Exercise 3.1

The Python code for performing the fitting is given below in listing 1.

(a) See Fig. 1. The coefficients are 15.39 cm and -0.032 s^{-1}.

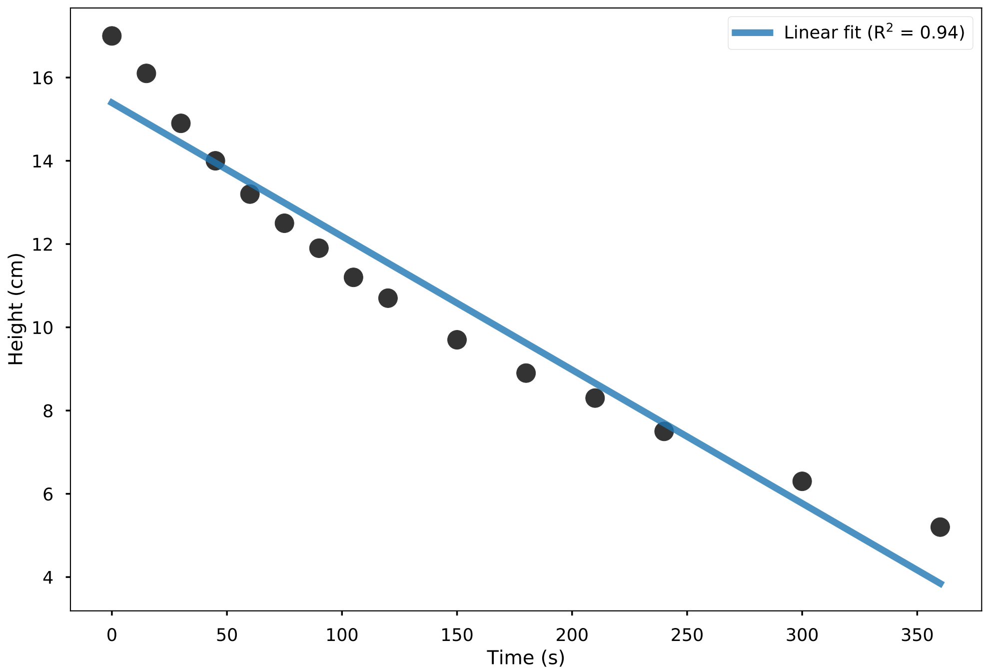

Figure 1: *Linear fit to the data.*

(b) See Fig. 1.

(c) Starting from:

$$h(t) = h(0)\exp\left(-\frac{t}{\tau}\right),$$

we take the natural logarithm ("ln") on both sides:

$$\ln\left(\frac{h(t)}{h(0)}\right) = \ln\left(\frac{h(t)\times 1\text{ cm}}{h(0)\times 1\text{ cm}}\right)\ln\left(\exp\left(-\frac{t}{\tau}\right)\right) = -\frac{t}{\tau},$$
$$\ln\left(\frac{h(t)}{1\text{ cm}}\right) = \ln\left(\frac{h(0)}{1\text{ cm}}\right) - \frac{t}{\tau}.$$

Comparing this with $y = a + bx$ gives:

$$
\begin{aligned}
y &= \ln\left(\frac{h(t)}{1\ \text{cm}}\right), \\
a &= \ln\left(\frac{h(0)}{1\ \text{cm}}\right), \\
b &= -\frac{1}{\tau}, \\
x &= t.
\end{aligned}
$$

Note: The "1 cm" is included here to avoid taking the logarithm of a number with a unit.

(d) See Fig. 2. The coefficients are $h(0) = 16.2$ cm and $\tau = 310$ s.

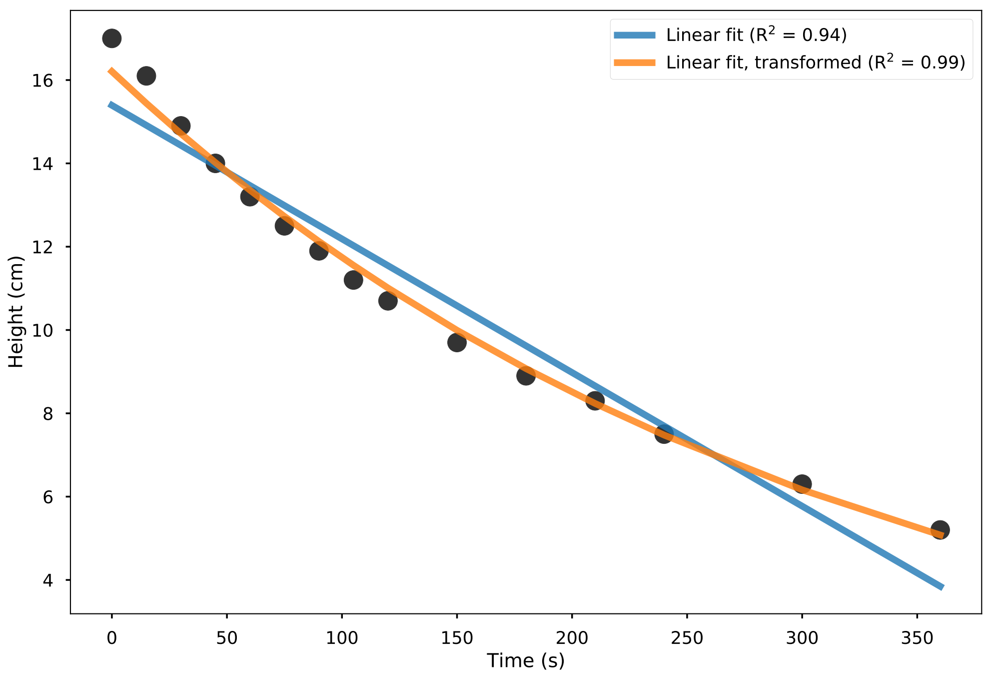

Figure 2: *Linear fit to the data after transformation.*

(e) The residual is given by $r_i = y_i' - bx_i$. The error is,

$$S = \sum_{i=1}^{N} r_i^2.$$

Let us minimize the error. We have:

$$\frac{\partial S}{\partial b} = -2\sum_{i=1}^{N} r_i x_i, \quad \frac{\partial^2 S}{\partial b^2} = 2\sum_{i=1}^{N} x_i^2 \geq 0,$$

Note that the second derivative is positive (except for the trivial case when $x_i = 0$) and we are indeed going to find a minimum.

Requiring that $\frac{\partial S}{\partial b} = 0$ gives,

$$-2\sum_{i=1}^{N} r_i x_i = 0 \implies \sum_{i=1}^{N} (y_i' x_i - b x_i^2) = 0 \implies b = \frac{\sum_{i=1}^{N} y_i' x_i}{\sum_{i=1}^{N} x_i^2}.$$

(f) See Fig. 3. The coefficients are $h(0) = 17.0$ cm and $\tau = 290$ s.

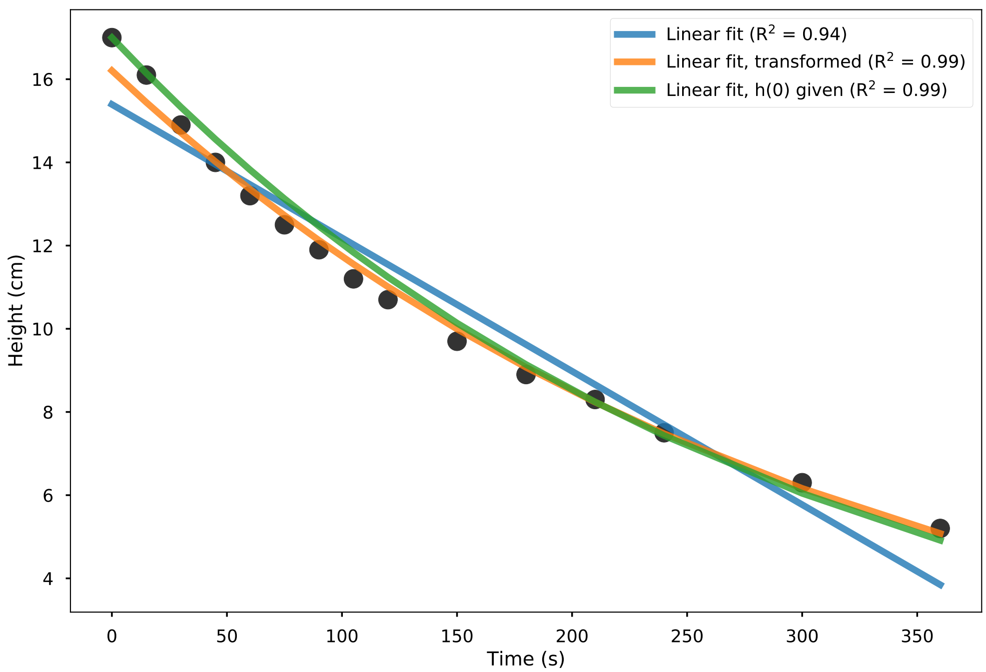

Figure 3: *Linear fit to the data after transformation with the initial height fixed.*

Exercise 3.2

(a) We rewrite the least squares expression to save some typing:

$$b = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{B},$$

where we let $B = \sum_{i=1}^{n}(x_i - \bar{x})^2$. Next, we are considering b as a function of the y_i's. To use the error propagation approach, we now determine $\frac{\partial b}{\partial y_j}$. First, it is helpful to determine the following:

$$
\begin{aligned}
\frac{\partial y_i}{\partial y_j} &= \delta_{ij}, \\
\frac{\partial \bar{y}}{\partial y_j} &= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial y_i}{\partial y_j} = \frac{1}{n}\sum_{i=1}^{n}\delta_{ij} = \frac{1}{n},
\end{aligned}
$$

where δ_{ij} is the Kronecker delta which is 1 if $i = j$ and 0 otherwise. We then get,

$$
\begin{aligned}
\frac{\partial b}{\partial y_j} &= \frac{1}{B}\sum_{i=1}^{n}(x_i - \bar{x})\left(\frac{\partial y_i}{\partial y_j} - \frac{\partial \bar{y}}{\partial y_j}\right) = \frac{1}{B}\sum_{i=1}^{n}(x_i - \bar{x})\left(\delta_{ij} - \frac{1}{n}\right) \\
&= \frac{1}{B}\left[\sum_{i=1}^{n} x_i\delta_{ij} - \sum_{i=1}^{n}\frac{x_i}{n} - \sum_{i=1}^{n}\bar{x}\delta_{ij} + \sum_{i=1}^{n}\frac{\bar{x}}{n}\right] \\
&= \frac{1}{B}\left[x_j - \bar{x} - \bar{x} + \bar{x}\right] \\
&= \frac{1}{B}(x_j - \bar{x}).
\end{aligned}
$$

The error in b is then:

$$
\begin{aligned}
\sigma_b^2 &= \sum_{i=1}^{n}\left(\frac{\partial b}{\partial y_i}\right)^2\sigma_y^2 = \frac{\sigma_y^2}{B^2}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{\sigma_y^2}{B^2}B = \frac{\sigma_y^2}{B} \\
&= \frac{\sigma_y^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2},
\end{aligned}
$$

which is the expression we were asked to derive.

(b) For the error in a, we take the derivative of the least squares solution $a = \bar{y} - b\bar{x}$:

$$
\begin{aligned}
\frac{\partial a}{\partial y_j} &= \frac{\partial \bar{y}}{\partial y_j} = \frac{1}{n}, \\
\frac{\partial a}{\partial b} &= -\bar{x}.
\end{aligned}
$$

The error in a is then:

$$
\begin{aligned}
\sigma_a^2 &= \sum_{i=1}^{n} \left(\frac{\partial a}{\partial y_i} \right)^2 \sigma_y^2 + \left(\frac{\partial a}{\partial b} \right)^2 \sigma_b^2 \\
&= \frac{\sigma_y^2}{n^2} \left(\sum_{i=1}^{n} 1 \right) + \bar{x}^2 \times \frac{\sigma_y^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \\
&= \frac{\sigma_y^2}{n \sum_{i=1}^{n} (x_i - \bar{x})^2} \left[\sum_{i=1}^{n} (x_i - \bar{x})^2 + n\bar{x}^2 \right] \\
&= \frac{\sigma_y^2}{n \sum_{i=1}^{n} (x_i - \bar{x})^2} \left[\sum_{i=1}^{n} x_i^2 - 2\bar{x} \sum_{i=1}^{n} x_i + \bar{x}^2 \sum_{i=1}^{n} 1 + n\bar{x}^2 \right] \\
&= \frac{\sigma_y^2}{n \sum_{i=1}^{n} (x_i - \bar{x})^2} \left[\sum_{i=1}^{n} x_i^2 - 2n\bar{x}^2 + n\bar{x}^2 + n\bar{x}^2 \right] \\
&= \frac{\sigma_y^2}{n} \times \frac{\sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2},
\end{aligned}
$$

which is the expression we were asked to derive.

Python code

Exercise – Erdinger

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
plt.style.use('seaborn-talk')


def get_rsquared(yval, yre):
    """Obtain R^2 for fitted data.

    Parameters
    ----------
    yval : numpy.array
        The y-values used in the fitting.
    yre : numpy.array
        The estimated y-values from the fitting.

    Returns
    -------
    rsq : float
        The estimated value of R^2.

    Notes
    -----
    https://en.wikipedia.org/wiki/Coefficient_of_determination

    """
    ss_tot = np.sum((yval - yval.mean())**2)
    ss_res = np.sum((yval - yre)**2)
    rsq = 1.0 - (ss_res / ss_tot)
    return rsq


def fit_linear(xdata, ydata):
    """Fit a linear function.

    Parameters
    ----------
    xdata : numpy.array
        The x-values for the raw data.
    ydata : numpy.array
        The y-values for the raw data.

    Returns
    -------
    yre : numpy.array
        The y-values estimated by the fitted function.
```

```
    pfit : numpy.array
        Estimated coefficients from the fit.
    rsq : float
        The estimated value of R^2 for the fit.

    """
    print('Fitting linear function')
    pfit = np.polyfit(xdata, ydata, 1)
    yre = np.polyval(pfit, xdata)
    print('\t- Coefficients: {}'.format(pfit))
    rsq = get_rsquared(ydata, yre)
    print('\t- R^2: {}'.format(rsq))
    return yre, pfit, rsq


def plot_xy(xdata, ydata, lines=None, output=None):
    """Plot the given data and lines."""
    fig = plt.figure()
    ax1 = fig.add_subplot(111)
    ax1.scatter(xdata, ydata, color='black', alpha=0.8, s=200)
    ax1.set_xlabel('Time (s)')
    ax1.set_ylabel('Height (cm)')
    if lines is not None:
        for line in lines:
            ax1.plot(line['x'], line['y'],
                     label=line['label'], alpha=0.8, lw=5)
        ax1.legend()
    fig.tight_layout()
    if output is not None:
        fig.savefig(output)
    return fig, ax1


def estimate_only_b(xdata, ydata):
    """Least-squares estimate of b when intercept is zero."""
    yprime = ydata - ydata[0]
    return sum(yprime * xdata) / sum(xdata * xdata)


def main():
    """Read in the data and run the fitting."""
    data = pd.read_csv('Data/erdinger.txt', delim_whitespace=True)
    xdata = data['time']
    ydata = data['height']

    # Plot the raw data:
    plot_xy(
        xdata,
        ydata,
        output='raw_data.pdf',
```

```
)

# Fit a linear model to the raw data:
yre1, _, rsq1 = fit_linear(xdata, ydata)
lines = [
    {
        'x': xdata,
        'y': yre1,
        'label': r'Linear fit (R$^2$ = {:4.2f})'.format(rsq1)
    },
]
# Plot the linear model:
plot_xy(
    xdata,
    ydata,
    lines=lines,
    output='linear1.pdf'
)

# Transform using log and fit a linear equation for the transformed
# variable:
ytrans = np.log(ydata)
yre2, pfit2, rsq2 = fit_linear(xdata, ytrans)
print('Estimated h(0):', np.exp(pfit2[1]))
print('Estimated tau:', -1.0 / pfit2[0])
lines.append(
    {
        'x': xdata,
        'y': np.exp(yre2),
        'label': r'Linear fit, transformed (R$^2$ = {:4.2f})'.format(rsq2)
    }
)
# Plot the two models found so far.
plot_xy(
    xdata,
    ydata,
    lines=lines,
    output='linear2.pdf'
)

# Do last the last estimate, where we are forcing h(0) to be a
# specified value:
only_b = estimate_only_b(xdata, ytrans)
yre3 = ytrans[0] + only_b * xdata
rsq3 = get_rsquared(ytrans, yre3)
lines.append(
    {
        'x': xdata,
        'y': np.exp(yre3),
        'label': r'Linear fit, h(0) given (R$^2$ = {:4.2f})'.format(rsq3)
```

```python
        }
    )
    print('Second fit, h(0):', np.exp(ytrans[0]))
    print('Second estimate of tau:', -1.0 / only_b)
    plot_xy(
        xdata,
        ydata,
        lines=lines,
        output='linear3.pdf'
    )

    plt.show()


if __name__ == '__main__':
    main()
```

Listing 1: *Python code for performing the fitting of the Erdinger data.*