Chemical Kinetics Library

Harvard University

CS 207 project

Hongxiang Qiu
Riddhi Shah
Yijun Shen
Yuyue Wang

December 11, 2017

Contents

Chapter 1

Introduction

Chemical kinetics is the study of rates of chemical processes. Different experiment settings and conditions, such as temperature or concentration of reactants, can influence the speed of chemical reactions as well as the yield of products.

The library *chemkin* we wrote aims to make such calculation in chemical kinetics easier.

1 Reaction Rate Coefficients

There are several kinds of reaction rates, and here we are introducing three types of reaction rate coefficients:

1.1 Constant Reaction Rate Coefficient

$$k_{const} = k$$

1.2 Arrhenius Reaction Rate Coefficient

$$k_{arr} = A \exp\left(-\frac{E}{RT}\right)$$

1.3 Modified Arrhenius Reaction Rate Coefficient

$$k_{modarr} = AT^b \exp\left(-\frac{E}{RT}\right)$$

Note:

- The Arrhenius prefactor A is strictly positive.
- The modified Arrhenius parameter b must be real.
- R=8.314 is the ideal gas constant. It should never be changed (except to convert units)
- The temperature T must be positive (assuming a Kelvin scale)

2 Irreversible Reactions

Consider a system consisting of N species undergoing M irreversible, elementary reactions of the form

$$\sum_{i=1}^{N} \nu'_{ij} \mathcal{S}_i \text{ -> } \sum_{i=1}^{N} \nu''_{ij} \mathcal{S}_i, \qquad \text{j} = 1, \ldots, \text{M}$$

the rate of change of specie i (**the reaction rate**) can be written as

$$f_i = \sum_{j=1}^{M} \nu_{ij} \omega_j, \qquad \text{i} = 1, \ldots, \text{N}$$

where **the progress rate** for each reaction is given by

$$\omega_j = k_j \prod_{i=1}^{N} x_i^{\nu'_{ij}}, \qquad \text{j} = 1, \ldots, \text{M}$$

and k_j is **the forward reaction rate coefficient**.

3 Reversible Reactions

In reality, it is often the case that the products can react and produce the reactants. This is called a reversible reaction. It may have the form

$$\sum_{i=1}^{N} \nu'_{ij} \mathcal{S}_i \rightleftharpoons \sum_{i=1}^{N} \nu''_{ij} \mathcal{S}_i \qquad j = 1, \ldots, M$$

where $\rightleftharpoons$ indicts that the forward and backward reactions occur.

The total progress rate is now given by

$$\omega_j = k_j^{(f)} \prod_{i=1}^{N} x_i^{\nu'_{ij}} - k_j^{(b)} \prod_{i=1}^{N} x_i^{\nu''_{ij}}, \qquad j = 1, \ldots, M.$$

where $k_j^{(f)}$ is **the forward reaction rate coefficient**, and $k_j^{(b)}$ is **the backward reaction rate coefficient**.

For an elementary reaction (and only elementary reactions), we have

$$k_j^{(b)} = \frac{k_j^{(f)}}{k_j^e}, \qquad j = 1, \ldots, M$$

where k_j^e is **the equilibrium coefficient** for reaction j.

The final expression for **the equilibrium coefficient** is

$$k_j^e = \left(\frac{p_0}{RT}\right)^{\gamma_j} \exp\left(\frac{\Delta S_j}{R} - \frac{\Delta H_j}{RT}\right), \qquad j = 1, \ldots, M$$

where $\gamma_j = \sum_{i=1}^{N} \nu_{ij}$ and p_0 is the pressure of the reactor (take it to be 10^5 Pa).

We call ΔS_j **the entropy change** of reaction j and ΔH_j **the enthalpy change** of reaction j. We have:

$$\Delta S_j = \sum_{i=1}^{N} \nu_{ij} S_i \quad \text{and} \quad \Delta H_j = \sum_{i=1}^{N} \nu_{ij} H_i, , \qquad j = 1, \ldots, M.$$

To calculate these terms, we use **the NASA Polynomials**.

The 7**th order NASA polynomials** are given by

$$\frac{C_{p,i}}{R} = a_{i1} + a_{i2}T + a_{i3}T^2 + a_{i4}T^3 + a_{i5}T^4$$

$$\frac{H_i}{RT} = a_{i1} + \frac{1}{2}a_{i2}T + \frac{1}{3}a_{i3}T^2 + \frac{1}{4}a_{i4}T^3 + \frac{1}{5}a_{i5}T^4 + \frac{a_{i6}}{T}$$

$$\frac{S_i}{R} = a_{i1}\ln(T) + a_{i2}T + \frac{1}{2}a_{i3}T^2 + \frac{1}{3}a_{i4}T^3 + \frac{1}{4}a_{i5}T^4 + a_{i7}$$

for $i = 1, \ldots, N$.

The coefficients a_{ik} are called **the NASA Polynomial Coefficients**. They are given in databases for multiple temperature ranges. For example, the NASA polynomial coefficients for H_2 are given as (T in Kelvin)

k	Low Range: $300 < T < 1000$	High Range: $1000 \leq T < 5000$
1	3.33727920	2.344331122
2	$-4.94024731 \times 10^{-5}$	$7.98052075 \times 10^{-3}$
3	$4.99456778 \times 10^{-7}$	$-1.94781510 \times 10^{-5}$
4	$-1.79566394 \times 10^{-10}$	$2.01572094 \times 10^{-8}$
5	$2.00255376E \times 10^{-14}$	$-7.37611761 \times 10^{-12}$
6	$-9.50158922 \times 10^{2}$	$-9.17935173 \times 10^{2}$
7	-3.20502331	$6.83010238 \times 10^{-1}$

Symbol Notes

Symbol	Meaning
$\mathcal{S}_i$	Chemical symbol of specie i
ν'_{ij}	Stoichiometric coefficients of reactants
ν''_{ij}	Stoichiometric coefficients of products
N	Number of species in system
M	Number of elementary reactions
f_i	Rate of consumption or formation of specie i (reaction rate)
ω_j	Progress rate of reaction j
x_i	Concentration of specie i
k_j	Reaction rate coefficient for reaction j

Chapter 2

Installation

1 Dependencies

- numpy
- pandas
- pytest-runner
- flask
- flask-jsonpify
- flask-restful
- h5py
- matplotlib

2 Installation Steps

- Open the terminal.
- Clone the repository: **git clone https://github.com/cs207G6/cs207-FinalProject.git** to your desired directory.
- Change working directory to the root directory of the cloned repository
- Install using **pip install .** or **python setup.py install**

```
$ git clone https://github.com/cs207G6/cs207-FinalProject.git
$ cd cs207-FinalProject/
$ pip install .
```

- If desired, run tests using **python setup.py test**

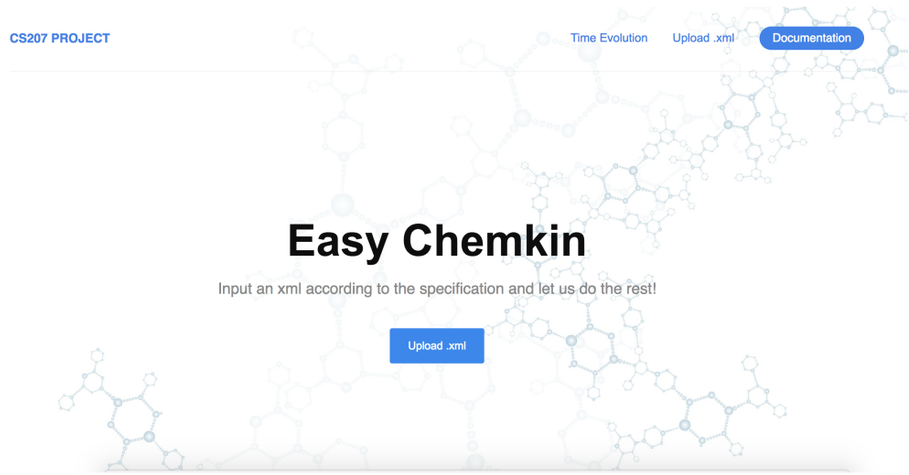
```
$ python setup.py test
```

- To start the web UI, type:
  python -c "import chemkin.webserver; chemkin.webserver.WebServer(8080).start()"

  The terminal will look like this:

```
$ python -c "import chemkin.webserver; chemkin.webserver.WebServer(8080).start()"
/Users/user/Desktop/Github/cs207-FinalProject/chemkin/webserver.py:6: ExtDeprecationWa
  from flask.ext.jsonpify import jsonify
 * Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
```

  Copy the link **http://127.0.0.1:8080/** to your web browser, and you will see the following page:



3 To Contribute to the Development Version

Clone the project using git clone, and then install all dependencies ('pip install .' should take care of this).

Please follow git protocols and conventions (work on a separate branch, etc.).

Chapter 3

Basic Usage and Examples

1 Usage Requirements

In order to use this package you must have an `.xml` of reaction which has the format specified below.

```
<?xml version="1.0"?>

<ctml>

    <phase>
        <speciesArray> H O OH H2 H2O O2 </speciesArray>
    </phase>

    <reactionData id="test_mechanism">

        <!-- Reaction 01  -->
        <reaction reversible="no" type="Elementary" id="
           reaction01">
            <equation>H + O2 =] OH + O</equation>
            <rateCoeff>
                <Arrhenius>
                    <A>3.52e+10</A>
                    <E>7.14e+04</E>
                </Arrhenius>
            </rateCoeff>
            <reactants>H:1 O2:1</reactants>
            <products>OH:1 O:1</products>
        </reaction>

        <!-- Reaction 02 -->
        <reaction reversible="no" type="Elementary" id="
           reaction02">
            <equation>H2 + O =] OH + H</equation>
            <rateCoeff>
                <modifiedArrhenius>
                    <A>5.06e-2</A>
                    <b>2.7</b>
                    <E>2.63e+04</E>
                </modifiedArrhenius>
            </rateCoeff>
            <reactants>H2:1 O:1</reactants>
            <products>OH:1 H:1</products>
        </reaction>

        <!-- Reaction 03 -->
        <reaction reversible="no" type="Elementary" id="
           reaction03">
```

```
            <equation>H2 + OH =] H2O + H</equation>
            <rateCoeff>
                <Constant>
                    <k>1.0e+03</k>
                </Constant>
            </rateCoeff>
            <reactants>H2:1 OH:1</reactants>
            <products>H2O:1 H:1</products>
        </reaction>

    </reactionData>

</ctml>
```

2 Package Usage with Examples

- Get Progress Rate

```
import chemkin.parser
import chemkin.nasa
nasa = chemkin.nasa.NASACoeffs()
# create a data parser class
data_parser = chemkin.parser.DataParser()
# parse the data file and return an instance of ReactionData
    class
reaction_data = data_parser.parse_file('chemkin/example-data/
    rxns.xml', nasa)
progress_rates = reaction_data.get_progress_rate
    ([1,2,3,4,5,6],100)
print(progress_rates)
```

where the arguments of *get_progress_rate* are chemical concentrations for each of the six species, ordered as specified in the *.xml* file ([1,2,3,4,5,6] in the example above) and reaction temperature (100 in the example above).

output:

```
[1.06613928e-26   1.85794997e-09   1.20000000e+04]
```

where each value above is corresponding to the progress rate of each elementary reaction.

- Get Reaction Rate

```
reaction_rates = reaction_data.get_reaction_rate(
    progress_rates)
print(reaction_rates)
```

where the arguments of *get_reaction_rate* is the result of *get_progress_rate* we got above.
output:

```
[1.20000000e+04  -1.85794997e-09  -1.20000000e+04
    -1.20000000e+04 1.20000000e+04  -1.06613928e-26]
```

where each value above is the reaction rate of each species in the order given in the *.xml* file.

- Get Reaction Rate Coefficients

```
ks = reaction_data.get_k(20)
print(ks)
```

where the argument of *get_k* is the temperature (20 in the example above) of the reaction. output:

```
1 [1.15383261e-176    3.35659594e-067    1.00000000e+003]
```

where each value above is the reaction rate coefficient calculated based on the type specified in the *.xml* file.

3 UI Usage

- Upload an `.xml` file in the specified format
- Input species concentrations and temperature
- Click "Plot Reaction Rate" to see how reaction rate changes within the temperature range. The user can specify a desired temperature range by typing in minimum and maximum temperature, then click "confirm".
- Click "Plot Progress Rate" to see how progress rate changes within the temperature range. The user can specify a desired temperature range by typing in minimum and maximum temperature, then click "confirm".
- Click "Get Rates" to see the numerical results of progress rates and reaction rates.

4 WebAPI

All requests and responses should be `json` objects. All responses will include a "status" attribute. When it's "success" then the response will contain valid response data, if it's a "failed" there will be a "reason" attribute containing reason of the failure.

- POST /session
  Request: data: plain text string of the xml document
  Response: id: session id, species: array of species names, equations: array of reaction equations
- POST /rates/<session id>
  Request: * (species names): concentration of the species, _temp: temperature
  Response: progress_rates: progress rates, reaction_rates: reaction rates
- POST /plots/<session id>
  Request: * (species names): concentration of the species, _temp: temperature (for marking on the plot)
  Response: progress_rates: plot for progress rates, reaction_rates: plot for reaction rates
- POST /timeevosession
  Request: data: base64 encoded h5 file content
  Response: id: session id, scenarios: array of scenarios
- GET /timeevo/<session id>/<scenario name>
  Response: plot: base64 encoded plot

Chapter 4

New Feature: UI & Web API

1 Motivation

Chemical kinetics can be intimidating for those who are new to this area, with all the complex equations and coefficients. We want to propose a user-friendly interface where the user can access all our package functionality without needing to deal with the terminal/-command prompt. The users can upload an `.xml` and `.hdf5` file containing information of chemical reactions they are interested in, and we will calculate progress/reaction rates for them as well as provide visualization. We give the user the option to visualize the progress and reaction rate over a range of temperature. The `.hdf5` file is to show temperature over time. We provide visualization as we think plots can help users to visualize the chemical process.

2 Implementation Details

2.1 Plot Generation Module

In this `plot` module, we are making plots of progress rates and reaction rates with respect to a range of temperatures. These plots intend to intuitively show our users how progress rates and reaction rates would change within the range of temperatures they provide. The data point of the current temperature is also marked on the plot as a reference. In this way, the users can get an idea of how their rates would change if they would like to change the reaction temperature.

Specifically, there are three functions in the `plot` module: `range_data_collection`[1], `progress_rate_plot_generation`[2], and `reaction_rate_plot_generation`[3]. [1] takes the parsed `ReactionData` object and other specifications of the plots, and returns all processed data required for generating the plot. [2] and [3] create the plot of progress rates and reaction rates respectively with the data returned from [1] and return a base64-encoded version of the plots that will be decoded shown on the webpage.

The inputs and outputs of each function in the `plot` module is shown below:

1. **range_data_collection**

```
def range_data_collection(user_data, input_concentration,
    lower_T, upper_T, current_T):
    ...
    return temp_range, progress_rates_list,
        reaction_rates_list, current_T, species,
        progress_rates_current, reaction_rates_current,
        equations
```

2. **progress_rate_plot_generation**

```
def progress_rate_plot_generation(temp_range,
    progress_rate_list, current_T, progress_rates_current,
    equations, pic_width, pic_length):
```

```
2       ...
3       return encoded_png
```

3. **reaction_rate_plot_generation**

```
1  def reaction_rate_plot_generation(temp_range,
       reaction_rate_list, current_T, reaction_rates_current,
       species, pic_width, pic_length):
2      ...
3      return encoded_png
```

2.2 Time Evolution Plot Generation Module

We have a `TimeEvo` class that is used to read `hdf5` file, and generate plots in `base64` format. The class will open a `h5` file in read mode, get all subsections (scenarios), and user can call plot with a specified scenario. The class will get 'truth' and 'time' tables (y and x values) and generate a plot using `matplotlib` and export as a `base64` string. We use `base64` because it's simpler to transport using `json`.

2.3 Web Server

We are using `Flask` and `flask-rest` as our framework as they are simple, lightweight, and contains all necessary functionalities. The detailed structure can be found in Chapter 3/WebAPI section. Generally we split the functionalities into different endpoints, and each endpoint is implemented using classes inheriting `flask_restful.Resource`. These classes will parse inputs (both from url and request body) and call relevant modules and return the results in `JSON`. `JSON` is becoming the de facto standard for web apis nowadays.