Notebook for Michaelis-Menten Kinetics

Written by N. Henkenhaf together with G. Gygli.

Contact gudrun.gygli@kit.edu in case of questions.

Usage:

To run this notebook, it is neccesary to create a specific folder structure. The notebook must be in a folder together with two other folders:

```
'data' and 'EnzymeKinetics'
```

'data' contains the csv files you want to work with and 'EnzymeKinetics' must contain all scripts needed to run this notebook (for a complete list seee below).

The notebook assumes that the experimental data was measured in a 96 well microtiter plate.

The folder structure must be as followed:

```
- yourDirectory:
    - data:
        - anyFilename.csv
    - EnzymeKinetics:
        -  __init__.py
        - Concentration.py
        - config.py
        - DataHandler.py
        - FitHandler.py
        - helper.py
        - InhibitionModels.py
        - InhibitionParentClass.py
        - Microplate.py
        - parameters.py
        - PlotHandler.py
        - Substrate.py
        - Variable.py
        - Well.py
    - MichaelisMentenNotebook.ipynb
```

Also, check 'requirements.txt' for a list of the version of the packages used and install any missing packages on your computer.

The notebook will do as follows:

```
1. Import data
    - Import your data as exported from the measurement. Several parameters will
        be asked to get the best result.
2. Fit initial rates and plot data
    - In this step, the rates for each substrate concentration will be calculated.
        The function will return data for further analysis.
3. Optional: Remove individual measurements and/or change the range for each
    individual experiment
```

4. Michaelis-Menten-Fit
5. Optional: Fit other models to your data.

Michaelis-Menten Kinetics

This notebook is a template for calculating the parameters K_{M} (the Michaelis(-Menten) constant) and k_{cat} (turnover number) of the Michaelis-Menten equation

$$v = \frac{k_{\mathrm{cat}}[S]}{K_{\mathrm{m}} + [S]},$$

where v is the rate at which product is formed and substrate is consumed and $[S]$ the concentration of the substrate.

To use this model, certain conditions must be met:

[S_0] >> K_m

[S_0] >> [E_0]

[S_0] >> [P]

where [S_0] and [E_0] are the substrate and enzyme concentration (respectively) at the beginning of the experiment, and [P] is the concentration of product formed in the experiment.

Note that this model assumes that only a single substrate and a single product are involved in the **irreversible reaction (uni-uni)**.

The data used in the example is for a reaction involving two substrates, of which only one (NADPH) is spectrophotometrically active. [NADPH] at the beginning of the reaction is always the same. The concentration of the other substrate (NDK) is varied and we follow the reaction by observing the change in [NADPH] as catalyzed by the enzyme (Gre2p). These data are therefore conditional on [NADPH].

0. Import everything the notebook needs from the "EnzymeKinetics" folder

In [ ]:
```
import EnzymeKinetics as ek
params = ek.parameters
```

1. Import data

For the notebook to work, you need to tell it the name of the csv file you want to work with and give some additional input.

1.1 Give the filename and other input:

Filename:

The name of the data file. The file must be in the 'data' directory and it has to be a .csv file. Replace 'FILENAME' with the name of the data file. Do not forget to write the name in quotations marks.

Make sure your data is as in the example and/or tidy (https://towardsdatascience.com/what-is-tidy-data-d58bb9ad2458).

In [ ]:
```
params.FILENAME = 'MM_25nM_Gre2p_NDK_NADPH_absorbance_340nm_cleaned.csv'
```

Replica names:

Enter the name of the replicas below in the list by changing the text in the brackets and make sure to write them between quotation marks.

The order of the replica names in the list must be the same order as on microtiter plate!

Extend or shorten the list seperated by *commas*.

In [ ]:
```
params.REPLICA_NAMES = ['Replica_1', 'Replica_2', 'Replica_3', 'Replica_4']
```

Spectrophotometrically active molecule:

The name of the spectrophotometrically active molecule you are observing in the experiment. In the example, this is NADPH.

In [ ]:
```
params.ABS_NAME = 'NADPH'
```

Molar extinction-coefficient:

Give the molar extinction-coefficient of your spectrophotmetrically active molecule in the same units as your substrate concentration (e.g. $mM^{-1}cm^{-1}$)

In [ ]:
```
params.MOLAR_EXTINCTION_COEFFICIENT = 6.22
```

Substrate concentrations:

Enter the substrate concentrations used as shown below. Therefore change the numbers in the brackets and make sure to set decimals with dot (0.1) not comma (0,1). In Python, comma is used to separate elements of a list.

If you have a negative control without any substrate and/or enzyme (as you should), insert '0.0' into the list.

The order of the substrate concentrations and the control groups in the list must be the same order as on plate!

Extend or shorten the list separated by *commas*.

In [ ]:
```
params.SUBSTRATE_CONCENTRATIONS  = [0, 0, 0.05, 0.1, 0.25, 0.5, 1, 2.5, 5, 10, 25, 50]
```

Unit:

Enter the unit of substrate concentrations. This unit will be used for labeling axes in the plot.

Make sure to formate units as LaTeX strings, e.g. µM as '\mu M'.

Fill in below.

In [ ]:
```
params.SUBSTRATE_UNIT  = 'mM'
```

Enzyme concentration in the assay:

Give the enzyme concentration you used in the same units as your substrate concentration

Fill in below.

In [ ]:
```
params.ENZYME_CONCENTRATION = 0.000025
```

Control:

If there is a control to be substracted from the data, enter it below. If there is none, enter 'None' without quotation marks.

In [ ]:
```
params.CONTROL = "C2"
```

1.2 Run the code below to import the data

In [ ]:
```
importer = ek.Microplate(params)

importer.import_csv()
importer.clean_dataframe()

data = importer.write_substrates()
```

```
---------------
 Data imported
---------------
```

2. Fit initial rates and plot data

Now that we got our data imported, it is time to calculate the initial rates of the reaction in every well on the plate.

2.1 Give fitting and plotting parameters:

2.1.1. Fitting parameter

Cut:

You must give a time-window (range) to use for the fits of the initial rates. As you can see in the example, there can be noise in the beginning of the measurement, and you can cleary see that the reaction is not linear for the entire duration of the measurement.

Also, depending on [E] or [S] used, the rates will decrease faster or slower and you will not measure initial rates for the entire duration of the experiment.

Enter a range (in seconds) to specify the data which should be used to fit the initial rates.

In [ ]:
```
params.CUT = [428, 748]
```

2.1.1. Plotting parameters

Title:

This will be the title of the plot(s) the notebook creates for you.

In [ ]:
```
params.TITLE = '25nM Gre2p NDK NADPH absorbance 340nm'
```

Show fit:

Set '*False*' if the fit curves should not be displayed in plots.

In [ ]:
```
params.SHOW_FIT = True
```

Labels:

Set '*False*' if the labels of each well should not be displayed in plots.

If you set 'params.Labels = True', you will see names for each well next to it's concentration in the label box.

These names will be needed for further data handling.

The names assume the following pipetting layout of the microtiter plate: the rows (A-H) are replicates and the columns (1-12) contain different [S]. A1, B1, C1, D1 are therefore replicates of one [S].

In [ ]:
```
params.LABELS = True
```

Save PNG:

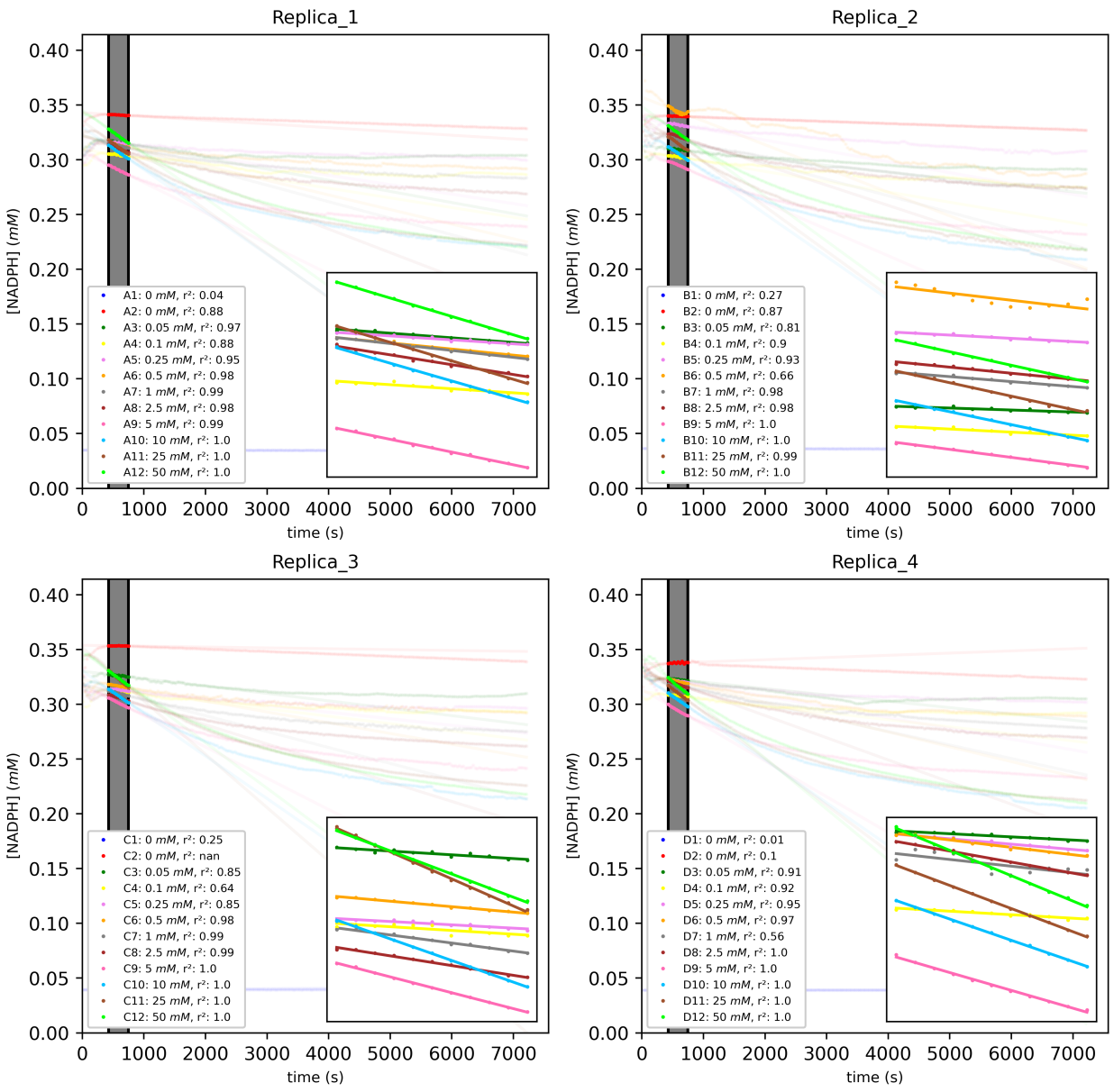
Set '*True*' if the plot should be saved as a png file. The png will be stored in the 'plots' folder.

In [ ]:
```
params.SAVE_FIG = True
```

2.2. Run the code below to fit the initial rates to your data and plot the fit

In [ ]:
```
data_cleaner = ek.DataHandler(params, data)
data_cleaner.set_cut(params.CUT)
data_cleaner.substract_control_rate()

rate_fitter = ek.FitInitialRate(params, data_cleaner)
rate_fitted_data = rate_fitter.do_fit()

rate_plotter = ek.PlotRateFit(params, rate_fitted_data)
rate_plotter.do_plot()
```

```
Control C2 substracted from every well.

Plot saved as png
```



3. Optional: Remove individual measurements and/or change the range for each individual experiment

3.1. Remove individual measurements

If it you need to discard data from individual experiments due to large noise or pipetting errors, you can do so:

You can check the data by running the code below and enter the substrate you want to check.

- Name: name of the well
- Concentration: concentration of well
- Rate: rate of the product
- Rate_std: standard error of the rate
- r-squared: goodness of fit
- Dropped: wheter the well is used for the fit or not

In [ ]:
```
substrate = "Replica_2"
data_cleaner.print_substrate_data(substrate)
```

```
Data for Replica_2:
Control rate substraced: True
Name      Concentration (mM)  Rate (mM/s)                Rate_std (mM/s)             r²  Dropped
------  --------------------  -------------------------  -------------------------  ----  ---------
B1                         0  {-3.1971691073897666e-07}  {1.7425348635664167e-07}  0.27  False
B2                         0  {-2.009645436355983e-06}   {2.5656918784215657e-07}  0.87  False
B3                      0.05  {-6.02893970851027e-06}    {9.81485074324977e-07}    0.81  False
B4                       0.1  {-9.363123784798099e-06}   {1.036502694583645e-06}    0.9  False
B5                      0.25  {-9.91120915803725e-06}    {8.76042654323123e-07}    0.93  False
B6                       0.5  {-2.1649372238805635e-05}  {5.177794253912769e-06}   0.66  False
B7                         1  {-1.5072348325202023e-05}  {7.48950933384581e-07}    0.98  False
B8                       2.5  {-1.8452206611242592e-05}  {9.609599294050628e-07}   0.98  False
B9                         5  {-2.4252777151155147e-05}  {5.444787820179952e-07}      1  False
B10                       10  {-3.909675516408012e-05}   {4.726985130526719e-07}      1  False
B11                       25  {-4.074101097067747e-05}   {1.029884923525109e-06}   0.99  False
B12                       50  {-4.096938044917866e-05}   {4.2273954141111296e-07}     1  False
```

If you want to remove individual data, you can run the code below.

You can run this code mutliple times to remove (deactivate) data in several steps.

Deactivated data will be activated again, if the well is in the list a second time.

Add or delete well names in the list.

In [ ]:
```
wells_to_drop = ["B6", "D4"]
data_cleaner.drop_by_well_names(wells_to_drop)
```

```
B6 deactivated
D4 deactivated
```

3.2 Change the range for each individual experiment

You can change the time-window (range) for the fits of the initial rates for each individual experiment (well) if it is not possible to fix a time-window where all the experimental data are linear.

Add or delete entries of this dictionary. The formatting is:

"name of the well" : [start of range, end of range]

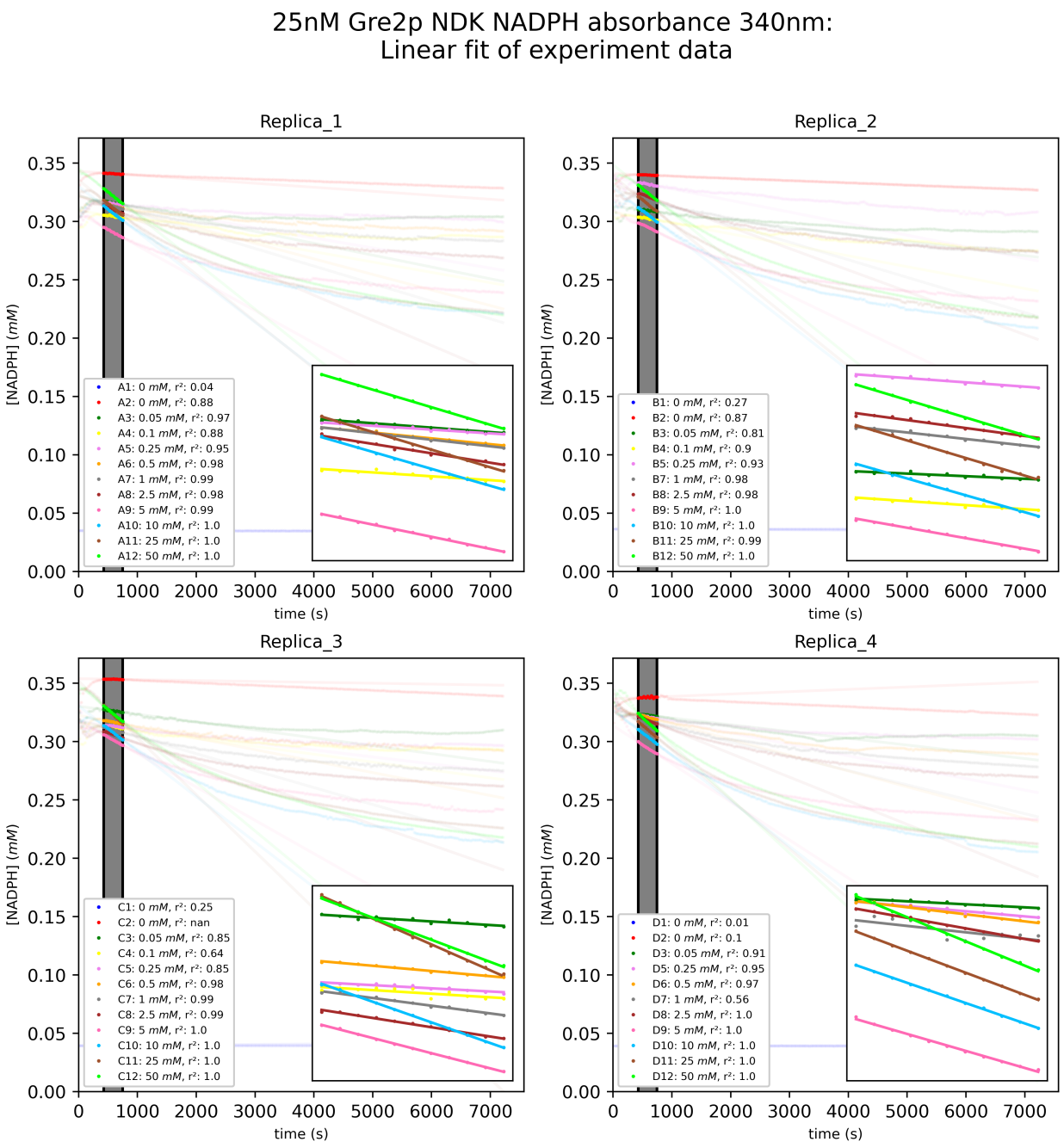
Make sure to separate them by comma.

In [ ]:
```
# wells_to_cut = {
#     "D9": [200, 3000],
#     "A5": [3450, 4000]
#     }

# data_cleaner.cut_by_well_names(wells_to_cut)
```

3.3. If you made your adjustments in 3.1 and/or 3.2, run the code below for fitting again.

In [ ]:
```
rate_fitter = ek.FitInitialRate(params, data_cleaner)
rate_fitted_data = rate_fitter.do_fit()

rate_plotter = ek.PlotRateFit(params, rate_fitted_data)
rate_plotter.do_plot()
```

```
Plot saved as png
```



3.1 Summarize the initial rates from your data

This will create a "summary plot" of the averaged initial rates for your data.

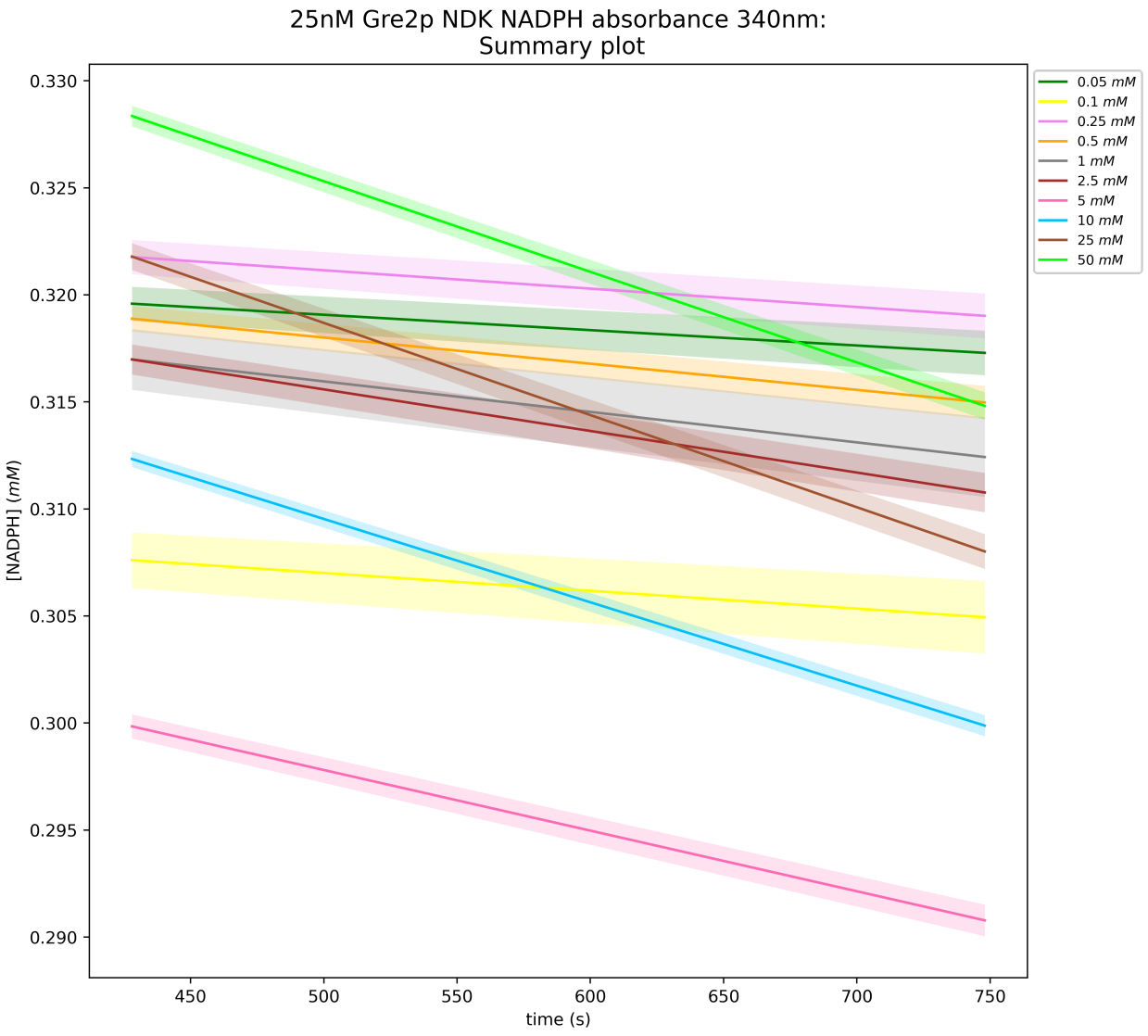
Initial rates are printed below for each concentration with a standard deviation and other statistic values.

In [ ]:

```
mm_preparer = ek.ConcentrationSummerizer(params, rate_fitted_data)
concentrations = mm_preparer.calculate_averages()

average_rate_plotter = ek.PlotAverageRateFit(params, concentrations)
average_rate_plotter.do_plot()
```

Concentration (mM)	Rate (mM/s)	Rate_std (mM/s)	y-intercept	y-intercept_std
0	-5.30957e-07	4.92274e-07	0.1905	0.000293712
0.05	-7.1822e-06	7.73959e-07	0.322661	0.000461778
0.1	-8.32785e-06	1.26528e-06	0.311172	0.000754922
0.25	-8.58667e-06	7.78945e-07	0.325441	0.000464753
0.5	-1.22253e-05	5.76522e-07	0.324118	0.000343978
1	-1.42502e-05	1.38227e-06	0.323082	0.00082472
2.5	-1.94228e-05	6.83052e-07	0.325294	0.000407539
5	-2.83177e-05	5.47791e-07	0.311963	0.000326836
10	-3.89255e-05	3.66465e-07	0.328994	0.000218649
25	-4.30475e-05	6.05019e-07	0.340213	0.000360981

```
                50    -4.23168e-05        4.70132e-07        0.346466        0.000280502
Plot saved as png
```



4. Michaelis-Menten-Fit

Now that you have made sure that your initial rates are truly based on linear fits, you can use the intital rates to perform the Michaelis-Menten-Fit, plot the data and calculate K_{m} and k_{cat}.

The Michaelis-Menten fit will be performed with the data from the "summary plot".

4.1 Fitting

requires four steps:

1. Defining the underlying model (the variable 'michaelis_menten' in the example below).

   It's arguments are 'params, concentrations, inhibitor_concentration (if needed, see 4.)'

2. Define the fitting instance (the variable 'michaelis_menten_fitter' in the example below). The inhibition model created in 2. must be in the fitters argument.

   It's arguments are 'params, concentrations, model = **your model**'

3. Calling the 'do_fit()'-method of the fitting instance to perform the fit

1. Optional: Additionally, you can give the inhibitior concentration in the same units as the substrate concentrations if needed for an inhibition model (see 1.).

In [ ]:
```
#1.
michaelis_menten = ek.MichaelisMenten(params, concentrations)
#2.
michaelis_menten_fitter = ek.MMFit(params, concentrations, model = michaelis_menten)
#3.
michaelis_menten_fitter.do_fit()
```

```
Michaelis Menten model fitted.
```

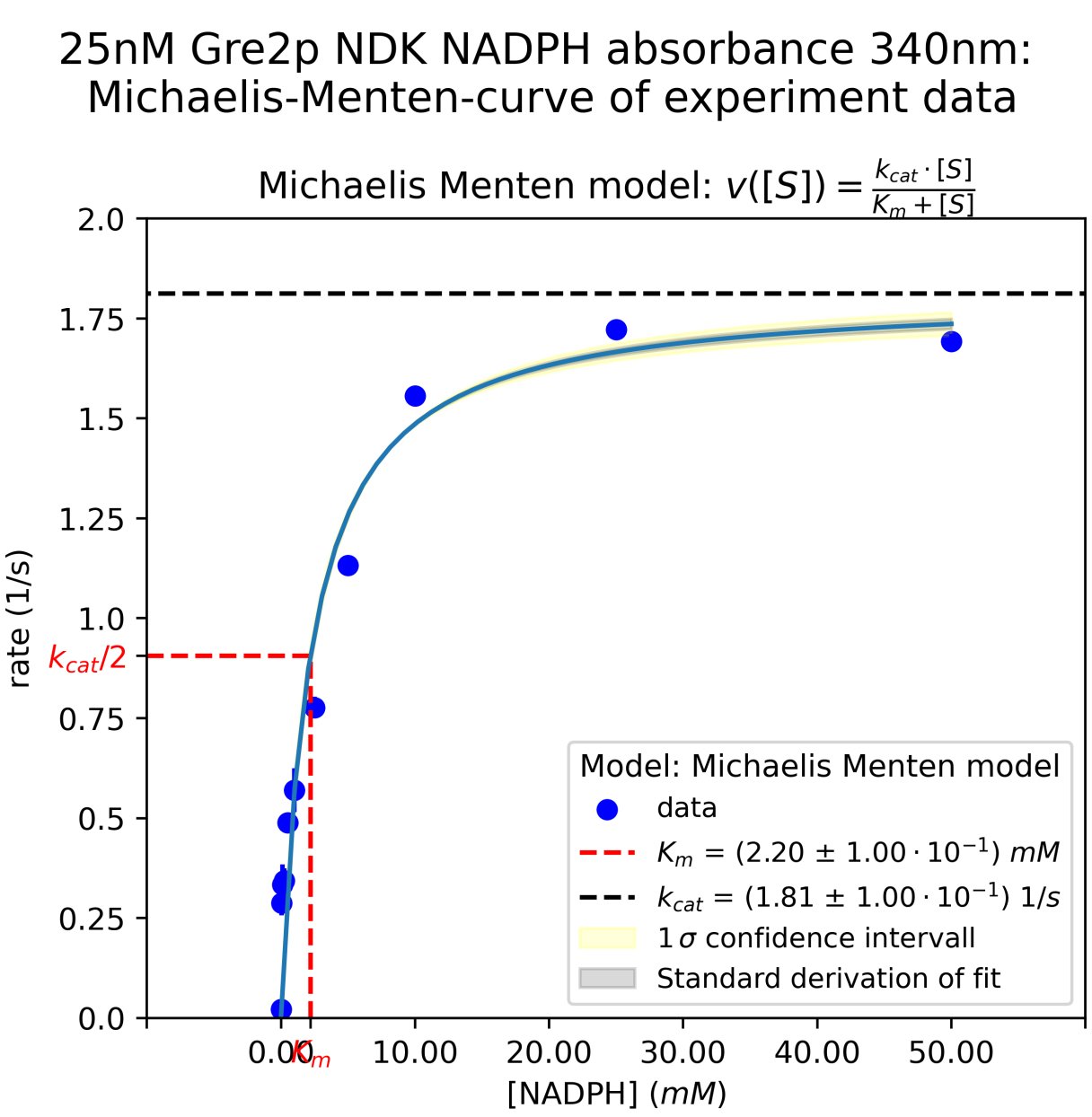
4.2 Plotting

requires three steps:

1. Create a 'plotter' instance, which will create a plot for the model that was used to fit the data.
2. To plot the fits, you need to create an instance of 'PlotMMFit' for every model you have performed a fit with. It's arguments are 'params, plotter, concentrations, model = **your model**'
3. The method 'do_plot()' of the plotter will do everything else.

In [ ]:
```
#1.
plotter = ek.Plotter(params)
#2.
michaelis_menten_plot = ek.PlotMMFit(params, plotter, concentrations, model = michaelis_menten)
#3.
plotter.do_plot()
```

```
Plot saved as png
```



You can add some text here to refer to other fits by other people or add an interpretation of your data...

Results as published by Ott et al. (https://pubs.acs.org/doi/10.1021/acscatal.1c02076):

$\mathrm{K_m} = (2.4 \pm 0.6)\,\mathrm{mM}$

$\mathrm{k_{cat}} = (1.8 \pm 0.1)\,1/\mathrm{s}$

5. Optional: Fit other models to your data

The Michaelis-Menten model (for an irreversible reaction (uni-uni)) is NOT a silver bullet that describes ALL enzyme kinetics!

So it may be necessary to use a different model to describe your data. We have included six different inhibition models in this notebook:

Inhibition model	Class	Arguments	Equation
No inhibition	NoInhibition(*arguments)	params, concentrations	$v([S]) = \frac{k_{cat} \cdot [S]}{K_m + [S]}$
Substrate inhibition	SubstrateInhibition(*arguments)	params, concentrations	$v([S]) = \frac{k_{cat} \cdot [S]}{K_m + [S]\left(1 + \frac{[S]}{K_i}\right)}$

Inhibition model	Class	Arguments	Equation
Uncompetitive inhibition	UncompetitiveInhibition(*arguments)	params, concentrations, inhibitor_concentration	$v([S]) = \frac{k_{cat}}{\left(1+\frac{[I]}{K_i^u}\right)+\frac{[Km]}{[S]}}$
Competitive inhibition	CompetitiveInhibition(*arguments)	params, concentrations, inhibitor_concentration	$v([S]) = \frac{k_{cat}}{1+\frac{Km}{[S]}\left(1+\frac{[I]}{K_i}\right)}$
Non competitive inhibition	NonCompetitiveInhibition(*arguments)	params, concentrations, inhibitor_concentration	$v([S]) = \frac{k_{cat}}{\left(1+\frac{Km}{[S]}\right)\left(1+\frac{[I]}{K_i}\right)}$
Mixedinhibition	MixedInhibition(*arguments)	params, concentrations, inhibitor_concentration	$v([S]) = \frac{k_{cat}}{\left(1+\frac{[I]}{K_i^u}\right)+\frac{Km}{[S]}\left(1+\frac{[I]}{K_i^c}\right)}$
Competitive product inhibition	CompetitiveProductInhibition(*arguments)	params, concentrations, inhibitor_concentration	$v([P]) = \frac{k_{cat}\cdot[P]}{Km\cdot\left(1+\frac{[I]}{K_P}\right)+[P]}$

Each model can be chosen by calling the corresponding class.

AGAIN:

Just because these models are defined here, does NOT make them the "correct" model for your data. We do not guarantee, that the parameters you will obtain do describe your enzyme kinetics. It is up to you to choose the correct model, and verify the conditions of the model are met.

If you want to add a model to your analytics, you need to go through two steps:

1. Extend the code at 4.1 for fitting the model:

```
#1.
michaelis_menten = ek.MichaelisMenten(params, concentrations)
#2.
michaelis_menten_fitter = ek.MMFit(params, concentrations, model = michaelis_menten)
#3.
michaelis_menten_fitter.do_fit()

competitive_inhibition_model = ek.CompetitiveInhibition(params, concentrations,
inhibitor_concentration = YOUR_INHIBITOR_CONCENTRATION)
competitive_inhibition_fitter = ek.MMFit(params, concentrations, model =
competitive_inhibition_model)
competitive_inhibition_fitter.do_fit()
```

Make sure to change "YOUR_INHIBITOR_CONCENTRATION" to the inhibitor concentration of your model in the same units as your substrate concentration.

Run the code and continue to the next step.

2. Extend the code at 4.2 for plotting the model:

```
#1.
plotter = ek.Plotter(params)
#2.
michaelis_menten_plot = ek.PlotMMFit(params, plotter, concentrations, model =
michaelis_menten)
competitive_inhibition_plot = ek.PlotMMFit(params, plotter, concentrations, model =
competitive_inhibition_model)
#3.
plotter.do_plot()
```

Now run the code and you see your model fitted and plotted next to the other plot.

How to create custom models

Disclaimer: You need a basic understanding of the Python coding language

To create our custom inhibition model, we'll take an existing inhibition model as an example. Open the file 'Models.py' in the 'EnzymeKinetics' directory. There are the default inhibition classes. One of them is the Class "CompetitiveInhibition":

```
class CompetitiveInhibition(Model):
def __init__(self, parameters: Parameters, concentrations: List[Concentration],
inhibitor_concentration: float):
        super().__init__(parameters, concentrations)
        self.name: str = "competitive inhibition"
        self.variables: List[Variable] = [
            Variable("$K_m$", f"{parameters.SUBSTRATE_UNIT}"),
            Variable("$k_{cat}$", "1/s"),
            Variable("$K_i$", f"{parameters.SUBSTRATE_UNIT}"),
        ]
        self.fit_function_latex: str = r"$v \left( [S] \right) = \frac{k_{cat}}{1+
\frac{K_m}{[S]} \left( 1+ \frac{[I]}{K_i} \right)}$"
        self.inhibitor_concentration: float = inhibitor_concentration

    def fit_function(self, x, a, b, c) -> float:
        return b / (1 + a / x * (1 + self.inhibitor_concentration / c))
```

Let's look over that class part by part.

1. Definition of the class

```
class CompetitiveInhibition(Model):
```

The class inherits from the 'Model' class. This parent class contains some methods, which are used to perform the fit. The parent file is in the script "ModelParentClass.py"

2. The instantiation of the class

```
def __init__(self, parameters: Parameters, concentrations: List[Concentration],
inhibitor_concentration: float):
        super().__init__(parameters, concentrations)
        self.name: str = "competitive inhibition"
        self.variables: List[Variable] = [
            Variable("$K_m$", f"{parameters.SUBSTRATE_UNIT}"),
            Variable("$k_{cat}$", "1/s"),
            Variable("$K_i$", f"{parameters.SUBSTRATE_UNIT}"),
        ]
        self.fit_function_latex: str = r"$v \left( [S] \right) = \frac{k_{cat}}{1+
\frac{K_m}{[S]} \left( 1+ \frac{[I]}{K_i} \right)}$"
        self.inhibitor_concentration: float = inhibitor_concentration
```

The __init__ function is called once the class is created. It's arguments are:

- parameters: these are parameters, like enzyme concentration. **Should not be changed**
- concentrations: this list contains the data to be fitted. **Should not be changed**
- inhibitor_concentration: the inhibitor concentration. If not needed in your model, delete it.

Extend the arguments as needed for your model.

The __init **function itself runs the parent-classes __init** function, sets a name for your model, variables, a LaTeX version of the function (for a nicer plot) and other parameters of the model.

Arguments of the variable class are it's name (in LaTeX for a nicer plot) and it's unit. In the template, the concentrations are the same unit as the substrate concentrations. Changing the units won't effect the fit and is just for plotting.

3. The fit function

```
def fit_function(self, x, a, b, c) -> float:
        return b / (1 + a / x * (1 + self.inhibitor_concentration / c))
```

The 'fit_function' function is used for fitting the model. Make sure to not rename this function. Arguments can be adjusted to fit to your model.

Arguments must be in following order: self, your x-axis-variable, other variables (**same order as in self.variables list in __init__**)

4. Own x- and y- values

By default it takes substrate concentrations (x-values) and returns rates in units of 1/s by dividing the enzyme concentration from the absolute averaged rates (y-values).

If you want to create your own x- or y-values for fitting, use the generate_fit_values function, which will overwrite the parents function. *This will need some study into the EnzymeKinetics module. Especially into the Concentration-class in Concentration.py and MMFit-class in FitHandler.py.* The parents class generate_fit_values looks like this:

```
def generate_fit_values(self):
        """Generates x and y values and their standard derivations to be fitted."""
        self.x_values = np.array([concentration.concentration for concentration in
self.concentrations]        )
        self.y_values = np.abs(np.array([concentration.rate /
self.parameters.ENZYME_CONCENTRATION for concentration in self.concentrations]))
        self.y_std_values = np.array([concentration.rate_std /
self.parameters.ENZYME_CONCENTRATION for concentration in self.concentrations])
```

You can use the "YourInhibition" class in the Models.py script as a template for your own model.