C207 Final Project
Report/Documentation

Chemical Kinetics Library

Submitted in fulfillment as per the requirements of the course

CS207 : Systems Development for Computational Science

Submitted by
Group 11

Team Members

Karan R. Motwani
Hannah Sim
Shiyu Huang
Haixing Yin

Under the guidance of
Prof. David Sondak

Harvard University
Cambridge, Massachusetts – 02138

Fall Semester, 2017

Abstract

This repository is a collection of Python algorithms and classes intended for research in Reaction Rate quantification, which includes the calculation of Reaction Rate Coefficients and Progress Rate for a system of chemical reactions. The library also includes a special feature that performs a graphical visualization of the reaction. This software tool is capable of receiving an XML file as input and extracting the chemical species and other quantities of interest such as NASA polynomial coefficients from an SQL database for further manipulation. The library is designed for flexibility, extensibility and ease of use. Its software design follows an object-oriented approach and its code is written on Python over an Anaconda Platform.

The library consists of three levels of documentation: a model document available in PDF format, comprehensive code documentation within individual Python files and a lower-level user's guide in the README file available in the repository. This is the model document: it gives a comprehensive overview of the library's capabilities, provides procedures for software execution, and includes examples for ease of use.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Definition

It is widely accepted that computation has now emerged as the third pillar of science alongside the pillars of theory and experiment. Computational science is maturing rapidly and has found considerable and significant use in supporting scientists from various disciplines. The Chemical Kinetics Library available in this repository aims to ease the process of calculation of Reaction Rate, Reaction Rate Coefficients and Progress Rate of a system of chemical reactions while also including a means for visualization of the reaction. The tool simplifies the process by orders of magnitude by simply taking a standard XML file as input and returning the desired parameters in suitable data structures. This library has great application in the field of Ordinary Differential Equation (ODE) solving for chemical reactions and can be further extended to more complex applications such as learning new reaction pathways in an artificial neural net based code library.

1.2 Chemical Kinetics : Brief Overview

In the follow subsections, the essential Chemical Kinetics concepts behind the algorithms and their implementation have been discussed.

1.2.1 Reaction Rate and Reaction Rate Coefficient

In chemical kinetics, a Reaction Rate constant or Reaction Rate coefficient, k, quantifies the progression of a chemical reaction. For a reaction between reactants A and B to form product C :

$$\nu_A A + \nu_B B \longrightarrow \nu_C C.$$

The reaction rate is often found to have the form :

$$r = k(T)[A]^m[B]^n$$

Here k(T) is the Reaction Rate constant which depends on temperature. [A] and [B] are the molar concentrations of substances A and B in moles per unit volume of solution, assuming the reaction is taking place throughout the volume of the solution. (For a reaction taking place at a boundary one would use instead moles of A or B per unit area).

The exponents **m** and **n** are called partial orders of reaction and are not generally equal to the Stoichiometric Coefficients 'a' and 'b'. Instead they depend on the reaction mechanism and can be determined experimentally.

The Reaction Rate Coefficient has multiple forms :

$$k_{\text{const}} = k \qquad \textbf{Constant}$$

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

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

The symbols used above are :

- The Arrhenius prefactor : A
- The Modified Arrhenius parameter : b
- The Temperature : T in Kelvin.
- Activation Energy : E in Joule/mol
- $R = 8.314$ is the ideal gas constant (in Joules/ mol K).

1.2.2 System of Reactions : Irreversible

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

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

The Rate of Change of Specie i (the Reaction Rate) can be written as :

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

And the Progress Rate for each reaction is given by :

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

and k_j is the forward reaction rate coefficient.

1.2.3 System of Reactions : Reversible

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 foward and backward reactions occur. The total progress rate is now given by :

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

The only thing that changes in the mathematics is that we must specify the backward progress rate, and in partcular, the backward reaction rate coefficient $k_j^{(b)}$.

For an elementary reaction, 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 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).

The symbols used above are :

- Chemical symbol of the Specie i : $\mathcal{S}_i$

- Stoichiometric Coefficients of the Reactants : ν'_{ij}
- Stoichiometric coefficients of Products : ν''_{ij}
- Rate of consumption or formation of specie i (Reaction Rate) : f_i
- Progress Rate of reaction j : ω_j
- Concentration of Specie i : x_i
- Reaction Rate Coefficient for reaction j : k_j
- Forward Reaction Rate Coefficient : $k_j^{(f)}$
- Backward Reaction Rate Coefficient : $k_j^{(b)}$
- Equilibrium Constant : $k_j^{(e)}$

Chapter 2

Code Design

2.1 Directory Structure

```
README.md                                   This file.
Requirements.txt                            Library requirements and dependencies.
CS207_Documentation.pdf                     Final Documentation.
LICENSE                                     License Information.
MANIFEST.in                                 Package File.
setup.py                                    Package Installation/Setup Python file.
src/chemkinlib/                             Main directory tree.
        data/
            *.xml                           Miscellaneous XML files consisting of reaction parameters and coefficients.
            NASA_poly_coeff.sqlite          SQL Database of NASA Polynomials.

        reaction/
                 Reactions.py               Module consists of class 'Reaction', 'Reversible' and 'Irreversible'.
                 test_Reactions.py          Testing module for Reactions.py.
                 ReactionSystems.py         Module consists of class 'ReactionSystem'.
                 test_ReactionSystems.py    Testing module for ReactionSystems.py.
                 ReactionRateCoeffs.py      Module consists of class 'ReactionCoeff'.
                 test_ReactionRateCoeffs.py Testing module for ReactionRateCoeffs.py.
        utils/
              Parser.py                     Module consists of class 'ReactionParser'.
              test_Parser.py                Testing module for Parser.py.

examples/
             irrev_rxns.py                  Examples use for Irreversible Reaction System.
             rev_rxns.py                    Examples use for Reversible Reaction System.
```

Figure 2.1: Directory Tree

2.2 Data Structures

Classes

- **class 'ReactionParser'** : This class is present in **Parser.py**. The user instantiates an object of this class with the XML filename as an argument, it consists of methods that facilitate the extraction of

Type of Reaction, Rate Parameters, Species and Stoichiometric Coefficients from the file. Furthermore, methods for the extraction of the NASA Polynomial Coefficients from a pre-created SQL Database are also present.

Methods :

- ➔ get_species(self) : Returns reaction species.
- ➔ get_rxn_type(self, reaction) : Returns reaction type.
- ➔ get_is_reversible(self, reaction) : Returns boolean about whether the reaction is reversible.
- ➔ get_rxn_equation(self,reaction) : Returns reaction equation.
- ➔ get_rate_coeffs_components(self,reaction) : Returns reaction rate coefficient components based on type of coefficient.
- ➔ get_reactant_stoich_coeffs(self,reaction) : Returns reactant Stoichiometric coefficients.
- ➔ get_product_stoich_coeffs(self, reaction) : Returns product Stoichiometric coefficients.
- ➔ get_coeffs(self, species_name, temp_range) : Returns NASA polynomial coefficients from SQL database from appropriate temperature range.
- ➔ get_NASA_poly_coefs(self) : Parses all coefficient information for all reactions.
- ➔ get_Tmid(self, species_name) : Returns middle Temperature value.
- ➔ get_reaction_list(self) : Appends a Reaction/Reaction-inherited object for each reaction described by input XML file to self.reaction_list.

- **class 'ReactionSystem'** : This class is present in **ReactionSystem.py**. An object of this class is instantiated by the user with the reaction list, NASA Polynomial Coefficients, Temperature and species concentrations as attributes. It consists of methods that return, sum and sort the Reaction Rates for a given system of reactions.

  Methods :

  - ➔ sort_reaction_rates(self): Return sorted dictionary of reaction rates.
  - ➔ get_nasa_matrix(self, NASA_poly_coef) : Get temperature range lower/higher than T_mid for each coefficient.

➔ get_reaction_rate(self) : Fetches reaction rate for each reaction in the system.

- **class 'Reaction'** : This class is present in **Reactions.py**. An object of this class is instantiated by the **class ReactionParser** or **class ReactionSystem** with the reaction type, equation, Stoichiometric Coefficients, Rate Coefficients and Species list as attributes. It forms the parent class structure of methods which is overloaded by the **class ReversibleReaction** or **class IrreversibleReaction** depending on the reaction type.

  Methods :

  ➔ get_unique_species(self): Helper function to return unique species involved in the reaction.

  ➔ set_temperature(self, T): Sets the Temperature of the reaction.

  ➔ set_concentrations(self, X): Sets concentrations of the reaction.

  ➔ order_dictionaries(self, dictionary): Helper function to order dictionaries (of concentrations, Stoichiometric Coefficients) based on ordering from species_list.This is to ensure a consistent ordering scheme.

  ➔ compute_reaction_rate(self, T=None): Computes reaction rate of the single reaction object passed.

  [It also consists of overloaded dummy parent methods 'compute_reaction_rate_coeff' and 'compute_progress_rate']

- **class 'IrreversibleReaction'** : This class is present in **Reactions.py**. It inherits the **class Reaction** and alters the Method Resolution Order using the super() command. It implements all methods necessary for returning and/or calculating the Reaction rate Coefficient, Reaction Rate and Progress Rate for an Irreversible Reaction.

  Methods :

  ➔ compute_reaction_rate_coeff(self, T=None) : Computes reaction rate coefficients of the reaction.

  ➔ compute_progress_rate(self, T=None): Computes progress rates of the individual reaction species.

  ➔ compute_reaction_rate(self, T=None): Computes reaction rate of the single reaction object passed.

- **class 'ReversibleReaction'** : This class is present in **Reactions.py**. It inherits the **class Reaction** and alters the Method Resolution Order using the super() command.It implements all methods necessary for returning and/or calculating the Reaction rate Coefficient, Reaction Rate and Progress Rate for an Reversible Reaction.

  Methods :

  ➔ compute_reaction_rate_coeff(self, T=None) : Computes reaction rate coefficients of the reaction. Both forward and backward.

  ➔ compute_progress_rate(self, T=None): Computes progress rates of the individual reaction species for both the forward and backward reaction and returns the difference.

- **class 'ReactionCoeff'** : This class is present in **ReactionRateCoeffs.py**. An object of this class is called by method 'compute_reaction_rate_coeff' of class 'Reaction' with the parameters for the calculation of Rate Coefficient passed in the form of a dictionary and Temperature as arguments.

  Methods :

  ➔ get_coeff(self, k_parameters, T): Defines Reaction Rate Coefficient parameters and calls appropriate function for calculation.

  ➔ const(self, k): Returns Constant Reaction Rate Coefficient 'k'.

  ➔ arr(self, A, E, T, R=8.314): Returns Arrhenius Reaction Rate Coefficient 'k'.

  ➔ mod_arr(self, A, b, E, T, R=8.314): Returns Modified Arrhenius Reaction Rate Coefficient 'k'.

- **class 'BackwardCoeff'** : This class is present in **ReactionRateCoeffs.py**. An object of this class is called by method 'compute_reaction_rate_coeff' of class 'ReversibleReaction' with the parameters for the calculation of Backward Reaction Rate passed in the form of a dictionary and Temperature as arguments.

  Methods :

  ➔ Cp_over_R(self, T): Returns specific heat of each specie given by the NASA polynomials.

  ➔ H_over_RT(self, T): Returns the enthalpy of each specie given by the NASA polynomials.

➔ S_over_R(self, T): Returns the entropy of each specie given by the NASA polynomials.

➔ backward_coeffs(self, kf, T): Returns the backward Reaction Rate Coefficient for reach reaction.

Dictionaries

- **Reaction Rate Parameters** : Parameters such as Activation Energy, Arrhenius pre-factor and Rate Constant are passed as a single dictionary for ease of calculation of Rate Coefficients.
- **Stoichiometric Reactant Coefficients** : The values of the Stoichiometric Reactant Coefficients are stored in a dictionary with the unique species as an index for ease of calculation of Reaction/Progress Rate.
- **Stoichiometric Product Coefficients** : The values of the Stoichiometric Product Coefficients are stored in a dictionary with the unique species as an index for ease of calculation of Reaction/Progress Rate.
- **NASA Polynomial Coefficients** : The values of the NASA Polynomial Coefficients are stored in a dictionary with the unique species as an index for ease of calculation of Backward Reaction Rate Coefficient.
- **Specie Concentrations** : The specie concentrations of a given system of reactions are stored in a dictionary with the unique species as an index. This enables ease of attribute transfer during method calls and calculations.

Other Data Structures

- **Lists** : are used to store a collection of the unique species of the system of reactions.
- **Sets** : are used for easy unique-ness check operations at the reaction level.

2.3 Flow of Control

From the theoretical overview, it is clear that the terms/parameters of a system of reactions need to be calculated in an specified order. The Reaction and Progress Rate (ω_j) depend on the Rate Coefficient (k_j) and the Concentrations of each Specie (ν_{ij}) depends on the Reaction Rate (f_i) and Time (t)

for which the reaction is allowed to progress.

The Initial Concentrations of the Reactants (ν'_{ij}) and the Temperature (T) need to be input by the user dynamically for a specific reaction to observe the variance of the results for different initial conditions. The Species involved in the system and the type of reaction, however, is pre-defined.

Abiding by this model, the usage of the library needs to follow an ordered set of operations :

- The **Parser** modules extract the Parameters and Species of a chemical reaction from the XML file and NASA Polynomial Coefficients from the pre-created SQL Database.
- The **ChemKin** modules consist of classes and methods that aid in the calculation of Reaction Rate Coefficient for an input Temperature using extracted Parameters.
- Finally, the respective Reaction Rate/Progress Rate can be calculated based on Initial Concentrations, provided as input through the **ReactionSystem** object and implicitly called using the compute_reaction_rate() method overloaded in the **class 'IrreversibleReaction'** or **class 'ReversibleReaction'**.

Chapter 3

Installation

3.1 Cloning the Repository

The repository is available on **github** through URL : `https://github.com/cs207-group11/cs207-FinalProject.git`. To download :

- Below 'Contributors', a green button to 'Clone or download' the repository is present.
- Use the 'Download ZIP' option to begin the download.
- Once completed, the ZIP file will be present in 'Downloads' folder of your computer (default).
- Unzip this file into a directory of your choice using WinRAR or equivalent software.
- Now, follow instructions in Chapter 3 : Procedure for usage.

This link can also be used to clone the repository to your machine through the Mac Terminal/GitBash.

- Enter directory of your choice using appropriate 'cd' commands.

```
$ git clone <repository URL mentioned above>
$ cd cs207-FinalProject
```

3.2 Package Installation

For simplicity, the library is available as a package that can be directly installed using the following commands.

```
#After cloning the repository into your local system.
#Execute in terminal
>>> python setup.py install

(OR)

#Execute in terminal
>>> pip3 install chemkinlib11

#Enter your Python project and import to test
#the installation in your interpreter.
>>>import chemkinlib
```

3.3 Test Suite

The testing is an important phase of development of any project and a variety of tests are necessary before the design is released. We have defined test modules called 'test_chemkin.py' and 'test_Parser.py' within this repository.

3.3.1 Unit Tests

Multiple assertion tests, Doc-Tests and PyTests have been defined in the modules mentioned above such that the entire library has been completely verified for functionality.

- **TravisCI** : Build : Passing.
- **Coveralls** : 97% Code Coverage.

3.3.2 Types of Errors

- ValueError: check the user's input values for incorrect variable or parameter definitions. Ex : Negative Temperature/Molar Concentration.

- AttributeError: check if the execution of the program is in the right order. Ex : If the Rate Coefficient has been calculated before attempting to calculate the Reaction Rate.
- TypeError : checks if the entered parameters are real and belong to a certain data type. Ex : Modified Arrhenius factor 'b' is real and a float.
- NotImplementedError : checks if a certain feature/function has been designed or is due for implementation.

The above types of errors are tested for within the module 'test_chemkin.py' for erroneous input to test the robustness of the 'chemkin.py' module.

3.3.3 Running the Test Suite

The test suite can be run from the **Terminal** to view the results of the testing.

```
#Enter repository directory
>>> python -m pytest <test_module_name>
```

Chapter 4

Usage and Examples

4.1 Procedure

- Install the package or clone the repository from **github** onto your local machine.
- Copy your target XML file to the directory of the cloned repository or your target Python Project directory.
- Instantiate an object of **class ReactionParser** with the filename.
- The '__call__' method defined for your convenience allows you to begin the extraction of parameters from the XML file.
- If the package is installed, import the library into your Python project using the following command.

```
>>> import chemkinlib
```

- In addition to the parameters, information about the Stoichiometric Coefficients and Type of Reaction is derived from the XML File for the reactants and products separately.
- An object of **class ReactionSystem** with the reaction list, NASA polynomial coefficients, Temperature and species concentrations as attributes should be created.
- The object of **class Reaction** accepts the parameters extracted from the XML file directly through the **class ReactionSystem** to return the reaction_list.

- The Reaction Rate Coefficient is calculated by calling a method 'compute_reaction_rate_coeff' of **class Reaction** which instantiates an object of **class ReactionCoeff** and passes a dictionary of parameters necessary for calculation. This method is overloaded by the **class ReversibleReaction** or **class IrreversibleReaction** depending on the reaction type.
- The **class ReactionCoeff** consists of methods for individual forms of the Reaction Rate Coefficient and returns the coefficient according to the parameters passed as input.
- Using the Reaction Rate Coefficient, Unique Specie Concentrations and Stochiometric Coefficients, the Reaction Rate and Progress Rate can be obtained using methods 'compute_reaction_rate' and 'compute_progress_rate' defined in the **class Reaction** and overloaded by the 'children classes' **ReversibleReaction** or **IrreversibleReaction**

4.2 Examples

```
import os

from chemkinlib.utils import Visualization
from chemkinlib.utils import Parser
from chemkinlib.reactions import ReactionSystems
from chemkinlib.config import DATA_DIRECTORY

#USER INPUT: reaction (xml) file
xml_filename = os.path.join(DATA_DIRECTORY, "rxns_reversible.xml")
parser = Parser.ReactionParser(xml_filename)

#USER INPUTS (concentrations and temperatures)
concentration = ({'H':1, 'H2':1, 'H2O':1, 'H2O2':1, 'HO2':1, 'O':1, "O2":1, "OH":1})
temperature = 1000

#Set up reaction system
rxnsys = ReactionSystems.ReactionSystem(parser.reaction_list,
                                        parser.NASA_poly_coefs,
                                        temperature,
                                        concentration)

#Compute and sort reaction rates
rxnrates_dict = rxnsys.sort_reaction_rates()

#Display reaction rates by species
for k, v in rxnrates_dict.items():
    print("d[{0}]/dt : \t {1:e}".format(k, v))
```

Figure 4.1: Code : Reversible Reaction Example.

4.2.1 Case 1 : Reversible Reaction

The code above shows the work-flow of the code for a System of Reversible Reactions.

The result of the above example module is shown below :

```
>python rev_rxns.py
d[H]/dt :        5.523733e+15
d[O]/dt :        -5.455678e+15
d[OH]/dt :       -5.917805e+15
d[H2]/dt :       2.853787e+13
d[H2O]/dt :      2.689407e+14
d[O2]/dt :       5.713759e+15
d[HO2]/dt :      -1.220901e+14
d[H2O2]/dt :     -3.939778e+13
```

Figure 4.2: Result

4.2.2 Case 2 : Irreversible Reaction

The code below shows the work-flow of the code for a System of Irreversible Reactions.

```
import os

from chemkinlib.utils import Parser
from chemkinlib.reactions import ReactionSystems
from chemkinlib.config import DATA_DIRECTORY

# USER INPUT: reaction (xml) file
xml_filename = os.path.join(DATA_DIRECTORY, "rxnset_long.xml")

parser = Parser.ReactionParser(xml_filename)

# USER INPUTS (concentrations and temperatures)
concentration = ({'H':1, 'H2':1, 'H2O':1, 'H2O2':1, 'HO2':1, 'O':1, "O2":1, "OH":1})
temperature = 1000


# Set up reaction system
rxnsys = ReactionSystems.ReactionSystem(parser.reaction_list,
                        parser.NASA_poly_coefs,
                        temperature,
                        concentration)

# Compute and sort reaction rates
rxnrates_dict = rxnsys.sort_reaction_rates()

# display reaction rates by species
for k, v in rxnrates_dict.items():
    print("d[{0}]/dt : \t {1:e}".format(k, v))
```

Figure 4.3: Code : Irreversible Reaction Example.

The result of the above example module is shown below :

```
>python irrev_rxns.py
d[H]/dt :         -1.387086e+14
d[O]/dt :         -1.238406e+13
d[OH]/dt :        1.874537e+14
d[H2]/dt :        2.627650e+13
d[H2O]/dt :       4.979461e+13
d[O2]/dt :        4.905580e+13
d[HO2]/dt :       -1.220885e+14
d[H2O2]/dt :      -3.939938e+13
```

Figure 4.4: Result

Chapter 5

Future Work

5.1 Definition

The feature we intend to implement can be unambiguously stated as a "visualization tool for a system of reactions that depicts the reaction path taken by the individual species of the reaction from their initial to final state."

For a system of elementary reactions, the nodes of this graph represent chemical species, and the arrows (or, edges) connecting the nodes represent the rate of flow of a conserved quantity between the corresponding reactant and product species.

5.2 Motivation

As discussed earlier, modern scientific study is very inter-disciplinary and tools that allow ease of understanding of internal chemical kinetics and specie exchange are gaining significance.

- Visualization deciphers the interpretation of a chemical reaction as a "black box".
- Aids in understanding of chemical reaction dynamics.
- Provides different types of information using a single diagram.
- Provides a more appealing visual aid for presentation.

5.3 Feature Compatibility

The Directory Structure including the feature module we intend to implement can be seen below. We view this visualization tool as a significant utility of our chemical library.

```
README.md                                   This file.
Requirements.txt                            Library requirements and dependencies.
CS207_Documentation.pdf                     Final Documentation.
LICENSE                                     License Information.
MANIFEST.in                                 Package File.
setup.py                                    Package Installation/Setup Python file.
src/chemkinlib/                             Main directory tree.
        data/
            *.xml                           Miscellaneous XML files consisting of reaction parameters and coefficients.
            NASA_poly_coeff.sqlite          SQL Database of NASA Polynomials.

        reaction/
                 Reactions.py               Module consists of class 'Reaction', 'Reversible' and 'Irreversible'.
                 test_Reactions.py          Testing module for Reactions.py.
                 ReactionSystems.py         Module consists of class 'ReactionSystem'.
                 test_ReactionSystems.py    Testing module for ReactionSystems.py.
                 ReactionRateCoeffs.py      Module consists of class 'ReactionCoeff'.
                 test_ReactionRateCoeffs.py Testing module for ReactionRateCoeffs.py.
        utils/
              Parser.py                     Module consists of class 'ReactionParser'.
              test_Parser.py                Testing module for Parser.py.
              Visualization.py              Module consists of class 'ReactionPathDiagram'.
              test_Visualization.py         Testing Module for Visualization.py.
examples/
             irrev_rxns.py                  Examples use for Irreversible Reaction System.
             rev_rxns.py                    Examples use for Reversible Reaction System.
```

Figure 5.1: Directory Tree with Visualization Feature Module.

The requirement of developing a Reaction Path Diagram include:

- Unique species of the system of reactions.
- Reaction Rate Coefficient.
- Reaction Rates of the individual species.
- Reaction Rate of the system.
- *Initial and final concentrations.

Given these requirements, we would need to import the methods and attributes of **class ReactionSystem**, **class Reaction** and **class ReactionCoeff**.

[Note* : For calculating final concentrations, we would need to implement an integrator that calculates the concentration of each specie at different intervals of time given its Reaction rate. Forward and Backward Euler implementations have been discussed with the Instructor (Prof. Sondak) as future steps after a more basic graphical model is ready. A simpler alternative to use SciPy packages was also recommended.]

5.4 Code Design

```
class ReactionPathDiagram():

    def __init__(self, rxn_sys_obj):
        #Get unique species of the reaction system
        self.unique_species = rxn_sys_obj.involved_species
        #Get Reaction Rate of the system
        self.sys_reaction_rate = rxn_sys_obj.get_reaction_rate()
        #Sort Rates of the unique species
        self.reaction_rates = rxn_obj.sort_reaction_rates(self.unique_species)

    def create(self):
        #Method to define graphical nodes for each unique
        #specie at the reactant and product end.
        *Code Here*
        #Call connect function after nodes are defined
        self.connect_nodes(self.unique_species)

    def connect_nodes(self):
        #Method to connect graphical nodes of each unique
        #specie between the reactant and product end.
        *Code Here*

    def add_graphics(self, boolean_dict):
        #Method to add graphics such as color, text and other reaction parameters.
        #Boolean dictionary contains information to add.
        *Code Here*

    def plot(self):
        #Method to plot graph object in new figure window
        *Code Here*

    *def calculate_product_concentrations(self, time):
        #Method to calculate specie concentration at specific time interval using integrator
        *Code Here*

    *def integrator(self, time, concentrations, reaction_rate_coeff, reaction_rates):
        #Method for Integration
        *Code Here*

*optional methods
```

Figure 5.2: Code Design for Feature

5.5 Functionality Example

The code below shows a demonstration of the functionality of the feature (or library in general) as envisioned by our team.

```
import os

from chemkinlib.utils import Visualization
from chemkinlib.utils import Parser
from chemkinlib.reactions import ReactionSystems
from chemkinlib.config import DATA_DIRECTORY

#USER INPUT: reaction (xml) file
xml_filename = os.path.join(DATA_DIRECTORY, "rxns_reversible.xml")
parser = Parser.ReactionParser(xml_filename)

#USER INPUTS (concentrations and temperatures)
concentration = ({'H':1, 'H2':1, 'H2O':1, 'H2O2':1, 'HO2':1, 'O':1, "O2":1, "OH":1})
temperature = 1000

#Set up reaction system
rxnsys = ReactionSystems.ReactionSystem(parser.reaction_list,
                                    parser.NASA_poly_coefs,
                                    temperature,
                                    concentration)

#Compute and sort reaction rates
rxnrates_dict = rxnsys.sort_reaction_rates()

#Display reaction rates by species
for k, v in rxnrates_dict.items():
    print("d[{0}]/dt : \t {1:e}".format(k, v))

#Visualization
graph_obj = ReactionPathDiagram(rxnsys)
graph_obj.create()
graph_obj.add_graphics({'rate':True,'concentration':True})
graph_obj.plot()
```

Figure 5.3: Feature Functionality Example.

5.6 External Dependencies

The most obvious external dependencies we define include graphing tools and computation libraries.

- **Graphviz** : To draw the nodes and make arrow connections in the Raction Path Diagram.
- **Numpy** : For computation of necessary reaction parameters.
- ***SciPy** : For calculating final specie concentration using pre-packaged integrator.

5.7 Sample Result

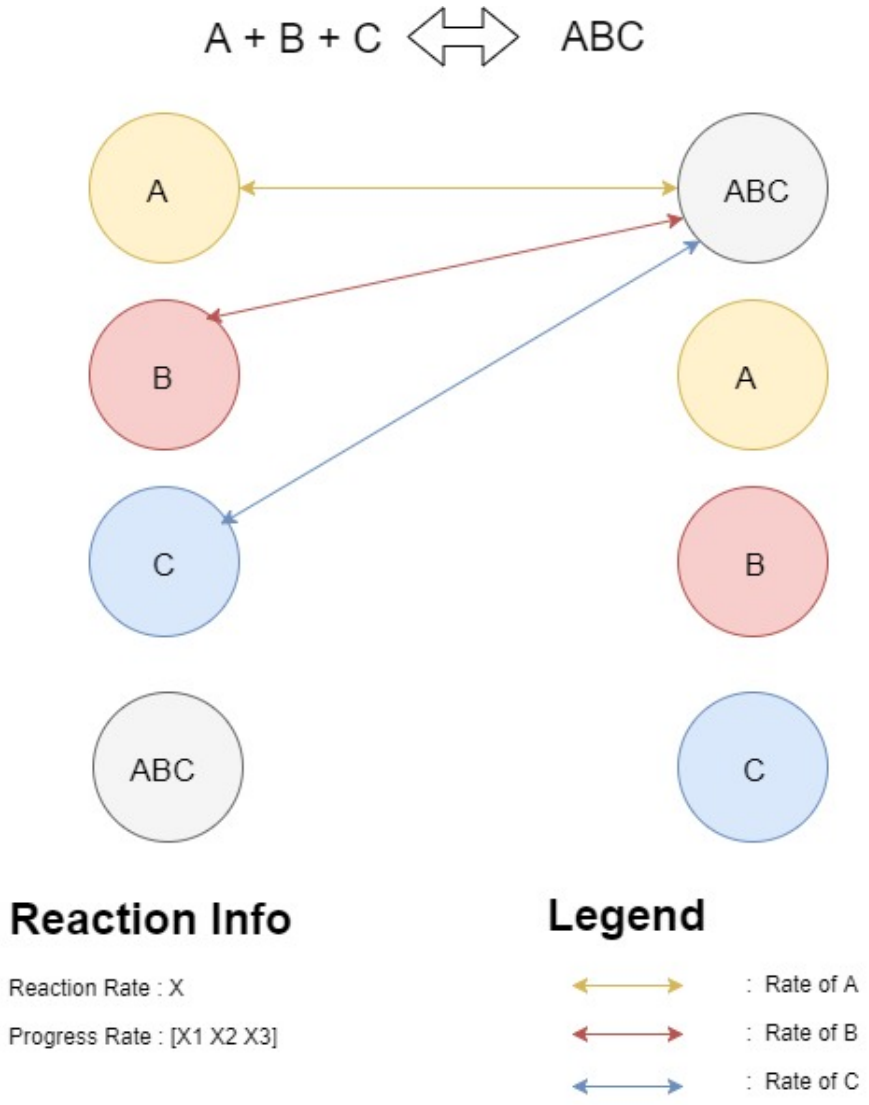

Figure 5.4: Sample Output

References

[1] Chemical Kinetics : Wikipedia, https://en.wikipedia.org/wiki/Chemical_kinetics

[2] Cantera : Chemical Library, http://www.cantera.org/docs/sphinx/html/cython/kinetics.html#reaction-path-analysis