Chemical Kinetics Library

1. Introduction

`chemkin` stands for chemical kinetics and is an objected-oriented library for modeling the kinetics of chemical reactions.

1.1 Key Chemical Kinetics Concepts

Chemical kinetics is the study of rates of chemical processes such as reaction rates, reaction mechanisms, etc. as well as the construction of mathematical models that can describe the characteristics of a chemical reaction (wikipedia). Typically, amounts of molecular species reacted (consumed)/formed and the rates of their consumption/formation are of interest.

Chemical reactions can be categorized as elementary or non-elementary and reversible or irreversible. Each class of chemical reactions can be modeled by ODEs (ordinary differential equations) or PDEs (partial differntial equations) using different strategies.

Irreversible Elementary Reaction

For a system consisting of N species undergoing M **irreversible**, **elementary** reactions of the form:

$$\sum_{i=1}^{N} \nu'_{ij} s_i \longrightarrow \sum_{i=1}^{N} \nu''_{ij} s_i, \qquad \text{for } j = 1, \ldots, M$$

where
s_i = Chemical symbol of specie i
ν'_{ij} = Stoichiometric coefficients of reactants
ν''_{ij} = Stoichiometric coefficients of products

The **rate of change of species** i (i.e. the **reaction rate of species** i) can be written as

$$f_i = \frac{d[i]}{dt} = \sum_{j=1}^{M} \nu_{ij} \omega_j \qquad i = 1, \ldots, N$$

where
$\nu_{ij} = \nu''_{ij} - \nu'_{ij}$
ω_j = Progress rate of reaction j

The **progress rate** ω_j for reaction j is given by

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

where
k_j = Forward reaction rate coefficient for reaction j
x_i = Concentration of specie i

There are several types of **forward reaction rate coefficient** k_j, including -

Constant coefficient: coefficient is constant for all reaction temperatures

$$k_j = k_j$$

Arrhenius coefficient:

$$k_j = A \cdot exp^{-E_a/(RT)}$$

Modified Arrhenius coefficient:

$$k_j = AT^b exp^{-E_a/(RT)}$$

where
A = Arrhenius prefactor
b = Modified Arrhenius paramter
E_a = Activation energy
R = Ideal gas constant
T = Temperature

Note: A complete table of notation of **irreversible elementary** reaction

Symbol	Meaning
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
A	Arrhenius prefactor
b	Modified Arrhenius paramter
R	Ideal gas constant
E_a	Activation energy
T	Temperature

Reversible Elementary Reaction

For a system consisting of N species undergoing M **reversible**, **elementary** reactions of the form:

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

where

s_i = Chemical symbol of specie i
ν'_{ij} = Stoichiometric coefficients of reactants
ν''_{ij} = Stoichiometric coefficients of products

The **rate of change of species** i (i.e. the **reaction rate of species** i) can be written as

$$f_i = \frac{d[i]}{dt} = \sum_{j=1}^{M} \nu_{ij} r_j \qquad i = 1, \ldots, N$$

where
$\nu_{ij} = \nu''_{ij} - \nu'_{ij}$
r_j = Total progress rate of reaction j

The **total progress rate** r_j of reaction j is given by

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

where
$k_j^{(f)}$ = Forward reaction rate coefficient for reaction j
$k_j^{(b)}$ = Backward reaction rate coefficient for reaction j
x_i = Concentration of specie i

The **backward reaction rate coefficient** $k_j^{(b)}$ is given by

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

where
k_j^e = Equilibrium constant for reaction j

The **equilibrium constant** k_j^e is related to the equilibrium thermochemistry of the elementary reactions, and it is given by

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

where
$\gamma_j = \sum_{i=1}^{N} \nu_{ij}$
p_0 = Pressure of the reactor (e.g. 10^5 Pa)
ΔS_j = Entropy change of reaction j
ΔH_j = Enthalpy change of reaction j

Read more about Equilibrium constant

The **entropy change** ΔS_j and the **enthalpy change** ΔH_j of reaction j is given by

$$\Delta S_j = \sum_{i=1}^{N} \nu_{ij} S_i = \sum_{i=1}^{N} \nu_{ij} \int_{T_0}^{T} \frac{C_{p,i}(T)}{T} \, \mathrm{d}T \qquad j = 1, \ldots, M$$

$$\Delta H_j = \sum_{i=1}^{N} \nu_{ij} H_i = \sum_{i=1}^{N} \nu_{ij} \int_{T_0}^{T} C_{p,i}(T) \, \mathrm{d}T \qquad j = 1, \ldots, M$$

where
$C_{p,i}$ = Specific heat at constant pressure of species i

The **specific heat at constant pressure** $C_{p,i}$ is given by a polynomial in T (called the NASA polynomial),

$$C_{p,i} = \left(\sum_{k=1}^{5} a_{ik} T^{k-1} \right) R, \qquad i = 1, \ldots, N.$$

where
T = Temperature
R = Ideal gas constant

The integrated forms of ΔS_j, ΔH_j, $C_{p,i}$, using 7^{th} order NASA polynomials are:

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

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

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

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

Note: A complete table of notation of **reversible elementary** reacion

Symbol	Meaning
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)
r_j	Total progress rate of reaction j
x_i	Concentration of specie i
$k_j^{(f)}$	Forward reaction rate coefficient for reaction j
$k_j^{(b)}$	Backward reaction rate coefficient for reaction j
k_j^e	Equilibrium constant for reaction j
p_0	Pressure of the reactor
ΔS_j	Entropy change of reaction j
ΔH_j	Enthalpy change of reaction j
S_i	Entropy of species i
H_i	Enthalpy of species i

Symbol	Meaning
$C_{p,i}$	Specific heat at constant pressure of species i
T	Temperature
R	Ideal gas constant

1.2 The `chemkin` Library

The high level functionality of the **chemkin** module is to take an XML file with reaction data as input and outputs the RHS (right-hand-side) of the ODE describing the rate of change of all molecular species involved in the chemical reaction(s) of interest (i.e. for **irreversible elementary** reaction: $\sum_{j=1}^{M} \nu_{ij}\omega_j, \text{ for } i = 1, \ldots, N$, and for **reversible elementary** reaction: $\sum_{j=1}^{M} \nu_{ij}r_j, \text{ for } i = 1, \ldots, N$).

Features of the **chemkin** module include:

- Parsing XML file with chemical reaction data to extract relevant parameters
- Handling the calculation of 3+ classes of forward reaction rate coefficients (e.g. constant, Arrhenius and modified Arrhenius) given the appropriate parameters
- Handling the calculation of backward reaction rate coefficients for **reversible elementary** reactions
- Handling the calculation of progress rates (ω_j or r_j) and reaction rates (f_i) for a system consisting of N species undergoing M **irreversible** or **reversible elementary** reactions
- Computing species concentrations given a end time point t_{end}
- Visualizing time evolution of species concentration in the reaction system as well as visualizing of the time to reach a dynamic equilibrium for each reaction

Overall structure of the `chemkin` library

```
chemkin/
    __init__.py
    chemkin_errors.py
    preprocessing/
        __init__.py
        parse_xml.py
        tests/
            test_parse_xml.py
    reaction/
        __init__.py
        base_rxn.py
        elementary_rxn.py
        non_elementary_rxn.py
        reaction_coefficients.py
        tests/
            test_base_rxn.py
            test_elementary_rxn.py
            test_non_elementary_rxn.py
            test_reaction_coefficients.py
    thermodynamics/
        __init__.py
        thermo.py
```

```
        NASA_coef.sqlite
        tests/
            test_thermo.py
    viz/
        __init__.py
        summary.py
        tests/
            test_summary.py
    solver/
        __init__.py
        ODEint_solver.py
        tests/
            __init__.py
            test_ODEint_solver.py
    xml-files/
```

A brief description of each subdirectory:

- `chemkin_errors` module hosts functions to detect library-related errors.
- `preprocessing` package contains modules to parse input files, extracts and returns relevant reaction parameters into a python dictionary. Currently, the library only parses .xml input files.
- `reaction` package contains modules to handle different reaction types (calculating progress rates, reaction rates, species concentrations and time to equilibrium) as well as calculating reaction rate coefficients
- `thermodynamics` package contains module to process thermodynamics-related parameters using the NASA_coef SQL database
- `solver` package contains an ODE solver, upon which reaction objects call to solve for the concentrations of reaction species as a function of time as well as the time to reach reaction equilibrium (both for individual reactions in the system and the overall equilibrium)
- `viz` package contains modules that allow the user to visualize reaction kinetics (e.g. printing reaction rates in a prettified, tabular format, plotting species concentration evolution)

2. Installation

The neccessary code can be found at and downloaded from here.

It is also pip-installable using the following command:

```
$ pip install chemkin
```

You can specify the directory of installation using the following command:

```
$ pip install -t [installation directory] chemkin
```

You can run the test suite on your local machine by typing the following command in your command line when in the directory of the installed module.

```
$ cd [installation directory]
```

```
$ pytest
```

If you wish to contribute to the further development of this library, please do not hesitate to consult with the IACS department at Harvard University Graduate School of Arts and Sciences here.

3. Basic Usage

3.1 Structure of the input file

Chemical reaction data should be stored in XML format with the following specifications:

1. a `<phase>` element with a `<speciesArray>` child element which lists the molecular species invovled in the reaction
2. a `<reactionData>` element that stores relevant parameters of the reaction:
    - `<reaction reversible>` tag indicates whether a reaction is reversible or irreversible, possible values = ["yes", "no"]
    - `<type>` tag indicates whether a reaction is elementary or non-elementary, possible values = ["Elementary", "Non-Elementary]
    - `<equation>` tag specifies the chemical reaction
    - `<rateCoeff>` tag stores parameters relevant to calculate the rate coefficient. It has a child tag indicating the rate coefficient class, which can be one of three acceptable types, each of which dictates what coefficient values are parsed by the `XmlParser` class:
        1. `<Arrhenius>` : Coefficients [A, E] will be retrieved.
        2. `<modifiedArrhenius>` : [A, b, E] will be retrieved.
        3. `<Constant>` : k will be retrieved.
        4. `<equation>` : Equation expression for each reaction will be retrieved.

Note There should be only 1 system of reactions per XML file (i.e. 1 `<reactionData>` element). If more than 1 `<reactionData>` element resides within the XML, only the first `<reactionData>` element (i.e. first system of reactions) will be considered.

Note While it is not recommended, there can be a mix reversible and irreversible reactions in a system of chemical reactions. To check if more than 1 type of reaction is specified in the XML file and to count the number of reversible reactions in the system, please see **Example 3.6**.

Note If no recognized child tag of `<rateCoeff>` is encountered, then `XmlParser` will raise a `ChemKinError`. Also, its retrieval of elements is case-sensitive. So, for example, `<A>`, `<b>`, `<E>`, and `<k>` must be used to store the appropriate coefficients; elements named `<a>`, `<B>`, `<e>` or `<K>` would not be recognized and would lead to an error.

Note If the user inputs `<type = Non-elementary>` or `<type = Nonelementary>`, the `XmlParser` will raise a `ChemKinError` as our library does not handle this type of reactions.

Example 3.1. XML file for the following chemical reaction:

$$2H_2 + O_2 \quad \longrightarrow \quad 2OH + H_2$$

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

    <phase>
```

```
        <speciesArray> H2 O2 OH HO2 H2O </speciesArray>
    </phase>

    <reactionData id="test_mechanism">

        <!-- reaction 01  -->
        <reaction reversible="no" type="Elementary" id="reaction01">
            <equation>2H2 + O2 [=] 2OH + H2</equation>
            <rateCoeff>
                <modifiedArrhenius>
                    <A units="m3/mol/s">1e+8</A>
                    <b>0.5</b>
                    <E units="J/mol">5e+04</E>
                </modifiedArrhenius>
            </rateCoeff>
            <reactants>H2:2 O2:1</reactants>
            <products>OH:2 H2:1</products>
        </reaction>
    </reactionData>

</ctml>
```

3.2 Reading and parsing the input file

Reading and parsing the input XML file is handled in the `preprocessing` package, where there are two related classes that allow the user to work with reaction data stored in XML files.

- `XmlParser`
- `RxnData`

3.2.1 `XmlParser`

The `XmlParser` class pulls reaction data from XML files and preprocesses the data (e.g. calculates the reaction rate coefficients from given paramters) with its `parsed_data_list(Ti)` method. `parsed_data_list(Ti)` takes a list of temperatures as input and returns a `list` of `dictionaries` that contain relevant reaction parameters at each of the temperatures.

Each dictionary in the returned list contains the following attributes:

- `species` : a list of moleucular species invovled in the reaction
- `ki` : a list of forward reaction rate coefficients, the i^{th} entry in the list is the coefficient for the i^{th} reaction in the system
- `b_ki` : a list of backward reaction rate coefficients, the i^{th} entry in the list is the coefficient for the i^{th} reaction in the system (this attribute is 0 for irreversible reactions)
- `sys_vi_p` : a list of stoichemotetric coefficients of the reactants, the i^{th} entry in the list is the coefficients for the i^{th} reaction in the system
- `sys_vi_dp` : a list of stoichemotetric coefficients of the products, the i^{th} entry in the list is the coefficients for the i^{th} reaction in the system
- `is_reversible` : an indicator of whether the reaction is reversible (takes on values of True or False)

- `T` : a float of reaction temperature

Example 3.2. Read in, parse and preprocess an input XML file

```
from chemkin import pckg_xml_path
from chemkin.preprocessing.parse_xml import XmlParser

Ti = [100, 750, 1500, 2500, 5000]
xi = [2., 1., .5, 1., 1., 1., .5, 1.]

xml_parser = XmlParser(pckg_xml_path('rxns_reversible'))
parsed_data_list = xml_parser.parsed_data_list(Ti)
```

The `parsed_data_list(Ti)` method abstracts much of the preprocssing stage, including calculating the reaction rate coefficients. However, there are instances where the user might want to simply extract XML elements (e.g. if the user wishes to calculate the reaction rate coefficients in an user-specified manner). The `XmlParser` module also allows the user to work directly with XML elements with its `load()` method. `load` returns a `tuple` of two lists:

- the species invovled in all the reactions in the file
- list of `RxnData` objects, with each object containing the data for an individual reaction (discussed in next section)

Example 3.3. Read in and parse XML file to work with XML elements directly

```
from chemkin import pckg_xml_path
from chemkin.preprocessing.parse_xml import XmlParser

xml_parser = XmlParser(pckg_xml_path('rxns_reversible'))
species, reaction_data = xml_parser.load()
```

3.2.2 `RxnData`

The reaction data parsed by the XmlParser `load()` function from XML files is returned as a list of `RxnData` objects. This class encapsulates relevant information from the XML file in a way that allows the caller to easily process reactions differently according to their features.

`RxnData` objects have the following attributes:

- `rxn_id` : a string for reaction ID
- `reversible` : a boolean indicating whether the reaction is reversible
- `type` : a RxnType object from the Enum class indicating whether a reaction is elementary or non-elementary
- `rate_coef` : a list of parameters for reaction rate coefficient
- `reactants` : a dictionary with molecular species of the reactants as keys and their respective stoichiometric coefficient as values
- `products` : a dictionary with molecular speicies of the products as keys and their respective stoichiometric coefficient as values
- `equation` : a list of equations for each reaction.

Example 3.4. Working with `RxnData` objects

```
from chemkin import pckg_xml_path
from chemkin.preprocessing.parse_xml import XmlParser

xml_parser = XmlParser(pckg_xml_path('rxns_reversible'))
_, reaction_data = xml_parser.load()

for rxn in reaction_data:
    if rxn.reversible:
        # Handle special reversible reaction logic
```

3.3. Calculating kinetic parameters of interest

Two families of modules in the `reaction` package allow you to compute kinetic paramters such as reaction rate coefficients, progress rates and reaction rates.

- `reaction_coefficients` module handles calculations of reaction rate coefficients
- `base_rxn`, `elementary_rxn` and `non_elementary_rxn` modules handle calculations of progress rates, reaction rates, species concentrations at a given time, evolution of species concentrations and time to reach equilibrium.

3.3.1. `reaction_coefficients` module

The `reaction_coefficients` module contains a base class `RxnCoefficientBase` for forward reaction coefficients, from which then the three subclasses (`ConstantCoefficient`, `ArrheniusCoefficient`, and `ModifiedArrheniusCoefficient`) inherit its basic properties (such as `init`, `__repr__` and `__eq__`). When creating the instances of these classes, their parameters are based on inputs extracted using the `load` method from the `XmlParser` class and stored in `RxnData` object. As documented above, `RxnData` objects have a `rate_coef` attribute, which is a list of parameters for reaction rate coefficient. The list can have the following entires: temperature T, Arrhenius constant A (where applicable), modified constant b (where applicable), ideal gas constant R, and Activation energy E.

Since different classes of rate coefficients have different number of input parameters, we can utilize the length of the `rate_coef` to determine the class to use to calculate the reaction rate coefficients. The `get_coeff()` method handls the calculation of the the rate coefficients ki for reaction i.

Example 3.5. Calculating forward reaction rate coefficients

```
from chemkin import pckg_xml_path
from chemkin.preprocessing.parse_xml import XmlParser

xml_parser = XmlParser(pckg_xml_path('rxns_reversible'))

_, reaction_data = xml_parser.load()

coef_params = reaction_data.rate_coeff

        if isinstance(coef_params, list):
            if len(coef_params) == 3: # modified arrhenius coef
                A = coef_params[0]
                b = coef_params[1]
                E = coef_params[2]
                ki.append(ModifiedArrheniusCoefficient(A, b, E, T).get_coef())
```

```
            else: # arrhenius coef
                A = coef_params[0]
                E = coef_params[1]
                ki.append(ArrheniusCoefficient(A, E, T).get_coef())
        else: # const coef
            ki.append(ConstantCoefficient(coef_params).get_coef())
```

The `reaction_coefficients` module also contains a separate class of `BackwardCoefficient`, which calculates the backward reaction rate coefficient for reversible chemical reactions.

3.3.2 `base_rxn` module

The `base_rxn` module contains a single `RxnBase` class, from which the subclasses in `elementary_rxn` and `non_elementary_rxn` modules inherit. The `RxnBase` base class and its subclasses have methods to calculate the progress rates, reaction rates, species concentrations at a given time, time evolution of species concentrations and time to reach equilibrium given data on a system of chemical reactions and associated parameters.

`RxnBase` objects have the following attributes:

- `ki` : a list of forward reaction rate coefficients
- `b_ki` : a list of backward reaction rate coefficients
- `xi` : a list of concentrations of molecular species
- `vi_p` : a list of stoichiometric coefficients of the reactants
- `vi_dp` : a list of stoichiometric coefficients of the product
- `wi` : a list of total progress rates
- `f_wi` : a list of forward progress rates
- `b_wi` : a list of backward progress rates
- `rates` : a list of reaction rates
- `is_reversible` : a list of booleans indicating whether i^{th} reaction is reversible

Note These attributes are initialized when `RxnBase` objects are created. For example,

```
reaction1 = RxnBase(ki=[10, 10], b_ki=[20, 20], xi=[1.0, 2.0, 1.0], vi_p=[[1.0, 2.0,
0.0], [2.0, 0.0, 2.0]], vi_dp=[[0.0, 0.0, 2.0], [0.0, 1.0, 1.0]])
```

`RxnBase` objects have the following methods:

- `progress_rate()` : Returns a list of progress rates for the system (Not implemented in the base class)
- `reaction_rate()` : Returns a list of reaction rates for the system (Not implemented in the base class)
- `species_concentration(self, T, end_t, n_steps=101)` : Returns a list of species concentrations at temperature = T and end time = end_t (n_steps specifies the number of time steps for the ODE solver)
- `species_concentration_evolution(self, T, end_t, n_steps=101)` : Returns a matrix of species concentration evolution at temperature = T and from start to end_t (n_steps specifies the number of time steps for the ODE solver)
- `time_to_equilibrium(self, T, n_steps=101)` : Returns the list of time to equilibrium of all the reactions and the time to equilibrium of the overall system at temperature = T (n_steps specifies the number of time steps for the ODE solver)

3.3.3 `elementary_rxn` module

The `elementary_rxn` module deals with elementary chemical reactions. It contains an `ElementaryRxn` base class

which inhertis from `RxnBase` and handles both **irreversible** and **reversible** elementary reactions.

The `ElementaryRxn` class

This class handles reversible, irreversible, and a mix thereof reactions in a chemical system.

Irreversible Reactions

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

$$\sum_{i=1}^{N} \nu'_{ij} s_i \longrightarrow \sum_{i=1}^{N} \nu''_{ij} s_i, \qquad \text{for } j = 1, \dots, M$$

The progress rate ω_j is given by

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

The reaction rate $f_i = \frac{d[i]}{dt}$ is given by

$$f_i = \frac{d[i]}{dt} = \sum_{j=1}^{M} \nu_{ij} \omega_j, \qquad \text{for } i = 1, \dots, N$$

Reversible Reactions

This class handles a system consisting of N species undergoing M **reversible**, **elementary** reactions of the form:

$$\sum_{i=1}^{N} \nu'_{ij} s_i \rightleftharpoons \sum_{i=1}^{N} \nu''_{ij} s_i \qquad j = 1, \dots, M$$

The total progress rate r_j is given by

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

The reaction rate $f_i = \frac{d[i]}{dt}$ is given by

$$f_i = \frac{d[i]}{dt} = \sum_{j=1}^{M} \nu_{ij} r_j, \qquad \text{for } i = 1, \dots, N$$

`ElementaryRxn` shares the same class attributes as the base class, inherits `species_concentration(self, T, end_t, n_steps=101)`, `species_concentration_evolution(self, T, end_t, n_steps=101)`, `time_to_equilibrium(self, T, n_steps=101)` methods and implements the two methods in the following manner:

- `progress_rate()`: Returns a list of $k_j^{(f)} \prod_{i=1}^{N} x_i^{\nu'_{ij}} - k_j^{(b)} \prod_{i=1}^{N} x_i^{\nu''_{ij}}$
- `reaction_rate()`: Returns a list of $\sum_{j=1}^{M} \nu_{ij} r_j$

3.3.4 `non_elementary_rxn` module

The implementation for non-elementary reactions is TBD.

3.4. Visualizing kinetic parameters of interest

The `viz` package contains the `summary` module allows various visualizations of kinetic parameters of interest.

3.4.1. `summary` module

The methods in the `summary` module is roughly divided into 2 high-level functions: 1) printing kinetic parameters of interest in prettified tabular format; and 2) plotting kinetic parameters of interest

For printing tables of kinetic parameters:

- `print_reaction_rate(parsed_data_list, xi)` : Takes in parsed reaction data from the output of `XMLparser` object's `parsed_data_list(Ti)` method and species concentrations `xi` . The method prints reactions rates in tabular format.
- `print_species_concentration(parsed_data_list, xi, n_steps=101, end_t=1e-12)` : Takes in parsed reaction data from the output of `XMLparser` object's `parsed_data_list(Ti)` method, species concentrations `xi` , and an end-time `end_t` . User can also specify `n_steps` , which the number of time steps the ODE solver uses to integrate differential equations. The method prints species concentration at `end_t` in tabular format.
- `print_time_to_equilibrium(parsed_data_list, xi, n_steps=101)` : Takes in parsed reaction data from the output of `XMLparser` object's `parsed_data_list(Ti)` method and species concentrations `xi` . User can also specify `n_steps` , which the number of time steps the ODE solver uses to integrate differential equations. The method prints time to reach equilibrium for each reaction in the system.

For plotting kinetic parameters:

- `plot_species_concentration(parsed_data_list, xi, n_steps=101, end_t=1e-12)` : Takes in parsed reaction data from the output of `XMLparser` object's `parsed_data_list(Ti)` method, species concentrations `xi` , and an end-time `end_t` . User can also specify `n_steps` , which the number of time steps the ODE solver uses to integrate differential equations. The method saves the line plot to the `viz/examples` directory.
- `plot_time_to_equilibrium(parsed_data_list, xi, n_steps=101)` : Takes in parsed reaction data from the output of `XMLparser` object's `parsed_data_list(Ti)` method and species concentrations `xi` . User can also specify `n_steps` , which the number of time steps the ODE solver uses to integrate differential equations. The method saves the bar chart to the `viz/examples` directory.

4. Examples

4.1. A system of irreversible, elementary chemical reactions

- The system of chemical reactions of interest:

$$
\begin{aligned}
2H_2 + O_2 &\longrightarrow 2OH + H_2 \\
OH + HO_2 &\longrightarrow H_2O + O_2 \\
H_2O + O_2 &\longrightarrow HO_2 + OH
\end{aligned}
$$

- Input XML file:

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

<ctml>
    <phase>
        <speciesArray> H2 O2 OH HO2 H2O </speciesArray>
```

```
    </phase>

    <reactionData id="test_mechanism">
        <!-- reaction 01  -->
        <reaction reversible="no" type="Elementary" id="reaction01">
            <equation>2H2 + O2 [=] 2OH + H2</equation>
            <rateCoeff>
                <modifiedArrhenius>
                    <A units="m3/mol/s">1e+8</A>
                    <b>0.5</b>
                    <E units="J/mol">5e+04</E>
                </modifiedArrhenius>
            </rateCoeff>
            <reactants>H2:2 O2:1</reactants>
            <products>OH:2 H2:1</products>
        </reaction>

        <!-- reaction 02 -->
        <reaction reversible="no" type="Elementary" id="reaction02">
            <equation>OH + HO2 [=] H2O + O2</equation>
            <rateCoeff>
                <Constant>
                    <k>1e+4</k>
                </Constant>
            </rateCoeff>
            <reactants>OH:1 HO2:1</reactants>
            <products>H2O:1 O2:1</products>
        </reaction>

        <!-- reaction 03 -->
        <reaction reversible="no" type="Elementary" id="reaction03">
            <equation>H2O + O2 [=] HO2 + OH</equation>
            <rateCoeff>
                <Arrhenius>
                    <A units="m3/mol/s">1e+7</A>
                    <E units="J/mol">1e+04</E>
                </Arrhenius>
            </rateCoeff>
            <reactants>H2O:1 O2:1</reactants>
            <products>HO2:1 OH:1</products>
        </reaction>
    </reactionData>
</ctml>
```

4.1.1 Printing reaction rates

This example demonstrates the highest-level and most-abstracted use of the library. The user simply specifies the temperatures and initial species concentrations with the corresponding input XML file which contains the reaction data, and the following code prints reaction rates in a prettified, tabular format.

- Given the the species concentration $xi = [2.0, 1.0, 0.5, 1.0, 1.0]$ in units of (mol/dm^3), calculate the reaction rate of each species at $Ti = [750, 1500, 2500]$ in units of K.

```
from chemkin import pckg_xml_path
from chemkin.preprocessing.parse_xml import XmlParser
from chemkin.viz import summary
```

```
Ti = [100, 750, 1500, 2500, 5000]
xi = [2., 1., .5, 1., 1., 1., .5, 1.]

xml_parser = XmlParser(pckg_xml_path('rxns_reversible'))
parsed_data_list = xml_parser.parsed_data_list(Ti)
summary.print_reaction_rate(parsed_data_list, xi)
```

- The expected result:

```
------At Temperature 100 K------
Backward reaction coefficients not defined: T=100 is not in some specie's temperature
range.
--------------------------------
------At Temperature 750 K------
     H : 3.42304562795e+16
     O : -3.38308977852e+16
     OH : -3.52979102957e+16
     H2 : 4.07078984572e+13
     H2O : 5.86479724945e+14
     O2 : 3.44028050967e+16
     HO2 : -7.63606068937e+13
     H2O2 : -5.52803118677e+13
--------------------------------
------At Temperature 1500 K------
     H : 1.32279654839e+14
     O : -3.12877268431e+14
     OH : -1.37621783394e+14
     H2 : 6.07049182e+13
     H2O : 6.45299057466e+13
     O2 : 3.36486898686e+14
     HO2 : -4.18771319555e+13
     H2O2 : -1.01625193691e+14
--------------------------------
------At Temperature 2500 K------
     H : -2.4081673376e+14
     O : -1.5885953972e+13
     OH : 3.22268792909e+14
     H2 : 6.41189812893e+13
     H2O : 4.70126660046e+13
     O2 : -2.74831634318e+13
     HO2 : 5.28617565897e+12
     H2O2 : -1.54500764698e+14
--------------------------------
------At Temperature 5000 K------
Backward reaction coefficients not defined: T=5000 is not in some specie's temperature
range.
--------------------------------
```

4.2 A system of reversible, elementary chemical reactions

- The system of chemical reactions of interest:

$$
\begin{aligned}
H + O_2 &\longrightarrow O + OH \\
O + H_2 &\longrightarrow H + OH \\
H_2 + OH &\longrightarrow H_2O + H \\
O + H_2O &\longrightarrow OH + OH \\
HO_2 + H &\longrightarrow H_2 + O_2 \\
HO_2 + H &\longrightarrow OH + OH \\
HO_2 + O &\longrightarrow O_2 + OH \\
HO_2 + OH &\longrightarrow H_2O + O_2 \\
H_2O_2 + H &\longrightarrow H_2O + OH \\
H_2O_2 + H &\longrightarrow HO_2 + H_2 \\
H_2O_2 + O &\longrightarrow OH + HO_2
\end{aligned}
$$

- Input XML file:

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

<ctml>

  <phase>
      <speciesArray> H O OH H2 H2O O2 HO2 H2O2 </speciesArray>
  </phase>

  <reactionData id="hydrogen_air_mechanism">
    <!-- reaction 01  -->
    <reaction reversible="yes" type="Elementary" id="reaction01">
      <equation>H + O2 =] O + OH</equation>
      <rateCoeff>
        <modifiedArrhenius>
          <A>3.547e+15</A>
          <b>-0.406</b>
          <E>1.6599e+04</E>
        </modifiedArrhenius>
      </rateCoeff>
      <reactants>H:1 O2:1</reactants>
      <products>O:1 OH:1</products>
    </reaction>

    <reaction reversible="yes" type="Elementary" id="reaction02">
    <!-- reaction O2  -->
      <equation>O + H2 =] H + OH</equation>
      <rateCoeff>
        <modifiedArrhenius>
          <A >5.08e+4</A>
          <b>2.67</b>
          <E >6.29e+03</E>
        </modifiedArrhenius>
      </rateCoeff>
      <reactants>O:1 H2:1</reactants>
      <products>H:1 OH:1</products>
    </reaction>

    <reaction reversible="yes" type="Elementary" id="reaction03">
    <!-- reaction 03  -->
      <equation>H2 + OH =] H2O + H</equation>
      <rateCoeff>
        <modifiedArrhenius>
          <A >2.16e+08</A>
```

```
      <b>1.51</b>
      <E >3.43e+03</E>
    </modifiedArrhenius>
  </rateCoeff>
  <reactants>H2:1 OH:1</reactants>
  <products>H2O:1 H:1</products>
</reaction>

<reaction reversible="yes" type="Elementary" id="reaction04">
<!-- reaction 04  -->
  <equation>O + H2O =] OH + OH</equation>
  <rateCoeff>
    <modifiedArrhenius>
      <A >2.97e+06</A>
      <b>2.02</b>
      <E >1.34e+04</E>
    </modifiedArrhenius>
  </rateCoeff>
  <reactants>O:1 H2O:1</reactants>
  <products>OH:2</products>
</reaction>

<reaction reversible="yes" type="Elementary" id="reaction05">
<!-- reaction 10  -->
  <equation>HO2 + H =] H2 + O2</equation>
  <rateCoeff>
    <Arrhenius>
      <A >1.66e+13</A>
      <E >8.23e+02</E>
    </Arrhenius>
  </rateCoeff>
  <reactants>HO2:1 H:1</reactants>
  <products>H2:1 O2:1</products>
</reaction>

<reaction reversible="yes" type="Elementary" id="reaction06">
<!-- reaction 11  -->
  <equation>HO2 + H =] OH + OH</equation>
  <rateCoeff>
    <Arrhenius>
      <A >7.079e+13</A>
      <E >2.95e+02</E>
    </Arrhenius>
  </rateCoeff>
  <reactants>HO2:1 H:1</reactants>
  <products>OH:2</products>
</reaction>

<reaction reversible="yes" type="Elementary" id="reaction07">
<!-- reaction 12  -->
  <equation>HO2 + O =] O2 + OH</equation>
  <rateCoeff>
    <Arrhenius>
      <A >3.25e+13</A>
      <E >0.0</E>
    </Arrhenius>
  </rateCoeff>
  <reactants>HO2:1 O:1</reactants>
```

```
      <products>O2:1 OH:1</products>
    </reaction>

    <reaction reversible="yes" type="Elementary" id="reaction08">
    <!-- reaction 13  -->
      <equation>HO2 + OH =] H2O + O2</equation>
      <rateCoeff>
        <Arrhenius>
          <A >2.890e+13</A>
          <E >-4.970e+02</E>
        </Arrhenius>
      </rateCoeff>
      <reactants>HO2:1 OH:1</reactants>
      <products>H2O:1 O2:1</products>
    </reaction>

    <reaction reversible="yes" type="Elementary" id="reaction09">
    <!-- reaction 17  -->
      <equation>H2O2 + H =] H2O + OH</equation>
      <rateCoeff>
        <Arrhenius>
          <A >2.41e+13</A>
          <E >3.97e+03</E>
        </Arrhenius>
      </rateCoeff>
      <reactants>H2O2:1 H:1</reactants>
      <products>H2O:1 OH:1</products>
    </reaction>

    <reaction reversible="yes" type="Elementary" id="reaction10">
    <!-- reaction 18  -->
      <equation>H2O2 + H =] HO2 + H2</equation>
      <rateCoeff>
        <Arrhenius>
          <A >4.82e+13</A>
          <E >7.95e+03</E>
        </Arrhenius>
      </rateCoeff>
      <reactants>H2O2:1 H:1</reactants>
      <products>HO2:1 H2:1</products>
    </reaction>

    <reaction reversible="yes" type="Elementary" id="reaction11">
    <!-- reaction 19  -->
      <equation>H2O2 + O =] OH + HO2</equation>
      <rateCoeff>
        <modifiedArrhenius>
          <A >9.55e+06</A>
          <b>2.0</b>
          <E >3.970e+03</E>
        </modifiedArrhenius>
      </rateCoeff>
      <reactants>H2O2:1 O:1</reactants>
      <products>OH:1 HO2:1</products>
    </reaction>

  </reactionData>
</ctml>
```

The following examples demonstrate the visualization functionality of the `chemkin` library. The user specifies the temperature and initial species concentrations with the corresponding input XML file which contains the reaction data, and the following code plots the time evolution of the species concetration until the reaction reaches equilibrium, as well as the the time until each reaction reaches equilibrium.

- Given the the species concentration $xi = [2.0, 1.0, 0.5, 1.0, 1.0]$ in units of (mol/dm^3), calculate the concentrations of species over a pre-specified with temperatures held at two levels: $Ti = [100, 1500]$ (Please note we assume the temperature is held constant in our calculation).

4.2.1 Print Concentration of Species at Time end_t

```
from chemkin import pckg_xml_path
from chemkin.preprocessing.parse_xml import XmlParser
from chemkin.viz import summary

Ti = [900, 2500]
xi = [2., 1., .5, 1., 1., 1., .5, 1.] # specie concentrations
xml_parser = XmlParser(pckg_xml_path('rxns_reversible'))

parsed_data_list = xml_parser.parsed_data_list(Ti)

summary.print_species_concentration(parsed_data_list, xi, end_t=1e-12)
```

- The expected result:

```
------At Temperature 900 K------
Specie Concentration at time = 1e-12
  H: start = 2.0, end = 1.723901712597415
  O: start = 1.0, end = 0.23872213644765622
  OH: start = 0.5, end = 0.027139067877998888
  H2: start = 1.0, end = 1.1618557607172413
  H2O: start = 1.0, end = 2.4626238490452628
  O2: start = 1.0, end = 2.385757473314534
  HO2: start = 0.5, end = 6.056885759728584e-14
  H2O2: start = 1.0, end = 6.5724287182009025e-19
--------------------------------

------At Temperature 2500 K------
Specie Concentration at time = 1e-12
  H: start = 2.0, end = 1.2148570014216886
  O: start = 1.0, end = 0.9791302354930477
  OH: start = 0.5, end = 0.9111019320202484
  H2: start = 1.0, end = 0.7430549709243639
  H2O: start = 1.0, end = 2.6939438877946014
  O2: start = 1.0, end = 1.4578690352649595
  HO2: start = 0.5, end = 4.252504205019055e-05
  H2O2: start = 1.0, end = 4.120390407757191e-07
--------------------------------
```

4.2.2 Print Time to Reach Equilibrium

```
from chemkin import pckg_xml_path
```

```
from chemkin.preprocessing.parse_xml import XmlParser
from chemkin.viz import summary

Ti = [900, 2500]
xi = [2., 1., .5, 1., 1., 1., .5, 1.] # specie concentrations
xml_parser = XmlParser(pckg_xml_path('rxns_reversible'))

parsed_data_list = xml_parser.parsed_data_list(Ti)

summary.print_time_to_equilibrium(parsed_data_list, xi)
```

- The expected result:

```
------At Temperature 900 K------
Time to Equilibrium (end_t = 10000000000.0)
  Reaction #0: 1.7710385756438292e-05
  Reaction #1: 1.7710385756438292e-05
  Reaction #2: 1.773464177804173e-07
  Reaction #3: 1.773464177804173e-07
  Reaction #4: 2.1808938174999524e-11
  Reaction #5: 2.1808938174999524e-11
  Reaction #6: 1.743709840167672e-11
  Reaction #7: 1.4245072821790747e-11
  Reaction #8: 8.557334002807858e-13
  Reaction #9: 1.3523084464099764e-12
  Reaction #10: 7.872345307371005e-13

Overall Time to Equilibrium: 1.7710385756438292e-05
--------------------------------

------At Temperature 2500 K------
Time to Equilibrium (end_t = 10000000000.0)
  Reaction #0: 3.28928631333613e-08
  Reaction #1: 32892533.73554649
  Reaction #2: 3289253.373554649
  Reaction #3: 0.0032892533738840468
  Reaction #4: 3.322193155035267e-11
  Reaction #5: 3.292547351702711e-10
  Reaction #6: 3.292547351702711e-10
  Reaction #7: 3.292547351702711e-10
  Reaction #8: 3.322193155035267e-11
  Reaction #9: 3.322193155035267e-11
  Reaction #10: 3.292547351702711e-10

Overall Time to Equilibrium: The system has not yet reached equilibrium.
--------------------------------
```

4.2.3 Plot Time Evolution of Concentration Species

```
from chemkin import pckg_xml_path
from chemkin.preprocessing.parse_xml import XmlParser
from chemkin.viz import summary

Ti = [900]
```

```
xi = [2., 1., .5, 1., 1., 1., .5, 1.] # specie concentrations
xml_parser = XmlParser(pckg_xml_path('rxns_reversible'))

parsed_data_list = xml_parser.parsed_data_list(Ti)

summary.plot_species_concentration(parsed_data_list, xi)
```

- See the expected image here

4.2.4 Plot Time to Equilibrium

```
from chemkin import pckg_xml_path
from chemkin.preprocessing.parse_xml import XmlParser
from chemkin.viz import summary

Ti = [900]
xi = [2., 1., .5, 1., 1., 1., .5, 1.] # specie concentrations
xml_parser = XmlParser(pckg_xml_path('rxns_reversible'))

parsed_data_list = xml_parser.parsed_data_list(Ti)

summary.plot_time_to_equilibrium(parsed_data_list, xi)
```

- See the expected image here

The demo code for this example can be found in the library GitHub repo in the module `demo_ODEsolver.py` here.

5. New Feature

Our new implemented feature is a differential equation solver that calculates species concentrations as a function of time as well as determines the time for each reaction to reach an equilibrium.

We implemented a numerical iterative solver of an Ordinary Differential Equation (ODE) under the assumption an ODE models the time evolution of the species concentration in the chemical system at hand.

This new feature enables the library to:

1. Given an end-time (t_{end}) and reaction data, output the concentrations of each species at t_{end}.
2. Given reaction data, output the time to reach equilibrium (in the case of reversible elementary reactions) or the time for the reaction to reach completion (in the case of irreversible elementary reactions).
3. Given an end-time (t_{end}), function plots the time evolution of species concentrations from t_0 to t_{end}.

5.1 Motivation

We motivate our feature by the following:

- Visualize the change in species concentration over time.
- Identify time for each reaction to reach equilibrium as well as the time when the overall chemical system achieves equilibrium (i.e., reaction completion).
- Get the concentrations at some end time (t_{end}) for each specie.

For the end-user, it may be useful to learn about the concentration of the various species at any given point in time (as opposed to just the reaction rates). Additionally, visualization of the gradient of change in concentration may help the

user make a decision about an experiment design in e.g., time to cut off a titration experiment or a batch chemical synthesis.

In the future, the library could be enhanced by relaxing the condition of fixed temperature to further enable the user to get insights on more complex systems.

5.2 Implementation Details

In the next section, we discuss the functionality of the `chemkin` ODE solver as well as new methods that realize the 3 additional features listed above (i.e. 1 - Calculate the concentrations of each species at a given t_{end}; 2- Calculate the time to reach equilibrium; and 3 - Plot time evolution of species concentration and time to equilibrium.

5.2.1 `ODE_int_solver`

The class `ODE_int_solver` is contained in the `ODEint_solver.py` module in the `solver` package. The `ODE_int_solver` class is the workhorse to solve concentration of species over time as well as time to reach the equilibrium.

Attributes of `ODE_int_solver`:

- `temp`: float, temperature for reaction (assumed to be held constant).
- `rxn`: an instance of the `ElementaryRxn()` object
- `species_equil_thresh`: (float, default 1e-5): Species concentration evolution is defined to reach equilibrium once `np.abs(bw - fw) < species_equil_thresh`, where `(bw - fw)` is the difference in backward and forward progress rates for that species.
- `overall_equil_thresh`: (float, default 1e-2), overall reaction is defined to reach equilibrium once `np.linalg.norm(prograte_diff) < overall_equil_thresh`, where `prograte_diff` is the difference in the backward and forward progress rate vectors of the reaction.
- `critical_t`: List of floats of len(ki). Stores time(s) at which reaction's component species' concentrations reach equilibrium.
- `overall_critical_t`: float, stores time at which overall reaction reaches equilibrium
- `max_t`: float, maximum time allowed for the solver

`ODE_int_solver` objects have the following method:

- `solve(time_int)` method:
    - Input: the time interval (a list of floats) over which the `odeint` numerical integrator will iteratively solve for the concentration of reaction species over time.
    - Output:
      1) sol: the list of lists (type: numpy array) for concentrations of each specie over the `time_int` time interval.
      2) critical_t: time for each reaction to reach equilibrium
      2) overall_t: time for the overall system to reach equilibrium
    - `rxn_rate()`: function (dx/dt) passed to the `odeint` solver to perform the numerical integration. Once the difference of backward and forward coefficients falls below a pre-defined threshold `species_equil_thresh`, the respective species time to reach equilibrium and in similar manner for `overall_equil_thresh` the overall reaction equilibrium time are recorded.
    - the `solve()` method utilizes an external ODE solver, namely the `odeint` from `scipy.integrate`.
    - The solver initializes with list of starting species concentrations stored as the attribute `xi` of the `ElementaryRxn` object. Then, in each iteration, the solver numerically integrates the concentration `xi` attribute of the passed `ElementaryRxn` object via the `rxn_rate()` function. Finally, the new reaction rates of the `ElementaryRxn` based on updated concentrations are calculated via `ElementaryRxn` object method `reaction_rate`.

Note In our implementation, the methods `species_concentration()`, `species_concentration_evolution()`,

and `time_to_equilibrium()` from the `RxnBase` class create an instance of the `ODE_int_solver` object in order to solve for concentration time evolution and equilibrium, respectively. These methods are discussed in the next section.

5.2.2 Added methods in `RxnBase` class

- `species_concentration(self, T, end_t, n_steps=101)` : Returns a list of species concentrations at temperature = T and end time = end_t (n_steps specifies the number of time steps for the ODE solver)
- `species_concentration_evolution(self, T, end_t, n_steps=101)` : Returns a matrix of species concentration evolution at temperature = T and from start to end_t (n_steps specifies the number of time steps for the ODE solver)
- `time_to_equilibrium(self, T, n_steps=101)` : Returns the list of time to equilibrium of all the reactions and the time to equilibrium of the overall system at temperature = T (n_steps specifies the number of time steps for the ODE solver)

5.2.2 `viz` package

The `viz` package contains the `summary` module allows various visualizations of kinetic parameters of interest.

The methods in the `summary` module is roughly divided into 2 high-level functions: 1) printing kinetic parameters of interest in prettified tabular format; and 2) plotting kinetic parameters of interest

For printing tables of kinetic parameters:

- `print_reaction_rate(parsed_data_list, xi)` : Takes in parsed reaction data from the output of `XMLparser` object's `parsed_data_list(Ti)` method and species concentrations `xi` . The method prints reactions rates in tabular format.
- `print_species_concentration(parsed_data_list, xi, n_steps=101, end_t=1e-12)` : Takes in parsed reaction data from the output of `XMLparser` object's `parsed_data_list(Ti)` method, species concentrations `xi` , and an end-time `end_t` . User can also specify `n_steps` , which the number of time steps the ODE solver uses to integrate differential equations. The method prints species concentration at `end_t` in tabular format.
- `print_time_to_equilibrium(parsed_data_list, xi, n_steps=101)` : Takes in parsed reaction data from the output of `XMLparser` object's `parsed_data_list(Ti)` method and species concentrations `xi` . User can also specify `n_steps` , which the number of time steps the ODE solver uses to integrate differential equations. The method prints time to reach equilibrium for each reaction in the system.

For plotting kinetic parameters:

- `plot_species_concentration(parsed_data_list, xi, n_steps=101, end_t=1e-12)` : Takes in parsed reaction data from the output of `XMLparser` object's `parsed_data_list(Ti)` method, species concentrations `xi` , and an end-time `end_t` . User can also specify `n_steps` , which the number of time steps the ODE solver uses to integrate differential equations. The method saves the line plot to the `viz/examples` directory.
- `plot_time_to_equilibrium(parsed_data_list, xi, n_steps=101)` : Takes in parsed reaction data from the output of `XMLparser` object's `parsed_data_list(Ti)` method and species concentrations `xi` . User can also specify `n_steps` , which the number of time steps the ODE solver uses to integrate differential equations. The method saves the bar chart to the `viz/examples` directory.