pdom

Release 1.0.0

Jul 21, 2020

Contents

simulation toolkit for Photocatalytic Degradation of Organic Molecules

CHAPTER 1

Table of contents

1.1 Setup

1.1.1 CLI

Most features of pdom can be accessed directly from its command-line tools. The easiest way to install them on your system is via pipx. *pipx* is a packagemenager for tools written in python that helps to keep them isolated and up to date. Some notes on how to get *pipx* running can be found at the end of this section. If you just need the CLI, use the following line to install pdom.

```
$ pipx install pdom
```

1.1.2 Library

You can also install the `pdom` library directly through the Python Package Index (PyPI) for use in your own projects. The use of a virtual environment is recommended.

```
$ pip install pdom
```

If the stable version of pdom on PyPI is missing a particular function, you can install the latest development version directly from the GitHub repository.

```
$ pip install -U git+https://github.com/theia-dev/pdom.git#egg=pdom
```

1.1.3 Developing and documentation

To work with the source code clone the repository from GitHub and install the requirements. The source code is accompanied by the documentation and an extensive collection of test cases.

```
$ git clone https://github.com/theia-dev/pdom.git
$ python3 -m venv pdom_env
$ . pdom_env/bin/activate
(pdom_env) $ pip install --upgrade pip
(pdom_env) $ pip install -r pdom/requirements.txt
```

Building the documentation locally needs a few extra python packages. They can also be installed in the same virtual environment with the following command.

```
(pdom_env) $ pip install -r pdom/docs/requirements.txt
```

The HTML version of the documentation can then be built:

```
(pdom_env) $ sphinx-build -b html pdom/docs pdom/docs_html
```

The tests are located under `pdom/tests` can be started with through:

```
(pdom_env) $ python -m unittest discover -s pdom/tests
```

1.1.4 pipx

Under macOS:

For macOS the Homebrew package manager is the easiest way to install pipx.

```
$ brew install pipx
$ pipx ensurepath
```

Under Linux:

For some distributions the python package system *pip* is not installed by default. On Debin/Ubuntu systems it can be quickly installed.

```
$ sudo apt update
$ sudo apt install python3-pip
```

Then *pipx* can be added.

```
$ python3 -m pip install --user pipx
$ python3 -m pipx ensurepath
```

Under Windows:

Python is not installed by default under Windows. You can get the installer from the Python download page. The python package system *pip* is already included in the latest releases.

In the windows commandline *pipx* can then be installed.

```
$ python3 -m pip install --user pipx
$ python3 -m pipx ensurepath
```

For more information on *pipx* refer to its documentation.

1.2 Usage

Here you can find a quick summary about `pdom` command-line tools and the `pdom` library. A more in-depth overview can be found in the *Examples* section

1.2.1 CLI pdom

A simple command line interface to run pdom.

```
usage: pdom [-h] [-d DATA] [-o OUT] config [config ...]
```

Positional Arguments

config config file (.ini)

Named Arguments

-d, --data experimental data for fit (.json)

-o, --out output folder (absolute or relative, will be created if it does not exit)

1.2.2 CLI pdom.config

A command line interface to create pdom config files.

```
usage: pdom.config [-h] [-s STRUCTURE] [-o OUTFILE]
```

Named Arguments

-s, --structure xyz molecule structure file

-o, --outfile output file

How to create a config is described in the *Configuration* section.

1.2.3 Library

```
import pdom

simulation = Simulate('simple.ini')
simulation.run()
```

See also the source code documentation of *pdom.Simulate*.

1.3 Configuration

1.3.1 Simulation

To run the simulation `pdom` needs a couple of information from the user. For `pdom`, this data is saved in an `.ini` file. This ensures that the results of a simulation can always be accompanied by the parameters that created them.

A simple config for `pdom` looks like this:

Listing 1: simple.ini

```
[SIMULATION]
id = simple_degradation_methylene_blue
multi = False
fit = False
duration = 5 h

[SYSTEM]
concentration_solution = 0.08 mmol/L
k_ads = 9.0E-10 m/s
k_des = 1.0E-3 1/s
k_reac = 1.1E-2 1/s

[CATALYST]
concentration = 2.0 g/L
surface = 50 m^2/g
volume = 1e-3 m^3

[MOLECULE]
name = methylene blue
composition = C16H18S1N3
molar_volume = 226.6 Ang^3/molecule
molar_surface = 99.7 Ang^2/molecule
diffusion_model = s
```

The parameters are arranged in different sections. Not all sections need to exist for each simulation.

To create a new `.ini` file you can use the configuration tool of `pdom`.

```
$ pdom.config --out 'my_new.ini'
```

It will guide you through the process by collecting all relevant information. For the parameters of the initial molecule, it is helpful to look up its chemID in PubChem before you start the process. This enables the automatic gathering of the molecule parameters from this database.

Most of the time, the automatic process should suffice. But if you want to build your `.ini` file from scratch, take a look into the available *Configuration settings*.

1.3.2 Experimental data

To extract parameters, we need to compare experimental data to the simulation. This data needs to be provided in a structured way. For **pdom** we use a `.json` file. Depending on the type of fit you want to carry out, the available features differ slightly.

Adsorption-Desorption experiments

Adsorption-Desorption experiments in the dark are analyzed with the single-species model. As it is common to have multiple repetitions that are based on the same setup. **pdom** supports multiple time series in its fits. The initial concentration and time steps can be different between the series. Below are examples of such a data file.

Listing 2: ads_des_multi.json

```
{
  "time_series": [
    [
      [0, 10, 15, 30, 60, 120, 240], [0.000, 0.076, 0.143, 0.175, 0.199, 0.207, 0.209]
    ], [
      [0, 10, 15, 30, 60, 120, 240], [0.000, 0.251, 0.452, 0.570, 0.609, 0.637, 0.633]
    ], [
      [0, 10, 15, 30, 60, 120, 240], [0.000, 0.598, 0.750, 0.898, 0.998, 1.021, 1.020]
    ], [
      [0, 10, 15, 30, 60, 120, 240], [0.000, 0.700, 0.996, 1.248, 1.396, 1.415, 1.415]
    ]
  ],
  "time_series_meta": [
    {
      "unit": "min",
      "type": "t"
    },
    {
      "unit": "mo/mc",
      "factor": 1E-3,
      "type": "surface"
    }
  ],
  "initial_concentration": [5.0, 15.0, 25.0, 35.0],
  "initial_concentration_meta": {
    "unit": "mg/L",
    "type": "solution"
  }
}
```

Degradation experiments

For degradation experiments all time series have to start with the same initial concentration set in the `.ini` file. This is the data file from the example *Degradation fit*.

Listing 3: fit_reac_mutli.json

```
{
  "time_series": [
    [
      [0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 20, 25],
      [3.442, 2.847, 2.428, 2.229, 1.949, 1.801, 1.705, 1.535,
        1.315, 1.021, 0.9690, 0.8238, 0.5114, 0.2839, 0.1353]
    ], [
      [0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 20, 25],
      [3.505, 2.876, 2.586, 2.323, 2.242, 2.097, 1.781, 1.681,
        1.461, 1.249, 0.9543, 1.0852, 0.7665, 0.4324, 0.2368]
    ]
```

(continues on next page)

(continued from previous page)

```
  ],
  "time_series_meta": [
    {
      "unit": "min",
      "type": "t"
    }, {
      "unit": "mg/L",
      "type": "solution"
    }
  ]
}
```

Multi species model

To compare multi-species model simulations to experiments, TOC (or NPOC) can be used. In general, the fit to TOC data is the last step in the experiment analytics. This fit is limited to a single TOC curve, due to the usually demanding experimental process. If multiple TOC experiments are available, the results should be average before simulation. As just one initial concentration is needed, it is taken from the `.ini` file.

This is the data file from the example *TOC fit*.

Listing 4: fit_toc.json

```
{
  "time_series": [
    [0, 60, 120, 180, 360],
    [12.6, 8.8, 6.78, 4.1, 2.77]
  ],
  "time_series_meta": [
    {
      "unit": "min",
      "type": "t"
    }, {
      "unit": "mg/L",
      "type": "toc"
    }
  ]
}
```

1.3.3 Configuration settings

SIMULATION

- ***id*** → **str**

  default `example_mb`

- ***multi*** → **bool**

  default `True`

  note MULTI section needed if True

- ***fit*** → **bool**

  default `False`

note FIT section needed if True

- ***duration*** → **float**

  default `5 h`

  units `h`, `min`, `s`

SOLVER

Relative and absolute tolerances for the LSODA solver

- ***rtol*** → **float**

  default `1e-9`

- ***atol*** → **float**

  default `1e-5`

ENVIRONMENT

- ***temperature*** → **float**

  default `20 C`

  units `K`, `C`

CATALYST

- ***concentration*** → **float**

  default `2.5 g/L`

  units `g/m^3`, `g/L`, `mg/L`

- ***surface*** → **float**

  default `56e3 m^2/g`

  units `m^2/g`, `cm^2/g`

- ***volume*** → **float**

  default `1e-3 m^3`

  units `m^3`, `L`, `cm^3`, `mL`

MOLECULE

- ***name*** → **str**

  default `methylene blue`

- ***composition*** → **chem_formula**

  default `C16H18S1N3`

- ***excess_bonds*** → **int**

  default `14`

- ***molar_volume*** → **float**

  default `226.6 Ang^3/molecule`

  units `Ang^3/molecule`, `nm^3/molecule`, `cm^3/mol`

- ***molar_surface*** → **float**

  default `99.7 Ang^2/molecule`

units `Ang^2/molecule`, `nm^2/molecule`, `m^2/molecule`

- ***diffusion_model*** → **str**

  default `s`

  note s: Stokes (default), wc: Wilke-Chang, hm: Hayduk-Minhas

SYSTEM

- ***concentration_solution*** → **float**

  default `10 mg/L`

  units `molecule/m^3`, `molecule/L`, `mol/m^3`, `mmol/L`, `M`, `mol/L`, `mo/mc`, `g/L`, `mg/L`, `g/m^3`

- ***concentration_surface*** → **float**

  default `0`

  units `molecule/m^2`, `mol/m^2`, `g/m^2`, `mg/m^2`

  note if concentration_surface is not set system is considered in equilibrium (dark)

- ***k_ads*** → **float**

  default `1E-9 m/s`

  unit `m/s`

- ***k_des*** → **float**

  default `1E-3 1/s`

  unit `1/s`

- ***k_reac*** → **float**

  default `1E-2 1/s`

  unit `1/s`

FIT

This section is just active if fit is True

- ***type*** → **str**

  default `dark`

  note dark, reaction or toc

- ***search*** → **str**

  default `relative`

  note minima search absolute, relative or relative_square

  note does not apply to fit type toc

- ***max_step*** → **int**

  default `100`

  note does not apply to fit type toc

MULTI

This section is just active if multi is True

- ***split_model*** → **str**

  default `fragmentation`

  note incremental, fragmentation or excess_bonds

- ***desorption_model*** → **str**

  default `weak`

  note weak or strong

- ***TOC_estimation*** → **str**

  default `all`

  note all or volume

- ***segment_export*** → **str**

  default `mass`

  note mass or molecule_count

MULTI_WEAK

This section is just active if desorption_model is set to weak. Just one value needed if k_des is set.

- ***beta_0*** → **float**

  default `-0.029 1/s`

  unit `1/s`

- ***beta_1*** → **float**

  default `0.8 1/s`

  unit `1/s`

MULTI_STRONG

This section is just active if desorption_model is set to strong

- ***E_0*** → **float**

  default `44.0 kJ/mol`

  unit `kJ/mol`

- ***E_1*** → **float**

  default `3.0 kJ/mol`

  unit `kJ/mol`

- ***alpha_0*** → **float**

  default `1.51e8 1/s`

  unit `1/s`

- ***alpha_1*** → **float**

  default `0.412`

1.4 Examples

1.4.1 Adsorption - Desorption equilibrium

In this first example, the goal is to simulate a system reaching equilibrium in the dark. To run the simulation, we are going to create a configuration file for **pdom** first. After calling **pdom.config** we need to answer a few questions. The parts which require user input are highlighted in yellow.

```
$ pdom.config
ID of the system (avoid spaces): example_ads_des

Should data be fitted to the simulation?
    1: fit
    2: just simulation
Your choice: 2

What kind of simulation?
    1: Adsorption-Desorption
    2: Degradation
    3: TOC
Your choice: 1

How can you identify the initial molecule?
    1: chemID (https://pubchem.ncbi.nlm.nih.gov)
    2: name
Your choice: 1
Molecule: 2764
Found ciprofloxacin (C17H18FN3O3)

What is the catalyst concentration?
  the allowed unis are: g/m^3, g/L, mg/L
Value: 1 g/L

What is the catalyst surface area?
  the allowed unis are: m^2/g, cm^2/g
Value: 56 m^2/g

What is the overall volume?
  the allowed unis are: m^3, L, cm^3, mL
Value: 0.3 L

How long should the simulation be?
  the allowed unis are: h, min, s
Value: 15 min

What is the adsorption constant?
  the allowed unis are: m/s
Value: 3.7E-8 m/s

What is the desorption constant?
  the allowed unis are: 1/s
Value: 2.0E-2 1/s

What is concentration in the solution?
  the allowed unis are: molecule/m^3, molecule/L, mol/m^3, mmol/L, M, mol/L, mo/mc, g/
↪L, mg/L, g/m^3
```

(continues on next page)

(continued from previous page)

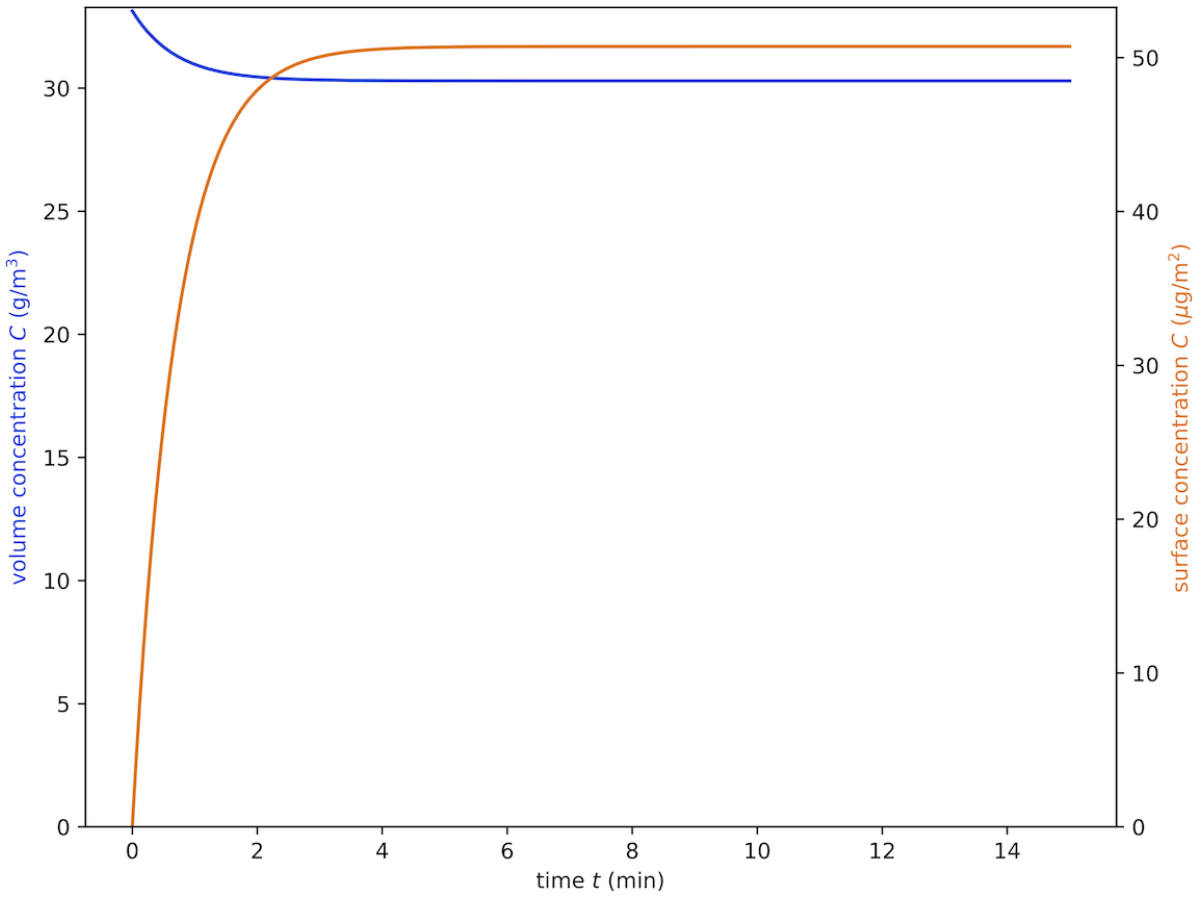
```
Value: 0.1 mmol/L

What is concentration on the surface?
  the allowed unis are: molecule/m^2, mol/m^2, g/m^2, mg/m^2
Value: 0 mol/m^2
```

The resulting file `example_ads_des.ini` is now in your working directory. To start the simulation simply call **`pdom`**:

```
$ pdom example_ads_des.ini
Start calculating single species model
Calculation finished!
Results saved in <your_working_dir>/example_ads_des
```

In the newly created folder *<your_working_dir>*`/example_ads_des`, you find the raw data files with corresponding units and a `plot` of the concentration development over time.



1.4.2 Degradation fit

Using the *single species* model, we fit in this example the reaction constant k_{reac} to experimental data. With **`pdom.config`** we can create `example_reac_fit.ini`. The parts which require user input are highlighted in yellow.

```
$ pdom.config
ID of the system (avoid spaces): example_reac_fit

Should data be fitted to the simulation?
        1: fit
        2: just simulation
Your choice: 1

What kind of experiment was conducted?
        1: Adsorption-Desorption
        2: Degradation
        3: TOC
Your choice: 2

How can you identify the initial molecule?
        1: chemID (https://pubchem.ncbi.nlm.nih.gov)
        2: name
Your choice: 1
Molecule: 2764
Found ciprofloxacin (C17H18FN3O3)

What is the catalyst concentration?
  the allowed unis are: g/m^3, g/L, mg/L
Value: 1.0 g/L

What is the catalyst surface area?
  the allowed unis are: m^2/g, cm^2/g
Value: 56 m^2/g

What is the overall volume?
  the allowed unis are: m^3, L, cm^3, mL
Value: 1 L

How long should the simulation be?
  the allowed unis are: h, min, s
Value: 0.8 h

Which constant is known?
        1: k_ads
        2: k_des
Your choice: 1

What is the adsorption constant?
  the allowed unis are: m/s
Value: 3.7E-8 m/s

What is concentration in the solution?
  the allowed unis are: molecule/m^3, molecule/L, mol/m^3, mmol/L, M, mol/L, mo/mc, g/
↪L, mg/L, g/m^3
Value: 3.8 mg/L
```

The resulting file `example_reac_fit.ini` is now in your working directory. Next, the experimental data needs to be stored as `example_reac_fit.json`. In this example, two time series with the same initial concentrations are used.

Listing 5: example_reac_fit.json

```
{
  "time_series": [
    [
      [0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 20, 25],
      [3.442, 2.847, 2.428, 2.229, 1.949, 1.801, 1.705, 1.535,
        1.315, 1.021, 0.9690, 0.8238, 0.5114, 0.2839, 0.1353]
    ], [
      [0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 20, 25],
      [3.505, 2.876, 2.586, 2.323, 2.242, 2.097, 1.781, 1.681,
        1.461, 1.249, 0.9543, 1.0852, 0.7665, 0.4324, 0.2368]
    ]
  ],
  "time_series_meta": [
    {
      "unit": "min",
      "type": "t"
    }, {
      "unit": "mg/L",
      "type": "solution"
    }
  ]
}
```

With both files prepared **pdom** can be started.

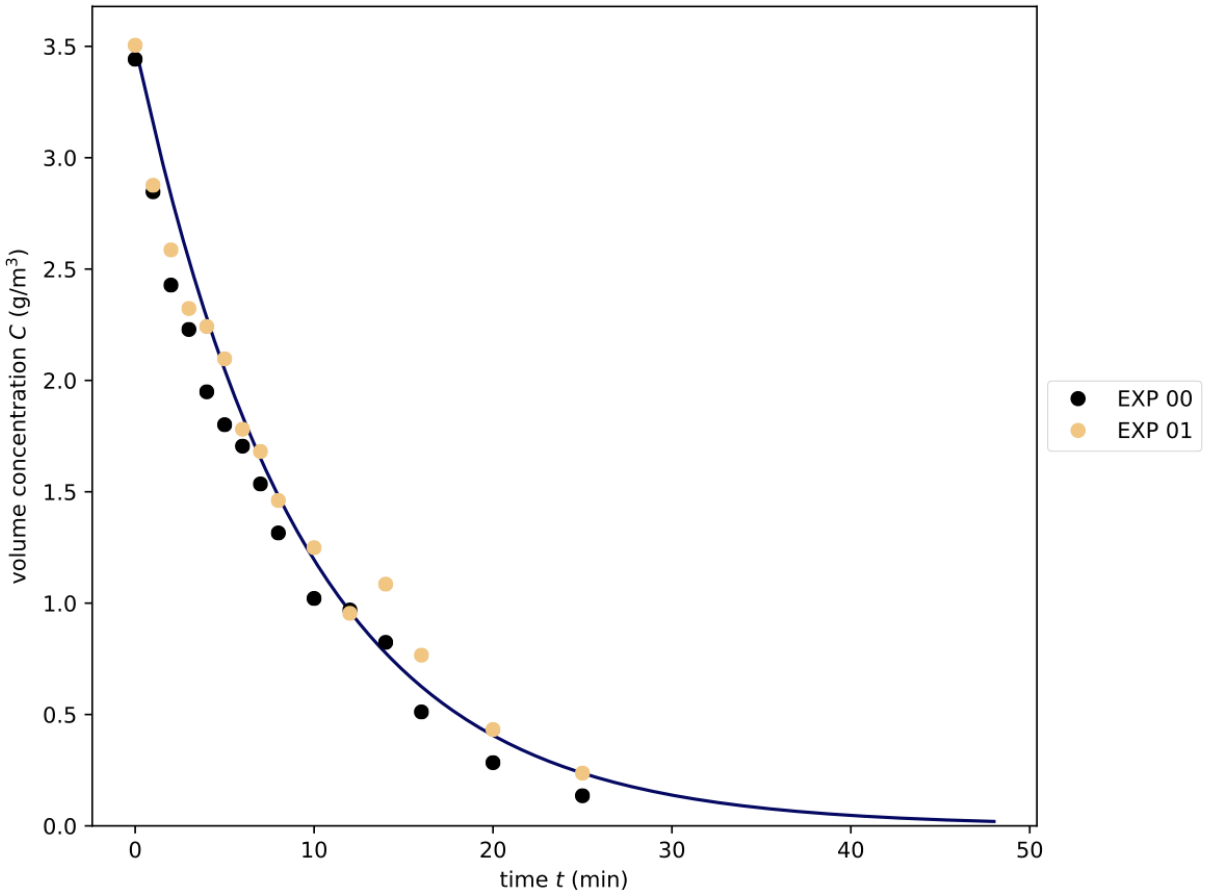
```
$ pdom example_reac_fit.ini --data example_reac_fit.json
Start fitting to data from reaction experiment.
Fit finished after 24 iterations.
    k_ads: 3.700E-08 m/s
    k_des: 2.180E-02 1/s
    k_reac: 1.452E-01 1/s
    error: 3.136E+00
Results saved in <your_working_dir>/example_reac_fit
```

The result of the fit is stored under *<your_working_dir>*/example_reac_fit/fit_single.json.

Listing 6: <your_working_dir>/example_fit_reac/fit_single.json


```
{
    "k_ads": "3.700E-08 m/s",
    "k_des": "2.180E-02 1/s",
    "k_reac": "1.452E-01 1/s",
    "error": "3.136E+00",
    "iterations": 19
}
```


In the same folder, you find the raw data files with corresponding units. The saved `plot` shows the concentration development over time compared to the experimental results.



1.4.3 TOC simulation

Fragmentation

When the overall degradation of an organic molecule should be simulated, one of the *multi-species* models must be used. Three different split models are available in **pdom**:

- incremental
- fragmentation

- excess bonds

To model the generalized split species, the size dependents of the desorption constant k_{ads} must be described. You can either choose *strong* or *weak* dependents for your simulation. We chose a *weak* dependents in this example, due to the fewer needed parameters.

As before the config file `example_toc_sim.ini` can be generated with **`pdom.config`**. Lines with require user input are highlighted in yellow.

```
$ pdom.config
ID of the system (avoid spaces): example_toc_sim

Should data be fitted to the simulation?
        1: fit
        2: just simulation
Your choice: 2

What kind of simulation?
        1: Adsorption-Desorption
        2: Degradation
        3: TOC
Your choice: 3

How can you identify the initial molecule?
        1: chemID (https://pubchem.ncbi.nlm.nih.gov)
        2: name
Your choice: 1
Molecule: 4139
Found methylene blue cation (C16H18N3S+)

What is the catalyst concentration?
  the allowed unis are: g/m^3, g/L, mg/L
Value: 1 g/L

What is the catalyst surface area?
  the allowed unis are: m^2/g, cm^2/g
Value: 56 m^2/g

What is the overall volume?
  the allowed unis are: m^3, L, cm^3, mL
Value: 0.4 L

How long should the simulation be?
  the allowed unis are: h, min, s
Value: 4 h

Which split model should be used?
        1: incremental
        2: fragmentation
        3: excess_bonds (slow)
Your choice: 2

Is the system in equilibrium (dark)?
        1: Yes
        2: No
Your choice: 1

Which model should be used
```

(continues on next page)

(continued from previous page)

```
for the estimation of size dependent k_des?
        1: Strong
        2: Weak
Your choice: 2

What is the adsorption constant?
  the allowed unis are: m/s
Value: 2.1E-8 m/s

What is the desorption constant?
  the allowed unis are: 1/s
Value: 2.85E-2 1/s

What is the reaction constant?
  the allowed unis are: 1/s
Value: 2.85E-1 1/s

What is the value of beta_1?
  the allowed unis are: 1/s
Value: 0.2 1/s

What is concentration in the solution?
  the allowed unis are: molecule/m^3, molecule/L, mol/m^3, mmol/L, M, mol/L, mo/mc, g/
↪L, mg/L, g/m^3
Value: 4 mg/L
```

With the generated config `example_toc_sim.ini` we can start the simulation with **pdom**.

```
$ pdom example_toc_sim.ini
Start calculating multi species model
Calculation finished!
Results saved in <your_working_dir>/example_toc_sim
```

In the folder *<your_working_dir>*/`example_toc_sim`, you find the raw data files with corresponding units and three plots. The first plot is an `overview` with TOC and the concentration of the initial molecule in solution. The other two show concentration developments of the segments in `solution` and on the `surface`.

Incremental

Because **pdom** is based on configuration files, you can simply make a copy of `example_toc_sim.ini` and change the settings quickly for a new simulation. You can alter the `split_model` key in the section `MULTI` to *incremental* for example:

```
...
[MULTI]
split_model = incremental
desorption_model = weak
...
```

This generates the following results:

1.4.4 TOC fit

To fit TOC data with **pdom**, one of the *multi-species* models must be selected:

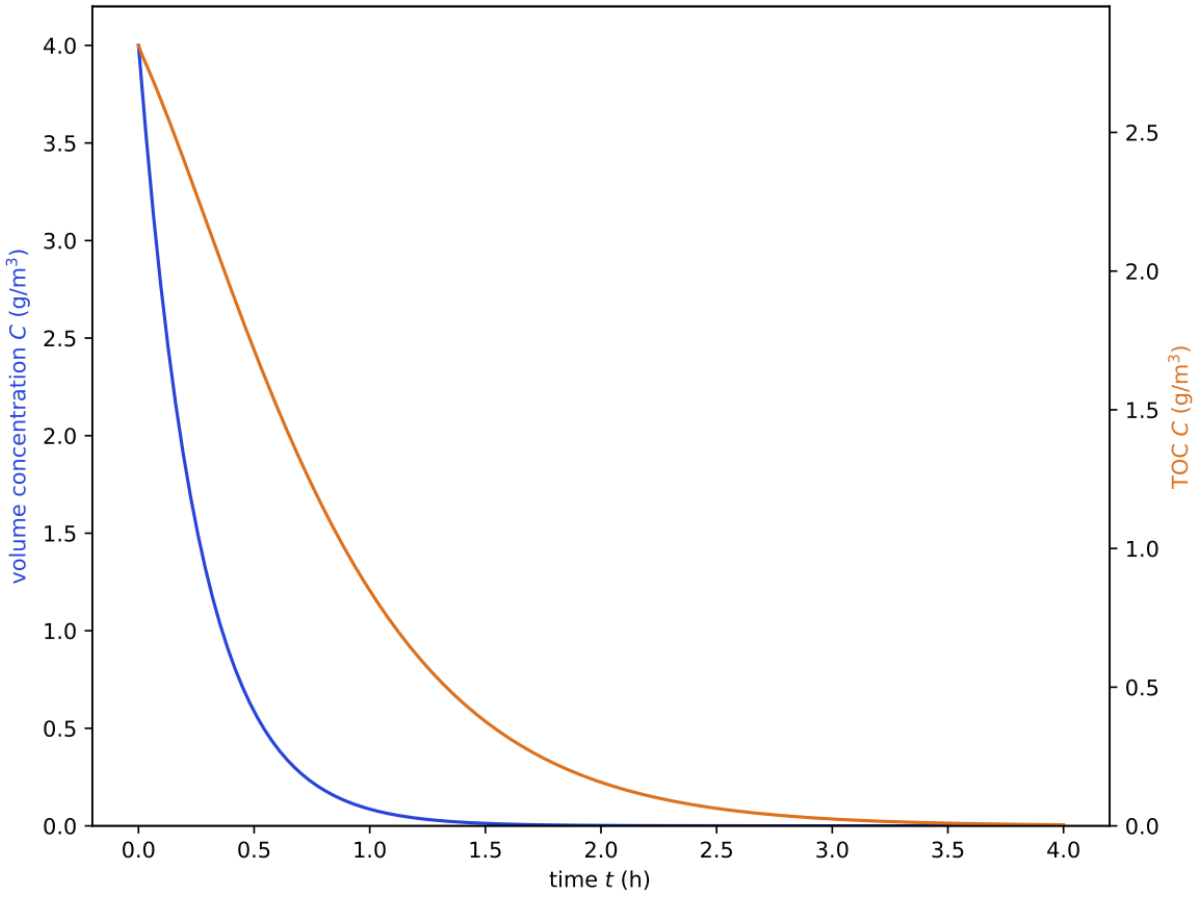


Fig. 1: Development of the concentration in solution and the total organic carbon (TOC) using *fragmentation* split model.

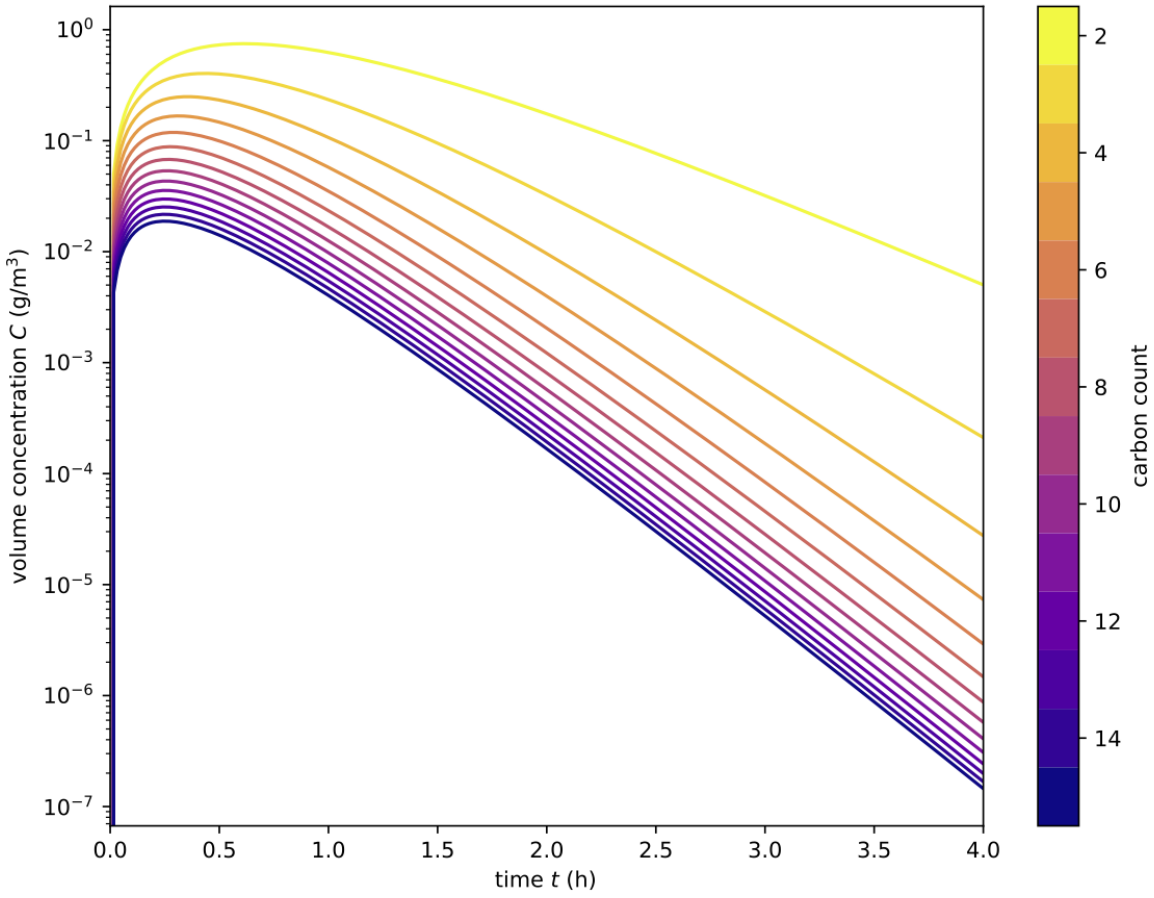


Fig. 2: Development of the segments in solution when using the *fragmentation* split model.

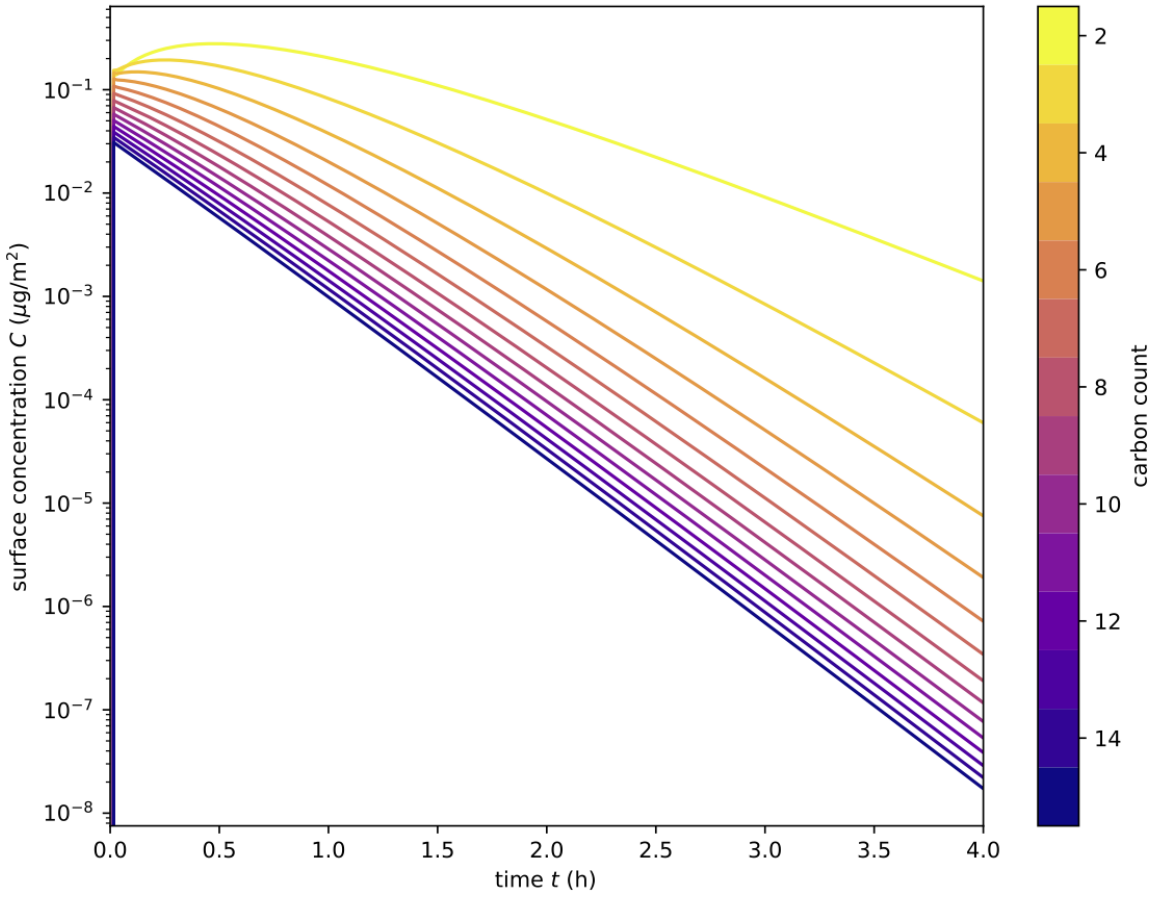


Fig. 3: Development of the segments on the surface when using the *fragmentation* split model.

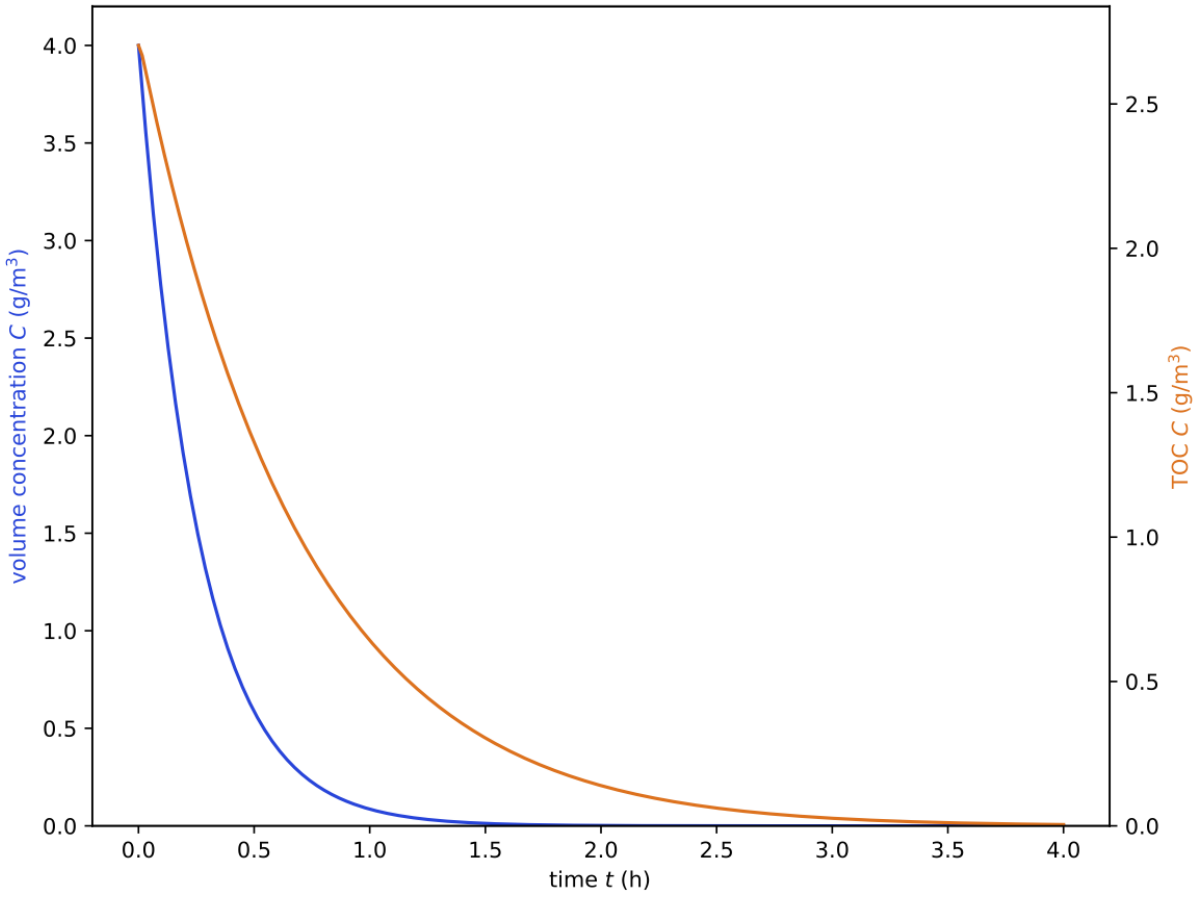


Fig. 4: Development of the concentration in solution and the total organic carbon (TOC) using the *incremental* split model.

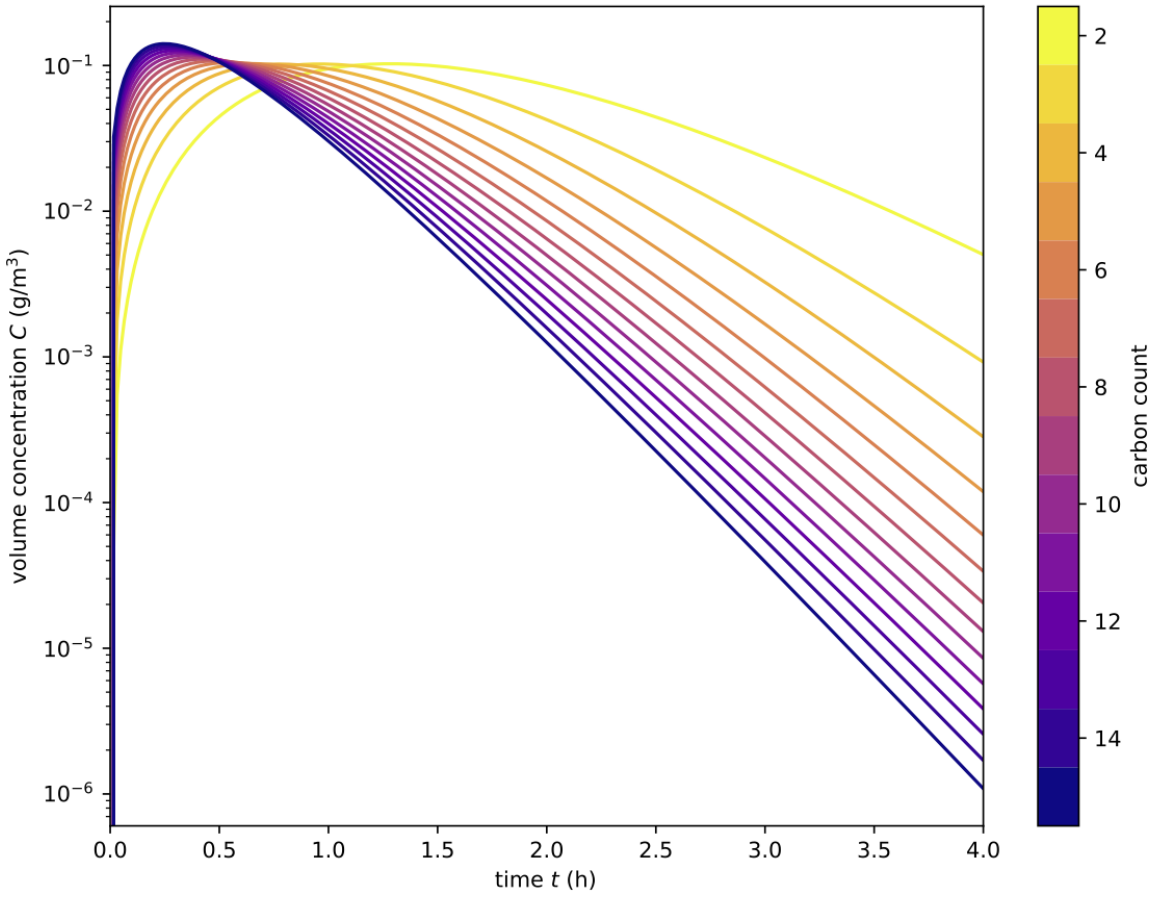


Fig. 5: Development of the segments in solution when using the *incremental* split model.

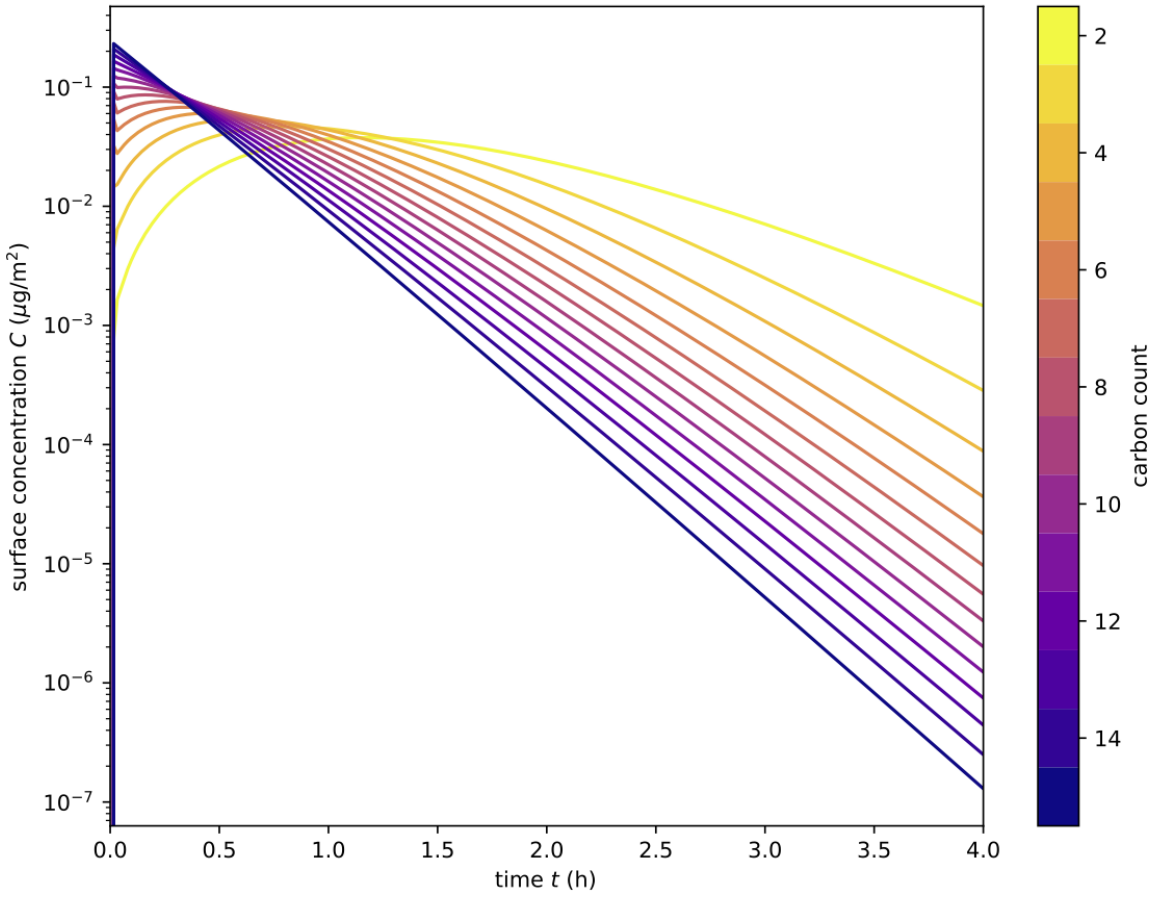


Fig. 6: Development of the segments on the surface when using the *incremental* split model.

- incremental
- fragmentation
- excess bonds

For this example, we will use *incremental*. The model for the desorption constant k_{ads} is always *weak* when a TOC experiment is fitted.

The generation of the config file `example_toc_fit.ini` is again carried out with **`pdom.config`**. Lines with require user input are highlighted in yellow.

```
$ pdom.config
ID of the system (avoid spaces): example_toc_fit

Should data be fitted to the simulation?
        1: fit
        2: just simulation
Your choice: 1

What kind of experiment was conducted?
        1: Adsorption-Desorption
        2: Degradation
        3: TOC
Your choice: 3

How can you identify the initial molecule?
        1: chemID (https://pubchem.ncbi.nlm.nih.gov)
        2: name
Your choice: 1
Molecule: 4139
Found methylene blue cation (C16H18N3S+)

What is the catalyst concentration?
  the allowed unis are: g/m^3, g/L, mg/L
Value: 2.5 g/L

What is the catalyst surface area?
  the allowed unis are: m^2/g, cm^2/g
Value: 56 m^2/g

What is the overall volume?
  the allowed unis are: m^3, L, cm^3, mL
Value: 1 L

How long should the simulation be?
  the allowed unis are: h, min, s
Value: 6 h

Which split model should be used?
        1: incremental
        2: fragmentation
        3: excess_bonds (slow)
Your choice: 1

Is the system in equilibrium (dark)?
        1: Yes
        2: No
Your choice: 1
```

(continues on next page)

(continued from previous page)

```
Which parameter should be fitted?
        1: k_reac
        2: beta_1
Your choice: 2

What is the adsorption constant?
  the allowed unis are: m/s
Value: 3.0E-9 m/s

What is the desorption constant?
  the allowed unis are: 1/s
Value: 6.8E-3 1/s

What is the reaction constant?
  the allowed unis are: 1/s
Value: 6.8E-2 1/s

What is concentration in the solution?
  the allowed unis are: molecule/m^3, molecule/L, mol/m^3, mmol/L, M, mol/L, mo/mc, g/
↪L, mg/L, g/m^3
Value: 0.069 mmol/L
```

After the config is generated, the experimental `data set` is created. In this example, values published by Houas (2001) [Hou01] will be used.

Listing 7: example_reac_fit.json

```
{
  "time_series": [
    [0, 60, 120, 180, 360],
    [12.6, 8.8, 6.78, 4.1, 2.77]
  ],
  "time_series_meta": [
    {
      "unit": "min",
      "type": "t"
    }, {
      "unit": "mg/L",
      "type": "toc"
    }
  ]
}
```

With both files prepared, **pdom** can be started.

```
$ pdom example_toc_fit.ini --data example_toc_fit.json
Start fitting to toc
   Iteration     Total nfev        Cost      Cost reduction    Step norm     ␣
↪Optimality
       0              1         2.0681e-02                                    3.05e+00
       1              2         4.6195e-03      1.61e-02       1.00e-01       6.13e-01
       2              3         2.7331e-03      1.89e-03       6.05e-02       5.52e-02
       3              4         2.7079e-03      2.52e-05       8.61e-03       8.37e-04
       4              5         2.7079e-03      1.79e-08       1.39e-04       1.54e-04
       5              6         2.7079e-03      1.99e-09       2.55e-05       5.45e-04
```

(continues on next page)

(continued from previous page)

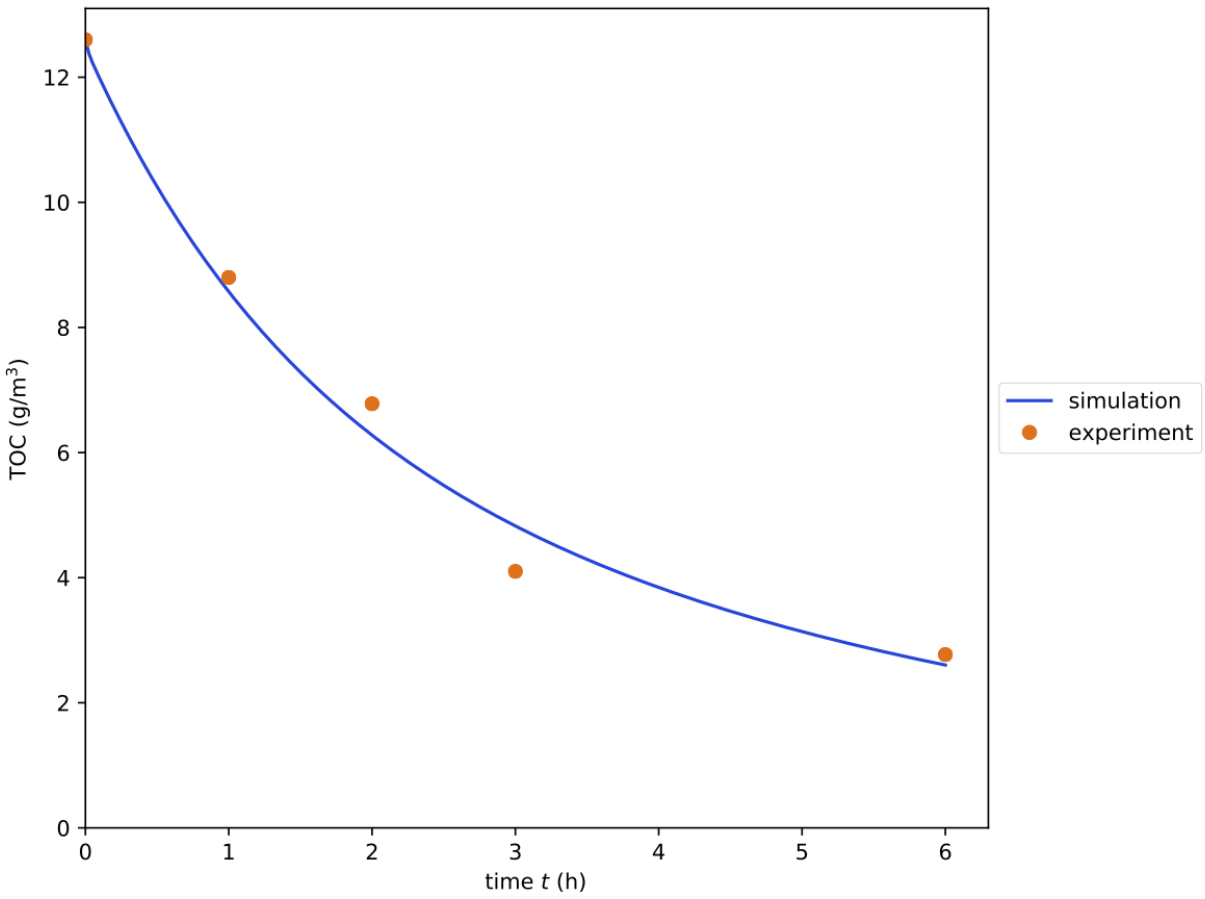
```
`xtol` termination condition is satisfied.
Function evaluations 6, initial cost 2.0681e-02, final cost 2.7079e-03, first-order␣
↪optimality 5.45e-04.
Fit finished
    k_ads: 3.000E-09 m/s
    k_des: 6.800E-03 1/s
    k_reac: 6.800E-02 1/s
    beta_0: -1.003E-02 1/s
    beta_1: 2.693E-01 1/s
    error: 4.657E-02
Results saved in <your_working_dir>/example_toc_fit
```

The result of the fit is stored under *<your_working_dir>*/example_toc_fit/fit_toc.json.

Listing 8: <your_working_dir>/example_toc_fit/fit_toc.json

```
{
    "k_ads": "3.000E-09 m/s",
    "k_des": "6.800E-03 1/s",
    "k_reac": "6.800E-02 1/s",
    "beta_0": "-1.003E-02 1/s",
    "beta_1": "2.693E-01 1/s",
    "sd_error": "4.657E-02"
}
```

In the same folder, you find the raw data files with corresponding units. The saved `plot` shows the TOC development over time compared to the experimental results.



1.5 Source documentation

1.5.1 pdom.Simulate

class pdom.**Simulate**(*config_file*, *out_folder=None*, *data_file=None*, *overwrites=None*, *resolution=250*)

Class to simulation different degradation models

Parameters

- **config_file** (*str, Path*) – .ini file to load
- **out_folder** (*str, Path, optional*) – folder to save the results
- **data_file** (*str, Path, optional*) – .json file containing experimental data
- **overwrites** (*dict, optional*) – overwrite settings from the config file
- **resolution** (*int, optional*) – time resolution for data export

cfg = None

the simulation configuration created with pdom.data.Parameter from the config file.

export_t = None

time steps used for export - filled after *run()*

rhs_bonds(*t*, *N_flat*, *k*, *N_shape*)
Right-hand site for multi species model *excess_bonds*.

Calculate the derivative for a given concentration profile.

Parameters

- **N_flat** (*ndarray*) – number of molecules | 3-dim flattened
- **t** (*float*) – time (not used)
- **k** (*ndarray*) – simulation constants (k_{ads}, k_{des}, k_{reac})
- **N_shape** (*tuple*) – original shape of N

- first dimension of N corresponds to the number of Carbon atoms `carbon_count=index+1`
- second dimension of N corresponds to the number of excess bonds `excess_bonds=index`
- `max(excess_bonds) < max(carbon_count)`
- `N[:, :, 0]` is the number of molecules on the surface
- `N[:, :, 1]` is the number of molecules in solution
- the simulation constants must have the shape `(3, max(excess_bonds)+1, max(carbon_count))`

Returns first derivative

Return type ndarray

rhs_multi(*t*, *N_flat*, *k*, *N_shape*)
Right-hand site for multi species models *fragmentation* and *incremental*.

Calculate the derivative for a given concentration profile.

Parameters

- **N_flat** (*ndarray*) – number of molecules | 2-dim flattened
- **t** (*float*) – time (not used)
- **k** (*ndarray*) – simulation constants (k_{ads}, k_{des}, k_{reac})
- **N_shape** (*tuple*) – original shape of N

- first dimension of N corresponds to the number of Carbon atoms `carbon_count=index+1`
- `N[:, 0]` is the number of molecules on the surface
- `N[:, 1]` is the number of molecules in solution
- the simulation constants must have the shape `(3, max(carbon_count))`

Returns first derivative

Return type ndarray

rhs_single(*t*, *N*, *k*)
Right-hand site for single species model.

Calculate the derivative for a given concentration profile.

Parameters

- **N** (*ndarray*) – number of molecules | 1-dim
- **t** (*float*) – time (not used)
- **k** (*ndarray*) – simulation constants (k_{ads}, k_{des}, k_{reac})

- `N[0]` is the number of molecules on the surface
- `N[1]` is the number of molecules in solution

Returns first derivative

Return type ndarray

run()
Runs either a simulation or performs a parameter fit based on the information stored in *cfg*. The results are saved to the file system.

t = None
time steps used by the solver - filled after *run()*

1.5.2 pdom.data.Parameter

class pdom.**Parameter**(*config_file*, *data_file=None*, *overwrites=None*)
Collection of constants, parameters and helper functions.

This class loads the configuration and handles experimental data. Provide functions to convert data into pdom base units and alter settings.

Parameters

- **config_file** (*str, Path*) – .ini file to load
- **data_file** (*str, Path, optional*) – .json file containing experimental data
- **overwrites** (*dict, optional*) – overwrite settings from the config file

avogadro = 6.02214129e+23
N_A - Avogadro constant `[1/mol]`

static get_diffusion_constant_air(*mol_volume*, *molar_weight*, *temperature=293.15*, *pressure=101325.0*)
Diffusion constant in air depending on molar volume and weight Based on the FSG model with constants from the Handbook of chemical property estimation methods.

Source Fuller (1966) [FSG66], Tucker & Nelken (1982) [TN82]

Parameters

- **mol_volume** (*float*) – molar volume [cm^3/mol]
- **molar_weight** (*float*) – molar weight [g/mol]
- **temperature** (*float, int, optional*) – temperature [K]
- **pressure** (*float, optional*) – pressure [Pa]

Returns D - diffusion constant [cm^2/s]

Return type float

classmethod get_diffusion_constant_water(*mol_volume*, *temperature=293.15*, *model='s'*)

Diffusion constant in water depending on molar volume. Based on models from either Stokes, Wilke-Chang, or Hayduk-Minhas.

Source Wilke & Chang (1955) [WC55], Hayduk & Minhas (1982) [HM82]

Parameters

- **mol_volume** (*float*) – molar volume [cm^3/mol]
- **temperature** (*float, int, optional*) – temperature [K]
- **model** (*str, optional*) – model abbreviation | "s" Stokes (default), "wc" Wilke-Chang, "hm" Hayduk-Minhas

Returns D - diffusion constant [m^2/s]

Return type float

classmethod get_molecular_weight(*composition*)

Calculate the weight of a molecule from its chemical formula.

Parameters **composition** (*dict*) – key - atomic symbol | value - atom count

Returns molecular weight [u]

Return type float

k_boltzmann = 1.3806488e-23

k_B - Boltzmann constant `[Nm/K]`

light_speed = 2997927458

c - light speed `[m/s]`

planck = 6.62606957e-34

h - Planck constant `[Ws^2]`

static scale_ten(*value*)

Decimal scaling

Parameters **value** (*int, float*) – input value

Returns scale factor, prefix

Return type tuple

static scale_time(*value*)

Temporal scaling

Parameters **value** (*int, float*) – input value

Returns scale factor, prefix

Return type tuple

standard_atomic_weight = {'Ac': 225.027, 'Ag': 107.87, 'Al': 26.982, 'Am': 241.061

Mapping element symbol to standard corresponding atomic weight.

Source "Atomic weights of the elements 2013" (IUPAC) [MCB+16]

Returns standard atomic weight [u]

Return type dict

classmethod unit_convert(*value*, *unit*)

Convert into `pdom` base units

Note Base units are `s`, `1/s`, `m`, `m/s`, `W/m^2`, `K`, `g/m^3`, `m^2/g`, `m^3`, `cm^3/mol`, `m^2/molecule`, `kJ/mol`

Parameters

- **value** (*float, int*) –
- **unit** (*str*) –

Returns converted value

Return type float

unit_convert_concentration (*value*, *unit*, *cfg=None*)
Concentration to number of molecules

Note A simulation config can be passed, to use alternative parameters, for example to convert TOC.

Parameters

- **value** (*float, int*) – value
- **unit** (*str*) – unit
- **cfg** (*dict, optional*) – simulation config

Returns N - number of molecules [1]

Return type float

static update_multi_k (*config*)
Update the config for multi species models

This needs to be called after a basic setting is changed in the config, to update parameters depending on species size.

Parameters **config** (*dict*) – simulation config

Returns simulation config

Return type dict

van_der_waals_radii = {'Ac': 2.0, 'Ag': 1.72, 'Al': 2.0, 'Am': 2.0, 'Ar': 1.88, '
Mapping element symbol to corresponding Van der Waals radius.

Note Radii that are not available in either of the sources have RvdW = 2.00.

Source "van der Waals Volumes and Radii" by Bondi (1964) [Bon64], the value for H is taken from Rowland & Taylor (1996) [RT96]

Returns Van der Waals radii [Ang]

Return type dict

viscosity_water = <scipy.interpolate.interpolate.interp1d object>
A smooth interpolation from 274 to 363 Kelvin. Viscosity is retuned in mPa*s | centipoise | (mN/m^2)*s.

Source Wikibooks - Stoffdaten Wasser

1.5.3 pdom.Molecule

class pdom.**Molecule** (*identifier*, *properties*, *save=True*)
This class should not be initialised directly. Use one of the following class methods instead: *from_chem_id()*, *from_folder()*, from_inchi(), from_inchi_key(), from_iupac_name(), *from_name()*

Parameters

- **identifier** (*dict*) –
  - `name` common name
  - `iupac_name` IUPAC name (can be the same as common name)
  - `chem_id` compound ID from PubChem
  - `inchi` IUPAC International Chemical Identifier (human readable)
  - `inchi_key` IUPAC International Chemical Identifier (hash)
- **properties** (*dict*) –
  - `chem_formula` chemical formula e.g. 'C10H22'
  - `chem_formdic` chemical formula as dict
  - `mol_weight` molecular weight in g/mol
  - `mol_volume` molecular volume in cm^3/mol
  - `mol_surface` largest projected surface area m^2/molecule
  - `excess_bonds` number of bonds in excess of a simple carbon chain
  - `structure_3d` atom position list
- **save** (*bool*) – save data to the user cache

classmethod from_chem_id(*chem_id*, *name=None*, *inchi_key=None*)

Create *Molecule* instance identified by a chemID from PubChem data.

Note if molecule is cached in `Molecule.db_folder` load from there

Parameters

- **chem_id** (*int*) – compound ID from PubChem
- **name** (*str, optional*) – overwrite compound name
- **inchi_key** (*str, optional*) – IUPAC International Chemical Identifier (to check cache quickly)

Returns *Molecule* instance

classmethod from_folder(*folder*)

create *Molecule* instance from a folder created by *save*

Parameters **folder** (*str, folder*) – molecule folder

Returns *Molecule* instance

classmethod from_name(*name*)

Create *Molecule* instance identified by a name if unique

Note queries `chem_id` and calls *from_chem_id()*

Parameters **name** (*str*) – unique compound name

Returns *Molecule* instance

identifier = None

dict of identifiers as described in *Molecule*

properties = None

dict of properties as described in *Molecule*

save (*folder=None*, *name=None*)
saves the molecule information to disk, can be loaded with *from_folder()*

Note If called without parameters the molecule is saved in an appropriate cache folder `Molecule.db_folder`.

Parameters

- **folder** (*str, Path*) – parent folder
- **name** (*str*) – molecule folder name (default INCHI key)

structure_3d (*rotated=True*)
Returned 3d structure of the molecule

Parameters **rotated** (*bool*) – if `True` the structure is rotated to cover the maximum surface in the xy space

Returns 3D structure (symbol, x, y, z)

Return type list

1.5.4 pdom.export

Helper functions to export simulation results.

`pdom.export.`**fit_dark** (*cfg*, *result*, *result_raw*)
Export function for fit: *Adsorption - Desorption* experiment in the dark.

Parameters

- **cfg** (*dict*) – simulation config
- **result** (*tuple*) – fit_t, fit_y, k_ads, k_des
- **result_raw** – raw fit function result

Result

- plot with fit and initial data points `fit_dark.pdf`
- space separated data file containing the fit `fit_dark-values_calculated.txt`
- space separated data file containing initial data points `fit_dark-values_experiment.txt`
- .json with the fitted parameters `fit_dark.json`
- .txt files with corresponding units (LaTeX formatted)

`pdom.export.`**fit_single** (*cfg*, *result*, *result_raw*)
Export function for fit: *single species* degradation model.

Parameters

- **cfg** (*dict*) – simulation config
- **result** (*tuple*) – fit_t, fit_y, k_ads, k_des, k_reac
- **result_raw** – raw fit function result

Result

- plot with fit and initial data points `fit_single.pdf`
- space separated data file containing the fit `fit_single-values_calculated.txt`

- space separated data file containing initial data points `fit_single-values_experiment.txt`
- .json with the fitted parameters `fit_single.json`
- .txt files with corresponding units (LaTeX formatted)

`pdom.export.`**`fit_toc`**(*cfg*, *sd_error*, *t*, *fit*)
Export function for fit: *TOC*.

Parameters

- **`cfg`** (*dict*) – simulation config
- **`sd_error`** (*float*) – standard derivation error
- **`t`** (*ndarray*) – fit time series
- **`fit`** (*ndarray*) – fit toc series

Result

- plot with fit and initial data points | `fit_toc.pdf`
- space separated data file containing the fit | `fit_toc-values_calculated.txt`
- space separated data file containing initial data points | `fit_toc-values_experiment.txt`
- .json with the fitted parameters | `fit_toc.json`
- .txt files with corresponding units (LaTeX formatted)

`pdom.export.`**`multi_species`**(*cfg*, *t*, *N_surf*, *N_vol*)
Export function for *incremental* and *fragmentation* multi species model simulations.

Parameters

- **`cfg`** (*dict*) – simulation config
- **`t`** (*ndarray*) – fit time series
- **`N_surf`** (*ndarray*) – number of molecules on the surface
- **`N_vol`** (*ndarray*) – number of molecules in solution

Result

- plot with concentration development in solution and TOC `c_volume-toc.pdf`
- plot of the segments in solution and on the surface `volume_segments.pdf`, `solution_segments.pdf`
- space separated data file containing the concentrations in solution `multi_species-values_volume.txt`
- space separated data file containing the concentrations on the surface `multi_species-values_surface.txt`
- .txt files with corresponding units (LaTeX formatted)

`pdom.export.`**`multi_species_bonds`**(*cfg*, *t*, *N_surf_detail*, *N_vol_detail*)
Export function for *incremental* and *fragmentation* multi species model simulations.

Parameters

- **`cfg`** (*dict*) – simulation config
- **`t`** (*ndarray*) – fit time series

- **N_surf_detail** (*ndarray*) – number of molecules on the surface
- **N_vol_detail** (*ndarray*) – number of molecules in solution

Result

- same files as *multi_species()*
- plot of the average excess bond count `excess_bonds.pdf`
- space separated data file containing the average number of excess bonds `multi_species-excess_bonds`

`pdom.export.`**single_species**(*cfg*, *t*, *N_surf*, *N_vol*)

Export function single species model simulations.

Parameters

- **cfg** (*dict*) – simulation config
- **t** (*ndarray*) – fit time series
- **N_surf** (*ndarray*) – number of molecules on the surface
- **N_vol** (*ndarray*) – number of molecules in solution

Result

- plot with concentration development in solution and on the surface `single_species.pdf`
- space separated data file containing the concentrations in solution `single_species-values_volume.txt`
- space separated data file containing the concentrations on the surface `single_species-values_surface.txt`
- .txt files with corresponding units (LaTeX formatted)

1.6 Bibliography

CHAPTER 2

Citation

If you use `pdom` in your work please cite the paper describing the basic models [EBT+15].

CHAPTER 3

Index

- genindex
- search

Bibliography

[Bon64] A. Bondi. van der Waals Volumes and Radii. *The Journal of Physical Chemistry*, 68(3):441–451, 1964. doi:10.1021/j100785a001.

[EBT+15] Hagen Eckert, Manfred Bobeth, Sara Teixeira, Klaus Kühn, and Gianaurelio Cuniberti. Modeling of photocatalytic degradation of organic components in water by nanoparticle suspension. *Chemical Engineering Journal*, 261:67–75, 2015. doi:10.1016/j.cej.2014.05.147.

[FSG66] Edward N. Fuller, Paul D. Schettler, and J. Calvin Giddings. New method for prediction of binary gas-phase diffusion coefficients. *Industrial & Engineering Chemistry*, 58(5):18–27, 1966. doi:10.1021/ie50677a007.

[HM82] W. Hayduk and B. S. Minhas. Correlations for prediction of molecular diffusivities in liquids. *The Canadian Journal of Chemical Engineering*, 60(2):295–299, 1982. doi:10.1002/cjce.5450600213.

[Hou01] Ammar Houas. Photocatalytic degradation pathway of methylene blue in water. *Applied Catalysis B: Environmental*, 31(2):145–157, 2001. doi:10.1016/S0926-3373(00)00276-9.

[MCB+16] Juris Meija, Tyler B. Coplen, Michael Berglund, Willi A. Brand, Paul De Bièvre, Manfred Gröning, Norman E. Holden, Johanna Irrgeher, Robert D. Loss, Thomas Walczyk, and Thomas Prohaska. Atomic weights of the elements 2013 (IUPAC Technical Report). *Pure and Applied Chemistry*, 88(3):265–291, 2016. doi:10.1515/pac-2015-0305.

[RT96] R. Scott Rowland and Robin Taylor. Intermolecular Nonbonded Contact Distances in Organic Crystal Structures: Comparison with Distances Expected from van der Waals Radii. *Journal of Physical Chemistry*, 100(18):7384–7391, 1996. doi:10.1021/jp953141+.

[TN82] William A. Tucker and Leslie H. Nelken. Diffusion Coefficients in Air and Water. In Warren J. Lyman, William F. Reehl, and David H. Rosenblatt, editors, *Handbook of chemical property estimation methods : environmental behavior of organic compounds*. McGraw-Hill, 1982. URL: https://archive.org/details/handbookofchemic0000lyma.

[WC55] C. R. Wilke and Pin Chang. Correlation of diffusion coefficients in dilute solutions. *AIChE Journal*, 1(2):264–270, 1955. doi:10.1002/aic.690010222.

Python Module Index

p

Index