chemical_kinetics

A python module to load, fit and plot chemical kinetics data. In particular, it can be used to extract reaction time constants from experimental data. This data should at least describe the evolution of some of the species concentration over time. Optionally, if an electrochemical measurement was performed, the fit can also include the charge passed over time in order to improve the fit of the model parameters.

Installation

Using pip:

```
pip install chemical_kinetics
```

Or download the chemical_kinetics folder on Github and add it to your python path.

Examples

- Simple theoretical example: demonstrates the fit of species concentration evolution over time.
- Example using real experimental data: published in this scientific paper. Demonstrates the simultaneous fit of the species concentrations and charge passed evolution over time in an electrochemical experiment.

Dependencies

This code relies on the use of the scipy, numpy, pandas and matplotlib packages. It also uses the lmfit package (Matt Newville et al., LMFIT: Non-Linear Least-Square Minimization and Curve-Fitting for Python).

License

Licensed under the MIT License.

Simple example

This example is meant to showcase the basic steps needed to fit the concentration evolution over time of different species. These steps are the following:

- create a kinetic model that can be used by the “chemical_kinetics” module
- loading and plotting the data
- fitting and displaying the fit results

Kinetic model

We consider in this example the following reactions:

- Forward reaction 1: A $\rightarrow$ B with time constant $k_{1,\,fw}$
- Backward reaction 1: B $\rightarrow$ A with time constant $k_{1,\,bw}$
- Forward reaction 2: B $\rightarrow$ C with time constant k_2

Which gives the following differential equations according to the law of mass action:

$$\frac{dA}{dt} = k_{1,bw}[B] - k_{1,fw}[A]$$

$$\frac{dB}{dt} = k_{1,fw}[A] - k_{1,bw}[B] - k_2[B]$$

$$\frac{dC}{dt} = k_2[B]$$

This system of differential equations will be computed at a given time t using the following function:

```
def derivatives(y, t, p):

    """Calculates the derivatives of the concentrations at t.

    Used scipy.integrate.odeint to numerically solve the differential
    equations in a given time range.

    Lists ("y" and "dy") used by scipy.integrate.odeint are converted
    to dictionaries ("c" and "dc") in order to make the differentials
    easier to write and read for humans.

    Arguments:
        y (list): concentration values at t
        t (float): time value where the derivatives are calculated
        p (dict): dictionary containing the parameters used to
        calculate the derivatives e.g. time constants
    """

    # list ("y") to dict ("c") conversion
    c = {"A" : y[0], "B" : y[1], "C" : y[2]}

    # calculate the differentials
    dc = dict()
    dc["A"] = p["k_1bw"]*c["B"] - p["k_1fw"]*c["A"]
    dc["B"] = p["k_1fw"]*c["A"] - p["k_1bw"]*c["B"] - p["k_2"]*c["B"]
    dc["C"] = p["k_2"]*c["B"]

    # dict ("dc") to list ("dy") conversion
    dy = [dc["A"], dc["B"], dc["C"]]

    return dy
```

Caution: when defining the “derivatives” function, do not use keys for the “p” parameters dictionary containing the string “c0_” (if you do an error will be raised). These keys are reserved for the initial concentrations and will be defined and used in the “fit.fit_dataset” function.

Loading and plotting the dataset

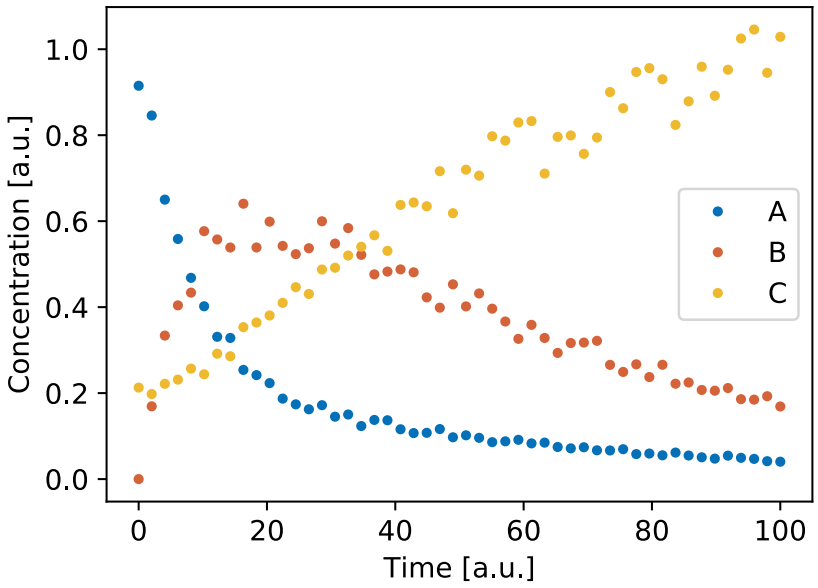
The dataset “data/concentrations vs time.csv” is loaded in an object of class “chemical_kinetics.data.Dataset”. This object stores the raw data and the fit results and makes these parameters easy to access.

Yon can consult the recomendations for the .csv files formatting in the “chemical_kinetics.data.Dataset.load_c” function documentation. The file loaded in this example can be found here.

```
from chemical_kinetics import data

ds = data.Dataset(
    files_c = ["data/concentrations vs time.csv"],
    t_label = "Time [a.u.]",
    c_label = "Concentration [a.u.]"
)
```

You can check if the data was loaded properly by plotting it:

```
from chemical_kinetics import plot

plot.plot_c(ds)
```



Fitting and displaying the fit results

We already defined the derivatives to be used by the fit in the derivatives function above. However, we also need to provide an initial guess for the time constants.

Note: In this case, only time constants constitute parameters stored in the “p” argument of the “residuals” function. However, a different “residuals” function definition can require parameters that are not time constants. These parameters can still be passed in the “p” dictionnary, the parameters in “p” are not required to be time constant, they can be any parameter needed by the model.

These parameters are given as a dictionary (“p” in the “residuals” function definition) where the keys are the time constant names. The corresponding values are a dictionary containing the parameter arguments, used to initialize a “lmfit.Parameter” object. The arguments that can be passed via this dictionary are in particular: value, vary, min, max and expr. Details on these arguments, and more generally on the “lmfit.Parameter” class can be found in the lmfit documentation.

```
parameters = {
    "k_1fw": dict(value = 0.1, min = 0),
    "k_1bw": dict(value = 0.1, min = 0),
    "k_2": dict(value = 0.1, min = 0)
}
```

Another argument to be passed to the fit function are the initial concentrations. These are defined in a similar way as the "parameters" variable defined above, since they are also fit parameters. It is mandatory to give the same names for theses parameters as the corresponding names given to the columns in the .csv file that was loaded in your dataset object. If you do not declare a value for the initial concentration for one of the species tracked in your .csv file then this value will be the first concentration value from this file by default.

For demonstration, in this example we consider that:

- the initial concentration of A is at least 0.5 and we default its initial value
- the initial concentration of B is known and fixed to 0
- the initial concentration of C is unknown and we use a default value for this parameter by not declaring it at all. The default value is the initial concentration for C in the dataset

```
c0 = {
    "A": dict(min = 0.5),
    "B": dict(value = 0., vary = False)
}
```

We can now pass these parameters to the fit function. Once the fit converged a message generated by the "lmfit.MinimizerResult" class is displayed (see the lmfit documentation for details on this message significance).

```
from chemical_kinetics import fit

fit.fit_dataset(
    dataset = ds,
    derivatives = derivatives,
    parameters = parameters,
    c0 = c0
)
```

```
Fit succeeded.
```

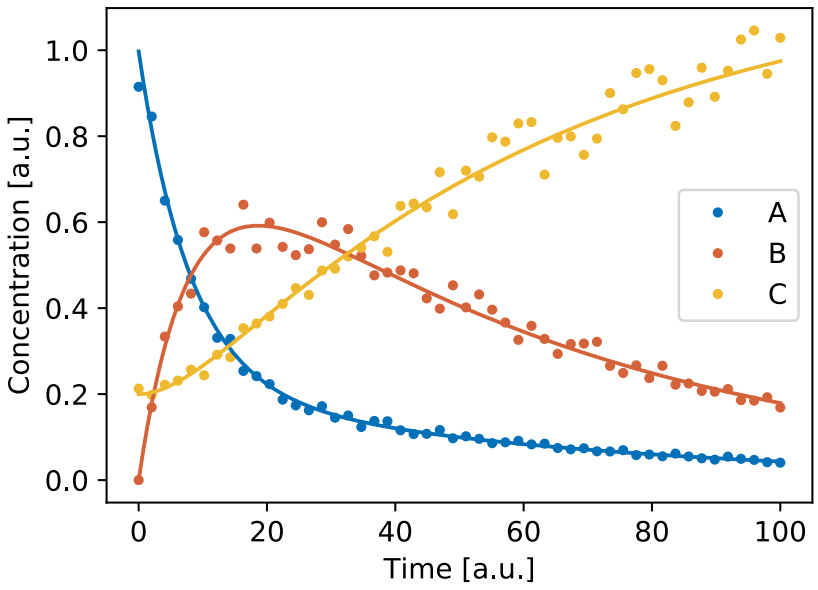
The fit results can be printed and plotted using the following functions:

```
fit.print_result(ds)
```

	name	value	stderr	stderr/value %	init. val.	vary	min	max

	name	value	stderr	stderr/value %	init. val.	vary	min	max
0	k_1fw	0.1	0.0	2.0	0.1	True	0.0	inf
1	k_1bw	0.0	0.0	3.1	0.1	True	0.0	inf
2	k_2	0.0	0.0	1.1	0.1	True	0.0	inf
3	c0_A	1.0	0.0	0.8	0.9	True	0.5	inf
4	c0_B	0.0	0.0	nan	0.0	False	-inf	inf
5	c0_C	0.2	0.0	2.3	0.2	True	-inf	inf

```
from chemical_kinetics import plot
plot.plot_c(ds)
```



For information, here is the code that was used to generate the raw data used in this example:

```
from chemical_kinetics import fit
import pandas as pd
import numpy as np
import lmfit

# Time constantsand initial concentrations definitions
params = lmfit.Parameters()
params.add("k_1fw", value = 0.1)
params.add("k_1bw", value = 0.02)
params.add("k_2", value = 0.02)
params.add("c0_A", value = 1)
params.add("c0_B", value = 0)
params.add("c0_C", value = 0.2)

# Generate time and concentration evolution over time
t = np.linspace(0,100,50)
c = fit.evaluate(derivatives, params, t)

# Add random noise
c += c*0.2*(np.random.random(c.shape) - 0.5)

# Create a DataFrame from t and c and save it as .csv
data = np.hstack((t.reshape(-1,1), c))
df = pd.DataFrame(columns = ["t", "A", "B", "C"], data = data)
df.to_csv(r"data/concentrations vs time.csv", index = False)
```

Advanced example

This example emphasizes two scenarii not covered in the simple example:

- using charge passed over time data, which can be acquired when performing electrochemical measurements (chronocoulometry), and that can help improve the accuracy of the time constants obtained from the fit
- fitting when the concentration evolution over time experimental data only covers some of the species considered in the model

Kinetic model

The kinetics model derived in this paper leads to the following set of differential equations:

$$\frac{\mathrm{d}[\mathrm{HMF}]}{\mathrm{dt}} = -(\mathrm{k_{11}} + \mathrm{k_{12}} + \mathrm{k_{D1}})[\mathrm{HMF}]$$

$$\frac{\mathrm{d}[\mathrm{DFF}]}{\mathrm{dt}} = \mathrm{k_{11}}[\mathrm{HMF}] - (\mathrm{k_{21}} + \mathrm{k_{D21}})[\mathrm{DFF}]$$

$$\frac{\mathrm{d}[\mathrm{HMFCA}]}{\mathrm{dt}} = \mathrm{k_{12}}[\mathrm{HMF}] - (\mathrm{k_{22}} + \mathrm{k_{D22}})[\mathrm{HMFCA}]$$

$$\frac{\mathrm{d}[\mathrm{FFCA}]}{\mathrm{dt}} = \mathrm{k_{21}}[\mathrm{DFF}] + \mathrm{k_{22}}[\mathrm{HMFCA}] - (\mathrm{k_3} + \mathrm{k_{D3}})[\mathrm{FFCA}]$$

$$\frac{\mathrm{d}[\mathrm{FDCA}]}{\mathrm{dt}} = \mathrm{k_3}[\mathrm{FFCA}] - \mathrm{k_{D4}}[\mathrm{FDCA}]$$

$$\frac{\mathrm{d}[\mathrm{D}]}{\mathrm{dt}} = \mathrm{k_{D1}}[\mathrm{HMF}] + \mathrm{k_{D21}}[\mathrm{DFF}] + \mathrm{k_{D22}}[\mathrm{HMFCA}]$$
$$+\mathrm{k_{D3}}[\mathrm{FFCA}] + \mathrm{k_{D4}}[\mathrm{FDCA}] - \mathrm{k_{D^*}}[\mathrm{D}]$$

$$\frac{\mathrm{d}[\mathrm{D^*}]}{\mathrm{dt}} = \mathrm{k_{D^*}}[\mathrm{D}]$$

In order to write a function describing this model we need to define all the species whose concentrations evolution over time are considered in this study. In that case 5 species concentrations were tracked by HPLC measurements (HMF, DFF, HMFCA, FFCA, FDCA). The degradation products (D and D*) concentrations could not be tracked. For the untracked species we need to consider 10 additional concentrations. Indeed, in the differential equations mentionned

above only the total concentration evolution of D and D* are considered. However, these total D and D* concentrations need to be splitted here into the individual concentrations coming from each individual degradation product formation reactions. This is required because humins species formed through different reaction pathways will involve a different amount of charge passed in our electrochemical measurement. With these individual concentrations we are able to calculate the amount of charge passed over time allowing us to include the corresponding experimental data in the fit.

We define the species names considered in this experiment as follow:

```
species_tracked = [
    "HMF", "DFF", "HMFCA", "FFCA", "FDCA"
]
species_untracked = [
    "D_HMF",  "D_DFF",  "D_HMFCA",  "D_FFCA",  "D_FDCA",
    "Dx_HMF", "Dx_DFF", "Dx_HMFCA", "Dx_FFCA", "Dx_FDCA"
]
species = species_tracked.copy()
species.extend(species_untracked)
```

HMF, DFF, HMFCA, FFCA and FDCA are the species whose concentrations are tracked experimentally. The D and D* coming from these species are denoted with the same names prepended respectively with a "D*" and a "Dx*". To sum up we can now write:

$$[\mathrm{D}] = [\mathrm{D\_HMF}] + [\mathrm{D\_DFF}] + [\mathrm{D\_HMFCA}] + [\mathrm{D\_FFCA}] + [\mathrm{D\_FDCA}]$$

$$[\mathrm{D^*}] = [\mathrm{Dx\_HMF}] + [\mathrm{Dx\_DFF}] + [\mathrm{Dx\_HMFCA}] + [\mathrm{Dx\_FFCA}] + [\mathrm{Dx\_FDCA}]$$

Therefore, the derivatives presented above for the concentration of humins are rewritten in the following way:

$$\frac{\mathrm{d[D\_HMF]}}{\mathrm{dt}} = \mathrm{k_{D1}}[\mathrm{HMF}] - \mathrm{k_{D^*}}[\mathrm{D\_HMF}]$$

$$\frac{\mathrm{d[D\_DFF]}}{\mathrm{dt}} = \mathrm{k_{D1}}[\mathrm{DFF}] - \mathrm{k_{D^*}}[\mathrm{D\_DFF}]$$

$$\frac{\mathrm{d[D\_HMFCA]}}{\mathrm{dt}} = \mathrm{k_{D1}}[\mathrm{HMFCA}] - \mathrm{k_{D^*}}[\mathrm{D\_HMFCA}]$$

$$\frac{\mathrm{d[D\_FFCA]}}{\mathrm{dt}} = \mathrm{k_{D1}}[\mathrm{FFCA}] - \mathrm{k_{D^*}}[\mathrm{D\_FFCA}]$$

$$\frac{\mathrm{d[D\_FDCA]}}{\mathrm{dt}} = \mathrm{k_{D1}}[\mathrm{FDCA}] - \mathrm{k_{D^*}}[\mathrm{D\_FDCA}]$$

The derivatives for the concentration of D* are rewritten in this way:

$$\frac{\mathrm{d}[\mathrm{Dx\_HMF}]}{\mathrm{dt}} = \mathrm{k_{D^*}}[\mathrm{D\_HMF}]$$

$$\frac{\mathrm{d}[\mathrm{Dx\_DFF}]}{\mathrm{dt}} = \mathrm{k_{D^*}}[\mathrm{D\_DFF}]$$

$$\frac{\mathrm{d}[\mathrm{Dx\_HMFCA}]}{\mathrm{dt}} = \mathrm{k_{D^*}}[\mathrm{D\_HMFCA}]$$

$$\frac{\mathrm{d}[\mathrm{Dx\_FFCA}]}{\mathrm{dt}} = \mathrm{k_{D^*}}[\mathrm{D\_FFCA}]$$

$$\frac{\mathrm{d}[\mathrm{Dx\_FDCA}]}{\mathrm{dt}} = \mathrm{k_{D^*}}[\mathrm{D\_FDCA}]$$

We can now write the “derivatives” function derived from these differential equations:

```
def derivatives(y, t, p):

    """Calculates the derivatives of the concentrations at t

    Used scipy.integrate.odeint to numerically solve the differential
    equations in a given time range.

    Lists ("y" and "dy") used by scipy.integrate.odeint are converted
    to dictionaries ("c" and "dc") in order to make the differentials
    easier to write and read for humans.

    Arguments:
        y (list): concentration values at t
        t (float): time value where the derivatives are calculated
        p (dict): dictionary containing the parameters used to
        calculate the derivatives e.g. time constants
    """

    c = {s:y[i] for i, s in enumerate(species)}

    dc = dict()

    dc["HMF"]      =  - (p["k11"] + p["k12"] + p["kD1"])*c["HMF"]
    dc["DFF"]      = p["k11"]*c["HMF"]                      - (p["k21"] +
p["kD21"])*c["DFF"]
    dc["HMFCA"]    = p["k12"]*c["HMF"]                      - (p["k22"] +
p["kD22"])*c["HMFCA"]
    dc["FFCA"]     = p["k21"]*c["DFF"] + p["k22"]*c["HMFCA"] - (p["k3"] + p["kD3"])*c["FFCA"]
    dc["FDCA"]     = p["k3"]*c["FFCA"]                      - p["kD4"]*c["FDCA"]

    dc["D_HMF"]    = p["kD1"]*c["HMF"]    - p["kDx"]*c["D_HMF"]
    dc["D_DFF"]    = p["kD21"]*c["DFF"]   - p["kDx"]*c["D_DFF"]
    dc["D_HMFCA"]  = p["kD22"]*c["HMFCA"] - p["kDx"]*c["D_HMFCA"]
    dc["D_FFCA"]   = p["kD3"]*c["FFCA"]   - p["kDx"]*c["D_FFCA"]
    dc["D_FDCA"]   = p["kD4"]*c["FDCA"]   - p["kDx"]*c["D_FDCA"]

    dc["Dx_HMF"]   = p["kDx"]*c["D_HMF"]
    dc["Dx_DFF"]   = p["kDx"]*c["D_DFF"]
    dc["Dx_HMFCA"] = p["kDx"]*c["D_HMFCA"]
    dc["Dx_FFCA"]  = p["kDx"]*c["D_FFCA"]
    dc["Dx_FDCA"]  = p["kDx"]*c["D_FDCA"]

    dy = [dc[name] for name in species]

    return dy
```

We can then convert the concentrations evolution over time into the charge passed over time using this equation:

$$Q = eN_A V \sum_i n_i C_i$$

With e the electron charge in Coulombs, N_A the Avogadro number, V the volume of solution, n_i the number of charge passed to make one molecule of i, and C_i the concentration of species i. This equation translates into code as follow:

```
import numpy as np
import scipy.constants as constants

def c_to_q(c):

    """Calculates the charge passed from the concentrations vs time

    Arguments:
        c (numpy.ndarray):
            Concentrations evolution over time, axis 0 is time,
            axis 1 is species.
    """

    # the concentrations are monitored in micromoles/L
    # so we convert them to moles/L
    c *= 1e-6

    # calculate the product ni*Ci for each species
    ni_Ci = list()
    for i, s in enumerate(species):
        ni_Ci.append(2*(i%5 + int(i/5))*c[:,i])

    # calculate the sum of ni*Ci for all species
    sum_ni_Ci = np.sum(ni_Ci, axis = 0)

    # solution volume in L
    V = 100e-3

    # charge passed in C
    q = constants.e*constants.N_A*V*sum_ni_Ci

    return q
```

Now that the model is defined we can load the raw data and fit it.

Load concentrations and charge passed evolution over time

Here three .csv files are used to get the measured evolution of HMF, DFF, HMFCA, FFCA and FDCA concentrations over time. Three additional .csv files are used for the charge passed over time.

```
from chemical_kinetics import data

folders = [f"data/run{i}/" for i in range(1,4)]
files_c = [f"{folder}Reaction Monitoring.csv" for folder in folders]
files_q = [f"{folder}Charge Passed.csv" for folder in folders]

ds = data.Dataset(
    files_c = files_c,
    files_q = files_q,
    t_label = "Time [h]",
    c_label = r"Concentration [:math:`\rm\mu`M]",
    q_label = "Charge passed [C]"
)
```

Fitting

The only parameters to be fitted are the time constants. The initial concentrations are fixed to the initial values recorded by HPLC for the tracked species. For the untracked species the initial concentrations are fixed to 0.

We first define the time constants parameters:

```
parameter_args = dict(value = 0.05, min = 0)
parameter_names = [
    "k11","k12","k21","k22","k3",
    "kD1","kD21","kD22","kD3","kD4",
    "kDx"
    ]
parameters = {name: parameter_args for name in parameter_names}
```

Then we define the initial concentrations parameter for the tracked species:

```
c0 = {name: dict(vary = False) for name in species_tracked}
```

And finally, we define the initial concentrations for the untracked species (note that we need here to define an ordered dictionary):

```
from collections import OrderedDict
c0_untracked = OrderedDict({
    name: dict(value = 0, vary = False) for name in species_untracked
})
```

With the data loaded, the model defined and the parameters / initial concentrations initialized we can proceed with the fit:

```
from chemical_kinetics import fit

fit.fit_dataset(
    dataset = ds,
    derivatives = derivatives,
    c_to_q = c_to_q,
    parameters = parameters,
    c0 = c0,
    c0_untracked = c0_untracked
)
```

```
Fit succeeded.
```

Fit results

The fit results are summarized in the table below. Note that all the initial concentrations (parameters starting with the string "c0_") are fixed so only the ks values are varying parameters in this fit.

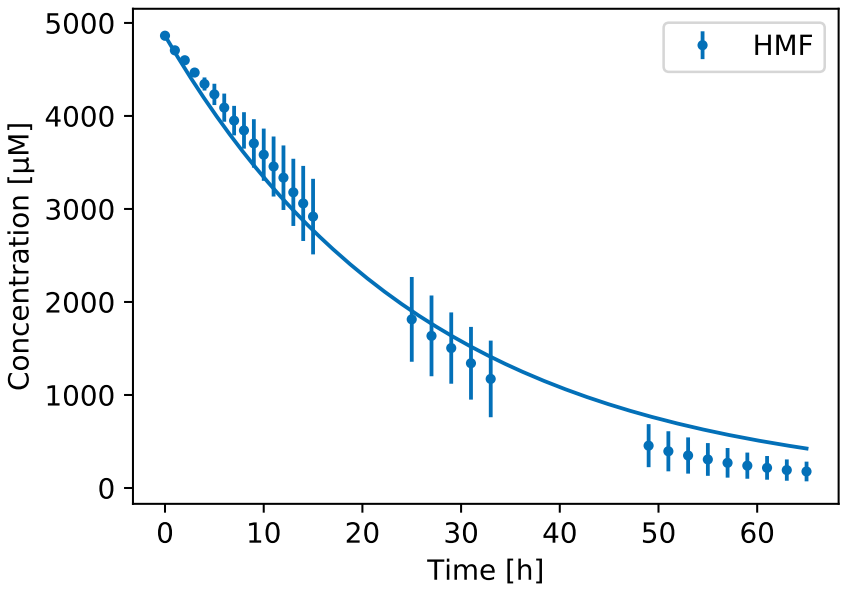
```
fit.print_result(ds)
```

	name	value	stderr	stderr/value %	init. val.	vary	min	ma
0	k11	0.0	0.0	3.5	0.1	True	0.0	inf
1	k12	0.0	0.0	2.9	0.1	True	0.0	inf
2	k21	0.1	0.0	6.3	0.1	True	0.0	inf
3	k22	0.0	0.0	22.4	0.1	True	0.0	inf
4	k3	0.0	0.0	5.4	0.1	True	0.0	inf
5	kD1	0.0	0.0	1.2	0.1	True	0.0	inf
6	kD21	0.0	0.0	16.3	0.1	True	0.0	inf
7	kD22	0.0	0.0	28625.5	0.1	True	0.0	inf
8	kD3	0.1	0.0	7.9	0.1	True	0.0	inf
9	kD4	0.1	0.0	8.1	0.1	True	0.0	inf
10	kDx	0.0	0.0	15.9	0.1	True	0.0	inf
11	c0_HMF	4862.5	0.0	0.0	4862.5	False	-inf	inf
12	c0_DFF	4.6	0.0	0.0	4.6	False	-inf	inf
13	c0_HMFCA	0.5	0.0	0.0	0.5	False	-inf	inf
14	c0_FFCA	0.2	0.0	0.0	0.2	False	-inf	inf
15	c0_FDCA	0.1	0.0	0.0	0.1	False	-inf	inf
16	c0_D_HMF	0.0	0.0	nan	0.0	False	-inf	inf
17	c0_D_DFF	0.0	0.0	nan	0.0	False	-inf	inf
18	c0_D_HMFCA	0.0	0.0	nan	0.0	False	-inf	inf
19	c0_D_FFCA	0.0	0.0	nan	0.0	False	-inf	inf

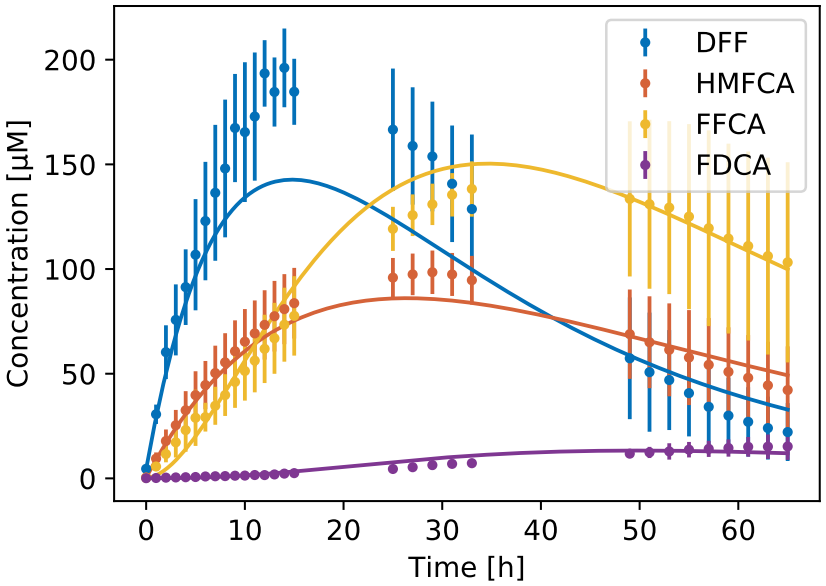
	name	value	stderr	stderr/value %	init. val.	vary	min	ma
20	c0_D_FDCA	0.0	0.0	nan	0.0	False	-inf	inf
21	c0_Dx_HMF	0.0	0.0	nan	0.0	False	-inf	inf
22	c0_Dx_DFF	0.0	0.0	nan	0.0	False	-inf	inf
23	c0_Dx_HMFCA	0.0	0.0	nan	0.0	False	-inf	inf
24	c0_Dx_FFCA	0.0	0.0	nan	0.0	False	-inf	inf
25	c0_Dx_FDCA	0.0	0.0	nan	0.0	False	-inf	inf

To assess the fit results the evolution of concentration over time and charge passed over time are plotted. For the concentrations evolution over time the lines correspond to the fit result and the points with errorbars to the experimental data.

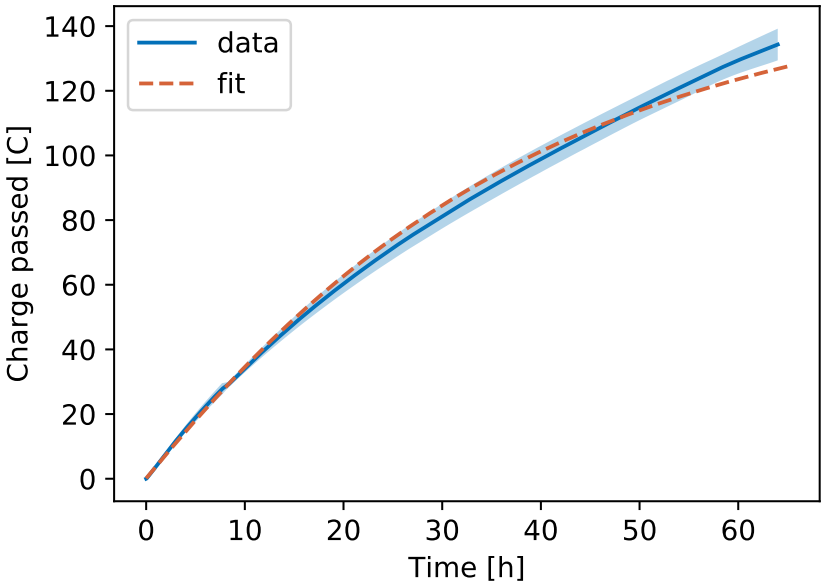
```
from chemical_kinetics import plot

plot.plot_c(ds, ["HMF"])
```




```
plot.plot_c(ds, ["DFF", "HMFCA", "FFCA", "FDCA"])
```




```
plot.plot_q(ds)
```



Loading data - data.py

This module defines the “Dataset” class that is used to load the .csv files holding the data to be fitted. After the fit is performed it also stores the initial parameters, the fitted parameters and the fit estimations.

This module also defines the function “load” used in the “Dataset.load_c” and “Dataset.load_q” functions to load the .csv files.

class **data.Dataset**(*files_c*, *files_q=None*, *t_label='t [s]'*, *c_label='C [M]'*, *q_label='Q [C]'*)

Loads and stores kinetics data, also stores fit results and axes labels.

Parameters

- **files_c** (*list*) – The path(s) of the concentration vs time files.
- **files_q** (*list*, *optional*) – The path(s) of the charge passed vs time files, if this argument is not set, further fitting will proceed without taking the charge passed in account.
- **t_label|c_label|q_label** (*str*, *optional*) – Set the corresponding attributes values. Used to set the x and y axes labels when plotting.

df_c|df_c_std|df_c_fit

Store respectively the mean concentration, the concentration standard deviations and the corresponding fit results. Defined by the load_c method or directly at initialization using the files_c argument.

Type

pandas.DataFrame

df_q|df_q_std|df_q_fit

Store respectively the mean charge passed, the charge passed standard deviations and the corresponding fit results. Defined by the load_q method or directly at initialization using the files_q argument.

Type

pandas.DataFrame

t_label|c_label|q_label

Labels to be used for the x and y axes when plotting the datasets.

Type

str

`fit_results`

Stores the results of the fit, for details see the lmfit.minimizer.MinimizerResult documentation: https://lmfit.github.io/lmfit-py/fitting.html

Defined when the fit.fit_dataset() function is run on the Dataset object.

Type

lmfit.MinimizerResult

`init_params`

Stores the initial parameters for the fit, for details on this object class see: https://lmfit.github.io/lmfit-py/parameters.html

Defined when the fit.fit_dataset() function is run on the Dataset object.

Type

lmfit.parameter.Parameters

`names`

Species names which concentration evolution over time is tracked. Used in particular to label the data when plotting. Defined in the load_c method.

Type

list

`load_c`(*files*)

Loads / processes .csv files holding concentration over time data.

Recommendations for the .csv file formatting: the files headers are formatted in this fashion: “t, species name 1, species name 2...”; first column is time, other columns are concentrations.

An example file can be found here: https://is.gd/GZPZFK

Relies on the load() function from this module, see also this function documentation for more details.

Parameters

files (*list*) – Path(s) of the .csv files.

load_q(*files*)

Loads / processes .csv files holding charge passed over time data.

Recommendations for the .csv file formatting: the files headers should be formatted in this fashion: “t,Q”; first column is the time, second column is the charge passed.

An example file can be found here: https://is.gd/0Ma7Ii

Relies on the load() function from this module, see also this function Docstring for more details.

Parameters

files (*list*) – Paths of the .csv files.

data.load(*files*)

Loads and processes data from .csv files, stores it in Dataframes.

The .csv files should have the same headers, the same number of columns and the same number of rows. The header should be held in the first row.

These .csv files are typically generated from raw data files from experiments using custom code to satisfy these requirements.

Parameters

files (*list*) – Path(s) to the file(s), each column in each file must have the same number of rows.

Returns

Averages DataFrame and standard deviation DataFrame.

Return type

pandas.DataFrame, pandas.DataFrame

Fitting - fit.py

This module defines a set of functions used to fit the species concentrations evolution over time and optionally the charge passed evolution over time, stored in an object of Dataset class. This fit proceeds via the "fit_dataset" function that uses the functions "residuals", "calculate_residuals" and "evaluate".

After the fit is performed the "print_result" function can be used to print the fit parameters initial values and setup (min, max, vary) and their fitted values and standard deviations.

The function "evaluate" can be used to get values from a kinetic model outside of the scope of fitting data, e.g. to test the influence of model parameters on the concentrations evolution over time.

`fit.calculate_residuals`(*df, fit, names*)

Calculates residuals values by comparing values in df and in fit.

> **Parameters**
>
> - **df** (*pandas.DataFrame*) – Holds the data to be fitted. Either concentrations vs time or charge passed vs time depending on the situation.
> - **fit** (*numpy.ndarray*) – Holds the fit evaluation.
> - **names** – Names of the columns in df that hold the data to be compared to the fit values. Necessary because in some cases not all of the data stored in df is fitted.

`fit.evaluate`(*derivatives, params, t*)

Evaluate the concentration(s) evolution(s) over time.

> **Parameters**
>
> - **derivatives** (*function*) – A function in the form dy = f(y, t, p) used to compute d(concentration)/dt at a time t for each species. Used by scipy.integrate.odeint
> - **params** (*lmfit.parameter.Parameters*) – The parameters values used to compute the derivatives function, for details on this object class see: https://lmfit.github.io/lmfit-py/parameters.html
> - **t** (*list*) – Time values at which the concentrations should be evaluated.

`fit.fit_dataset`(*dataset, derivatives, parameters, c0={}, c0_untracked={}, c_to_q=None*)

Fit a dataset holding concentration vs t data and optionally charge vs t.

The arguments “parameters”, “c0” and “c0_untracked” are dictionaries in which each value is a dictionary of the arguments to use in order to initialize objects of the lmfit.Parameter class. The arguments that can be passed via this dictionary are in particular: value, vary, min, max and expr. Details on the Parameter class can be found here: https://lmfit.github.io/lmfit-py/parameters.html

Parameters

- **dataset** (*chemical_kinetics.data.Dataset*) – Object holding the different DataFrames containing the data to be fitted.
- **derivatives** (*function*) – A function in the form dy = f(y, t, p) used to compute d(concentration)/dt at a time t for each species. Used by scipy.integrate.odeint
- **parameters** (*dict*) – Stores parameter names (str): arguments (dict) (e.g. value, min, max, vary) to be passed to the corresponding lmfit.Parameter. Represents all the parameters of the kinetic model.
- **c0** (*dict*, *optional*) – Stores species name (str): arguments (dict) (e.g. value, min, max, vary) to be passed to the corresponding lmfit.Parameter. Represent the concentrations at initial time for the species whose concentration evolution over time is stored in dataset.df_c.
- **c0_untracked** (*collections.OrderedDict*, *optional*) – Stores species name (str): arguments (dict) (e.g. value, min, max, vary) to be passed to the corresponding lmfit.Parameter. Represent concentrations at initial time for the species whose concentrations evolution over time is NOT stored in dataset.df_c. An ordered dictionary is necessary in this case to be able to pass arguments properly to the scipy.integrate.odeint solver.
- **c_to_q** (*function*, *optional*) – Used to convert the concentrations over time evolution into charge passed.

`fit.print_result`(*dataset*)

Pretty printing of the fit parameters stored in dataset.

Parameters

dataset (*chemical_kinetics.data.Dataset*) – Object holding the different DataFrames containing the initial parameters and the fitted parameters.

`fit.residuals`(*params*, *df_c*, *derivatives*, *tracked_species*, *df_q=None*, *c_to_q=None*)

Calculates residuals for concentrations vs t and optionally charge vs t.

Parameters

- **params** (*lmfit.parameter.Parameters*) – The parameters values used to compute the derivatives function, for details on this object class see: https://lmfit.github.io/lmfit-

py/parameters.html

- **df_c** (*pandas.DataFrame*) – Holds the concentration vs time data to be fitted.
- **derivatives** (*function*) – A function in the form dy = f(y, t, p) used to compute d(concentration)/dt at a time t for each species. Used by scipy.integrate.odeint
- **tracked_species** (*list*) – Column names in df_c corresponding to the fitted data (used to exclude e.g. the “t” column).
- **df_q** (*pandas.DataFrame*, *optional*) – Holds the charge passed vs time data to be fitted.
- **c_to_q** (*function*, *optional*) – Used to convert the concentrations over time evolution into charge passed.

Returns

Residuals values.

Return type

list

Plotting - plot.py

This module defines two functions to plot the data contained in an object of class chemical_kinetics.data.Dataset. “plot_c” plots the species concentration evolution over time and “plot_q” plots the charge passed evolution over time. If no fit was performed on the dataset object these functions only plot the raw data. If a fit was performed it also plot the fit estimation.

---

plot.plot_c(*dataset*, *names=None*)

Plots the species concentrations evolution over time and its fit.

Plots the fit results only if the data has been fitted once, i.e. chemical_kinetics.fit.fit_dataset function was run once on the dataset. Else it only plots the raw data.

> **Parameters**
>
> - **dataset** (*chemical_kinetics.data.Dataset*) – Object holding the DataFrame containing the data and the fit results.
> - **names** (*list*, *optional*) – List of names of the species concentration and fit to be plotted, if None all concentrations are plotted.

---

plot.plot_q(*dataset*)

Plots the charge passed over time and its fit.

Plots the fit results only if the data has been fitted once, i.e. chemical_kinetics.fit.fit_dataset function was run once on the dataset. Else it only plots the raw data.

> **Parameters**
>
> **dataset** (*chemical_kinetics.data.Dataset*) – Object holding the DataFrame containing the data and the fit results.