`chemical-kinetics` User Manual

Divyam Misra Shiyun Qiu Victor Zhao

2017-12-11

Table of Contents

Introduction

The subject of chemical kinetics studies the rates of chemical processes. Analyzing how reaction rates change with respect to reactant abundances, temperature, pressure, and other conditions gives insight into reaction mechanisms.

The present library, `chemical-kinetics`, is a Python 3 library for mathematical modeling of chemical reaction systems. Features include handling systems of elementary reactions, prediction of initial reaction rates, and simulation of reaction progress using deterministic and stochastic methods.

Reaction systems are specified in a standard XML format. The user may choose to use the provided interactive interface or program using the library directly.

Scientific background

A chemical equation describes a chemical reaction, relating the stoichiochetric ratios of reactants and products. For example, the following is the equation for the combustion of hydrogen gas:

$$2\mathrm{H}_{2(g)} + \mathrm{O}_{2(g)} \rightarrow 2\mathrm{H}_2\mathrm{O}_{(l)} \tag{1}$$

A chemical equation may describe an overall net reaction, indicating the ratio of reactants consumed to products produced. On the other hand, such a net reaction may occur by way of several *elementary* reaction steps, each of which can be written as a separate chemical equation. An elementary reaction step describes exactly which chemical species collide to produce which reaction products. A set of elementary reactions that together add up to yield the net reaction represents a *reaction mechanism* describing the exact reactions that lead to products of the overall reaction.

Laboratory and computational experiments studying chemical kinetics have several motivations, including...

- Determining reaction mechanisms; hypothesized mechanisms can be validated and tested.
- Finding rate-determining steps, the reaction step that controls the overall reaction rate.
- Measuring equilibrium: at what concentrations is reaction equilibrium reached and how quickly is that state reached.

Calculating reaction rates

In a system of N chemical species undergoing M *elementary* reactions, the reaction rate of species i, f_i, is computed by

$$f_i = \sum_{j=1}^{M} \nu_{ij}\omega_j, \qquad i = 1, \ldots, N \tag{2}$$

where ω_j is the *progress rate* of the jth reaction and ν_{ij} is the net stoichiometric coefficient for species i in the jth reaction. The progress rate (ω) for each reaction j is computed by

$$\omega_j = k_j^{(f)} \prod_{i=1}^{N} x_i^{\nu'_{ij}} - k_j^{(b)} \prod_{i=1}^{N} x_i^{\nu''_{ij}}, \qquad j = 1, \ldots, M \tag{3}$$

where, for the jth reaction, $k_j^{(f)}$ is the forward reaction rate coefficient and $k_j^{(b)}$ is the corresponding backward reaction rate cofficient. The x_i's represent abundances (usually concentrations) of the chemical species.

Given the particular nature of a chemical reaction, the reaction rate coefficient, k, can be a constant value or exhibit dependence on an activation energy, $E^{\ddagger}$, and reaction temperature, T. In the latter case, the reaction rate coefficient is often modeled using the Arrhenius equation:

$$k = AT^b \exp[-E^{\ddagger}/RT] \tag{4}$$

where A is the pre-exponential factor, b gives the temperature dependent of the pre-exponential factor ($b = 0$ indicating no dependence), and R is the universal gas constant (8.314 J $\mathrm{K}^{-1}\mathrm{mol}^{-1}$).

The forward and reverse reaction rate coefficients for a chemical reaction j are related to each other through the reaction equilibrium constant, K:

$$K_j = \frac{k_j^{(f)}}{k_j^{(b)}}, \qquad j = 1, \ldots, M \tag{5}$$

The equilibrium constant depends on temperature and the thermodynamic properties of the reactants and products in the following way:

$$K_j = \left(\frac{p_0}{RT}\right)^{\gamma_j} \exp\left(\frac{\Delta S_j}{R} - \frac{\Delta H_j}{RT}\right), \qquad j = 1, \ldots, M \tag{6}$$

$$\gamma_j = \sum_{i=1}^{N} \nu_{ij} \tag{7}$$

In particular, the equilibrium constant depends on the entropy and enthalpy change of the reaction, respectively ΔS and ΔH, which are calculated from entropy and enthalpy values of the chemical species: the change in entropy (enthalpy) is the sum of product entropies (enthalpies) minus the sum of reactant entropies (enthalpies).

$$\Delta S_j = \sum_{i=1}^{N} \nu_{ij} S_i \quad \text{and} \quad \Delta H_j = \sum_{i=1}^{N} \nu_{ij} H_i, , \qquad j = 1, \ldots, M \tag{8}$$

Entropy and enthalpy are thermodynamic quantities that are defined as follows, where C_p is the heat capacity at constant pressure.

$$H_i = \int_{T_0}^{T} C_{p,i}\left(T\right)\, \mathrm{d}T, \qquad i = 1, \ldots, N \tag{9}$$

$$S_i = \int_{T_0}^{T} \frac{C_{p,i}\left(T\right)}{T}\, \mathrm{d}T, \qquad i = 1, \ldots, N \tag{10}$$

Instead of determining the heat capacity of each chemical species from first principles, we approximate the heat capacity using the 7th order NASA polynomials, which are polynomial fits for the temperature dependence of the heat capacity and whose straighforward integration yields entropy and enthalpy:

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

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

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

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

Summary of notation

ν'_{ij}: Stoichiometric coefficients of reactants,

ν''_{ij}: Stoichiometric coefficients of products,

ν_{ij}: Net stoichiometric coefficient: $\nu''_{ij} - \nu'_{ij}$

ω_j: Progress rate of reaction j,

x_i: Concentration of species i,

$k_j^{(f)}$: Forward reaction rate coefficient for reaction j,

$k_j^{(b)}$: Backward reaction rate coefficient for reaction j,

K_j: Equilibrium constant for reaction j,

p_0: pressure of the reactor,

ΔS_j: the entropy change of reaction j,

ΔH_j: the enthalpy change of reaction j,

$C_{p,i}$: specific heat at constant pressure for reactant i.

Quantitative modeling of chemical reactions

There are two main methods for modeling the time evolution of a system of chemically reacting species: deterministic modeling, using ordinary differential equations and treating chemical abundances as continuous variables, and stochastic modeling, which accounts for the discrete nature of chemical species and the random nature of reactions. The latter case becomes relevant when chemical abundances are countable, and the time interval between reactions can be modeled as a stochastic variable.

`chemical-kinetics` offers both deterministic and stochastic simulation of reaction systems. Necessary inputs include description of the system of reactions, temperature, and the initial reactant concentrations or abundances. The methodology used for these simulations is described in this section.

Deterministic modeling

Deterministic modeling of system of chemical reactions occurs by numerically integrating reaction rates forward in time. The calculations to obtain instantaneous reaction rates are described in the preceding section.

For deterministic simulations, there are three ODE solvers available: the backward euler method, the backward differentiation formula, and the Runge-Kutta-Fehlberg method. The third method, Runge-Kutta-Fehlberg (rk45), is **not** recommended however, as the equations involved with are usually stiff.

It is recommended that the user sets the step size to be small (eg. 0.01) to achieve higher simulation accuracy when using the backward euler method and the backward differentiation formula.

Note, if numerical integration leads to negative chemical species conentrations, such concentrations will be truncated to 0, since chemical concentrations cannot be negative.

Stochastic modeling

Stochastic simulation of chemical abundances uses the Gillespie stochastic simulation algorithm. The principle behind the algorithm is that waiting times between reaction events are exponentially distributed; thus the time until the next reaction is drawn from an exponential distribution (the simulation is advanced in time by this amount). The particular reaction event that occurs is randomly selected from among the possible reactions in proportion to their probabilities.

The following is a summary of the simulation process:

1. Generate two random numbers r_1, r_2 uniformly distribution in (0,1).
2. Compute the propensity function of each reaction and compute

$$\alpha_0 = \sum_{i=1}^{q} \alpha_i(t)$$

3. Compute the time interval until the next chemical reaction via

$$\tau = \frac{1}{\alpha_0} \ln[1/r_1]$$

4. Compute which reaction occurs. Find j such that

$$r_2 \geq \frac{1}{\alpha_0} \sum_{i=1}^{j-1} \alpha_i$$

$$r_2 < \frac{1}{\alpha_0} \sum_{i=1}^{j} \alpha_i$$

5. The jth reaction takes place. Update the numbers of chemical species accordingly.
6. Continue simulation by returning to step 1.

To elaborate, the propensity function for a chemical reaction describes the probability of a particular reaction happening.

For instance, consider the following bimolecular reaction:

$$A + A \rightarrow B; \quad rate\ k \qquad (1)$$

The quantity of species `A` present at time `t` is given by `A(t)`. The propensity function for the above reaction is $A(t)(A(t)-1)k$. In a more complex reaction system, each elementary reaction will have its own propensity, and it is necessary to calculate the propensity of all elementary reactions in order to find the total "propensity" for a reaction to occur, in order to draw time τ until the next reaction from an exponential distribution.

Installation

A latest version of `chemical-kinetics` can downloaded from Github here.

Installation instructions

Obtain the latest version from github, change to the directory, and install using pip.

```
git clone https://github.com/cs207-2017-group13/cs207-FinalProject.git
cd cs207-FinalProject
pip3 install ./
```

Testing

Users can run the test suite by calling pytest from the main directory, e.g.

```
pytest --cov=src
```

Dependencies

Our package depends on `scipy`, `numpy`, and `matplotlib` packages.

Contributing to the development version

`chemical-kinetics` is hosted on Github and gladly accepts contributions.

Basic Usage and Library Class Descriptions

In this section, usage of `chemical-kinetics` is demonstrated. The various classes built into the library are enumerated.

The `chemical-kinetics` library top-level module is `chemkin`. All classes and functions are contained in `chemkin`.

Users have two ways of using `chemical-kinetics` currently: through an interface provided by the executable `process_reaction_system` and by accessing the classes and functions of the library directly. For an example of the latter, please see `examples/reversible_reaction_simulation.py`.

Note that input data on reaction systems must be provided in the standard CTML XML format. See `examples` as well as `tests` for examples of the XML format.

`process_reaction_system` executable: Menu-based user interface

Once the library is installed, `process_reaction_system` executable is available as an interface to our library. In addition to using the menu-based interface, it is also an option to start an IPython notebook at any time to continue working programmatically.

Example:

```
process_reaction_system <path to input XML file>
```

`chemkin.chemkin.XMLReader`: Read and parse XML input file

The user should have an XML input file containing all the chemical reactions. `XMLReader` will read and parse the input file. The user only needs to use the `get_reaction_systems` method. `XMLReader` creates a `ReactionSystem` instance for each system of reactions in the XML input file. Each `ReactionSystem` instance holds a list of `ElementaryReaction` instances, each one representing an elementary reaction.

Example:

```
import chemkin.chemkin as chemkin
reader = chemkin.XMLReader("tests/rxns.xml")
reaction_systems = reader.get_reaction_systems()
```

`reaction_systems` is a list containing multiple `ReactionSystem` instances. The length of `reaction_systems` is the number of reaction systems.

`chemkin.chemkin.ElementaryReaction`: Represents elementary reactions

Takes a dictionary of properties from the `XMLReader` class for each elementary reaction. This class calculates the rate coefficients and returns stoichiometric coefficients. Instances of this class are contained by the `ReactionSystem` class.

This class has thee public methods, two private methods, and a special method:

- `__repr__()`: Returns representation of the class
- `get_info()`: Returns a `str` with reaction information.
- `get_reactants()`: Returns a dictionary with reactants as key and the stoichiometric coeff of reactants as value for each elementary reaction.
- `get_products()`: Returns a dictionary with products as key and the stoichiometric coeff of products as value for each elementary reaction.
- `calculate_rate_coefficient(T)`: Calculates and returns the rate coeffiecient of the reaction based on the type of rate coefficient required.
- `_constant_rate(k=1.0)`: Returns a constant reaction rate coefficient with default value as 1.0
- `_k_arrhenius(A, E, T, b=0.0, R=8.314)`: Returns a reaction rate coefficient according to the Arrhenius equation. The Arrhenius equation relates the rate constant, k, of a chemical reaction to parameters A (pre-exponential factor), E (activation energy), T (absolute temperature), and b (exponential indicating temperature-dependence of pre-exponential factor)::

A nonzero value for b gives the modified Arrhenius equation.

$$k_{\text{mod arr}} = AT^b \exp\left(-\frac{E}{RT}\right) \qquad \text{(Modified Arrhenius)}$$

When b = 0, the above formula corresponds to the Arrhenius equation.

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

Example:

```
import chemkin.chemkin as chemkin
elementary_reaction = chemkin.ElementaryReaction(reaction_properties)
reactants = elementary_reaction.get_reactants()
products = elementary_reaction.get_products()
rate_coeff = elementary_reaction.calculate_rate_coefficient(1000)
```

`chemkin.chemkin.ReactionSystem`: Represents a system of reactions

Takes a list of ElementaryReaction instances and a list of species. Builds stoichiometric coefficient matrices for the reactants and products and calculates the corresponding progress rates and reaction rates.

This class has five methods, and two special methods:

- `__repr__`: Returns a string containing basic information for the reaction system
- `__len__`: Returns the number of species in the reaction system
- `calculate_progress_rate(concs, temperature)`: Returns the progress rate of a system of elementary reactions
- `calculate_reaction_rate(concs, temperature)`: Returns the reaction rate of a system of elementary reactions
- `get_rate_coefficients()`: Calculate reaction rate coefficients
- `get_backward_rate_coefficients()`: Calculate rate coefficients for reactions in the reverse direction (0 for irreversible reactions)
- `build_reactant_coefficient_matrix()`: Build a reactant coefficients matrix for the reaction system
- `build_product_coefficient_matrix()`: Build a product coefficients matrix for the reaction system
- `check_reversible`: Check if each elementary reaction is reversible
- `setup_reaction_simulator(self, simulation_type, abundances, temperature, t_span, dt=0.01, system_volume=1e-15)`: Quantitative modeling of chemical reactions by either deterministic simulator or stochastic simulator

Example:

```
import chemkin.chemkin as chemkin
reader = chemkin.XMLReader("tests/rxns.xml")
reaction_system = reader.get_reaction_systems()
len(reaction_system[0])
concs = [1., 2., 1., 3., 1.]
reaction_system[0].build_reactant_coefficient_matrix()
reaction_system[0].build_product_coefficient_matrix()
print(reaction_system[0])
reaction_system[0].calculate_progress_rate(concs, 300)
reaction_system[0].calculate_reaction_rate(concs, 300)
reaction_system[0].get_rate_coefficients()
reaction_system[0].get_backward_rate_coefficients()
```

`chemkin.thermodynamics.Thermochem`: Class for calculating the backward reaction rate

Construct with Rxnset class object, the default pressure of the reactor and the default ideal gas constant. The values of pressure of the reactor and ideal gas constant can be changed. It calculates backward reaction rate using the temperature passed into the functions and the corresponding NASA polynomial coefficients.

This class has four methods:

- `Cp_over_R(T)`: Returns specific heat of each species given by the NASA polynomials
- `H_over_RT(T)`: Returns the enthalpy of each species given by the NASA polynomials
- `S_over_R(T)`: Returns the entropy of each species given by the NASA polynomials
- `backward_coeffs(kf, T)`: Returns the backward reaction rate coefficient for reach reaction

Example:

```
import chemkin.chemkin as chemkin
import chemkin.thermodynamics as thermodynamics
reader = chemkin.XMLReader("tests/rxns_reversible.xml")
reaction_system = reader.get_reaction_systems()[0]
nu = reaction_system.product_coefficients - reaction_system.reactant_coefficients
rxnset = thermodynamics.Rxnset(reaction_system.species, nu)
thermo =  thermodynamics.Thermochem(rxnset)
thermo.Cp_over_R(800)
thermo.H_over_RT(800)
thermo.S_over_R(800)
kf = reaction_system.get_rate_coefficients(800)
thermo.backward_coeffs(kf, 800)
```

`chemkin.thermodynamics.Rxnset`: Read and store NASA polynomial coefficients

This class reads the NASA polynomial coefficients for all species in the reaction system from the SQL database which contains coefficients for all species. It stores the coefficients and temperature ranges in a dictionary of dictionaries where the name of species is the key. For each reaction system, the class just need to read from the database once, and check the range the given temperature is in every time the temperature of the reaction system changes afterwards. If the `get_nasa_coefficients` function is called twice for the same reaction system and the same temperature, the cached value is returned.

This class has two methods:

- `get_nasa_coefficients(T)`: Returns the corresponding NASA polynomial coefficients for all species at the given temperature
- `read_nasa_coefficients()`: Return NASA polynomial coefficients for all species involved in the reaction system

Example:

```
import chemkin.chemkin as chemkin
import chemkin.thermodynamics as thermodynamics
reader = chemkin.XMLReader("tests/rxns_reversible.xml")
reaction_system = reader.get_reaction_systems()[0]
nu = reaction_system.product_coefficients - reaction_system.reactant_coefficients
```

```
rxnset = thermodynamics.Rxnset(reaction_system.species, nu)
rxnset.get_nasa_coefficients(800)
```

Simulation classes (new feature for final project)

We elaborated the motivations for implementing a deterministic simulator and a stochastic simulator in the section of `Scientific Background`. To implement such simulators, we added two modules containing four classes as shown below:

`chemkin.simulator.ReactionSimulator`: Base class for simulations.

This class is a base class for `DeterministicSimulator` and `StochasticSimulator`. Common methods like validating arguments and saving data are implemented.

`chemkin.simulator.StochasticSimulator`: Class for stochastic simulation

This class inherits from base class `ReactionSimulator`. It carries out stochastic reaction simulations using the Gillespie stochastic simulation algorithm. Note that a reversible elementary reaction represents two different reactions in a stochastic simulation.

It has five methods:

- `calculate_state_change_matrix()`: Determine how abundances change with each reaction event.
- `calculate_stochastic_constants(temperature)`: Determine stochastic rate constants from deterministic rate constants.
- `calculate_reaction_propensities()`: Determine the propensity of each reaction, the probability of the reaction to occur in the next interval [t, t+dt).
- `simulate()`: Run stochastic simulation between `t_span[0]` and `t_span[1]`.
- `plot_simulation(show=True, savefig=None)`: Shows or saves a plot of the abundances of species over time.

Example:

```
import chemkin.chemkin as chemkin
import chemkin.simulator as simulator
abundances = np.array([10, 10, 10, 10, 10])
reader = chemkin.XMLReader("tests/rxns.xml")
reaction_system = reader.get_reaction_systems()[0]
stoch_sim = simulator.StochasticSimulator(
    reaction_system, abundances, 800, [0, 1], 1e-15)
stoch_sim.simulate()
stoch_sim.plot_simulation()
```

`chemkin.simulator.DeterministicSimulator`: Class for deterministic simulation

This class inherits from the `ReactionSimulator` class. It calls the `ODE_solver` class to numerically integrate reaction rates forward in time, and simulates the concentration of species deterministically.

This class has three methods:

- `simulate(method='bdf', epsilon = 1e-06)`: We implemented three methods to solve the ordinary differential equation. Backward differentiation formula and backward eulerare good for stiff functions, and rk45 is accurate for non-stiff functions. BDF is the most suitable for solving ODE problems in chemical kinetics, so our default method is set as BDF.
- `diff_func(t, y)`: In order for the ode solver to work, we need a function with t (time) and y as parameters. However, the original calculate reaction rate function is a function of y and temperature. We need to transform the original function.
- `plot_simulation(show=True, savefig=None)`: Shows or saves a plot of the concentrations of species over time.

Example:

```
import chemkin.chemkin as chemkin
import chemkin.simulator as simulator
concs = np.array([1., 2., 1., 3., 1.])*1e-05
reader = chemkin.XMLReader("tests/rxns.xml")
reaction_system = reader.get_reaction_systems()[0]
det_sim = simulator.DeterministicSimulator(
    reaction_system, concs, 800, [0, 1], dt=0.01)
det_sim.simulate()
det_sim.plot_simulation()
```

`chemkin.ode_solver.ODE_solver`: Solve ordinary differential equation with three methods

Implemented three ode solvers: backward euler method, Runge-Kutta-Fehlberg, and backward differentiation formula. Backward euler method and backward differentiation formula can be used to solve stiff ODE problems, while Runge-Kutta-Fehlberg moethod is more accurate for non-stiff problems. Notably, Runge-Kutta-Fehlberg moethod will adapt its step size according to the error.

For the purpose of solving ODE problems in chemical kinetics, our default method is backward differentiation formula with step size 0.01. If the abundances of species become negative in the simulation process, the simulation will set the negative abundances back to zero as abundances should never be negative.

This class has five methods:

- `backward_euler()`: Solve the ODE using backward euler method.
- `backward_euler_step()`: Solve the ODE one step/time forward using backward euler method. We use fixed point iterations to find the optimal root.
- `rk45()`: Solve the ODE using Runge-Kutta-Fehlberg method.
- `rk45_step()`: Solve the ODE one step/time forward using Runge-Kutta-Fehlberg method.
- `BDF()`: Solve the ODE using backward differentiation formula.

Example:

```
import chemkin.ode_solver as ode_solver
func = lambda t, y: 2*t
obj = ode_solver.ODE_solver(func, 0.8, [1, 2], 0.1)
obj.backward_euler()
obj.backward_euler_step()
obj.rk45()
```

```
obj.rk45_step()
obj.BDF()
```