(xtitle + " vs " + ytitle);
legend('Data Points', 'Line of Best Fit');
grid on;
hold off;
```