Statistics Notes

Paolo Raiteri

February 27, 2026

Contents

1. Basics of statistics

This chapter is not meant to replace a comprehensive statistics course, but it will just provide a quick overview of the main key statistics concepts routinely used in research projects.

1.1 Basic concepts and terminology

First, let's define the terminology used in this chapter and make some important distinctions.

- **Observable**
  A property of a substance or of a phenomenon that can be determined experimentally.
- **Measurement**
  The process of finding the values of an observable.
- **True value**
  The actual, usually unknown, value of the observable that is measured.
- **Error of the measurement**
  The difference between the true value of an observable and its measured value.
- **Uncertainty of a measurement**
  The interval within which the true value of an observable lies, with a given probability.
- **Precision of the instrument**
  The smallest difference between two signals that an instrument can distinguish. This is usually indicated by the last significant figure of the measure. Hence, a measurement usually has an uncertainty that is ± 0.5 of the last significant figure reported.
- **Detection limit**
  The smallest value of an observable that can be measured by the instrument.
- **Sample**
  A set of measurements of the same observable.
- **Population**
  This indicates the collection of all possible measurements of an observable. In other words, an infinitely large sample.

Experiments are used to measure observables, which are then used to test hypotheses, and variables are quantities that influence the outcome of the measurement.

- **Independent variables**
  Variables that can be changed independently from other variables.

- **Dependent variables**
  Variables whose value is determined by the independent variables.

- **Controlled variables**
  These are variables that affect the measurement and are effectively controlled during an experiment, *e.g.,* constant pH experiment.

- **Uncontrolled variables**
  These are variables that affect the measurement but are NOT controlled during an experiment, *e.g.,* the laboratory temperature.

The effect of uncontrolled variables on the measurement can be two-fold: add a false trend to the signal (interference) and/or add random variations to the signal (noise). This is illustrated in Figure 1.1.

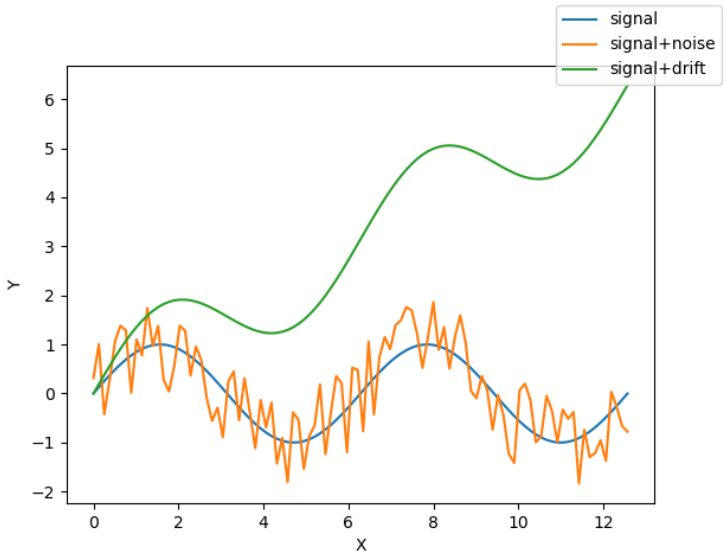


Figure 1.1: Examples of the effects of random noise and drift on a signal.

Often when discussing the solution to a problem it is mentioned:

- **Linearize an equation**
  An algebraic manipulation of a function to transform it into a linear equation. For example, the Arrhenius equation can be written as an exponential:

  $$k_r = Ae^{-E_a/RT}$$

  or as a linear relationship between $\ln K_r$ and $1/T$:

  $$\ln K_r = \ln A - \frac{E_a}{R}\frac{1}{T}$$

- **Analytic solution**
  The solution of a problem obtained by solving every equation using pen and paper.

- **Numerical solution**
  The approximate solution of a problem obtained using an algorithm, often run by a computer.

1.2 Significant figures

Significant figures in a measurement reflect the precision of the measurement and the certainty of the data. They include all the digits that are known with certainty plus one final digit, which is somewhat uncertain or estimated. Significant figures are crucial in scientific and engineering calculations because they communicate the accuracy of measurements and ensure that results are not over-interpreted. For instance, a measurement reported as 12.34 grams has four significant figures, indicating a higher level of precision compared to a measurement of 12 grams, which has only two significant figures.

Zeros in a measurement can be significant or insignificant depending on their position relative to the decimal point and other digits.

- Leading zeros, which are zeros to the left of the first non-zero digit, are not significant. For example, in the number 0.0025, the zeros are not significant, and the number has two significant figures.
- Captive (or embedded) zeros, which are zeros between non-zero digits, are always significant. For instance, in the number 1002, all four digits are significant.
- Trailing zeros, which are zeros to the right of the last non-zero digit, are significant if there is a decimal point. For example, in the number 50.00, all four digits are significant because the presence of the decimal point indicates that the zeros are measured and precise. Conversely, in the number 1500 without a decimal point, the trailing zeros may or may not be significant, often depending on the context or notation used.

In most cases, it does not make sense to report uncertainties with more than one significant figure. Because the uncertainty in a measurement reflects the precision of the measurement process, reporting it with an excessive number of significant figures can be misleading. For example, if a measurement is reported as 12.34 ± 0.05 grams, the uncertainty (0.05 grams) has one significant figure, which aligns with the precision of the measurement. Reporting an uncertainty with more than one significant figure, such as 12.34 ± 0.053 grams, would suggest a level of precision that the measurement process likely does not support. Therefore, keeping uncertainties to one significant figure is usually sufficient and avoids overstating the accuracy of the measurement. Examples of the correct use of significant figures:

- 12.3 ± 0.1
- 123 ± 1
- $120. \pm 1$
- 120 ± 10
- 12.3010 ± 0.0001

Examples of incorrect use of significant figures:

- 12.32 ± 1
- 12.3 ± 0.0001
- 12.3 ± 10

2. Error and uncertainty

Errors are not mistakes! While with careful experimentation mistakes can be avoided, errors can never be completely eliminated. There are in fact multiple sources of errors:

- the presence of uncontrolled variables
- random fluctuations of the instrument
- incorrect calibration of the instrument
- uncertainty of the measurement

Errors can be systematic or *random* and it is the responsibility of the researcher to try to minimize the errors and report the values with the observable with its uncertainty, so that the readers are not misled in the interpretation of the data. Typically, measurements are defined as

- **precise** if their spread is small, *i.e.,* the random errors are small.
- **accurate** if the systematic errors are small.

Systematic errors, such as the incorrect calibration of the instrument, can be difficult to identify and cannot be treated statistically. A good strategy to avoid these errors is to repeat the measurements using different instruments, or by having different users do the calibration and the measurements. Running the blank before and after the measurements of the samples can show whether the instrument has changed during the experiment. Another possible source of systematic errors is the use of incorrect data analysis techniques to post-process the measurements.

On the other hand, **random errors** are always present, and the simplest and most effective strategy to reduce them is to repeat the experiments as many times as possible and then use statistical tools to obtain the best possible estimate of the observable and its uncertainty. Indeed, the main scope of error analysis is to determine how large the uncertainty of an observable is given a finite number of measurements. In the following we will assume that any systematic errors have been eliminated or are negligible compared to the random errors.

2.1 Uncertainty due to random errors

The cornerstone of error analysis is the *central limit theorem*, which can be formulated as follows: Given a random sample of n independent observations ($\{X_1 \dots X_n\}$) of a population, in the limit of $n \to \infty$ the sample distribution converges to the *normal* (or Gaussian) distribution

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], \tag{2.1}$$

which implies that the average of the sample ($\overline{X}$) tends to the expected value (μ) of the population and the variance of the sample (σ_s^2) tends to the population variance (σ^2).

$$\overline{X} = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} X_i \to \mu \qquad \text{(average)} \tag{2.2}$$

$$\sigma_s^2 = \lim_{n\to\infty} \frac{1}{n-1}\sum_{i}^{n} (X_i - \overline{X})^2 \to \sigma^2 \qquad \text{(variance)} \tag{2.3}$$

The normal distribution is a probability distribution function, and its definite integral corresponds to the probability that a single observation of a quantity whose population is described by this function falls within a given interval (Figure 2.1).

$$\int_{\mu}^{\mu+\sigma} g(x)\,\mathrm{d}x = 0.341 \tag{2.4}$$

$$\int_{\mu}^{\mu+2\sigma} g(x)\,\mathrm{d}x = 0.478 \tag{2.5}$$

$$\int_{\mu-\sigma}^{\mu+\sigma} g(x)\,\mathrm{d}x = 0.682 \tag{2.6}$$

$$\int_{\mu-2\sigma}^{\mu+2\sigma} g(x)\,\mathrm{d}x = 0.956 \tag{2.7}$$

$$\ldots \tag{2.8}$$

$$\int_{-\infty}^{+\infty} g(x)\,\mathrm{d}x = 1 \tag{2.9}$$

$$\tag{2.10}$$

Hence, if we perform one measurement of an observable that has a population with overall expected value μ and variance σ^2, we can say that our measurement has a probability of 68.2% to be within $\pm\sigma$ of μ or 95.4% of being within $\pm 2\sigma$, where σ takes the name of *Standard Deviation*.

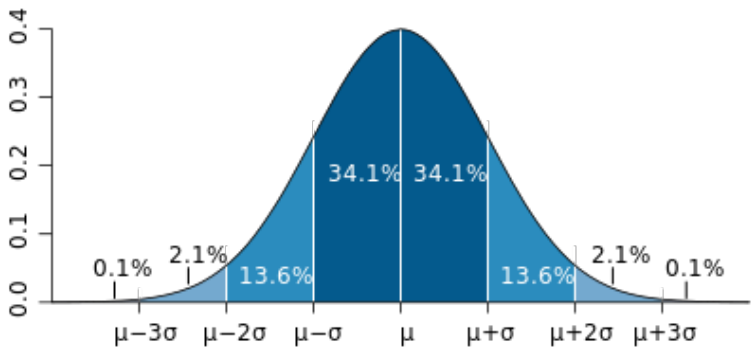


Figure 2.1: Normal distribution curve centered at μ with standard deviation σ. The percentages of the areas under the curve are shown for multiples of σ. By Ainali - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3141713

Using the central limit theorem, it is also possible to determine what are the best estimates for an observable X and its uncertainty, given a sample of n measurements. The best estimate of the observable is, obviously, the average of the sample

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{N} X_i \tag{2.11}$$

while its uncertainty can be defined using the standard deviation of the mean

$$\sigma_{\overline{X}} = \frac{\sigma_s}{\sqrt{n}} \tag{2.12}$$

where σ_s is the standard deviation of the sample

$$\sigma_s = \sqrt{\frac{\sum_i^n (X_i - \overline{X})^2}{(n-1)}} \tag{2.13}$$

In fact, using the central limit theorem it can be demonstrated that for n independent observations of the same variable, the variance of their average is the variance of one observation divided by the number of observations. The term $(n-1)$ is known as Bessel's correction, and it provides an unbiased estimate of the population standard deviation by accounting for the fact that the true mean of the population (μ) is unknown and it has been replaced with the sample mean ($\overline{X}$).

Again, from the central limit theorem it follows that the population of the means of an observable is *normally* distributed, which implies that the true mean of the population has a 68.27% probability of being within $\pm\sigma_{\overline{X}}$ of the computed mean ($\overline{X}$). However, in research papers measurements are typically reported with larger confidence intervals, which means that the standard error of the mean gets multiplied by a factor t.

$$\overline{X} \pm t\ \sigma_{\overline{X}}. \tag{2.14}$$

For a 95% confidence interval we have $t = 1.960$ and for a 99% confidence interval we have $t = 2.576$.

2.1.1 Weighting

Weighted averages and standard deviations are important statistical tools in physics and chemistry, where measurements often have varying levels of precision or importance. They can be calculated using the following equations, which are just generalizations of the well-known formulæ for the mean and standard deviation that we have seen before.

$$\bar{x} = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i} \tag{2.15}$$

$$s = \sqrt{\frac{\sum_{i=1}^n w_i (x_i - \bar{x})^2}{\frac{(n-1)\sum_{i=1}^n w_i}{n}}} \tag{2.16}$$

where $\overline{x}$ is the weighted average, s is the weighted standard deviation, x_i are the individual values, w_i are the corresponding weights and n is the number of values.

In physics and chemistry, these concepts are frequently applied in experimental data analysis. For instance, when measuring the speed of light using different methods, each measurement might have a different uncertainty. A more precise measurement would be given a higher weight in the final calculation. The weighted standard deviation in this context would indicate the spread of measurements around this average. These statistical methods are also vital in spectroscopy, where peak intensities have different weights based on factors like absorption coefficients, and in quantum mechanics, where energy states are weighted by their degeneracy in calculating average values.

Example 1

In chemistry, weighted averages are for example used in isotope abundance calculations. The average atomic mass of an element is a weighted average of its isotopes' masses, with weights based on their natural abundance. For example, chlorine has two main isotopes: chlorine-35 (mass 34.97 amu, abundance 75.77%) and chlorine-37 (mass 36.97 amu, abundance 24.23%). The weighted average atomic mass is

$$\frac{(34.97 \times 0.7577) + (36.97 \times 0.2423)}{0.7577 + 0.2423} = 35.45 \text{ amu}$$

2.1.2 Uncertainty for small sample sizes

For very small sample sizes ($n \leq 20$), which are common in most experimental works, the uncertainty defined using the standard deviation of the mean is typically an underestimate of the true uncertainty. In these cases, we should use the Student's t distribution, which has "bigger" tails and provides a better estimate of the uncertainty of the measurement (Figure 2.2). Hence, in order to report the number within a certain confidence interval we can still use Eq. 2.14 but the t is now derived from the Student's t distribution and it depends on the number of degrees of freedom of our sample $\nu = n - 1$. Some selected values for t are listed in Table 8.1. Note how for infinitely large samples t tends to the same values as the normal distribution. Below you can also find a Python code and the

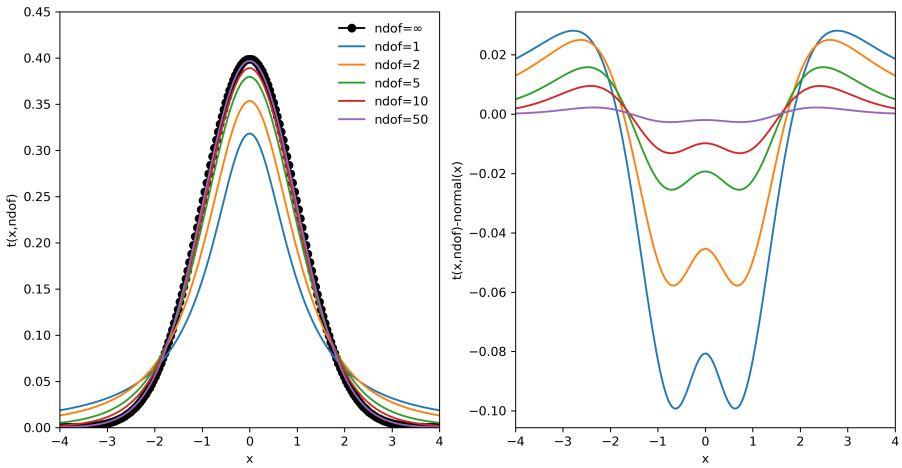


Figure 2.2: Comparison between the Student's t-distribution for different degrees of freedom ($ndof$) and the normal Gaussian distribution ($ndof = \infty$). The right panel shows the difference between the Student's t-distribution and the normal distribution.

`Excel` function to compute the t-value using `SciPy`.

Python Code Example

```
from scipy import stats
degrees_of_freedom = 10
confidence_level = 0.95

# t-value for a one-tailed distribution
```

```python
t_value_1t = stats.t.ppf(confidence_level, degrees_of_freedom)

# t-value for a two-tailed distribution
t_value_2t = stats.t.ppf((1 + confidence_level)/2, degrees_of_freedom)

print(f"Degrees of freedom = {degrees_of_freedom}")
print(f"One-tailed t value = {t_value_1t}")
print(f"Two-tailed t value = {t_value_2t}")
```

Output

```
Degrees of freedom = 10
One-tailed t value = 1.8124611228107335
Two-tailed t value = 2.2281388519649385
```

The corresponding `Excel` function for a confidence level of 95% and 10 degrees of freedom is given below. Note that for the one-tailed distribution `Excel` assumes the distribution is left-tailed, *i.e.,* the returned number is negative and $-t$ is to be used for a right-tailed distribution.

```
t_value_1t =T.INV( 0.95, 10 )
t_value_2t =T.INV.2T( (1-0.95), 10 )
```

Note that the one tailed and two tailed distributions take different inputs for the confidence value. While the second parameter is the number of degrees of freedom, *i.e.,* the number of values minus 1.

Notes about extracting the t-value

The `scipy.stats.t.ppf` and `Excel T.INV` functions calculate the percent point function (PPF), also known as the inverse cumulative distribution function or quantile function, for the t-distribution. In the context of statistical hypothesis testing, this function is incredibly useful for finding critical values, which are the boundaries of your rejection region in hypothesis testing. It's also key in constructing confidence intervals, helping you determine the range of values that likely contains the true population parameter. For a given probability and degrees of freedom, it returns the t-value that corresponds to that probability in the t-distribution. For a one-tailed test, we are only interested in one end of the distribution. We might use this when we are testing if something is significantly greater than or less than a certain value, but not both. For a two-tailed test, we are interested in both ends of the distribution. This is used when you want to know if there's a significant difference in either direction.

Note that while `Excel` has different functions for the two-tailed t-value (`T.INV.2T`), `SciPy` does not. However, because the Student's t distribution is symmetric, we can use the same function to get the two-tailed t-value. This is done by simply passing half of the desired significance level to `stats.t.ppf` (so 0.025 for a 95% confidence level). This is because we're splitting the probability between the two tails, which are equal by symmetry (2.5% each).

3. Error propagation

Having just learned how to compute the uncertainties of individual measurements, it becomes important to understand how the error propagates when we combine their values to compute a derived quantity through a mathematical expression. This step is indeed essential to report our final results with a meaningful confidence interval, which depends on the accuracy of all the individual measurements. While it is beyond the scope of this document to formally derive the equations, it is important to summarize the main findings and the assumptions behind them. Let's take a set of variables $(x_1, \ldots, x_n)$ and their uncertainties $(\sigma_1, \ldots, \sigma_n)$ that are combined through a function $f(x_1, \ldots, x_n)$. The uncertainty on the computed value of f can be approximated with:

$$\sigma_f^2 = \sum_i \left(\frac{\partial f}{\partial x_i} \right)^2 \sigma_i^2 \tag{3.1}$$

where ∂ indicates the partial derivative of the function with respect to one of the variables. This expression is valid under the assumption that the variables are independent and that their errors are "small enough". While there is no quantitative definition of "small," we can assume that this is true for all cases in these laboratories. We are also implicitly assuming that if the uncertainty used in these formulas corresponds to the 95% confidence interval, the derived uncertainty also corresponds to the 95% confidence interval of the derived quantity.

3.1 Linear combination of variables

$$f(x_1, \ldots, x_n) = \sum_i A_i x_i \tag{3.2}$$

In this case the partial derivatives are simply the coefficients of the linear combination: $\frac{\partial f}{\partial x_i} = A_i$, which gives:

$$\sigma_f^2 = \sum_i A_i^2 \sigma_i^2 \tag{3.3}$$

3.2 Product of variables

$$f(x_1, \ldots, x_n) = \prod_i x_i \tag{3.4}$$

In this case the partial derivatives are a bit more complicated:

$$\frac{\partial f}{\partial x_i} = \prod_{j \neq i} x_j = \frac{f(x_1, \ldots, x_n)}{x_i} \tag{3.5}$$

hence the uncertainty of the derived function is:

$$\sigma_f^2 = f^2 \sum_i \left(\frac{\sigma_i}{x_i}\right)^2 \tag{3.6}$$

3.3 Logarithm of a variable

$$f(x) = \ln(bx)^a = a\ln(bx) = a\ln b + a\ln x \tag{3.7}$$

where a and b are exact numbers. In this case the partial derivative is:

$$\frac{\partial f}{\partial x} = \frac{a}{x} \tag{3.8}$$

hence the uncertainty of the derived function is:

$$\sigma_f^2 = \left(\frac{a\sigma_x}{x}\right)^2 \tag{3.9}$$

3.4 Exponential of a variable

$$f(x) = ae^{bx} \tag{3.10}$$

where a and b are exact numbers. In this case the partial derivative is:

$$\frac{\partial f}{\partial x} = abe^{bx} = bf(x) \tag{3.11}$$

hence the uncertainty of the derived function is:

$$\sigma_f^2 = f^2(x)\,(b\sigma_x)^2 \tag{3.12}$$

3.5 Examples

Let's now make some examples of how this works in some common cases.

Example 2

Given two quantities with their uncertainties, $x = 12 \pm 3$ and $y = 7 \pm 1$, compute the expected value and uncertainty of the derived quantity $f = x - 2y$.

Solution:
The coefficients for the linear combination are 1 and -2 respectively, and the uncertainty on f is:

$$\begin{aligned} \sigma_f^2 &= (1)^2(3)^2 + (-2)^2(1)^2 = 13 \\ \sigma_f &= 3.6 \approx 4 \end{aligned}$$

Hence, our best estimate and uncertainty for the derived quantity f is $f = -2 \pm 4$.

More complicated cases can be solved by using Eq. 3.1 or by applying the formulas above recursively.

Example 3

Compute the rate constant and its uncertainty at 300 K, given the following values for the activation energy and pre-exponential factor: $E_a = 40.0 \pm 0.1$ kJ/mol and $A = 1.2 \pm 0.1$ THz.

Solution:
We have to use the Arrhenius equation to compute the rate constant:

$$k = A \ \exp\left[-\frac{E_a}{RT}\right]$$

Because A is not an exact number, we cannot directly use Eq. 3.12 but we need to compute first the uncertainty for $\exp\left[-\frac{E_a}{RT}\right]$ and then combine it with the uncertainty for A. Let:

$$\alpha = \exp\left[-\frac{E_a}{RT}\right] = 108 \times 10^{-9}$$

and using Eq. 3.12 we get:

$$\sigma_\alpha = \sqrt{\left[\alpha\left(\frac{-1}{RT}\right)0.1\right]^2} = 4 \times 10^{-9}$$

then using Eq. 3.6 we can compute the error on k:

$$k = A \ \alpha = 130 \times 10^3 Hz$$

$$\sigma_k = \sqrt{k^2\left[\left(\frac{\sigma_A}{A}\right)^2 + \left(\frac{\sigma_\alpha}{\alpha}\right)^2\right]} = 10 \times 10^3$$

Hence the number we should be reporting for the rate constant is:

$$k = 130 \ \pm 10 \text{ kHz}$$

4. Comparison with literature data

The Student's t distribution can also be used to determine whether the measured mean of an observable ($\overline{X}$) is consistent with its expected population average (μ), *e.g.,* a previously published value.

In this case, what we want to test is whether $\overline{X} - \mu = 0$ is within reasonable limits.

Although there are many different cases, the general procedure to compare two values requires us to:

- compute the t-statistics
- determine the number of degrees of freedom
- calculate the p-score

4.1 t-statistics

The general definition of the t-statistics is:

$$Z = \frac{|\overline{X} - \mu|}{CSE} \tag{4.1}$$

where CSE is the combined standard error for the comparison. In fact, the experimental value we want to compare our results with often has its uncertainty, which we need to take into account. In that case, we can compute CSE by simply using the error propagation formula for a difference between two numbers.

$$CSE = \sqrt{SE_{\overline{X}}^2 + SE_{\mu}^2} \tag{4.2}$$

If the reference value does not have an uncertainty, we can assume $SE_\mu = 0$.

4.1.1 Number of degrees of freedom

In many cases, the number of degrees of freedom (n_{dof}) is not well defined and some approximations are often required.

The most general formula is the Welch-Satterthwaite equation, which combines the variance of two independent samples and their sizes to obtain an approximate value of the degrees of freedom:

$$n_{dof} \approx \frac{\left(\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}\right)^2}{\frac{\left(\frac{\sigma_1^2}{n_1}\right)^2}{n_1 - 1} + \frac{\left(\frac{\sigma_2^2}{n_2}\right)^2}{n_2 - 1}} \tag{4.3}$$

where σ_1^2 is the variance of the first sample, σ_2^2 is the variance of the second sample, n_1 is the size of the first sample and n_2 is the size of the second sample.

However, we do not often know the number of samples that were used to determine the uncertainty of an experimental value, and one logical approximation would be to

assume that a much larger sample size was used, *i.e.*, $n_2 \to \infty$. In that case the Welch-Satterthwaite equation simplifies to:

$$n_{dof} \approx n_1 - 1 \tag{4.4}$$

which is the number of degrees of freedom of our sample.

4.1.2 *p*-score

Z and n_{dof} are needed to compute the p-score, which is the probability of obtaining a Z value for the t-statistics at least as extreme as the observed results, assuming that the null hypothesis is true, *i.e.*, the p-score quantifies how likely it is to observe such an extreme value by random chance. Using again 95% as our confidence interval, we accept or reject our hypothesis that the two values are statistically equivalent according to this criterion:

- p-score ≥ 0.05 our measurement is compatible with the literature value
- p-score < 0.05 suggests that our measurement is statistically different from the literature value.

The calculation of the p-score is not trivial, but there are `Excel` and Python functions that will do that for us. Here below there's a python code to compute the t-value and the p-score using `SciPy`.

Python Code Example

```python
from scipy import stats
ndof = 9
Z_value = 1.84
p_score = stats.t.sf(np.abs(Z_value), ndof)*2
print(f"p score 1 = {p_score}")

Z_value = 3.74
p_score = stats.t.sf(np.abs(Z_value), ndof)*2
print(f"p score 2 = {p_score}")
```

Output

```
p score 1 = 0.0989116094254279
p score 2 = 0.004625492598127718
```

and the corresponding `Excel` functions:

```
p_score1 =T.DIST.2T( 1.84 , 9 )
p_score2 =T.DIST.2T( 3.74, 9 )
```

Example 4

Given the ten measurements of an observable reported below, compute the best estimate for the observable and the 95% confidence interval. There are two previously published values for that quantity, 15.3 and 23.5. Is your measurement consistent with them?

Measurement	Value
1	10
2	12
3	23
4	23
5	16
6	23
7	21
8	16
9	18
10	20

Solution:
Let's compute the average:

$$\overline{X} = \frac{10 + 12 + 23 + 23 + 16 + 23 + 21 + 16 + 18 + 20}{10} = 18.2,$$

the standard deviation:

$$\sigma_s = \sqrt{\frac{1}{n-1}\sum (X_i - \overline{X})^2} = 4.66$$

and the standard deviation of the mean:

$$\sigma_{\overline{X}} = \sigma_s/\sqrt{n} = 1.47422.$$

Using the *t-value* from the table with 9 degrees of freedom and a 95% confidence interval (two tails) is 2.262, we can then compute the uncertainty of our average.

$$t\ \sigma_{\overline{X}} = 3.335$$

Finally, using the correct number of significant figures, we can say the result of our experiment is:

$$18 \pm 3$$

We can then compare our estimate with the previous experimental values:

$$Z_1 = \frac{|18.2 - 15.3|}{1.47} = 1.84 \qquad \rightarrow \qquad p_{score} = 0.099 > 0.05$$

and

$$Z_2 = \frac{|18.2 - 23.5|}{1.47} = 3.74 \qquad \rightarrow \qquad p_{score} = 0.046 < 0.05$$

Hence, our measurements are consistent with the first experimental estimate of the observable but not with the second.

Example 4

Let's say we measured the melting point of a compound:

- Our measurement: $T_{m1} = 85.0 \pm 2.0°\text{C}$ (from 5 measurements)
- Literature value: $T_{m2} = 86.3 \pm 0.4°\text{C}$

Here we assume that both uncertainties correspond to the same confidence interval.

Solution:
Since we want to assess whether the difference between the melting temperatures is significant, the error propagation formula that we need to use is Eq. 3.3:

$$SE_{combined} = \sqrt{2.0^2 + 0.4^2} = 2.04°\text{C}$$

We can then do the Student's t-test and compute the Z-score:

$$Z = \frac{|\bar{x}_1 - \bar{x}_2|}{SE_{combined}} = \frac{|85.0 - 86.3|}{2.04} = 0.64$$

We now need to choose an appropriate number for the degrees of freedom, which may be complex when using uncertainties that we don't know how they have been calculated. A conservative approach is to use the smallest number of degrees of freedom. If the literature value has an unknown number of degrees of freedom, we assume a very large number of measurements have been performed ($n_2 \to \infty$), and use the degree of freedom of our sample, in this case $n = 5 - 1 = 4$.
We compute the p-score using the two-tailed t-distribution with 4 degrees of freedom and compare it with the 95% confidence threshold:

$$p_{score} = 0.55 > 0.05$$

The p-score is greater than the typical significance level of 0.05, therefore, we fail to reject the null hypothesis, which means that there is not enough evidence to conclude that our measured melting point is significantly different from the literature value, considering both uncertainties.

4.2 Analysis of Variance

One-way ANOVA (Analysis of Variance) is a statistical method used to determine whether there are any statistically significant differences between the means of three or more independent (unrelated) groups. In a typical one-way ANOVA, we analyze the impact of a single independent variable, known as a factor, on a dependent variable. This method helps us understand if the variation in the dependent variable is greater between groups than within groups. For instance, if we are examining the effect of different types of fertilizers on plant growth, the fertilizer type would be the independent variable with multiple levels (e.g., Fertilizer A, B, and C), and plant growth, measured in height, would be the dependent variable. By comparing the mean growth of plants across different fertilizer groups, one-way ANOVA allows us to determine if any fertilizer leads to significantly different plant growth compared to others, thus providing insight into the effectiveness of

each fertilizer.

This is typically done by comparing the variance within the groups and between the groups, with the so called F-test, which is similar to what we discussed before, but it involves more than two data sets. These are the steps required to do the one-way ANOVA test:

1. Compute the mean within each group:

$$\bar{X}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} X_{ij}$$

2. Compute the overall mean:

$$\bar{X} = \frac{1}{N} \sum_{i=1}^{k} \sum_{j=1}^{n_i} X_{ij}$$

3. Compute the sum of squares between groups:

$$SSB = \sum_{i=1}^{k} n_i (\bar{X}_i - \bar{X})^2$$

4. Compute the sum of squares within groups:

$$SSW = \sum_{i=1}^{k} \sum_{j=1}^{n_i} (X_{ij} - \bar{X}_i)^2$$

5. Compute the total sum of squares:

$$SST = SSB + SSW$$

6. Compute the mean square between groups (MSB):

$$MSB = \frac{SSB}{k-1}$$

7. Compute the mean square within groups (MSW):

$$MSW = \frac{SSW}{N-k}$$

8. Compute the F-statistic:

$$F = \frac{MSB}{MSW}$$

9. Compare the calculated F-statistic to the critical value from the F-distribution table, based on the chosen significance level and the degrees of freedom for the numerator $(k-1)$ and denominator $(N-k)$.

Example 4

Perform the one-way ANOVA test on these three groups of data:

- Group 1: 5, 7, 8, 6, 7
- Group 2: 6, 9, 12, 8, 10
- Group 3: 7, 8, 9, 8, 7

Solution:
The averages:

$$\bar{X}_1 = \frac{5+7+8+6+7}{5} = 6.6$$
$$\bar{X}_2 = \frac{6+9+12+8+10}{5} = 9$$
$$\bar{X}_3 = \frac{7+8+9+8+7}{5} = 7.8$$
$$\bar{X} = \frac{(5+7+8+6+7)+(6+9+12+8+10)+(7+8+9+8+7)}{15} = 7.8$$

Between-group variability:

$$SSB = 5(6.6-7.8)^2 + 5(9-7.8)^2 + 5(7.8-7.8)^2$$
$$SSB = 5(1.2)^2 + 5(1.2)^2 + 5(0)^2$$
$$SSB = 5(1.44) + 5(1.44) + 0$$
$$SSB = 7.2 + 7.2 = 14.4$$

Within-group variability (SSW):

$$SSW = \sum (X_{ij} - \bar{X}_i)^2$$
$$SSW = (5-6.6)^2 + (7-6.6)^2 + (8-6.6)^2 + (6-6.6)^2 + (7-6.6)^2$$
$$+ (6-9)^2 + (9-9)^2 + (12-9)^2 + (8-9)^2 + (10-9)^2$$
$$+ (7-7.8)^2 + (8-7.8)^2 + (9-7.8)^2 + (8-7.8)^2 + (7-7.8)^2$$
$$SSW = 2.56 + 0.16 + 1.96 + 0.36 + 0.16$$
$$+ 9 + 0 + 9 + 1 + 1$$
$$+ 0.64 + 0.04 + 1.44 + 0.04 + 0.64$$
$$SSW = 5.2 + 20 + 2.8$$
$$SSW = 28$$

Total variability (SST):

$$SST = SSB + SSW = 14.4 + 28 = 42.4$$

Mean squares:

$$MSB = \frac{SSB}{k-1} = \frac{14.4}{3-1} = \frac{14.4}{2} = 7.2$$
$$MSW = \frac{SSW}{N-k} = \frac{28}{15-3} = \frac{28}{12} \approx 2.33$$

The F-statistic:

$$F = \frac{MSB}{MSW} = \frac{7.2}{2.33} \approx 3.09$$

The critical value for a 95% confidence level with 3 groups and a total of 15 points is 3.885. Because $F < 3.885$ we cannot reject the null hypothesis. This means that the means are statistically the same.

Python Code Example

```python
import scipy.stats as stats

# Define the data
group1 = [5, 7, 8, 6, 7]
group2 = [6, 9, 12, 8, 10]
group3 = [7, 8, 9, 8, 7]

# Perform the ANOVA
f_statistic, p_value = stats.f_oneway(group1, group2, group3)

# Significance level
confidence_level = 0.95
alpha = 1 - confidence_level

# Degrees of freedom
dfn = 2  # Degrees of freedom for the numerator (between groups)
dfd = 12  # Degrees of freedom for the denominator (within groups)

# Calculate the critical value using the one tail distribution
critical_value = stats.f.ppf(1 - alpha, dfn, dfd)

# Print the critical value
# Print the results
print(f"F-statistic: {f_statistic}")
print(f"Critical value: {critical_value}")
print(f"P-value: {p_value}")
```

Output

```
F-statistic: 3.0857142857142867
Critical value: 3.8852938346523933
P-value: 0.08293790262218739
```

In `Excel`, the `ANOVA` test can be done using the Analysis ToolPak add-in (Figure 4.1).

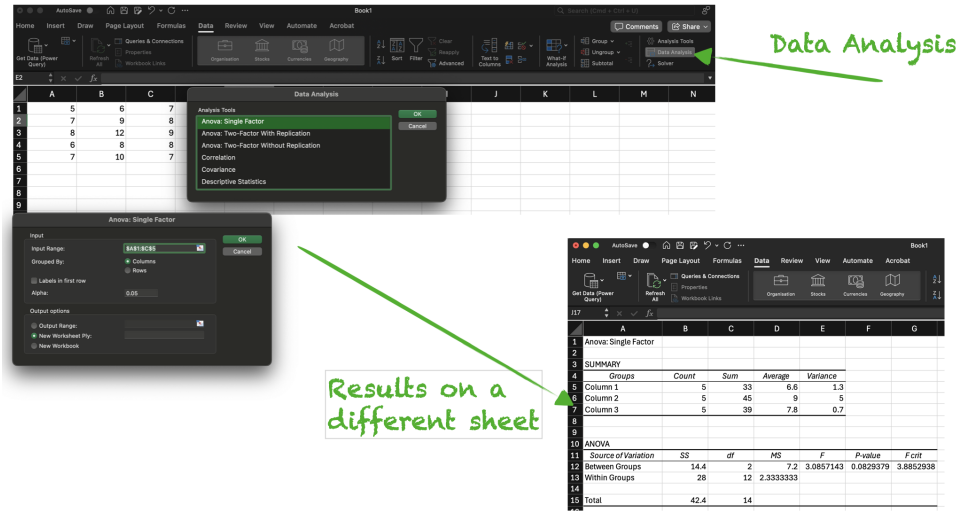


Figure 4.1: One-way `ANOVA` test in `Excel` using the Analysis ToolPak add-in.

5. Curve fitting

Let's imagine we have a set of observations of a quantity (y) that depends on a controlled variable (x), which changes between experiments, and that the dependent and independent variables are connected by a function $f(x_i; \text{params})$, which contains a few parameters. In this scenario we may want to know

- whether our dataset is consistent with the function $f(x_i; \text{params})$
- what are the optimal values of the parameters of $f(x_i; \text{params})$ to reproduce our data

The former can be simply assessed using various metrics such as the sum of the residuals squared (e), the coefficient of determination (R^2), the reduced chi-squared (χ^2), or other criteria depending on the context. The latter is instead achieved by performing a curve fitting, also called regression, which is the process of adjusting the parameters of the mathematical model so that it best describes a set of observations.

The least squares fitting is arguably the most commonly used mathematical method to find the best-fitting curve through a set of data points. By minimizing the sum of the squares of the residuals between the data points and the curve, the least squares method ensures that the overall error in the prediction is as small as possible. The same metrics mentioned above can then be used to assess the "goodness" of the fit; it is however important not to over-interpret the meaning of those metrics and always supplement them with plotting the data, fit function and residuals.

5.1 The sum of the residuals squared

This is the simplest possible measure of how well a given function reproduces our data

$$e = \sum_i^n w_i^2 \left[y_i - f(x_i)\right]^2 \tag{5.1}$$

where y_i are the observed values and $f(x_i)$ those predicted by the model, and w_i are the weights assigned to the observations. Typically values that have smaller uncertainties are assigned larger weights, $w_i = 1/\sigma_i$. If the weights are not known, it is assumed that they are all the same. Any value can be used, but for convenience we typically use $w_i = 1$.

5.2 The coefficient of determination

The coefficient of determination, usually indicated as R^2, is a statistical measure that indicates the proportion of the variance in the dependent variable that is explained by the independent variable(s) in a regression model. R^2 is typically defined as

$$R^2 = \frac{e}{\sum_i (y_i - \overline{y})^2} \tag{5.2}$$

where e is the sum of the residuals and $\overline{y}$ is the average of the observed values.

R^2 is commonly used as a measure of how well a model fits the data, indicating its accuracy. However, what it truly reflects is the proportion of variance in the dependent variable explained by the independent variable(s). If the dependent variable varies consistently with the independent variable, the R-squared value will tend to 1, suggesting a strong relationship. Conversely, an R^2 value close to 0 is often assumed to indicate that the model fails to explain the variance in the data, implying a poor fit; however, this is not always the case.

There are however many limitations in the use of the R^2 value, which also cannot be used to compare the goodness of two fits where different models are used. In fact, the value of R^2 can be increased arbitrarily by adding variables into the model, which may not necessarily mean that we have a "better" model. An overly high R^2 can sometimes indicate overfitting, *i.e.,* the model is too complex, having too many parameters relative to the number of observations.

Although the R^2 value is typically between 0 and 1, in some contexts, it can also be negative, which may indicate that the model fits the data worse than a horizontal line through the mean of the observed data. Essentially, this means that the model is a poor predictor of the dependent variable, performing worse than the simplest possible model (one that ignores all independent variables), *i.e.,* averaging the data.

Overall, there is a good amount of evidence on the limitations of the use of R^2 as a metric to assess the goodness of a fit, and one should always plot the data, the fitting function and the residuals to make an informed judgment on the usefulness of R^2, and also look at other metrics to assess the goodness of a fit.

5.3 The reduced chi-squared

The reduced chi-squared (χ^2_ν) can at times provide a better measure of how well a model fits the data, including the relative to the uncertainties in the data points, if they are known.

$$\chi^2_\nu = \frac{\chi^2}{N-k} = \frac{1}{N-k}\sum_{i=1}^{N}\left(\frac{y_i - f(x_i;\text{params})}{\sigma_i}\right)^2 \tag{5.3}$$

where:

- N is the number of data points.
- k is the number of fitted parameters.
- y_i are the observed data points.
- $f(x_i;\text{params})$ are the model predictions.
- σ_i are the uncertainties (inverse of the weights) of the data points.

Here are some practical considerations to keep in mind.

- **Good Fit**
  Ideally, χ^2_ν should be close to 1. This indicates that the residuals (differences between observed and predicted values) are consistent with the expected scatter due to measurement uncertainties.

- **Slight Deviations**
  Slight deviations from 1 are common and acceptable, especially when working with real-world data, which may have additional sources of variability not captured by the model or uncertainties.

- **Significant Deviations**
  If χ^2_ν is significantly different from 1, it may be necessary to revisit the model or the estimated uncertainties. For example:

  - $\chi^2_\nu \ll 1$: might indicate that the model is overfitting the data, the uncertainties are overestimated, or that your data have no noise.
  - $\chi^2_\nu \gg 1$: Indicates a poor fit. The model does not describe the data well, or the uncertainties might be underestimated.

5.4 Importance of plotting the residuals

Looking at the residuals can help us spot if the wrong function is used for a fit. In fact, the residuals should be randomly scattered around the zero value (Figure 5.1). On the other hand, if there are patterns where the residuals are always above or below the zero value, that is usually an indication that the wrong function has been used for the fit (Figures 5.2 and 5.3) even if the fit looks good by eye.

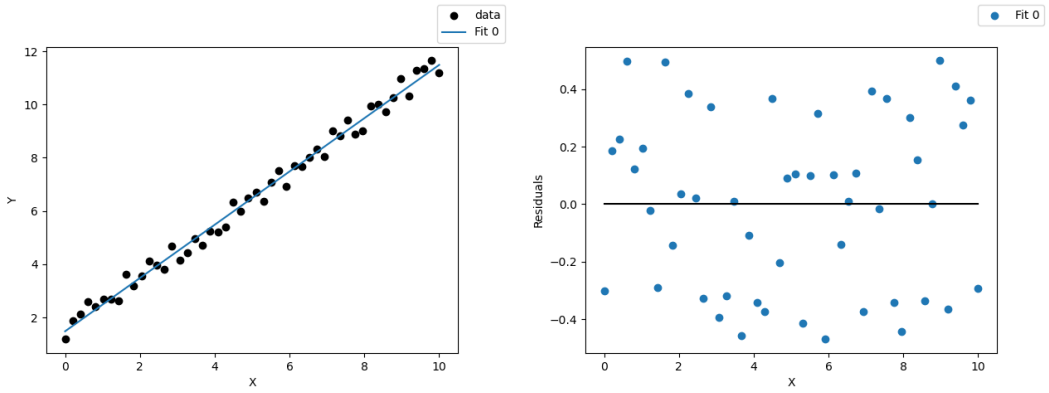


Figure 5.1: Example of using the correct function for a fit. Residuals are plotted in the right panel.

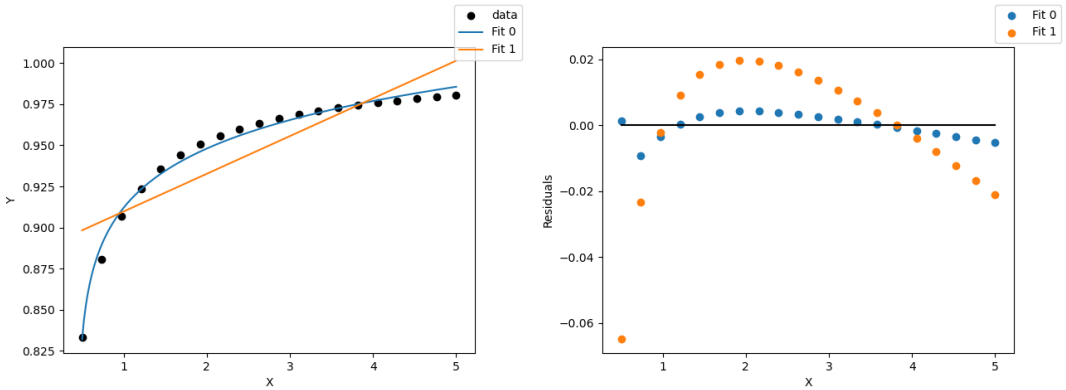


Figure 5.2: Example of using the wrong function for a fit. Residuals are plotted in the right panel.

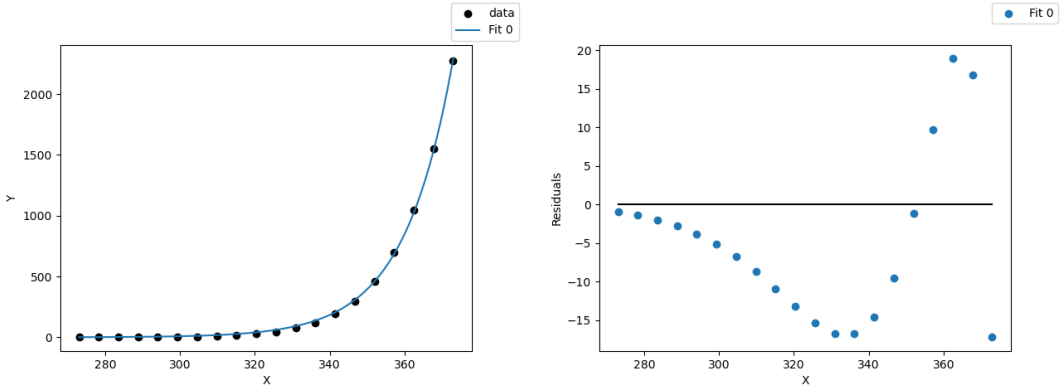


Figure 5.3: Example of using the wrong function for a fit. Residuals are plotted in the right panel.

5.5 Uncertainty on the fitting parameters

Computing the best-fitting parameters for a model using least squares methods is similar to determining the expectation value of an observable by calculating its average. Just as expectation values have uncertainties, the best-fitting parameters also come with uncertainties that should be reported together with their values. Although this process is a bit more complex than computing the standard error of a sample, most statistical tools, such as those available in Python or R can handle it. However, `Excel` is not designed for complex statistical analysis and has limited automatic functionalities for this purpose, and the calculation of the uncertainties is often left to the user, which is however far from trivial.

The uncertainty (standard error, 68.2% confidence interval) of the fitting parameters (β_i), under some assumptions, can be written in terms of the diagonal elements of the covariance matrix ($\mathbf{Cov}(\beta)_{ii}$).

$$\sigma_{\beta_i} = \sqrt{\mathbf{Cov}(\beta)_{ii}} \tag{5.4}$$

where

$$\mathbf{Cov}(\beta) = \sigma_e \ \left(\mathbf{J}^T\mathbf{J}\right)^{-1} \tag{5.5}$$

σ_e^2 is the variance of the residuals and $\mathbf{J}$ is the Jacobian matrix, which consists of the partial derivatives of the model with respect to each parameter, evaluated at each data point. This is an $M \times N$ matrix, where N is the number of fitting parameters and M is the number of points, hence the covariance matrix becomes an $N \times N$ matrix. The variance is divided by the number of degrees of freedom $(M - N)$ to get an unbiased estimate. The Jacobian can be calculated numerically or analytically.

5.6 How to practically do a curve fit

In this section we show how to fit a (custom) function using `Excel` and Python. We will also show how to do a weighted fit, where the inverse of the data uncertainties are used as weights, *e.g.,* points with smaller uncertainties have larger weights in the fit. Both approaches give equivalent values for the fitting parameters, but in Python it is arguably easier to extract uncertainties on the fitting parameters.

Example 4

A typical regression problem involves fitting a set of experiments to evaluate the fitting function at a chosen value for which we don't have a corresponding measurement. In order to illustrate this, we will do a linear fit of the data provided below, where x is the independent variable, y is the measurement and *error* is its uncertainty, and evaluate the fitting function at 5 and 15. Both the fitting parameters and the results of the function evaluation need to be reported with their uncertainties.

x	y	error
0.0	-0.5	0.4
0.5	5.5	0.4
1.1	3.2	0.1
1.6	5.6	0.2
2.1	20.9	0.2
2.6	20.6	0.1
3.2	16.8	0.4
3.7	20.3	0.4
4.2	25.4	0.4
4.7	30.4	0.4
5.3	31.9	0.2
5.8	23.0	0.2
6.3	33.5	0.3
6.8	35.9	0.1
7.4	44.1	0.2
7.9	44.4	0.2
8.4	32.6	0.2
8.9	43.4	0.2
9.5	56.2	0.3
10.0	48.3	0.2

5.6.1 Fitting any function in Python

While there are many Python packages that would allow us to do curve fitting, such as `NumPy` and `SciPy`, `lmfit` is one of the most complete tools available, particularly for error analysis, and it can also quickly produce plots of the fitted function and residuals. In the following sections we'll demonstrate how `SciPy` and `lmfit` can be used to perform linear and non-linear fitting using the same data. The main disadvantage of `lmfit` is that it is not immediately available on Google Colab and it would have to be installed every time a session is initiated.

Examples of how to create and use functions that can be included in a Jupyter Notebook to do a linear (or non-linear) fit using `SciPy` and `lmfit` are shown in the appendix. These functions may not be exactly suitable for all labs, but they may serve as a starting

point for you to achieve the goal of the labs. Here below only the plot of the fitted function, residuals and screen output are shown.

Output

```
-------------------------------------------------
Fit results SCIPY:
Parameter #1 = 4.8740 ± 0.8033
Parameter #2 = 2.7051 ± 4.6966
-------------------------------------------------
Function evaluation at 5 = 27.0750 ± 2.4345
Function evaluation at 15 = 75.8148 ± 8.3935
-------------------------------------------------
```

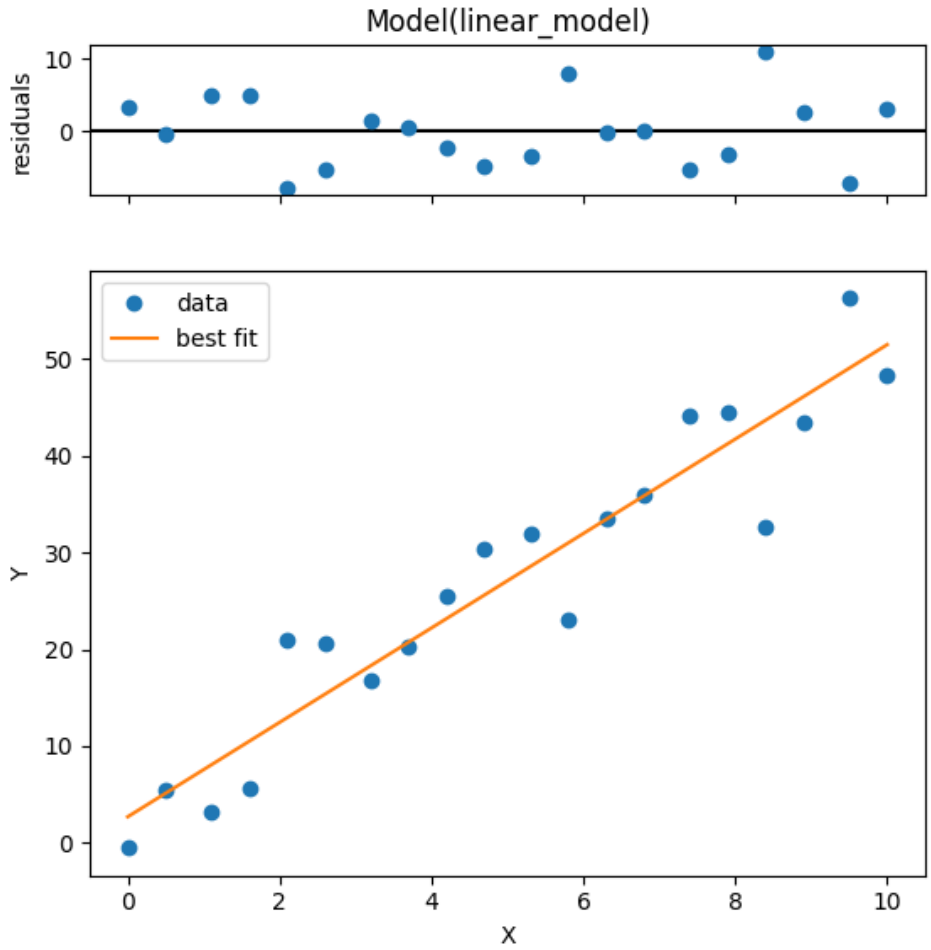


Figure 5.4: Plot of the fitted function and residuals.

5.6.2 Fitting in Excel

Unlike python where the same functions can be used to fit any functions, `Excel` can compute trend lines for a limited number of functions. However, statistical information

about the uncertainties of the fitting parameters can only be obtained using the `LINEST` and `LOGEST` functions that can be used for linear and exponential fitting, respectively. For any other non-linear fitting, you would need to use an external "Add-in" called `solver`, to find the optimal parameters that minimize the sum of the residuals squared. Similarly, the uncertainties of the fitting parameters would have to be computed by hand.

Linear fit

In `Excel` the `LINEST` function gives you more information than just the slope and intercept. It provides a set of statistics for the linear regression, which can be very useful for a thorough analysis. Note that using `Excel` we can get uncertainties only when doing a linear fit. If the function we want to fit is not a line, we need to linearize our data, often by taking the logarithm of the function or some other algebraic manipulation, to get the uncertainties. Using the procedure outlined below we can get:

- The full regression statistics from `LINEST`
- Predicted Y value for any given X
- Confidence interval for the fitting parameters
- Confidence interval for the prediction

It should then be pretty straightforward to compute the residuals and plot the data.

Let's imagine we have 20 (x, y) data points and we put them in columns A and B, starting from the second row; we leave the first row empty for some text headers. In order to use `LINEST` we select a cell *e.g.,* E2 and use this formula: `=LINEST(B2:B21,A2:A21,TRUE,TRUE)`. This will fill a 5x2 block of cells with the following information:

- E2: Slope
- F2: Intercept
- E3: Standard error of slope (68% CI)
- F3: Standard error of intercept (68% CI)
- E4: R-squared
- F4: Standard error of the y estimate
- E5: F statistic
- F5: Degrees of freedom
- E6: Regression sum of squares
- F6: Residual sum of squares

We can now calculate the fitted values for column C using this formula `=`$E2$`*A2+`$F2$ in cell C2 and double clicking on the square in the bottom right corner of the cell.

Let's now imagine we want to calculate the expectation value of the function for $X = 5$ and the 95% uncertainty. This can be done by evaluating the fitting function at the desired X value and computing the 95% confidence interval as

$$\sigma_{\hat{y}} = t \; s_{y,x} \sqrt{\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2}} \tag{5.6}$$

where t is the value from the Student's t table appropriate for the number of degrees of freedom. These are the formulae to fill in the cells in the spreadsheet:

B23 Number of values (`=COUNT(A2:A21)`)

A25 Average of the x values (`=AVERAGE(A2:A21)`)

E2:F6 Linear fit (`=LINEST(B2:B21,A2:A21,TRUE,TRUE)`)

F8 t-value for 95% CI (`=T.INV.2T((1-0.95),F5)`)

F9 95% uncertainty on the slope (`=F8*E3`)

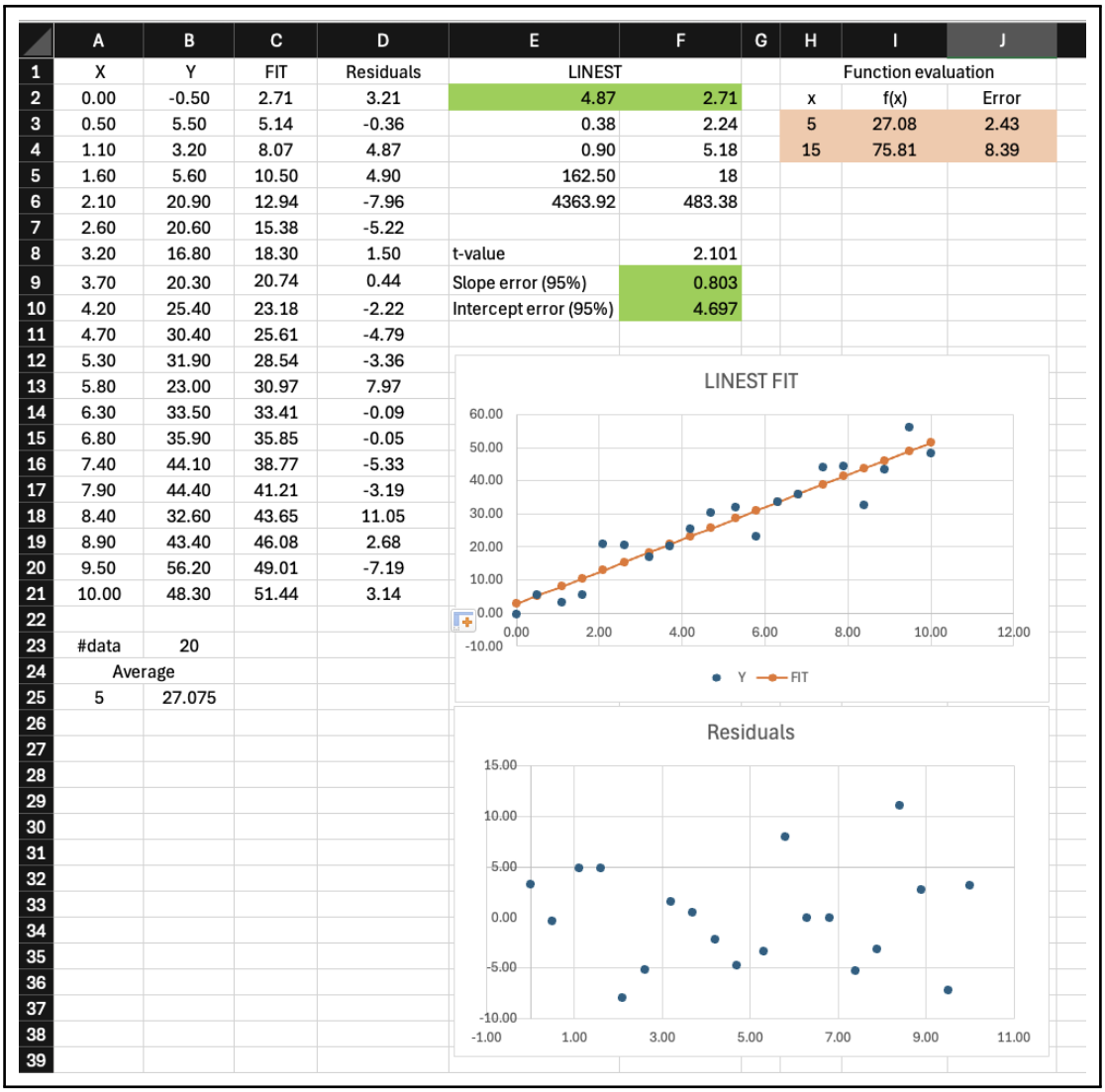
F10 95% uncertainty on the intercept (`=F8*F3`)

H2 desired X values for the prediction

I2 Predicted Y (`=H3*E2+F2`)

J2 Standard Error of Prediction, Eq. 5.6, (`=F8*F4*SQRT(1/B23+(H3-A25)^2/SUMSQ(A2:A21-A25))`)

We can then plot the data and the fit values (Figure 5.5)



	A	B	C	D	E	F	G	H	I	J
1	X	Y	FIT	Residuals	LINEST			Function evaluation		
2	0.00	-0.50	2.71	3.21	4.87	2.71		x	f(x)	Error
3	0.50	5.50	5.14	-0.36	0.38	2.24		5	27.08	2.43
4	1.10	3.20	8.07	4.87	0.90	5.18		15	75.81	8.39
5	1.60	5.60	10.50	4.90	162.50	18				
6	2.10	20.90	12.94	-7.96	4363.92	483.38				
7	2.60	20.60	15.38	-5.22						
8	3.20	16.80	18.30	1.50	t-value	2.101				
9	3.70	20.30	20.74	0.44	Slope error (95%)	0.803				
10	4.20	25.40	23.18	-2.22	Intercept error (95%)	4.697				
11	4.70	30.40	25.61	-4.79						
12	5.30	31.90	28.54	-3.36						
13	5.80	23.00	30.97	7.97						
14	6.30	33.50	33.41	-0.09						
15	6.80	35.90	35.85	-0.05						
16	7.40	44.10	38.77	-5.33						
17	7.90	44.40	41.21	-3.19						
18	8.40	32.60	43.65	11.05						
19	8.90	43.40	46.08	2.68						
20	9.50	56.20	49.01	-7.19						
21	10.00	48.30	51.44	3.14						
22										
23	#data	20								
24	Average									
25	5	27.075								
26										
27										
28										
29										
30										
31										
32										
33										
34										
35										
36										
37										
38										
39										

Figure 5.5: Example of an `Excel` spreadsheet to do a linear fit and evaluate the fitting function at a given point. The 95% confidence intervals for both the fitting parameters and the prediction for $X = 5$ are also calculated.

5.6.3 Non-linear fit in Excel

Fitting data to an arbitrary function using `Excel` is much more involved, and it is not possible to easily extract the uncertainties of the fitting parameters. Let's assume we have collected data from N measurements, with $\{X_1, \ldots, X_N\}$ being the independent variable, $\{Y_1, \ldots, Y_N\}$ the dependent variable and $\{\sigma_1, \ldots, \sigma_N\}$ the uncertainty of those measurements. We also know that these data should be modelled by an exponentially decaying function

$$f(x) = Ae^{-Bx} \tag{5.7}$$

where A and B are our fitting parameters, and we want to know what are the best parameters that reproduce our data.

In order to do a non-linear fit in `Excel` we need to recreate the least squares minimization algorithm in the spreadsheet and use the *Solver add-in*.

Starting from a spreadsheet that contains the same data used in the python example as three columns, we need to (Figure 5.6):

1. Define a starting guess for the parameters (`G2` and `G3`).
   It is very important for this guess to be reasonable, otherwise `Excel` will struggle to converge to the right solution.

2. Calculate the value of the fitting function for each point using those parameters (placed in column `C`)

3. Calculate the residuals (difference between columns `B` and `C`)

4. Calculate the sum of the residuals squared (`=SUMSQ(D2:D101)`) It is important that this number is "large", otherwise `Excel` may stop prematurely and not reach the correct solution. If the number is too small you can multiply it by a large arbitrary constant.

5. Plot the data and the function, and the residuals.

6. Launch the Solver Add-in (you may have to install it first)

7. Set the objective and parameters for the fit and run the solver

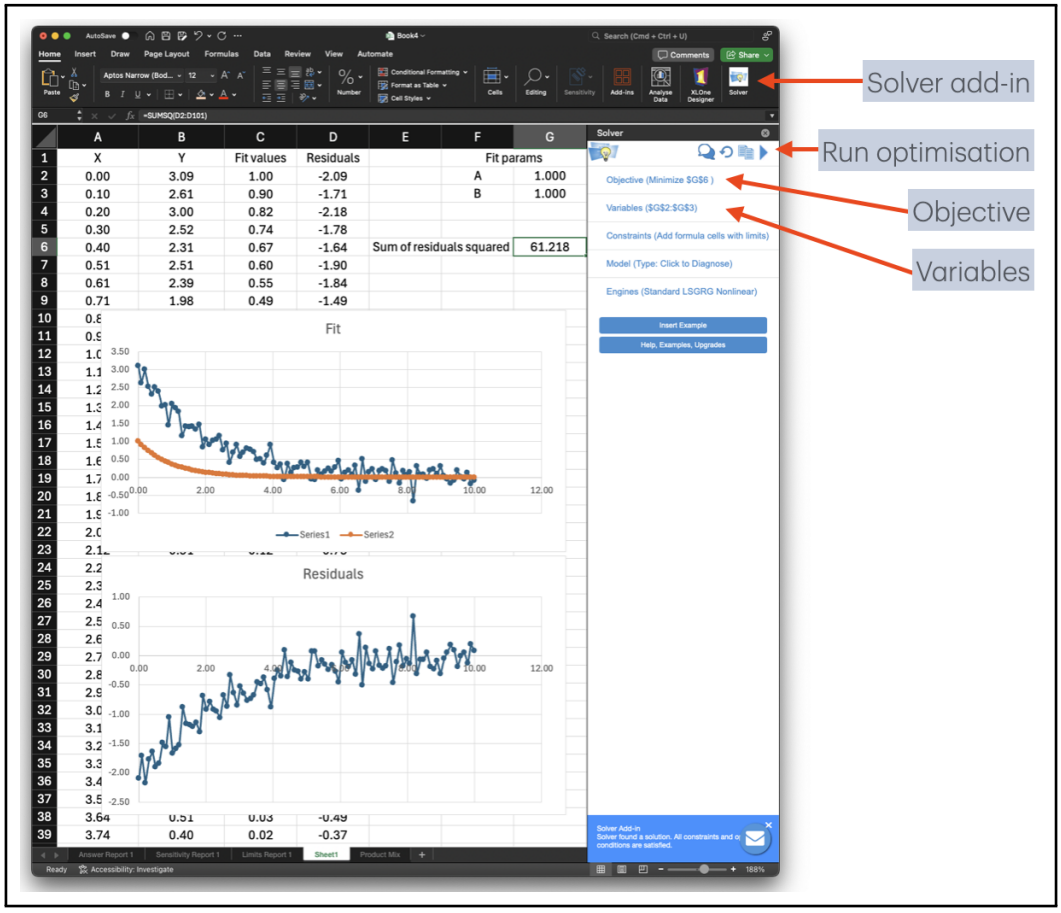
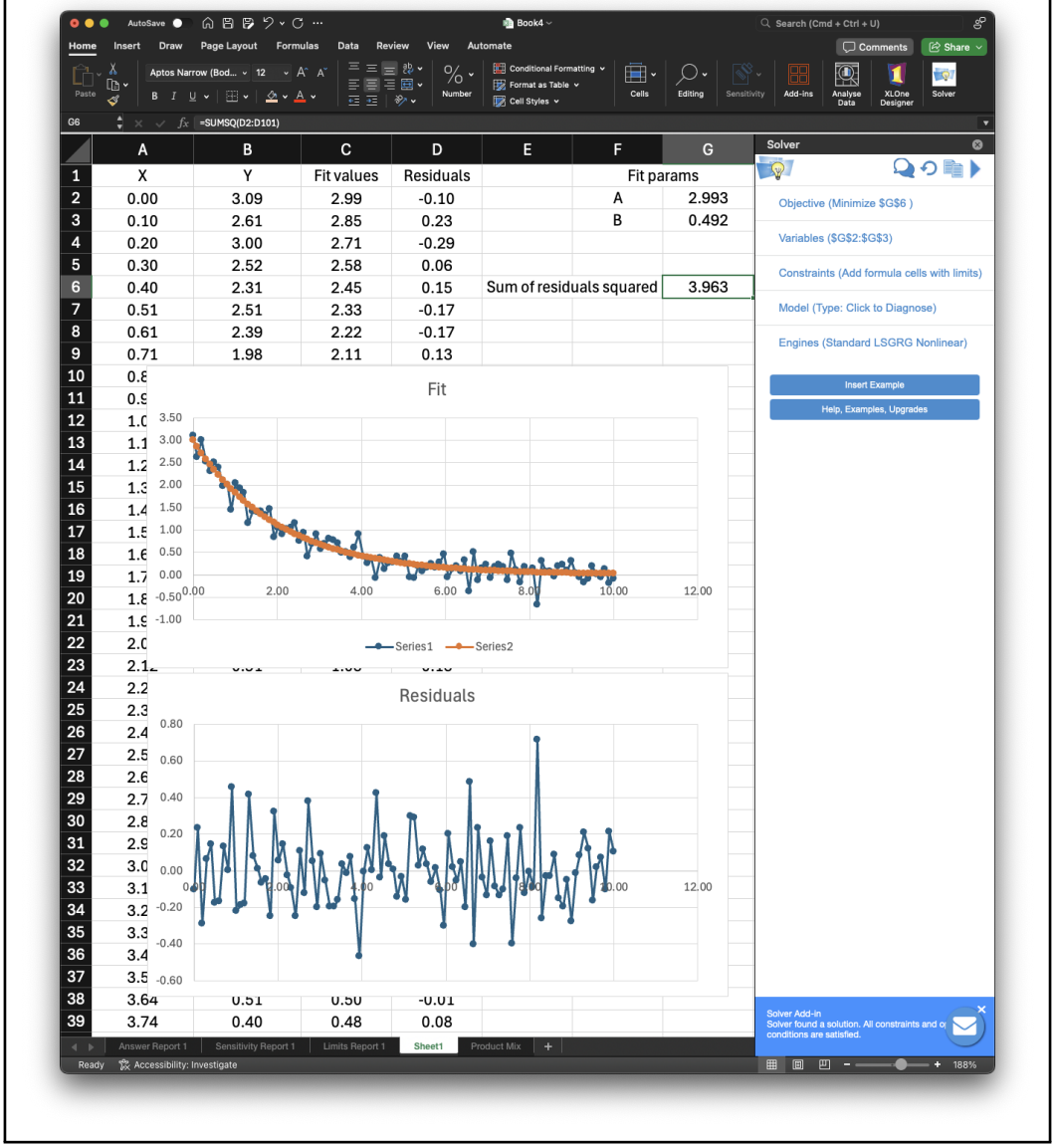


Figure 5.6: Screen capture of the `Excel` spreadsheet that we need to use to fit a custom function.

At this point, if everything worked, the parameters and the graph should be updated (Figure 5.7). If you have an older version of `Excel` the `Solver Add-in` may look different, check the Appendix.



	A	B	C	D	E	F	G
1	X	Y	Fit values	Residuals		Fit params	
2	0.00	3.09	2.99	-0.10		A	2.993
3	0.10	2.61	2.85	0.23		B	0.492
4	0.20	3.00	2.71	-0.29			
5	0.30	2.52	2.58	0.06			
6	0.40	2.31	2.45	0.15	Sum of residuals squared		3.963
7	0.51	2.51	2.33	-0.17			
8	0.61	2.39	2.22	-0.17			
9	0.71	1.98	2.11	0.13			
38	3.64	0.51	0.50	-0.01			
39	3.74	0.40	0.48	0.08			

Figure 5.7: Screen capture of the `Excel` spreadsheet after fitting a custom function.

6. Detection of outliers

Outlier detection is an important step in data analysis, particularly in experimental sciences like chemistry. Outliers are data points that deviate significantly from other observations in a dataset. They may arise from measurement errors, experimental anomalies, or genuine extreme values in the population being studied. Identifying outliers is important because they can disproportionately influence statistical analyses, potentially leading to skewed results and incorrect conclusions. Various statistical methods have been developed to detect outliers objectively, including Dixon's Q-test for small datasets and Grubbs' test for larger samples. These tests help researchers distinguish between data points that are likely to be genuine outliers and those that are simply at the extremes of normal variation. However, it's essential to approach outlier detection and removal with caution, always considering the context of the data and the potential scientific significance of extreme values.

6.1 Dixon's Q-test

Dixon's Q-test, also known as the Q-test, is a statistical method used to identify and reject outliers in small sample sizes, typically when the number of observations is between 3 and 10. The test compares the gap between the suspected outlier and its nearest neighbor to the range of the entire dataset. Obviously, the outliers can only be the minimum or the maximum value of a dataset. The Q-test statistic is calculated as follows:

$$Q = \frac{|x_{\text{suspect}} - x_{\text{nearest}}|}{\text{range}} \tag{6.1}$$

where:

- x_{suspect} is the suspected outlier value
- x_{nearest} is the nearest value to the suspected outlier
- range is the difference between the maximum and minimum values in the dataset

The calculated Q-value is then compared to a critical Q-value from Table 6.1, based on the sample size and desired confidence level. If the calculated Q-value exceeds the critical value, the suspected outlier is rejected.

6.2 Grubbs' Test

Grubbs' test, also known as the maximum normed residual test, is used to detect outliers in a univariate dataset that follows an approximately normal distribution. It can be applied to larger sample sizes compared to Dixon's Q-test. The Grubbs' test statistic (G) is calculated as:

$$G = \frac{\max_{i=1,\ldots,N} |Y_i - \bar{Y}|}{s} \tag{6.2}$$

where:

- Y_i is the ith observation in the dataset
- $\bar{Y}$ is the mean of the dataset

- s is the standard deviation of the dataset
- N is the number of observations

The calculated G-value is compared to a critical value based on the sample size and desired significance level. If the G-value exceeds the critical value, the null hypothesis is rejected, indicating the presence of an outlier.

Example 4

Consider the following dataset of reaction yields (%) from a chemical experiment:

$$92.3, 94.1, 93.7, 93.9, 94.2, 94.0, 93.8, 98.7$$

Determine if there are any outliers using both Dixon's Q-test and Grubbs' test at a 95% confidence level. **Solution:**

Dixon's Q-test

Arranging the data in ascending order:

$$92.3, 93.7, 93.8, 93.9, 94.0, 94.1, 94.2, 98.7$$

The suspected outlier is 98.7.

$$Q = \frac{|98.7 - 94.2|}{98.7 - 92.3} = \frac{4.5}{6.4} = 0.703 \tag{6.3}$$

For n = 8 and 95% confidence, the critical Q-value is 0.526. Since 0.703 > 0.526, we reject the null hypothesis and conclude that 98.7 is an outlier.

Grubbs' Test

Calculate the mean and standard deviation:

$$\bar{Y} = 94.34$$
$$s = 1.76$$

Calculate the G-statistic:

$$G = \frac{|98.7 - 94.34|}{1.76} = 2.34 \tag{6.4}$$

For n = 8 and $\alpha = 0.05$, the critical G-value is approximately 2.1266. Since 2.34 > 2.1266, we reject the null hypothesis and conclude that 98.7 is an outlier. Both tests identify 98.7 as an outlier in this dataset.

Unlike Dixon's Q-test, where the critical values need to be looked up in a table, the Grubbs' test critical values can be reliably calculated using Python and Excel. Here below you can find a sample Python code to solve the example above and the equivalent spreadsheet in Excel. For completeness, we also show the Grubbs' test for the minimum value and show that it is not an outlier.

Python Code Example

```python
import scipy.stats as stats
import numpy as np

def grubbs_critical_value(n, confidence_level=0.95):
    """
    Parameters:
        n: int
            sample size
        alpha: float
            confidence level, (default 95%)
    Returns:
        critical_value: float
            Grubbs' critical value for the sample size
    """
    alpha = 1 - confidence_level
    # alpha / (2*n) is the Bonferroni correction,
    # which accounts for multiple comparisons
    t_value = stats.t.ppf(1 - alpha / (2*n), n-2)
    numerator = (n - 1) * np.sqrt(t_value**2)
    denominator = np.sqrt(n) * np.sqrt(n - 2 + t_value**2)
    critical_value = numerator / denominator
    return critical_value

# Example usage
data = np.array([92.3, 93.7, 93.8, 93.9, 94.0, 94.1, 94.2, 98.7])
n = len(data)  # sample size

min_value = data.min()
max_value = data.max()
print(f"Average: {np.average(data)}")
print(f"StDev: {np.std(data,ddof=1)}")
print("Minimum value {} and its G value {:.4f}".format(
    min_value, np.abs(min_value-np.average(data))/np.std(data,ddof=1)))
print("Maximum value {} and its G value {:.4f}".format(
    max_value, np.abs(max_value-np.average(data))/np.std(data,ddof=1)))

confidence = 0.95  # significance level for two-sided test
critical_value = grubbs_critical_value(n, confidence)
print(f"Grubbs' test critical value for n={n}, CL={confidence}: {critical_value:.4f}")
```

Output

```
Average: 94.3375
StDev: 1.8615949382950412
Minimum value 92.3 and its G value 1.0945
Maximum value 98.7 and its G value 2.3434
Grubbs' test critical value for n=8, CL=0.95: 2.1266
```

In Excel, the Grubbs test can be done as shown in Figure 6.1.

6.3 Recursive Outlier Removal

Both Dixon's Q-test and Grubbs' test are designed to identify a single outlier in a dataset and should not be applied recursively to the same dataset. Removing multiple outliers through repeated application of these tests can lead to erroneous results and compromise the integrity of the data. The number of outliers that can be safely removed depends on the

	A	B	C	D	E
1	Raw data		Max	98.7	=MAX(A2:A9)
2	92.3		Min	92.3	=MIN(A2:A9)
3	93.7				
4	93.8				
5	93.9		Mean	94.3375	=AVERAGE(A2:A9)
6	94		St Dev	1.86159493829504	=STDEV(A2:A9)
7	94.1		G(MaxValue)	2.34342063907598	=(E1-E5)/E6
8	94.2		G(MinValue)	1.09449158787791	=(E5-E2)/E6
9	98.7				
10					
11			Confidence	0.05	0.05
12			Sample Size	8	=COUNT(A2:A9)
13			Significance	0.00625	=E11/E12
14			ndof	6	=E12-2
15			t_critical	2.44691185114497	=T.INV.2T(E11,E14)
16			G_critical	1.7490784053906	=(E12-1)*E15/SQRT(E12*(E14+E15^2))
17					

Figure 6.1: Example of how to do the Grubbs' test in Excel. Column E contains the formulæ used to compute the values in column D.

initial sample size and the nature of the data. As a general guideline, it is recommended to remove no more than one outlier for every 10-20 data points in the original dataset. However, the removal of any data point should be approached with caution and justified by sound scientific reasoning rather than relying solely on statistical tests. If multiple outliers are suspected, more robust methods designed for multiple outlier detection should be considered. Always document and report any data points removed as outliers, along with the justification for their removal.

6.4 Applying Grubbs' Test to Linear Regression Residuals

The Grubbs' test can be effectively applied to the residuals of a linear regression to identify potential outliers. The process simply involves the following steps:

1. Perform the linear regression on your dataset.
2. Calculate the residuals (the differences between observed and predicted values) for each data point.
3. Apply the Grubbs' test to these residuals.

This method assumes that the residuals are normally distributed, which is a common assumption in linear regression. The Grubbs' test will identify unusually large residuals, which correspond to data points that deviate significantly from the linear model. It's important to note that outliers in regression can be of two types: outliers in the y-direction (vertical outliers) and outliers in the x-direction (leverage points). The Grubbs' test on residuals will primarily identify vertical outliers. To detect leverage points, additional methods such as Cook's distance or hat values should be considered. As with any outlier detection method, results should be interpreted cautiously and in the context of the specific dataset and research question. Visualization techniques, such as residual plots, should be used in conjunction with statistical tests for a comprehensive analysis.

Table 6.1: Critical Values for Dixon's Q-test

n	90%	95%	99%
3	0.941	0.970	0.994
4	0.765	0.829	0.926
5	0.642	0.710	0.821
6	0.560	0.625	0.740
7	0.507	0.568	0.680
8	0.468	0.526	0.634
9	0.437	0.493	0.598
10	0.412	0.466	0.568
11	0.392	0.444	0.542
12	0.376	0.426	0.522
13	0.361	0.410	0.503
14	0.349	0.396	0.488
15	0.338	0.384	0.475
16	0.329	0.374	0.463
17	0.320	0.365	0.452
18	0.313	0.356	0.442
19	0.306	0.349	0.433
20	0.300	0.342	0.425
21	0.295	0.337	0.418
22	0.290	0.331	0.411
23	0.285	0.326	0.404
24	0.281	0.321	0.399
25	0.277	0.317	0.393
26	0.273	0.312	0.388
27	0.269	0.308	0.384
28	0.266	0.305	0.380

7. Calibration Curve Analysis and Uncertainty Calculation

7.1 Calibration Curve Construction

Calibration curves are fundamental tools in analytical chemistry and other scientific fields. They help us determine unknown concentrations of analytes by relating measured responses to known standard concentrations. This guide explains how to construct a calibration curve and, importantly, how to calculate the uncertainty in your measurements — a crucial aspect often overlooked in routine analysis.

$$y = mx + b$$

where m is the slope and b is the y-axis intercept. The linearity of your calibration curve is crucial. Always check the correlation coefficient (R^2) and visually inspect the residuals plot before proceeding.

7.1.1 Linear Regression Analysis

The method of least squares can be used to find the best-fitting line through our N calibration points (x_i, y_i). As explained before, this works by minimizing the sum of squares of the residuals (SS_{res}).

$$S_{res} = \sum [y_i - (mx_i + b)]^2$$

While this step is automatically done using `Excel` or `Python`, for completeness these are the formulas that can be used to compute the slope and the intercept. Their calculation can be greatly simplified by defining some quantities:

$$\bar{x} = \frac{\sum x_i}{N}$$

$$\bar{y} = \frac{\sum y_i}{N}$$

$$S_{xx} = \sum (x_i - \bar{x})^2$$

$$S_{yy} = \sum (y_i - \bar{y})^2$$

$$S_{xy} = \sum (x_i - \bar{x})(y_i - \bar{y})$$

Hence, we can define the slope of the calibration curve:

$$m = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{n\sum x_i^2 - (\sum x_i)^2} = \frac{S_{xy}}{S_{xx}}$$

and its y-axis intercept:

$$b = \frac{\sum y_i - m\sum x_i}{n} = \bar{y} - m\bar{x}$$

7.2 Uncertainty in Calibration Curve

Understanding the uncertainty in the fitting curve is also an essential step of the calibration procedure as it allows for:

- Assessing the quality of your calibration
- Determining confidence intervals for unknown samples
- Comparing results between different methods or laboratories

Using the quantities defined above, we can define the Standard Deviation of the Regression:

$$S_r = \sqrt{\frac{S_{yy} - m^2 S_{xx}}{N-2}} = \sqrt{\frac{S_{res}}{N-2}}$$

the Standard Deviation of the Slope:

$$S_m = \sqrt{\frac{S_r^2}{S_{xx}}}$$

and the Standard Deviation of the intercept:

$$S_b = S_r \sqrt{\frac{\sum x_i^2}{N S_{xx}}}$$

7.2.1 Sample Analysis

When analyzing unknown samples, it is normal practice to take multiple measurements to reduce random error. The average response ($\bar{z}$) can be used to obtain the best estimate of the sample concentration:

$$x_0 = \frac{\bar{z} - b}{m}$$

Assuming we have done M replicate measurements of the sample, the uncertainty in the concentration ($u_c(x_0)$) can be obtained by combining the uncertainty of the mean response with the uncertainty of the calibration curve:

$$S_c = \frac{S_r}{|m|} \sqrt{\frac{1}{M} + \frac{1}{N} + \frac{(\bar{z} - \bar{y})^2}{m^2 S_{xx}}}$$

Finally, the expanded uncertainty (U) at confidence level p can be calculated as:

$$U = t(p, \nu) S_c$$

where: - $t(p, \nu)$ is the Student's t-value - ν is the degrees of freedom (typically $N - 2$)

Hence, the final result should be reported as: $x_0 \pm U$ (units) at $p\%$ confidence level

7.2.2 Notes

1. All uncertainties should be calculated using the same units as the original measurements
2. The calibration range should bracket the expected sample concentrations
3. The residuals should be randomly distributed around zero
4. Check for outliers using statistical tests (e.g., Grubbs' test)

7.2.3 Examples

Python and Excel examples to compute the concentration of a sample using the following data for calibration and three replicate measurements of the sample concentration (0.148 ppb, 0.128 ppb and 0.156 ppb).

Concentration (ppb)	Absorbance
0	0.000
1.55E-03	0.050
3.16E-03	0.093
4.74E-03	0.143
6.34E-03	0.188
7.92E-03	0.236

Python Code Example

```
import numpy as np
from scipy import stats

concentration = np.array(
    [0, 1.55E-03, 3.16E-03, 4.74E-03, 6.34E-03, 7.92E-03]
    )
absorbance = np.array(
    [0, 0.05, 0.093, 0.143, 0.188, 0.236]
    )

samples = np.array(
    [0.148, 0.128, 0.156]
)

fit = np.polyfit( concentration, absorbance, 1 )

degrees_of_freedom = len(concentration)-2
residues = absorbance - (concentration*fit[0]+fit[1])
Sres = np.sum(residues**2)
Sxx = np.sum((concentration - np.mean(concentration))**2)
ybar = np.mean(absorbance)

Sr = np.sqrt(Sres/degrees_of_freedom)

Sc = Sr/fit[0]*np.sqrt(
    1/len(concentration) + 1/len(samples) + (np.mean(samples)-ybar)**2/fit[0]**2/Sxx
)

t_value_2t = stats.t.ppf((1 + 0.95)/2, degrees_of_freedom)

print("-"*50)
print(f"Slope     = {fit[0]:.5e}")
print(f"Intercept = {fit[1]:.5e}")
print(f"      Sxx = { Sxx:.5e}")
print(f"     ybar = {ybar:.5e}")
print(f"     Sres = {Sres:.5e}")
print(f"       Sr = {  Sr:.5e}")
print(f"       Sc = {  Sc:.5e}")
print(f"  t-value = {t_value_2t}")
```

```
print("-"*50)
print(f"Sample concentration {np.mean(samples):.5e} +/- {t_value_2t*Sc:.5e}")
print("-"*50)
```

Output

```
--------------------------------------------------
Slope     = 2.95927e+01
Intercept = 1.39272e-03
      Sxx = 4.40837e-05
     ybar = 1.18333e-01
     Sres = 1.59363e-05
       Sr = 1.99602e-03
       Sc = 4.85011e-05
  t-value = 2.7764451051977987
--------------------------------------------------
Sample concentration 1.44000e-01 +/- 1.34661e-04
--------------------------------------------------
```

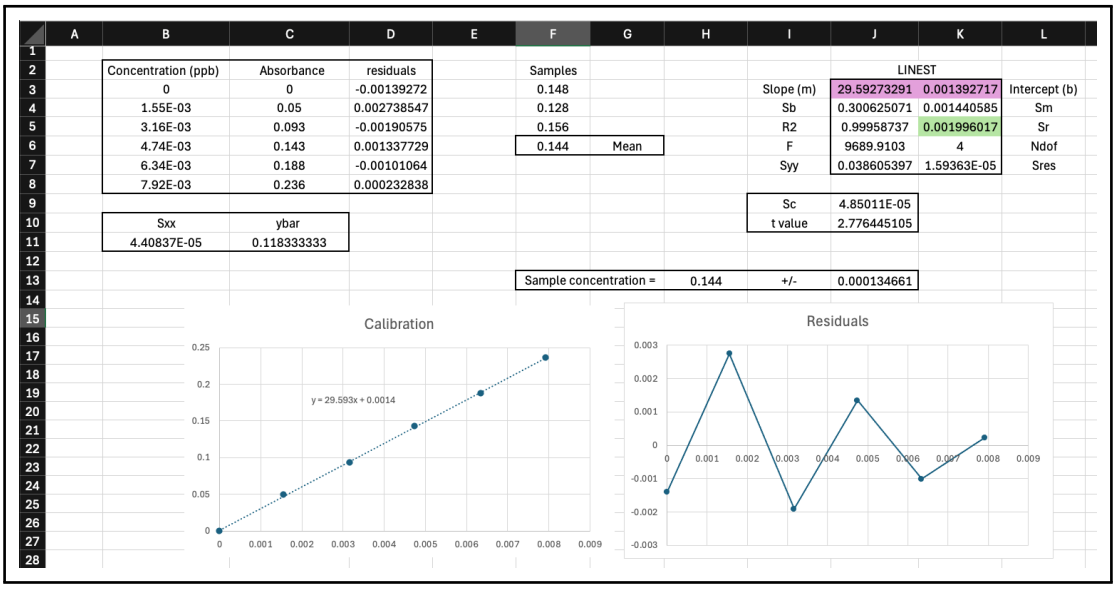


Figure 7.1: Example of a calibration curve done in Excel using LINEST.

8. Appendix

8.1 Student's t distribution

Table 8.1: List of tabulated values to compute the confidence interval using the Student's t test distribution for sample sizes of $\nu = n - 1$ degrees of freedom and different confidence intervals. "One-sided" and "Two-sided" refer to the cases where only one side of the distribution is excluded or both sides are excluded. The "Two-sided" case is the most commonly used.

	Confidence Interval										
"One-sided"	75%	80%	85%	90%	95%	97.5%	99%	99.5%	99.75%	99.9%	99.95%
"Two-sided"	50%	60%	70%	80%	90%	95%	98%	99%	99.5%	99.8%	99.9%
ν =1	1.000	1.376	1.963	3.078	6.314	12.706	31.821	63.657	127.321	318.309	636.619
2	0.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	14.089	22.327	31.599
3	0.765	0.978	1.250	1.638	2.353	3.182	4.541	5.841	7.453	10.215	12.924
4	0.741	0.941	1.190	1.533	2.132	2.776	3.747	4.604	5.598	7.173	8.610
5	0.727	0.920	1.156	1.476	2.015	2.571	3.365	4.032	4.773	5.893	6.869
6	0.718	0.906	1.134	1.440	1.943	2.447	3.143	3.707	4.317	5.208	5.959
7	0.711	0.896	1.119	1.415	1.895	2.365	2.998	3.499	4.029	4.785	5.408
8	0.706	0.889	1.108	1.397	1.860	2.306	2.896	3.355	3.833	4.501	5.041
9	0.703	0.883	1.100	1.383	1.833	2.262	2.821	3.250	3.690	4.297	4.781
10	0.700	0.879	1.093	1.372	1.812	2.228	2.764	3.169	3.581	4.144	4.587
11	0.697	0.876	1.088	1.363	1.796	2.201	2.718	3.106	3.497	4.025	4.437
12	0.695	0.873	1.083	1.356	1.782	2.179	2.681	3.055	3.428	3.930	4.318
13	0.694	0.870	1.079	1.350	1.771	2.160	2.650	3.012	3.372	3.852	4.221
14	0.692	0.868	1.076	1.345	1.761	2.145	2.624	2.977	3.326	3.787	4.140
15	0.691	0.866	1.074	1.341	1.753	2.131	2.602	2.947	3.286	3.733	4.073
16	0.690	0.865	1.071	1.337	1.746	2.120	2.583	2.921	3.252	3.686	4.015
17	0.689	0.863	1.069	1.333	1.740	2.110	2.567	2.898	3.222	3.646	3.965
18	0.688	0.862	1.067	1.330	1.734	2.101	2.552	2.878	3.197	3.610	3.922
19	0.688	0.861	1.066	1.328	1.729	2.093	2.539	2.861	3.174	3.579	3.883
20	0.687	0.860	1.064	1.325	1.725	2.086	2.528	2.845	3.153	3.552	3.850
21	0.686	0.859	1.063	1.323	1.721	2.080	2.518	2.831	3.135	3.527	3.819
22	0.686	0.858	1.061	1.321	1.717	2.074	2.508	2.819	3.119	3.505	3.792
23	0.685	0.858	1.060	1.319	1.714	2.069	2.500	2.807	3.104	3.485	3.767
24	0.685	0.857	1.059	1.318	1.711	2.064	2.492	2.797	3.091	3.467	3.745
25	0.684	0.856	1.058	1.316	1.708	2.060	2.485	2.787	3.078	3.450	3.725
26	0.684	0.856	1.058	1.315	1.706	2.056	2.479	2.779	3.067	3.435	3.707
27	0.684	0.855	1.057	1.314	1.703	2.052	2.473	2.771	3.057	3.421	3.690
28	0.683	0.855	1.056	1.313	1.701	2.048	2.467	2.763	3.047	3.408	3.674
29	0.683	0.854	1.055	1.311	1.699	2.045	2.462	2.756	3.038	3.396	3.659
30	0.683	0.854	1.055	1.310	1.697	2.042	2.457	2.750	3.030	3.385	3.646
40	0.681	0.851	1.050	1.303	1.684	2.021	2.423	2.704	2.971	3.307	3.551
50	0.679	0.849	1.047	1.299	1.676	2.009	2.403	2.678	2.937	3.261	3.496
60	0.679	0.848	1.045	1.296	1.671	2.000	2.390	2.660	2.915	3.232	3.460
80	0.678	0.846	1.043	1.292	1.664	1.990	2.374	2.639	2.887	3.195	3.416
100	0.677	0.845	1.042	1.290	1.660	1.984	2.364	2.626	2.871	3.174	3.390
120	0.677	0.845	1.041	1.289	1.658	1.980	2.358	2.617	2.860	3.160	3.373
$\nu = \infty$	0.674	0.842	1.036	1.282	1.645	1.960	2.326	2.576	2.807	3.090	3.291
	Confidence Interval										
"One-sided"	75%	80%	85%	90%	95%	97.5%	99%	99.5%	99.75%	99.9%	99.95%
"Two-sided"	50%	60%	70%	80%	90%	95%	98%	99%	99.5%	99.8%	99.9%

8.2 Collection of Python functions

Collection of useful functions to perform (non-)linear fits in Python using `SciPy` or `lmfit`.

Computing the Student's t-value

Python Code Example

```python
def get_t_value(ndof, confidence=0.95):
    """
    Calculate the two-tailed Student's t-value for a given number of degrees of
    freedom and confidence level.

    Parameters:
        ndof (int): Number of degrees of freedom
        confidence (float): Confidence level, default is 0.95 for 95% confidence

    Returns:
        float: The critical t-value for the specified parameters

    Example:
        For 95% confidence and 10 degrees of freedom:
        >>> get_t_value(10)
        2.2281388519495335
    """
    from scipy import stats

    # Calculate alpha (significance level) from confidence level
    # For 95% confidence, alpha = 0.05
    alpha = 1 - confidence

    # Calculate the t-value using the percent point function (PPF) of the
    # t-distribution. We use alpha/2 for two-tailed test and (1 - alpha/2) for
    # the upper tail
    tval = stats.t.ppf(1.0 - alpha/2., ndof)

    return tval
```

Fitting data using a model function - `SciPy`

Python Code Example

```python
def function_fit_scipy(x_data, y_data, model_func):
    """
    Fit a model function to data using scipy's curve_fit with optional weights
    and calculate confidence intervals.

    Parameters:
    x_data (array-like): Independent variable data points
    y_data (array-like): Dependent variable data points
    model_func (callable): The model function to fit, must be of form f(x, *params)

    Returns:
    dict: Dictionary containing:
        popt: Optimal parameter values from the fit
        pcov: Covariance matrix of parameters
        perr: Standard deviation of each parameter
```

```
        ndof: Number of degrees of freedom
        chi2: Chi-squared value of the fit
        rchi2: Reduced Chi-squared value of the fit


    Example:
    >>> def linear_model(x, m, b):
    ...     return m*x + b
    >>> x = np.array([1, 2, 3, 4, 5])
    >>> y = np.array([2.1, 4.0, 6.2, 7.9, 9.8])
    >>> uncertainties = np.array([0.2, 0.3, 0.2, 0.4, 0.3])
    >>> result = function_fit_scipy(x, y, linear_model, weights=uncertainties)
    """
    import numpy as np
    from scipy.optimize import curve_fit

    # Use scipy's curve_fit to find optimal parameters and covariance matrix
    popt, pcov = curve_fit(model_func, x_data, y_data)

    # Calculate degrees of freedom: number of data points minus number of parameters
    ndof = max(0, len(x_data) - len(popt))

    # Calculate standard error for each parameter from diagonal of covariance matrix
    perr = np.sqrt(np.diag(pcov))

    # Calculate chi-squared of the fit
    residuals = y_data - model_func(x_data, *popt)
    chi2 = np.sum(residuals**2)

    # Calculate reduced chi-squared
    reduced_chi2 = chi2 / ndof if ndof > 0 else np.inf

    # Package all results into a dictionary for easy access
    result = {
        'popt' : popt,          # Optimal parameters
        'pcov' : pcov,          # Covariance matrix
        'perr' : perr,          # Standard errors
        'ndof' : ndof,          # Degrees of freedom
        'chi2' : chi2,          # Chi-squared value of the fit
        'rchi2': reduced_chi2   # Reduced chi-squared
    }

    return result
```

Fitting data using a model function - `lmfit`

Python Code Example

```
def function_fit_lmfit(x_data, y_data, model_func):
    """
    Fit a model function to data using lmfit's Model class with optional weights
    and calculate confidence intervals.

    Parameters:
    x_data (array-like): Independent variable data points
    y_data (array-like): Dependent variable data points
    model_func (callable): The model function to fit, must be of form f(x, *params)
```

```
    Returns:
    dict: Dictionary containing:
        popt: Optimal parameter values from the fit
        pcov: Covariance matrix of parameters
        perr: Standard deviation of each parameter
        ndof: Number of degrees of freedom
        chi2: Chi-squared value of the fit
        rchi2: Reduced Chi-squared value of the fit


    Example:
    >>> def linear_model(x, m, b):
    ...     return m*x + b
    >>> x = np.array([1, 2, 3, 4, 5])
    >>> y = np.array([2.1, 4.0, 6.2, 7.9, 9.8])
    >>> uncertainties = np.array([0.2, 0.3, 0.2, 0.4, 0.3])
    >>> result = function_fit_lmfit(x, y, linear_model, weights=uncertainties)
    """

    try:
        import lmfit
    except:
        !pip install -q lmfit
        import lmfit

    # Create the model
    model = lmfit.Model(model_func)

    # Set up parameters
    params = model.make_params()

    # Perform the fit
    result = model.fit(y_data, params, x=x_data)

    # Print fit report
    # print(result.fit_report())

    # Get optimal parameters as array
    popt = np.array([param.value for param in result.params.values()])

    # Get covariance matrix
    pcov = result.covar

    # Get parameter uncertainties
    perr = np.array([param.stderr for param in result.params.values()])

    # Calculate degrees of freedom
    ndof = result.ndata - result.nvarys

    # Get chi-squared
    chi2 = result.chisqr

    # Get reduced chi-squared
    reduced_chi2 = result.redchi

    result = {
        'popt' : popt,          # Optimal parameters
        'pcov' : pcov,          # Covariance matrix
        'perr' : perr,          # Standard errors
```

```
        'ndof' : ndof,           # Degrees of freedom
        'chi2' : chi2,           # Chi-squared value of the fit
        'rchi2': reduced_chi2    # Reduced chi-squared
    }

    return result
```

Evaluate a function at a given point

Python Code Example

```
def function_eval_fit(x_point, popt, pcov, model_func, epsilon=1e-6):
    """
    Evaluate a fitted (non-)linear function at a specific point and calculate the
    confidence interval.

    Parameters:
        x_point (float or array-like): Point(s) at which to evaluate the function
        popt: Optimal parameter values from the fit
        pcov: Covariance matrix of parameters
        model_func (callable): The model function used in the fit
        epsilon (float): Step size for numerical Jacobian approximation

    Returns:
        tuple: (y_predicted, confidence_interval)
            y_predicted: Model prediction at x_point
            confidence_interval: Width of confidence interval at x_point
    """

    import numpy as np
    from scipy.optimize import approx_fprime

    # Ensure x_point is a numpy array
    xp = np.atleast_1d(x_point)

    # Evaluate model at x_point
    y_pred = model_func(xp, *popt)

    # Approximate Jacobian matrix numerically using finite differences
    def wrapper(params):
        return model_func(xp, *params)

    # Compute Jacobian and ensure it's a 2D array
    jacob = approx_fprime(popt, wrapper, epsilon)
    jacob = np.atleast_2d(jacob)  # Ensure Jacobian is 2D

    # Compute standard error using covariance matrix
    sigma = np.sqrt(np.einsum('ij,ji->i', jacob @ pcov, jacob.T))

    # Return predicted value and confidence interval width
    return y_pred[0], sigma[0]
```

Function to plot the residuals

Python Code Example

```python
def plot_residuals_scipyFit(x_data, y_data, fit_model, popt):
    """
    Create a residual plot to visualize the differences between observed
    and predicted values.
    Residuals are the differences between actual y values and model predictions.

    Parameters:
        x_data (array-like): Independent variable data points (input for the model)
        y_data (array-like): Observed dependent variable values (true values)
        fit_model (callable): The model function used in the fit
        popt: Optimal parameter values from the fit
            The 'result' dictionary should include at least the 'popt' key,
            which contains the optimal parameters for the fitted model.

    Returns:
        None: Displays a matplotlib plot of residuals

    Example:
        >>> fit_result = function_fit_scipy(x_data, y_data, linear_model)
        >>> plot_residuals_scipyFit(x_data, y_data, linear_model, fit_result['popt'])
    """

    import matplotlib.pyplot as plt

    y_pred = fit_model(x_data, *popt)

    # Calculate residuals (observed y values - predicted y values)
    residuals = y_data - y_pred

    # Create a new figure with a specified size for the plot
    plt.figure(figsize=(10, 6))

    # Scatter plot of the residuals vs the x_data points
    # Each point in the scatter plot corresponds to the residual
    # for a specific x_data value
    plt.scatter(x_data, residuals, color='blue', label='Data')

    # Add a horizontal line at y=0 for reference to see how residuals deviate from 0
    plt.axhline(y=0, color='black', linestyle='--', label='Zero Residuals')

    # Label the x and y axes
    plt.xlabel('x')  # x-axis represents the independent variable
    plt.ylabel('e')  # y-axis represents the residuals or errors

    # Add a title to the plot
    plt.title('Plot of the residuals')

    # Optionally, add a legend to clarify the plot's labels
    plt.legend()

    # Display the plot
    plt.savefig("fit_residuals.png")
    # plt.show()
```

Plot the best fitting function with the confidence band

Python Code Example

```python
def plot_fit_with_confidence_band(x_data, y_data, fit_model, popt, confidence=0.95):
    """
    Plot the data along with the best fit line and its associated confidence band.

    Parameters:
        x_data (array-like): Independent variable data points
        y_data (array-like): Observed dependent variable values
        fit_model (callable): The model function used in the fit
        popt: Optimal parameter values from the fit
        confidence (float, optional): Confidence level for the confidence band
                                      (default is 0.95)

    Returns:
        None: Displays a matplotlib plot with data, fit line, and confidence band

    Example:
        >>> fit_result = function_fit_scipy(x_data, y_data, linear_model)
        >>> plot_fit_with_confidence_band(x_data, y_data, linear_model, fit_result['popt'])
    """

    import matplotlib.pyplot as plt

    # Calculate degrees of freedom: number of data points minus number of parameters
    ndof = max(0, len(x_data) - len(popt))

    # t-statistic used to calculate the confidence interval
    tval = get_t_value(ndof, confidence)

    # Create the plot figure with a specified size
    plt.figure(figsize=(10, 6))

    # Plot the observed data points as a scatter plot
    plt.scatter(x_data, y_data, color='blue', label='Data')

    # Generate points for the fitted line
    # Linearly spaced between the minimum and maximum of x_data
    x_fit = np.linspace(min(x_data), max(x_data), 100)

    # Calculate the fitted y values using the optimal parameters
    y_fit = fit_model(x_fit, *popt)

    # Calculate the error for the confidence band based on the data and fit
    y_err = np.sqrt(1/len(x_data) + (x_fit - np.mean(x_data))**2 /
                    np.sum((x_data - np.mean(x_data))**2))

    # Calculate the upper and lower bounds of the confidence band
    bounds = np.sqrt(np.sum((y_data - fit_model(x_data, *popt))**2) / ndof)
    y_upper = y_fit + tval * y_err * bounds
    y_lower = y_fit - tval * y_err * bounds

    # Plot the fitted line
    plt.plot(x_fit, y_fit, 'r-', label='Best fit')

    # Plot the confidence band by shading the area between the upper and lower bounds
    plt.fill_between(x_fit, y_lower, y_upper,
                     color='gray', alpha=0.2,
```

```
                    label=f'{int(confidence*100)}% Confidence band')

    # Add labels and title to the plot
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'Data with Linear Fit and {int(confidence*100)}% Confidence Bands')

    # Display the legend
    plt.legend()

    # Show the plot
    plt.savefig("fit_scipy_band.png")

    plt.show()
```

Example of using the above functions

Python Code Example

```
import numpy as np

# Example data
data = np.array([
    [ 0.0, -0.5, 0.4],
    [ 0.5,  5.5, 0.4],
    [ 1.1,  3.2, 0.1],
    [ 1.6,  5.6, 0.2],
    [ 2.1, 20.9, 0.2],
    [ 2.6, 20.6, 0.1],
    [ 3.2, 16.8, 0.4],
    [ 3.7, 20.3, 0.4],
    [ 4.2, 25.4, 0.4],
    [ 4.7, 30.4, 0.4],
    [ 5.3, 31.9, 0.2],
    [ 5.8, 23.0, 0.2],
    [ 6.3, 33.5, 0.3],
    [ 6.8, 35.9, 0.1],
    [ 7.4, 44.1, 0.2],
    [ 7.9, 44.4, 0.2],
    [ 8.4, 32.6, 0.2],
    [ 8.9, 43.4, 0.2],
    [ 9.5, 56.2, 0.3],
    [10.0, 48.3, 0.2]
])

x_data = data[:,0]
y_data = data[:,1]
e_data = data[:,2]

# Define the model function (linear in this case)
# Remember to initialize the values
def linear_model(x, m=1, b=1):
    return m * x + b

# Fit the data - Scipy
result = function_fit_scipy(x_data, y_data, linear_model)

# Calculate degrees of freedom: number of data points minus number of parameters
```

```
ndof = max(0, len(x_data) - len(result['popt']))

# t-statistic used to calculate the confidence interval (95%)
tval = get_t_value(ndof, 0.95)

print(f"Fit results SCIPY:")
print(f"Chi^2         = {result['chi2']}")
print(f"Reduced Chi^2 = {result['rchi2']}")
for i in range( len(result['popt']) ):
    p = result['popt'][i]
    e = result['perr'][i] * tval
    print(f"Parameter #{i+1} = {p:.4f} ± {e:.4f}")
print("-"*50)

for xval in [5,15]:
  v,e = function_eval_fit(xval,result['popt'],result['pcov'],linear_model)
  print(f"Function evaluation at {xval} = {v:.4f} ± {e*tval:.4f}" )
print("-"*50)

# Fit the data - lmfit
result = function_fit_lmfit(x_data, y_data, linear_model)

print(f"Fit results LMFIT:")
print(f"Chi^2         = {result['chi2']}")
print(f"Reduced Chi^2 = {result['rchi2']}")
for i in range( len(result['popt']) ):
    p = result['popt'][i]
    e = result['perr'][i] * tval
    print(f"Parameter #{i+1} = {p:.4f} ± {e:.4f}")
print("-"*50)

for xval in [5,15]:
  v,e = function_eval_fit(xval,result['popt'],result['pcov'],linear_model)
  print(f"Function evaluation at {xval} = {v:.4f} ± {e*tval:.4f}" )
print("-"*50)
```

Output

```
Fit results SCIPY:
Chi^2         = 483.3801211213936
Reduced Chi^2 = 26.854451173410755
Parameter #1 = 4.8740 ± 0.8033
Parameter #2 = 2.7051 ± 4.6966
--------------------------------------------------
Function evaluation at 5 = 27.0750 ± 2.4345
Function evaluation at 15 = 75.8148 ± 8.3935
--------------------------------------------------
Fit results LMFIT:
Chi^2         = 483.3801211213936
Reduced Chi^2 = 26.854451173410755
Parameter #1 = 4.8740 ± 0.8033
Parameter #2 = 2.7051 ± 4.6966
--------------------------------------------------
Function evaluation at 5 = 27.0750 ± 2.4345
Function evaluation at 15 = 75.8148 ± 8.3935
--------------------------------------------------
```

Plotting the fitting results

Python Code Example

```
# Plot of the residuals
plot_residuals_scipyFit(x_data, y_data, linear_model, result['popt'])

# Plot of the data and best fitting function
plot_fit_with_confidence_band(
    x_data, y_data, linear_model, result['popt'], confidence=0.95)
```

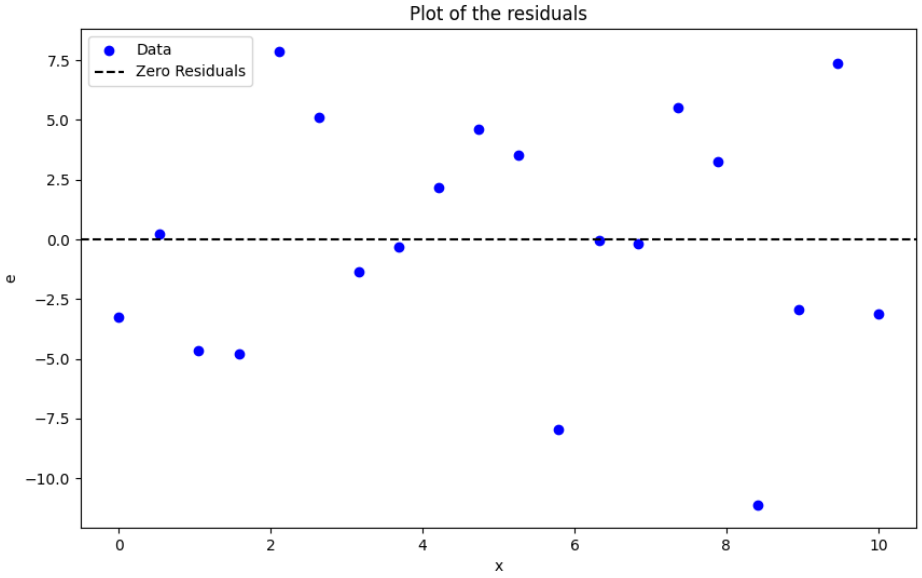


Figure 8.1: Plot of the residuals.

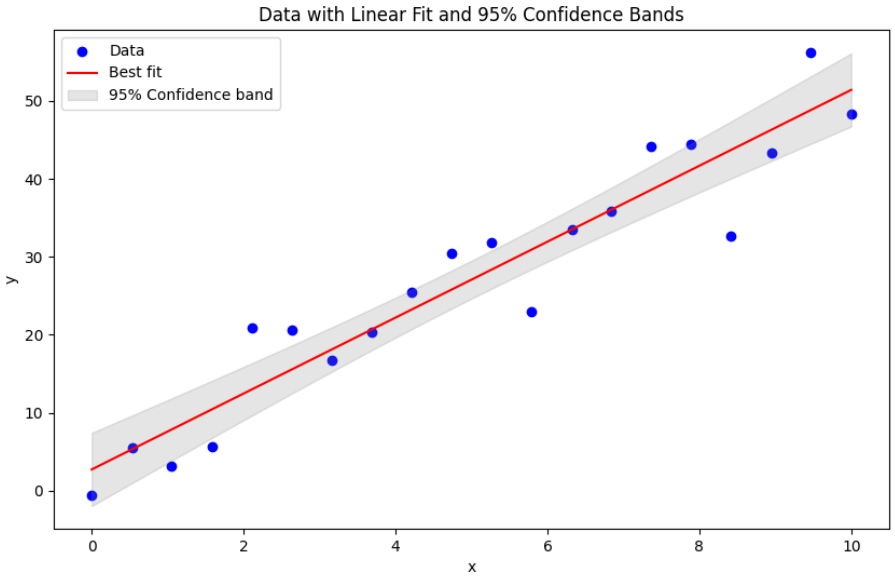


Figure 8.2: Plot of the fit with the 95% confidence band.