TØ 9. Laboratory Exercise 2

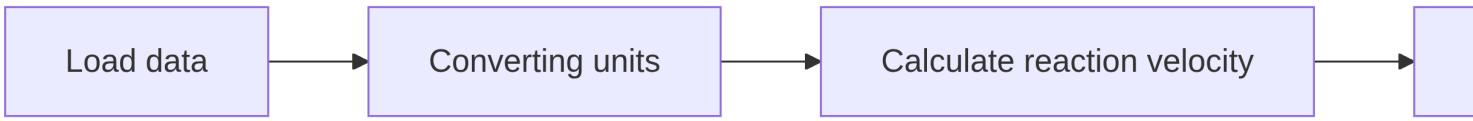
Table of contents

In this exercise you will analyze the data you've recorded in the laboratory. You will not be writing any Python code yourself - but you will walk through each of the step of analyzing this dataset.

In the "Fysiskbiokemi & Datanalayse"-course you will learn more about the theory behind this analysis and about the details of performing the analysis with Python.

1 Enzyme Kinetics

The diagram below describes the workflow we will be performing



1.1 Importing the data

The button below allows you to import your LØ dataset by uploading the `.xlsx`-file.

```
//| echo: false
viewof xlsx_file = Inputs.file({
  label: "Upload Excel file",
  accept: ".xlsx",
  required: false
})

xlsx_name = xlsx_file ? xlsx_file.name : null
xlsx_bytes = xlsx_file
  ? Array.from(new Uint8Array(await xlsx_file.arrayBuffer()))
  : null
```

! Question

What is shown in the different columns of the dataset?

1.2 Converting units

The measured absorbances are unitless, they are the log of the ratio between the incoming light intensity and the transmitted light intensity. We would like to work with concentrations instead, so we need to convert the absorbance to concentration. The conversion is done using Lambert-Beer's law,

$$A = \epsilon \cdot l \cdot c \Rightarrow c = \frac{\epsilon \cdot l}{A}$$

Where A is the measured absorbance, l is the path length, ϵ is the extinction coefficient and c is the concentration.

The cell below converts the absorbances to concentrations and creates a new dataframe.

Interactive Cell

```
df_conc = convert_absorbance(df)
display(df_conc)
```

Code

```
def lambert_beers(absorbance):
    L = 1 # cm
    ext_coeff = 18000 # 1 / (M cm)
    conc = absorbance / (L * ext_coeff) # M
    return conc


def convert_absorbance(df):
    data = {
        'time': df['time'],
    }

    for key in df.keys():
        if key == "time":
            continue

        new_key = f"S_{key}_mM"
        conc = lambert_beers(df[key])
        data[new_key] = conc

    df_conc = pd.DataFrame(data)

    return df_conc
```

The next cell plots the concentration as a function of time for each column, corresponding to a plot for each substrate concentration.

Interactive Cell

```
plot_dataframe(df_conc)
```

Code

```
def plot_dataframe(df):
    fig, ax = plt.subplots()

    for key in df.keys():
        if key == "time":
            continue

        ax.plot(df['time'], df[key], '-o')

    ax.set_ylabel('Concentration [M]')
    ax.set_xlabel('Time [s]')

    ax.patch.set_alpha(0.0)
    fig.patch.set_alpha(0.0)
    plt.show()
```

! Question

What happens to the slope of the curves as substrate concentration increases?

1.3 Calculate reaction velocity

Hopefully what you see in the plot of concentrations vs. time are approximately linear relationships, as the next step is to find the slope of each of these - as that describes the initial reaction velocity. Mathematically we fit a linear function to each dataset

$$C = V_0 \cdot t + C_0$$

Where V_0 is the intial reaction velocity that we are looking for.

Interactive Cell

```
# Uses fitting to the find the slope (reaction velocity)
# for each substrate concentration
substrate_conc, slopes = find_slopes(df_conc)

# Plots the substrate concentration vs. slope
fig, ax = plt.subplots()
ax.plot(substrate_conc, slopes, 'o')
ax.set_xlabel('Substrate concentration [M]')
ax.set_ylabel('Slope [M/s]')
plt.show()
```

Code

```
def linear_function(x, a, b):
    result = a * x + b
    return result

def find_slope(time_data, abs_data):
    ## This selects the times and absorbances that have been measured (not NaN)
    indices = np.where(~abs_data.isna())[0]
    time_data = time_data[indices]
    abs_data = abs_data[indices]

    ## Find the slope and the intercept using curve_fit and return the slope
    fitted_parameters, trash = curve_fit(linear_function, time_data, abs_data)
    slope = fitted_parameters[0]
    return slope

def find_slopes(df_conc):
    slopes = []
    substrate_concentrations = []

    for key in df_conc.keys():
        if key == "time":
            continue

        slope = find_slope(df_conc['time'], df_conc[key])
        conc = float(key.split('_')[1]) * 10**(-3)
        slopes.append(slope)
        substrate_concentrations.append(conc)

    substrate_concentrations = np.array(substrate_concentrations)
    slopes = np.array(slopes)
    return substrate_concentrations, slopes
```

1.4 Michaelis-Menten

As you will learn much more about in the "Fysiskbiokemi & datananalyse"-course this type of data can be described by a Michealis-Menten model

$$V_0 = \frac{V_{max} \cdot S}{K_M + S}$$

where V_0 is the initial reaction velocity, V_{max} is maximum reaction velocity occuring at high enough substrate concentrations and S is the substrate concentration.

Interactive Cell

```
K_M, V_max = fit_michealis_menten(substrate_conc, slopes)

plot_michealis_menten(K_M, V_max, substrate_conc, slopes)
```

Code

```
def michaelis_menten(S, K_M, V_max):
    result = (V_max * S) / (K_M + S)
    return result

def fit_michealis_menten(substrate_conc, slopes):
    K_M_estimate = 0.00001
    V_max_estimate = 8 * 10**(-8)
    initial_guess = [K_M_estimate, V_max_estimate]
        fitted_parameters,  trash  =  curve_fit(michaelis_menten,  substrate_conc,  slopes,
initial_guess)
    K_M_fit, V_max_fit = fitted_parameters

    return K_M_fit, V_max_fit

def plot_michealis_menten(km, vmax, substrate, slopes):
    fig, ax = plt.subplots()

    S_smooth = np.linspace(substrate.min(), substrate.max())
    V0_fit = michaelis_menten(S_smooth, km, vmax)

    ax.plot(substrate, slopes, 'o', label='Data')
    ax.plot(S_smooth, V0_fit, label='Fit')

    ## These add x and y axis labels.
    ax.set_xlabel('Substrate concentration (M)')
    ax.set_ylabel('$V_0$ (M/s)')

    ax.text(0.7, 0.5, f'$K_M=${km:0.4} M', transform=ax.transAxes, ha='right', va='center',
fontsize=14)
     ax.text(0.7, 0.4, f'$V_{{max}}=${vmax:0.4} M/s', transform=ax.transAxes, ha='right',
va='center', fontsize=14)

    plt.show()
```

! Question

What is an explanantion of this behaviour based on enzymekinetics?