A Data Science and Visualization Primer

First Steps – Plotting with Matplotlib and Numpy

assoc. Prof. Dr. Thomas Hofer
Department of Theoretical Chemistry
Email: T.Hofer@uibk.ac.at

Stefanie Kröll, MSc
Department of Theoretical Chemistry
Email: Stefanie.Kroell@uibk.ac.at

Welcome to the First Exercise Block

In this example, we will go over the basics how to:

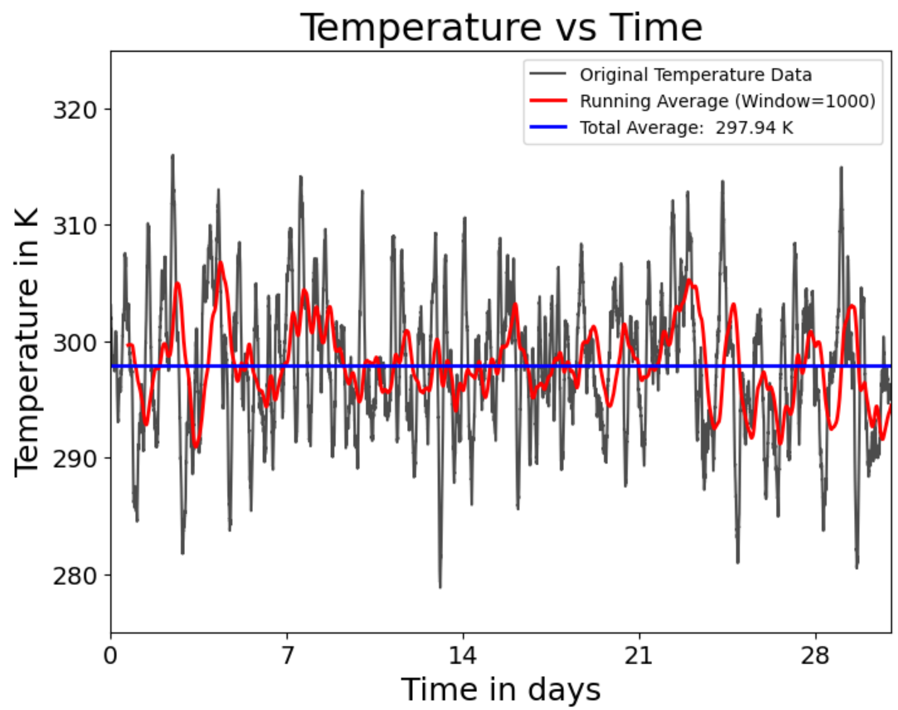
- Easily import files to Google Colab.
- Extract the data using Python.
- Quickly make scientific plots (that look like we want).

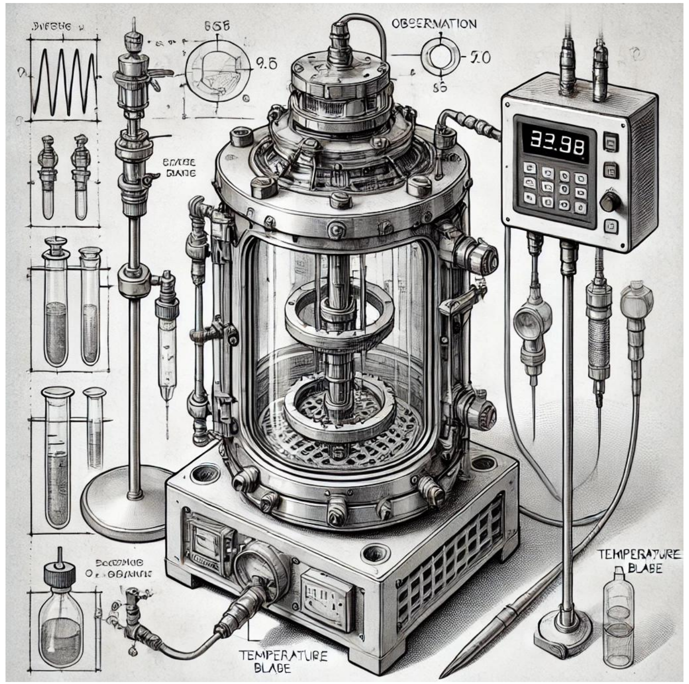
Also, we have a look of some simple data treatment like:

- Calculating and depicting running and total averages.
- Making a histogram from the data.
- Perform some analysis via sigmoid and Gaussian fits.

The idea behind this data set is that we are running a series of experiments that should maintain a constant temperature over several weeks.

The sensors give us one temperature measurement every minute. Let's see if our experiments were stable and start by uploading the temperature data files ...🤓





Uploading data files to Google Colaboratory (*zero difficulty*)

"*Colab is a hosted Jupyter Notebook service that requires no setup to use and provide free access to computing resources. Colab is especially well suited to machine learning, data science and education.*" (*https://colab.google/*)

There are two way to easily get data files into colab:

1) Import files directly from the computer:

Using the `google.colab` packages a text box will open to navigate the computer and select a file on the hard disk of the current computer.

```
from google.colab import files

uploaded = files.upload()
```

2) Access files via google drive:

Another option is to allow colab to access files located on google drive. (While this is convenient, knowledge how to navigate paths is required.)

```
from google.colab import drive

drive.mount('/content/drive/')
```

Either way data files can be easily uploaded to colab to access the data. Let's upload the files `Temperature_1.dat`, `Temperature_2.dat` and `Temperature_3.dat`.

My First Plot (*zero difficulty*)

After uploading the files, we have to `import` key python modules:

- NumPy, a library for numerical analyses
- Matplotlib, a library to generate the plots

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

data1 = np.loadtxt('Temperature_1.dat')

plt.plot(data1[:, 0], data1[:, 1])
```

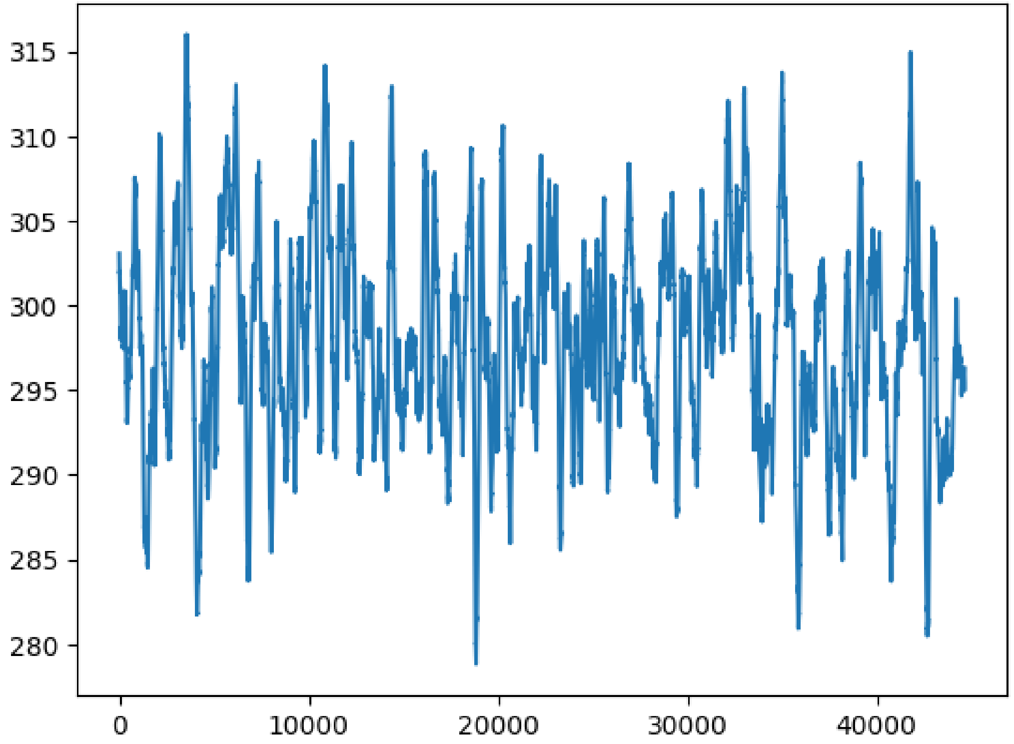
The lables `np` and `plt` are shorthand notations.

Using `np.loadtext()` we can easily load the contents of file `Temperature_1.dat` into the array `data1`.

An array is a container, which stores all entries on a grid. (Think of it as an advanced, interactive Excel sheet.)

The array data has two columns that can be accessed using `data[:, 0]` and `data[:, 1]`.

With this we can make our first xy-plot using the command `plt.plot()`.

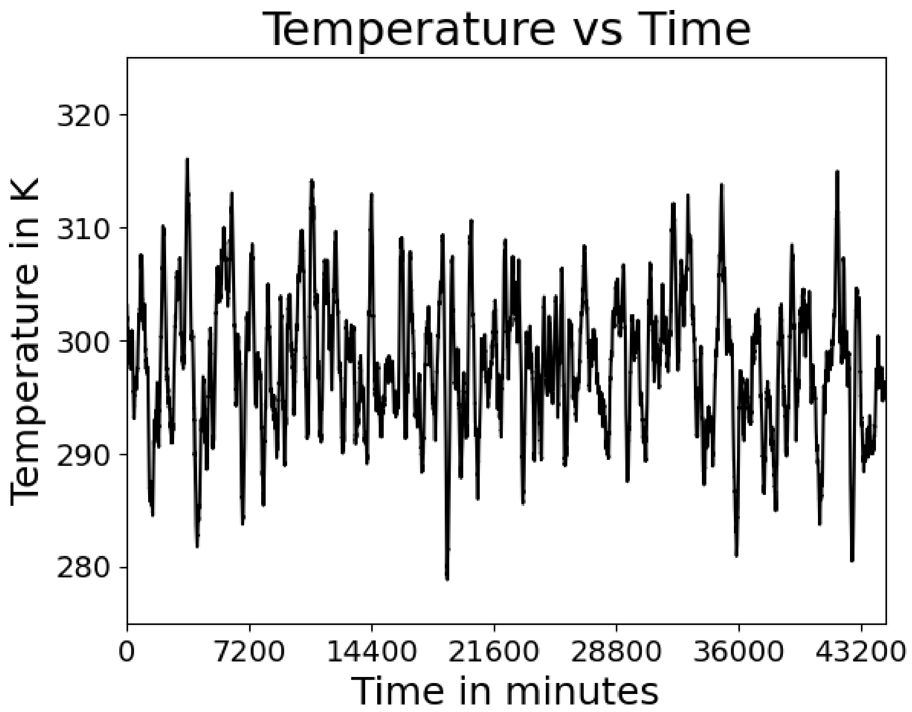


Customizing the Plots – Ticks, Labels and Fontsize (*very low difficulty*)

- Extract the data columns into separate arrays `time` and `temperature` (optional).
- Set up the plot by using the arrays *time* and `temperature` – `color = 'k'` is black
- Adjust x- and y-range using `plt.xlim(min,max)` and `plt.ylim(min, max)`
- Use `plt.xticks()` to adjust the x-range to 7200:

  `5 days · 24 hours · 60 min = 7200 min`
- Add labels with `plt.xlabel()`, `plt.ylabel()` and `plt.title()` (`fontsize` adjusts the size)
- Use `plt.tick_params()` to adjust the size of tick labels using `labelsize`
- After adjusting all settings, bring it home using `plt.show()`

```
time = data1[:, 0]
temperature = data1[:, 1]

plt.plot(time, temperature, color='k')

plt.xlim(0, 44640)
plt.ylim(275, 325)

plt.xticks(np.arange(0, 44640, step=7200))

plt.title('Temperature vs Time', fontsize=22)
plt.xlabel('Time in minutes', fontsize=18)
plt.ylabel('Temperature in K', fontsize=18)

plt.tick_params(axis='both', labelsize=14)

plt.show()
```



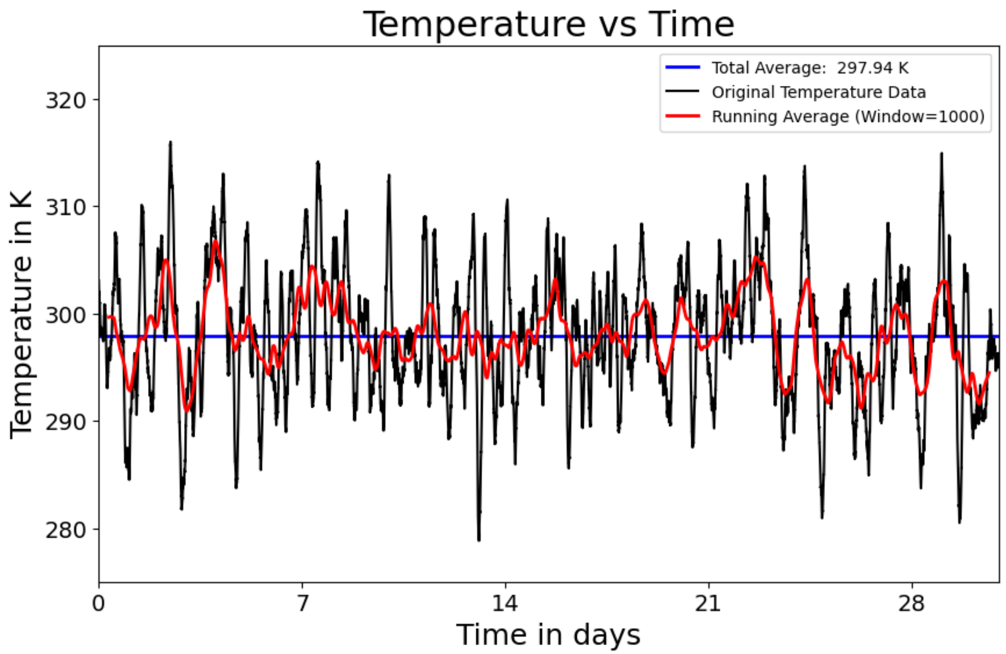
Further Customization – Adding Averages (*low difficulty*)

- Change x-axis from minutes to days using a new array:

```
days = time / 60 / 24
```

- Control the size of the plot using `plt.figure(figsize = (10, 6))`
- Add the average temperature using `np.mean()` and `plt.axhline()`
- Add a running average using `np.convolve()`
- Use different colors for the plots
- Adjust the axis range, labels and fontsize as before using: `plt.xlim`, `plt.ylim`, `plt.xlabel`, `plt.ylabel`, `plt.xticks` and `labelsplt.tick_params`
- Add a legend by adding `plt.legend()`
- When done, again: `plt.show()`

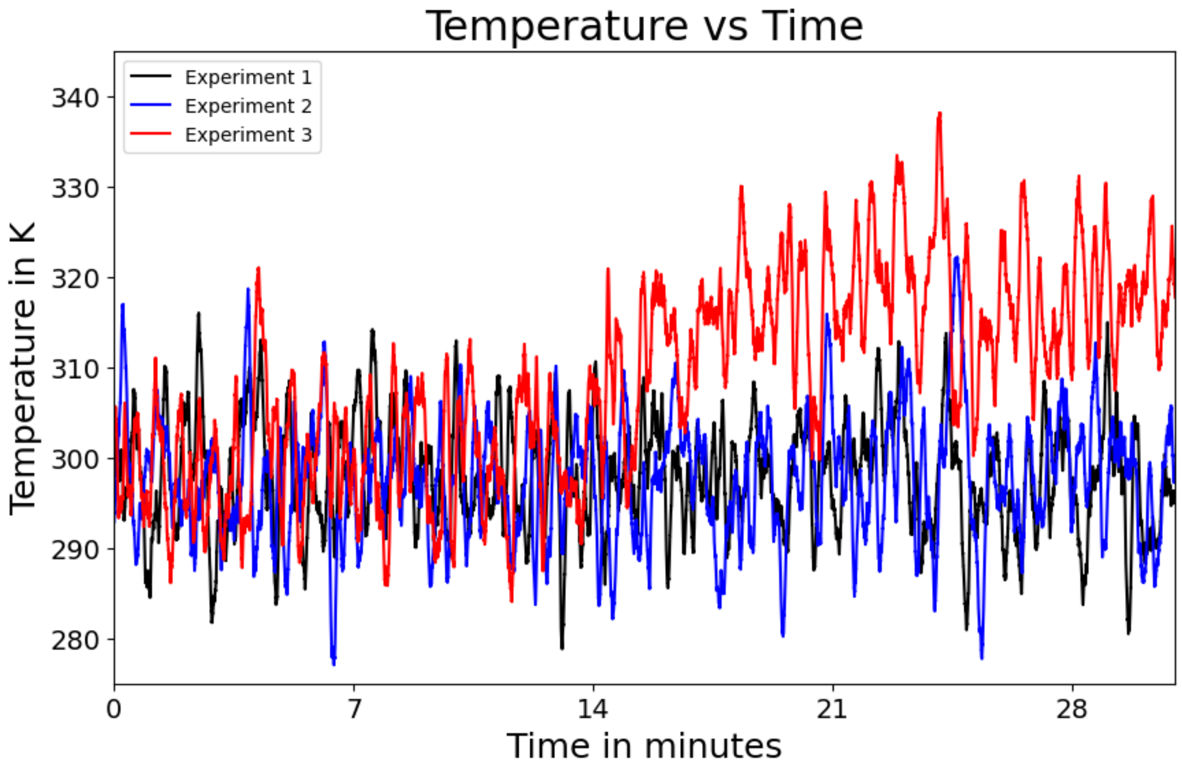
```
days1 = time1 / 60 /24

plt.figure(figsize=(10, 6))

plt.plot(days1, temperature1, color='k',
         label='Original Temperature Data')

temp1_avg = np.mean(temperature1)

plt.axhline(y=temp_avg, color='b',linewidth=2,
            label=f'Total Average: {temp_avg: 4.2f} K'

window = 1000
temp1_runavg = np.convolve(temperature1, np.ones(window)/window,
                           mode='valid')

plt.plot(days[(window//2-1):(-window//2)], temp1_runavg, color='r',
         linewidth=2, label=f'Running Average (Window={window})')
```



Data from Different Files (*low difficulty*)

- To import data from the other two files just re-use the commands from above.
- In this example, it is sufficient to only extract the y-data
- We can even re-use the array days calculated above for all three data sets.
- We plot all three temperature arrays and assign different colors.
- Adjust the axis range, labels and fontsize as before.
- Place the legend into the upper left corner using `plt.legend(loc='upper left')`.
- When done, again: `plt.show()`
- OH NO! Experiment 3 is messed up!!

```
data2 = np.loadtxt('Temperature_2.dat')
data3 = np.loadtxt('Temperature_3.dat')

temperature2 = data2[:, 1]
temperature3 = data3[:, 1]

plt.plot(days, temperature1, color='k', label = 'Experiment 1')
plt.plot(days, temperature2, color='b', label = 'Experiment 2')
plt.plot(days, temperature3, color='r', label = 'Experiment 3')

[...]

plt.legend(loc='upper left')
```



Data from Different Files in Different Subplots (*low difficulty*)

- Plotting the data in different `subplots` of a single `figure` is very simple.
- Define a figure *fig* with three subplots being `ax1`, `ax2` and `ax3`.
- In this example, the subplot is composed of 3 rows and 1 column.
- For example `plt.subplots(2,2)` is composed of 2 rows and 2 columns.
- To customize provide commands separately for each subplot.
- Use *for*-loops to apply the same settings to different plots.
- Assign axis labels individually.
- <u>Careful:</u> Commands for subplots (`ax`) are different from just a normal plot.😔

```
fig, (ax1, ax2, ax3) = plt.subplots(3,1,figsize=(8, 10))

ax1.plot(days1, temperature1, color='k', label='Experiment 1')
ax2.plot(days1, temperature2, color='b', label='Experiment 2')
ax3.plot(days1, temperature3, color='r', label='Experiment 3')

ax1.set_ylim(275, 335)
ax2.set_ylim(275, 335)
ax3.set_ylim(275, 345)

for i in [ax1, ax2, ax3]:
  i.set_xlim(0, 31)
  i.set_xticks(np.arange(0, 31, step=7))

ax1.set_title('Temperature vs Time', fontsize=22)
ax2.set_ylabel('Temperature in K', fontsize=20)
ax3.set_xlabel('Time in days', fontsize=20)

ax1.set_xticklabels([])
ax2.set_xticklabels([])
```

Sigmoid curve fitting (*intermediate difficulty*)

Let's analyze how bad the situation is in experiment 3 is. To do this, we will fit a sigmoid curve to the data set using `curve_fit()` from the SciPy module `scipy.optimize`.

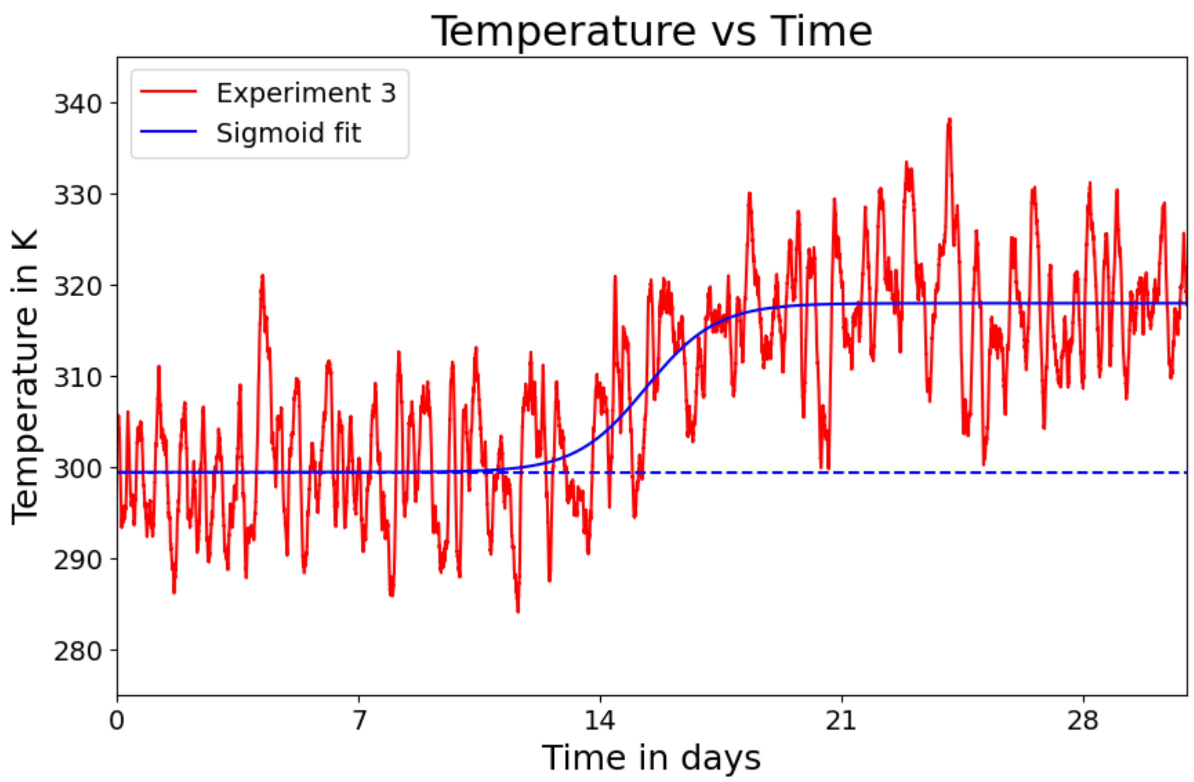
In our case the sigmoid function is given by:

$$T(t) = T_0 + \Delta T \frac{1}{1 + e^{-a(t-t_0)}}$$

The fit four fit parameters are:

- the baseline temperature T_0,
- the temperature increase ΔT,
- the growth rate a,
- the time at the midpoint of the temperature increase t_0

After fitting the curve, we can plot and overlay it with the original data for comparison.

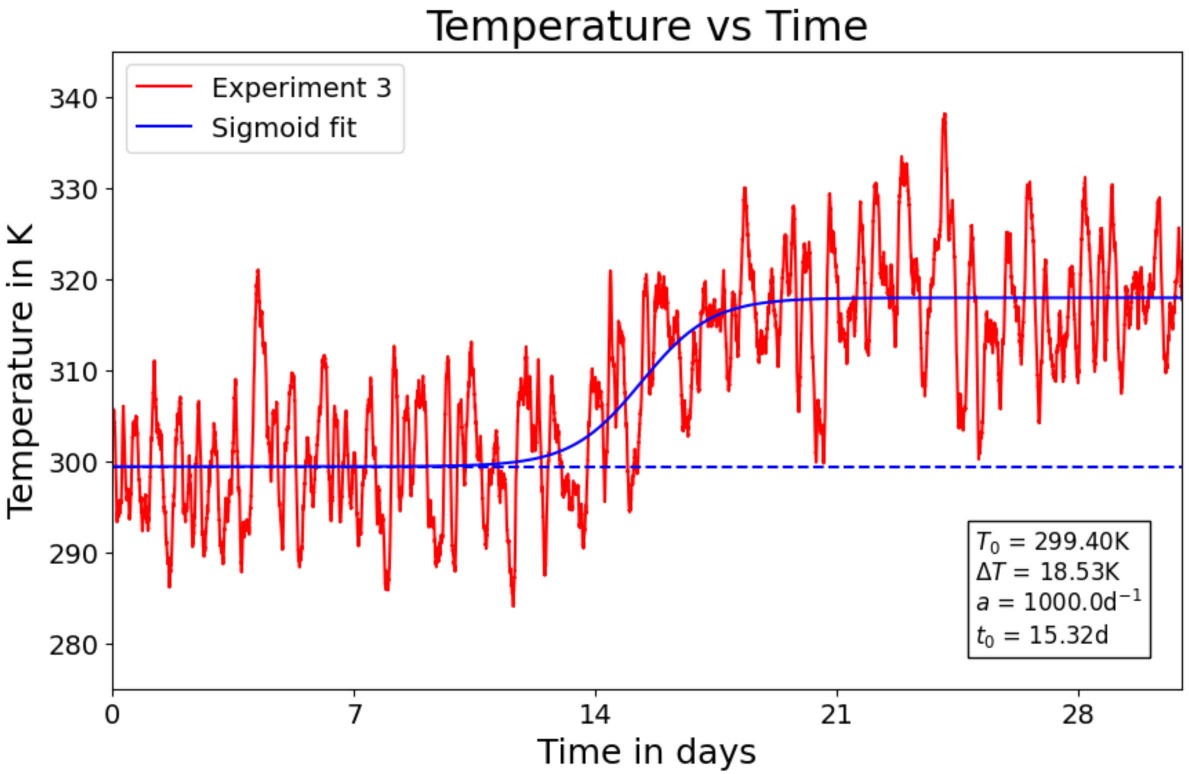


Sigmoid curve fitting (*intermediate difficulty*) – How To:

Only few commands are required:

- Make sure to import `numpy` and `curve_fit()` from `scipy.optimize`
- Next, we need to define the sigmoid:
  - It requires five input variables.
  - It gives one output variable.
- Next, execute `curve_fit()`. it requires:
  - The function to be fitted: `sigmoid`
  - The x- and y-data: `days1, temperature3`
  - Approximate starting values in `p0`
- Extract the optimized parameters from the fit results `T0_fit, dT_fit, a_fit, t0_fit`
- Using the optimized parameters we can then calculate sigmoid for the whole x-axis and store it in `T_sigmoid`

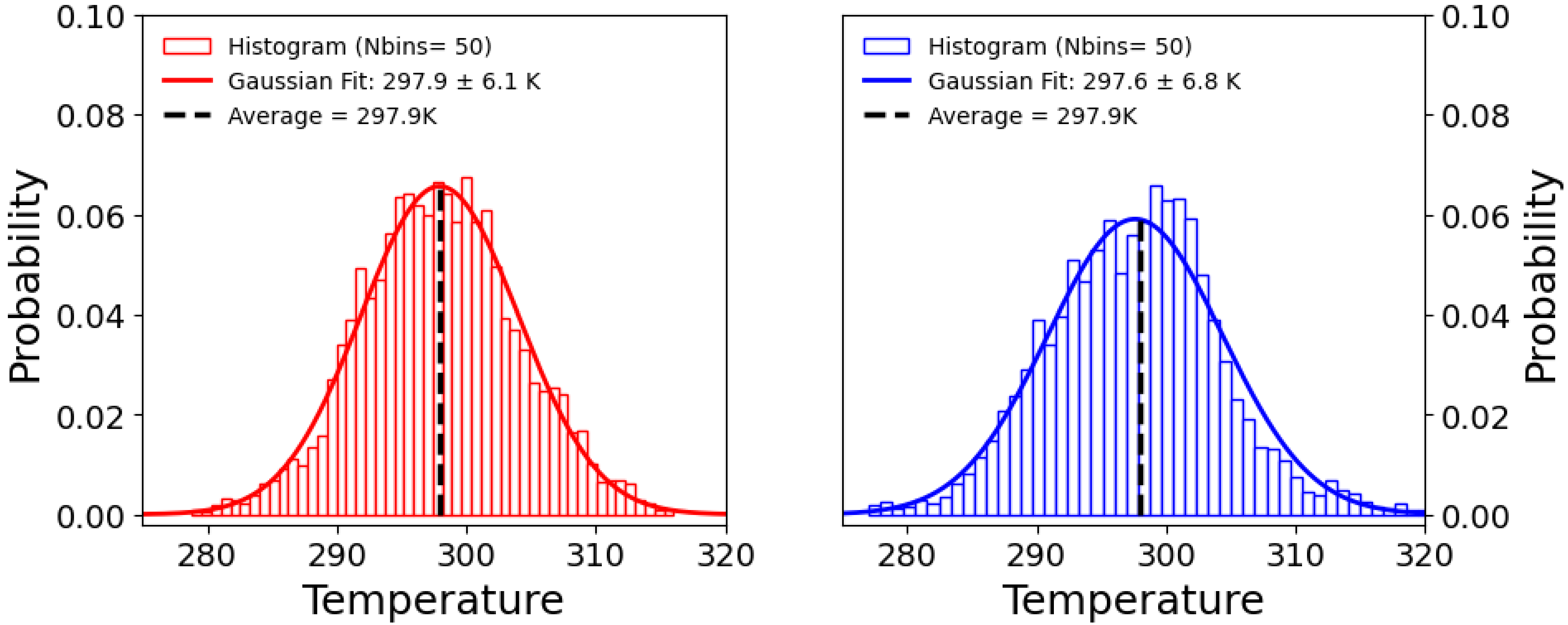
```
import numpy as np
parameters from scipy.optimize import curve_fit

def sigmoid(t, T0, dT, a, t0):
  return T0 + dT / (1 + np.exp(-a * (t - t0)))

fit_result, info = curve_fit(sigmoid, days, temperature3,
                             p0=[298, 20, 0.1, 10])

T0_fit, dT_fit, a_fit, t0_fit = fit_result

T_sigmoid = sigmoid(days1, T0_fit, dT_fit, a_fit, t0_fit)
```



Gaussian Fitting for Two Plots Simultaneously (*intermediate difficulty*)

For experiment 1 and 2 we can analyze the distribution of the temperature to get an idea of the temperature fluctuations in the reactor. For this we have to:

- Generate histograms of the data using `ax1.hist()`.
- Fit and plot Gaussian distributions using `norm.fit()` form the SciPy module `scipy.stats`
- Draw the average as vertical line to double check.
- Analyze both data sets in two separate subplots



Gaussian Fitting for Two Plots Simultaneously (*intermediate difficulty*)

Here, we only look at one data set (ax1):

- It is convenient to define the number of histogram bins as `nbin` (optional)
- Set up the subplots and directly plot a histogram via `ax1.hist()`. To normalize the histogram to 1 use `density = True`.
- Use `norm.fit()` to obtain the `mean1` and stddev1 of the `temperature1`.
- Plotting the Gaussian is a bit intricate:
  - Define axis limits first using `set_xlim()`.
  - Extract and store the limits `get_xlim()`.
  - Generate x-points at regular intervals using `np.linspace(xmin, xmax, 1000)`.
  - Generate the Gaussian curve using the linespace `x1` as input together with the parameters `mean1` and `stddev1`

```
nbin = 50

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))

ax1.hist(temperature1, bins=nbin, density=True,
         color='white',edgecolor='red',
         label=f'Histogram (Nbins= {nbin})')

mean1, stddev1 = norm.fit(temperature1)

ax1.set_xlim(275, 320)
ax1.set_ylim(-0.002, 0.10)

xmin, xmax = ax1.get_xlim()

x1 = np.linspace(xmin, xmax, 1000)

p1 = norm.pdf(x1, mean1, stddev1)

ax1.plot(x1, p1, 'r-', linewidth=2,
         label=f'Gaussian Fit:{mean1:.1f} ± {stddev1:.1f} K')

temp1_avg = np.mean(temperature1)

ax1.plot([temp1_avg, temp1_avg],
         [0, norm.pdf(temp1_avg, mean1, stddev1)],
         color='k', linestyle = '--', linewidth=2.5,
         label=f'Average = {temp1_avg:.1f}K')

[...]

ax2.yaxis.tick_right()
ax2.yaxis.set_label_position('right')

ax2.set_ylabel('Probability', fontsize=18)
```

Gaussian Fitting for Two Plots Simultaneously (*intermediate difficulty*)

Here, we only look at one data set (ax1):

- To double check the Gaussian fit let's calculate the average using `np.mean()`
- Instead of drawing a vertical line over the entire range of the plot using `plt1.axvline()` we use `plot1.plot()`
  - That way we can define the range as:
    x: `[temp1_avg, temp1_avg]`
    y: `[0, norm.pdf(temp1_avg, mean1, stddev1)]`
  - Now the vertical line is exactly aligned with the peak of the Gaussian
- Rinse and repeat for `plt2`
- Using `ax2.yaxis.tick_right()` and `ax2.yaxis.set_label_position('right')` the ticks and labels of `plt2` will appear on the right side of the figure.

```
nbin = 50

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))

ax1.hist(temperature1, bins=nbin, density=True,
         color='white',edgecolor='red',
         label=f'Histogram (Nbins= {nbin})')

mean1, stddev1 = norm.fit(temperature1)

ax1.set_xlim(275, 320)
ax1.set_ylim(-0.002, 0.10)

xmin, xmax = ax1.get_xlim()

x1 = np.linspace(xmin, xmax, 1000)

p1 = norm.pdf(x1, mean1, stddev1)

ax1.plot(x1, p1, 'r-', linewidth=2,
         label=f'Gaussian Fit:{mean1:.1f} ± {stddev1:.1f} K')

temp1_avg = np.mean(temperature1)

ax1.plot([temp1_avg, temp1_avg],
         [0, norm.pdf(temp1_avg, mean1, stddev1)],
         color='k', linestyle = '--', linewidth=2.5,
         label=f'Average = {temp1_avg:.1f}K')

[...]

ax2.yaxis.tick_right()
ax2.yaxis.set_label_position('right')

ax2.set_ylabel('Probability', fontsize=18)
```