Applied Chemical Engineering (2025) Volume 8 Issue 2 ISSN: 2578-2010 (O)
doi: 10.59429/ace.v8i2.5644

RESEARCH ARTICLE

The Law of mass action: Mathematical modelling and python implementation for chemical kinetics

Choon Kit chan[1], Pankaj Dumka[2,*], Rishika Chauhan[3], Altafhussain G Momin[4], Rajashree Bhokare[5], Neelashetty K[6] , Subhav Singh[7,8], Deekshant Varshaney[9,10], Feroz Shaik[11]

[1] *Faculty of Engineering and Quantity Surveying, INTI International University, Nilai, Negeri Sembilan, 71800, Malaysia*

[2] *Department of Mechanical Engineering, Jaypee University of Engineering and Technology, A.B. Road, Raghogarh-473226, Guna, Madhya Pradesh, India*

[3] *Department of Electronics and Communication Engineering, Jaypee University of Engineering and Technology, A.B. Road, Raghogarh-473226, Guna, Madhya Pradesh, India*

[4] *Department of Mechanical Engineering, L D College of Engineering, Ahmedabad, Gujarat, India*

[5] *Department of Electrical Engineering, Dr. D. Y. Patil Institute of Technology, Pimpri, Pune, Maharashtra, 411018, India*

[6] *Professor, EEE department, Guru Nanak Dev Engg College, Bidar, Karnataka, 585403, India*

[7] *Chitkara Centre for Research and Development, Chitkara University, Himachal Pradesh-174103, India*

[8] *Division of research and development, Lovely Professional University, Phagwara, Punjab, India*

[9] *Centre of Research Impact and Outcome, Chitkara University, Rajpura- 140417, Punjab, India*

[10] *Centre for Promotion of Research, Graphic Era (Deemed to be University), Uttarakhand, Dehradun, India*

[11] *Department of Mechanical Engineering, College of Engineering, Prince Mohammad Bin Fahd University, Al Khobar 31952, Saudi, Arabia*

***Corresponding author:** Pankaj Dumka, p.dumka.ipec@gmail.com

ARTICLE INFO

Received: 26 March 2025
Accepted: 25 June 2025
Available online: 30 June 2025

COPYRIGHT

Copyright © 2025 by author(s).
Applied Chemical Engineering is published by Arts and Science Press Pte. Ltd. This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License (CC BY 4.0).
https://creativecommons.org/licenses/by/4.0/

ABSTRACT

This article explores the mathematical framework and computational implementation of the "Law of Mass Action" to model the kinetics of chemical reaction. The study begins with a detailed explanation of the governing equations, emphasizing the role of stoichiometry and reaction orders in dynamic systems. Using Python, a generalized computational framework was developed to solve systems of ordinary differential equations (ODEs) that describe concentration changes over time. The function solve_ivp has been used from the SciPy module to perform the task of solving ODEs. The solver is capable of handling complex reaction networks by incorporating a stoichiometric matrix, reaction rate constants, and reaction orders as inputs. The results are plotted and tabulated with the help of Matplotlib.pylab and Pandas modules. Two representative examples, including real-world chemical reactions, were solved to demonstrate the versatility and accuracy of the approach. Results show that this generalized methodology provides an efficient and adaptable tool for chemical reaction modelling. This work highlights the power of combining mathematics with modern programming to solve practical chemical engineering problems.

Keywords: chemical kinetics; law of mass action; python programming; SciPy; NumPy; pandas; process innovation

1. Introduction

The "Law of Mass Action", a foundation of chemical kinetics which describes the relationship between the reactant concentrations and the reaction rates in a dynamic chemical system[1]. Python has widely variety applications in engineering sciences.[2]. Its mathematical formulation forms the foundation for modelling and understanding a wide range of chemical reactions in industries such as pharmaceuticals, petrochemicals, and environmental engineering[3]. However, solving these equations for complex reaction networks with multiple reactions and species can be computationally intensive[4–6].

With the increasing demand for accurate and efficient modelling tools, Python programming has emerged as a robust solution[7–10]. Python's extensive library ecosystem, including NumPy[11–13] and SciPy[11,14–16], enables rapid development and solving of nonlinear ODEs that govern reaction kinetics. Also, the modules like Matplotlib[17–20] and Pandas[15,21] are very handy when it comes to data plotting and file handling. This article presents a systematic methodology for modelling the "Law of Mass Action", combining theoretical derivations with computational implementation. Patel et al.[22] carried out drying analysis and it reflects underlying chemical kinetics, where moisture diffusion and evaporation rates are influenced by temperature-dependent reactions. It also enabling accurate modeling of drying behavior and effective moisture transport mechanisms. Mahesh kumar et al.[23] evaluated thin-layer drying kinetics of vermicelli. They carried out moisture removal rates to temperature-driven diffusion and evaporation processes governed by fundamental principles of chemical kinetics in greenhouse conditions.

In this work, first a detailed explanation of the "Law of Mass Action" mathematics is given, including its relationship to stoichiometric matrices and reaction orders. A generalized Python framework is then introduced which can solve systems of chemical reactions of arbitrary complexity. To demonstrate the effectiveness of this approach, two example problems were solved: one involving two reactions and another with multiple interconnected reactions, reflecting real-world complexity. This study bridges the gap between chemical theory and practical computation, offering a tool that can be easily adapted for diverse applications in chemical engineering, research, and education.

The novelty of this work lies in the development of a generalized and modular Python-based solver that systematically integrates the stoichiometric matrix, reaction orders, and rate constants to model chemical kinetics based on the Law of Mass Action. Unlike existing studies that often focus on specific reaction mechanisms or use hard-coded equations, this study offers a flexible and reusable framework capable of handling arbitrary reaction networks. Moreover, the solver emphasizes clarity, educational transparency, and expandability, making it particularly suitable for both academic instruction and research. The two diverse case studies further demonstrate its robustness across different chemical systems, showcasing its practical utility and versatility.

2. Mathematical background

The "Law of mass action" explains that how the chemical reaction rate is proportional to the reactants concentration products raised to their respective stoichiometric coefficients. Below is a step-by-step derivation of the law of mass action.

Consider a general chemical reaction:

$$a_1A_1 + a_2A_2 + \cdots \rightarrow b_1B_1 + b_2B_2 + \cdots \quad (1)$$

Here, A's are the reactants, B's are the products and, a's &b's are the stoichiometric coefficients of the reactants and products, respectively. For example, in a reaction like:

$$2H_2 + O_2 \rightarrow 2H_2O \quad (2)$$

Here in this reaction, $a_1 = 2$, $a_2 = 1$, and $b_1 = 2$

The "Law of mass action"expresses that the rate of this reaction depends on the concentrations of the reactants, raised to their stoichiometric powers. If $r_j \& k_j$ are the rate and rate constant for the j^{th} reaction, c_l is the concentration of the l^{th} reactant, and R_l is the stoichiometric coefficient of the l^{th} reactant and j^{the} reaction then the r_j can be mathematically expressed as[24]:

$$r_j = k_j \prod_l c_l^{R_l} \tag{3}$$

Eqn. (2) tells that the r_j is proportional to the reactant concentrationsproduct. Let's take the same reaction of water formation, then the reaction depends on the concentration of $[H_2]$ and $[O_2]$. So, to write the reaction rate the concentrations of their reactants are raised to their respective stoichiometric coefficients, as shown in Eqn. (4). It is important to note that in this study, the reaction orders are assumed to be equal to the stoichiometric coefficients, which is a valid assumption only for elementary reactions. For more complex reactions involving intermediate steps or catalytic pathways, the reaction order may deviate from stoichiometry and should be derived experimentally or mechanistically. The generalized solver, however, allows manual input of reaction orders, enabling flexibility beyond this assumption.

$$r_j = k_j[\mathrm{H_2}]^2[O_2]^1 \tag{4}$$

In general, for n reactions, the rate expression includes all reactant concentrations. Therefore, Eqn. (3) can also be understood as follows:

$$r_j = k_j \times c_1^{R_1} \times c_2^{R_2} \ldots \times c_n^{R_n} \tag{5}$$

Each chemical species has a concentration c_i, which changes with time as the reaction proceeds. The rate of change of c_i is governed by the stoichiometry of the reaction[24,25]:

$$\frac{dc_i}{dt} = \sum_j S_{ij}\, r_j \tag{6}$$

where, $\frac{dc_i}{dt}$ is the concentration rate change for the i^{th} specie and S_{ij} is the stoichiometric coefficient for i^{th} species in the j^{th} reaction. The expression for the r_j is taken from Eqn. (3). The properties of S_{ij} tensor are mentioned in **Table 1**.

Table 1. S_{ij} tensor properties.

S_{ij}	Explanation
> 0	i^{th} species is a **product** of the j^{th} reaction (produced)
< 0	i^{th} species is a **reactant** of the j^{th} reaction (consumed)
$= 0$	i^{th} species is **not involved** in the j^{th} reaction

Consider Eqn. (2), the stoichiometric coefficients for H_2, O_2, and H_2O can be arranged in S as: $S = [-2, -1, +2]$

And the rate of concentration change for reactants and products can be written as:

$$\frac{d[H_2]}{dt} = -2r_j, \frac{d[O_2]}{dt} = -r_j, \text{ and } \frac{d[H_2O]}{dt} = +2r_j$$

The relationship between the rates of change and the stoichiometry of the reactions can be represented in **matrix form** as shown in Eqn. (7)[28].

$$\frac{d\boldsymbol{c}}{dt} = \boldsymbol{S}\,.\,\boldsymbol{r} \tag{7}$$

Here, $\boldsymbol{c}$is the species concentrations vector $\left(\begin{bmatrix} c_1 \\ c_2 \\ \dots \end{bmatrix}\right)$, $\boldsymbol{S}$ si the stoichiometric matrix with entries S_{ij} (where each row corresponds to a species and the column corresponds to a reaction), and $\boldsymbol{r}$ is the reaction rates vector $\left(\begin{bmatrix} r_1 \\ r_2 \\ \dots \end{bmatrix}\right)$. Let us clear this with the help of following reactions:

$$\begin{cases} 2H_2 + O_2 \rightarrow 2H_2O \\ CO + \frac{1}{2}O_2 \rightarrow CO_2 \end{cases} \tag{8}$$

For these reactions the stoichiometric matrix $\boldsymbol{S}$will be as shown in **Figure 1**. Negative sign represent that these are reactants whose concentration is going to be reduced with time.

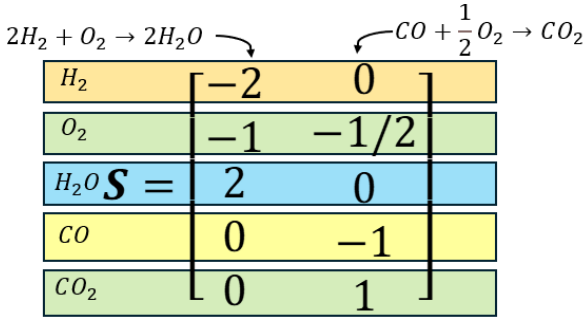


Figure 1. Formation of $\boldsymbol{S}$matrix.

And the reaction rates are as shown in vector r in Eqn. (9):

$$\boldsymbol{r} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \tag{9}$$

Key Assumptions in the Law of Mass Action formulation explained above are as follows:

- Reactants are uniformly distributed, so their concentrations are well-defined and do not vary spatially.
- The reaction proceeds in a single step, with no intermediate steps or complex mechanisms.
- The rate of reaction is proportional to the product of the reactant concentrations.

3. Python modelling

Following is the algorithm to model the above mathematics in Python:

i. Define the Inputs:

- Stoichiometric Matrix ($\boldsymbol{S}$): Describes how each species participates in each reaction. Rows represent species, columns represent reactions.
- Reaction Orders (R): Specifies the order of each reactant in each reaction. Typically derived from the reaction mechanism.
- ReactionRate Constants (k_j): Define how fast each reaction occurs.
- Initial concentrations (c_o): starting concentration of all the species.

ii. Define the Rate Equation:

- For each reaction j, calculate the reaction rate r_j for Eqn. 3. For the sake of programming use the following version of Eqn. 3:

$$r_j = k_j \prod_i c_i^{R_{ij}} \tag{10}$$

- For every reaction, iterate over all species, multiply their concentrations raised to their respective reaction orders.

iii. System of ODEs:

- The rate of change of species concentrations is computed using Eqn. 7.

iv. Simulate the Reaction:

- The ODE system is solved using solve_ivp for high accuracy.

v. Plot Results:

- The concentrations of all species are plotted over time to observe their dynamics.

Based on the above algorithm the following function is developed:

```
def mass_action_solver(S, κ, co, R, time_span, time_points):
"""
Solver for systems of chemical reactions using the law of mass action.

Inputs:
    - S (2D array): Matrix where rows represent species and columns represent reactions.
    - κ (1D array): Array of reaction rate constants (k_j) for each reaction.
    - co (1D array): Initial concentrations of all species.
    - R (2D array): Matrix with orders of reactants for each reaction.
    - time_span (tuple): Start and end times for the simulation (t0, tf).
    - time_points (1D array): Array of time points to evaluate the solution.

    Returns:
    - solution (object): Solution object from "solve_ivp".
    """

# Step 1: Define the reaction rate equations
def reaction_rates(c):
"""
        Compute the rate of each reaction based on the law of mass action.

        Parameters:
        - concentrations (1D array): Current concentrations of all species.

        Returns:
        - rates (1D array): Reaction rates for all reactions.
        """
        r = empty(len(κ))
for j in range(len(κ)):  # For each reaction
            r[j] = κ[j]  # Start with k_j
for i in range(len(c)):  # For each species
# Multiply by c_i raised to the power of its reaction order
                r[j] *= c[i] ** R[i, j]

return array(r)

# Step 2: Define the system of ODEs
def odes(t, c):
"""
        Compute the rate of change of concentrations for all species.

        Parameters:
        - t (float): Current time (not used here but required by ODE solver).
        - concentrations (1D array): Current concentrations of all species.

        Returns:
```

```
        - dCdt (1D array): Rate of change of concentrations.
        """
        r = reaction_rates(c)  # Compute reaction rates
    return S @ r  # Matrix multiplication S * r

# Step 3: Solve the ODE system
    soln = solve_ivp(odes, time_span, co, t_eval=time_points, method='RK45')
return soln
```

The function call solve_ivp(odes, time_span, co, t_eval=time_points, method='RK45') is part of the **SciPy** library's integrate module. It is used to solve systems of **ordinary differential equations (ODEs)**. The solve_ivp function numerically integrates a system of first-order ODEs over a specified time range. It produces the solution for each species at discrete time points based on the defined ODE system and initial conditions. The segmented explanation of the functions arguments is as follows:

- **odes** (Function)**:** This is the function that defines the system of ODEs to solve.
- **time_span** (Tuple)**:** Specifies the start and end times for the integration.
- **co** (Initial Conditions)**:** An array representing the initial concentrations of all chemical species in the system.
- **t_eval=time_points:**Specifies the specific time points at which the solution is desired.
- **method='RK45':** Defines the numerical integration method to be used. RK45 stands for Runge-Kutta method of order 4(5). It is an adaptive step-size method, meaning the solver dynamically adjusts the time step to balance accuracy and computational efficiency.It is suitable for non-stiff systems of ODEs and is the default method for solve_ivp.

The above input arguments are also tabulated in **Table 2.** for better understanding of coding aspects.

Table 2. Description of Inputs to mass_action_solver() Function.

Parameter	Description	Data Type / Shape	Units
S	Stoichiometric matrix: rows = species, columns = reactions	2D NumPy array (m×n)	Dimensionless
κ (kappa)	Reaction rate constants for each reaction	1D NumPy array (n,)	Depends on reaction order
co	Initial concentrations of each species	1D NumPy array (m,)	mol/L or any consistent unit
R	Reaction order matrix: order of each species in each reaction	2D NumPy array (m×n)	Dimensionless
time_span	Start and end times of simulation	Tuple (t0, tf)	seconds (s) or consistent unit
time_points	Discrete time points to evaluate the solution	1D NumPy array (k,)	seconds (s)

The result of solve_ivp is a solution object which primarily contain:

- **soln.t:**The data points in the time at which the concentrations are computed.
- **soln.y:**A 2-D array where each row corresponds to a species, and each column corresponds to the concentration at a specific time.

4. Code implementation

Now let's understand how to create the inputs for the function (**`mass_action_solver`**) with the help of chemical reactions given in Eqn. (8). The species present in all the reaction, along with the bifurcation of reaction, Reactant, and product is given in **Table. 2**.

Table 2. Species classification.

Species	Reaction	Is it a Reactant	Is it a Product
H_2	1	Yes	No
O_2	1 and 2	Yes	No
H_2O	1	No	Yes
CO	2	Yes	No
CO_2	2	No	Yes

Once this is finalized, the array for S has to be created by keeping the order of the species same as shown in **Table 2** (so that the programming remains flawless). The first column of the matrix is the first reaction, and the second column is the second equation. The reactant stoichiometric coefficients are negative, and products will be positive. The NumPy array for $\boldsymbol{S}$ is shown below:

```
S = array([
    [-2, 0],    # H₂
    [-1, 0],    # O₂
    [1, -1],    # H₂O
    [0, -1],    # CO
    [0, 1]      # CO₂
])
```

Then the array for the reaction rate constant (k_j) is to be created, which is a 1-D array having the same number of elements as the number of reactions. Let say that the rate of first reaction is 1.0 and that of second one is 0.5 units, respectively. The array will look like as follows:

```
# Reaction rate constants (k_j)
k = array([1.0, 0.5])
```

After this the initial concentration array (c_o) has to be created. This is again a 1-D array with the number of elements as the number of species. The array will look like as follows:

```
# Initial concentrations (c₀) of species [H₂, O₂, H₂O, CO, CO₂]
co = array([1.0, 1.0, 0.0, 1.0, 0.0])
```

Once this is over then the array for reaction order (R_{ij}) has to be created. This going to be a 2-D array as two equations are involved in it. First column for the first reaction and the second column will be the second reaction. Only the reactant stoichiometric coefficient (with positive sign) to be written for both the reactions and the products will be zero. The array will look very similar to the array for $\boldsymbol{S}$, as shown below:

```
S = array([
    [2, 0],    # H₂
    [1, 0],    # O₂
    [0, 1],    # H₂O
    [0, 1],    # CO
    [0, 0]     # CO₂
])
```

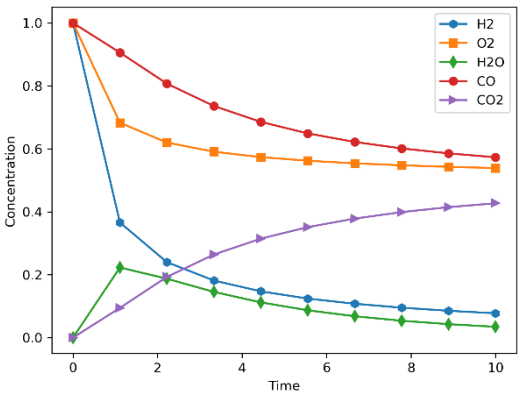
Once this is over, the time span of the reaction i.e. the starting and the ending time has to be supplied to the reaction along with the number of data points in between the time span. The time span will be a tuple and the data points will be created with the help of inbuilt NumPy array function *linspace*.The program statements will be as follows:

```
# Time span and time points
time_span = (0, 10)  # Start and end times
time_points = linspace(0, 10, 10)  # Data Points
```

After this, call the function which will solve for the concentration of species at each time step and then the solution can be plotted as well as tabulated (using Pandas DataFrame) by using the following code:

```
# Callin the function to solve the system
#---------------------------
soln = mass_action_solver(S, κ, co,R, time_span, time_points)

# Data Plotting
#---------------------------
figure(1, dpi =300)
species_labels = ['H2', 'O2', 'H2O', 'CO', 'CO2']
marker = ['h','s','d','o','>','*','<']
for i inrange(len(co)):
    plot(soln.t, soln.y[i],f'-{marker[i]}', label=species_labels[i])

xlabel('Time')
ylabel('Concentration')
legend()
grid()
show()

# Saving result in CSV
#------------------------
data = {'Time': soln.t}
for i, label inenumerate(species_labels):
    data[label] = soln.y[i]

df = DataFrame(data)

# Save to CSV
df.to_csv('species_concentrations.csv', index=False)
```

A proper care has to be taken to create the labels for the species, otherwise the wrong result will be reflected for them. The species labels in the list should be in the same order as in the concentration array. The plot and the data in the data fill will look as shown in the **Figure 2**.



(a)Concentration vs Time graph

	A	B	C	D	E	F
1	Time	H2	O2	H2O	CO	CO2
2	0	1	1	0	1	0
3	1.111	0.366	0.683	0.223	0.906	0.094
4	2.222	0.240	0.620	0.187	0.807	0.193
5	3.333	0.181	0.591	0.145	0.736	0.264
6	4.444	0.147	0.574	0.112	0.686	0.314
7	5.556	0.124	0.562	0.087	0.649	0.351
8	6.667	0.108	0.554	0.068	0.622	0.378
9	7.778	0.095	0.548	0.054	0.601	0.399
10	8.889	0.085	0.543	0.043	0.585	0.415
11	10.000	0.077	0.539	0.034	0.573	0.427

(b)Data saved in Excel

Figure 2. Plot and Excel file created by the code.

Now let us take one more example to demonstrate the strength of code. Consider the system of chemical reactions which are typically seen a combustion reaction (Ref. Eqn. 8).

$$\begin{cases} 2CH_4 + O_2 \rightarrow CO_2 + 2H_2O \\ CO_2 + H_2 \rightarrow CH_4 + O_2 \\ CH_4 + H_2O \rightarrow CO + 3H_2 \\ CO + H_2 \rightarrow CH_3OH \end{cases} \quad (8)$$

Consider the rate constants (for equations) along with the initial concentrations (of species) as given below:

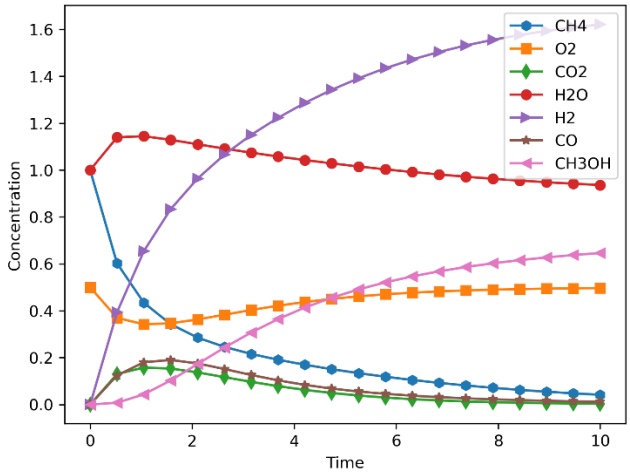
$$k = [1.0, 0.5, 0.3, 0.8]$$

$$c_o[CH_4, O_2, CO_2, H_2O, H_2, CO, CH_3OH] = [1.0, 0.5, 0.0, 1.0, 0.0, 0.0, 0.0]$$

Then the main program to handle this problem will be as follows:

```
# Example 2
# Define the stoichiometric matrix (S)

S = array([
    [-2, 1, -1, 0],    # CH4
    [-1, 1,  0, 0],    # O2
    [1, -1,  0, 0],    # CO2
    [2,  0, -1, 0],    # H2O
    [0,  1,  3,-1],    # H2
    [0,  0,  1,-1],    # CO
    [0,  0,  0, 1]     # CH3OH
])

# (k_j) for reactions
κ = array([1.0,0.5,0.3,0.8])

# (c₀) [CH4, O2, CO2, H2O, H2, CO, CH3OH]
co = array([1.0,0.5,0.0,1.0,0.0,0.0,0.0])

# Reaction orders (R)
R = array([
    [2, 0, 1, 0],    # CH4
    [1, 0, 0, 0],    # O2
    [0, 1, 0, 0],    # CO2
    [0, 0, 1, 0],    # H2O
    [0, 1, 0, 1],    # H2
    [0, 0, 0, 1],    # CO
    [0, 0, 0, 0]     # CH3OH
])

# Time span and time points
time_span = (0, 10)  # Start and end times
time_points = linspace(0, 10, 20)  #Data Points

# Solve the system
soln = mass_action_solver(S, κ, co,R, time_span, time_points)

# Plot the results
figure(1, dpi =300)
species_labels = ['CH4', 'O2', 'CO2', 'H2O', 'H2', 'CO', 'CH3OH']
marker = ['h','s','d','o','>','*','<','P']
for i inrange(len(co)):
    plot(soln.t, soln.y[i],f'-{marker[i]}', label=species_labels[i])

# plt.title('Concentration of Species Over Time')
xlabel('Time')
ylabel('Concentration')
legend()
show()
```

The plot along with the output data table is shown in **Figure 3**.



Concentration vs Time graph

	A	B	C	D	E	F	G	H
1	Time	CH4	O2	CO2	H2O	H2	CO	CH3OH
2	0	1	0.5	0	1	0	0	0
3	0.526	0.602	0.370	0.130	1.139	0.393	0.124	0.008
4	1.053	0.434	0.342	0.158	1.145	0.655	0.180	0.043
5	1.579	0.343	0.346	0.154	1.129	0.833	0.190	0.103
6	2.105	0.286	0.363	0.137	1.110	0.964	0.176	0.173
7	2.632	0.246	0.383	0.117	1.092	1.066	0.152	0.243
8	3.158	0.216	0.403	0.097	1.074	1.150	0.126	0.307
9	3.684	0.191	0.422	0.078	1.058	1.223	0.103	0.365
10	4.211	0.170	0.437	0.063	1.043	1.287	0.084	0.414
11	4.737	0.151	0.451	0.049	1.029	1.342	0.068	0.456
12	5.263	0.134	0.462	0.038	1.015	1.392	0.056	0.491
13	5.789	0.119	0.470	0.030	1.003	1.434	0.046	0.521
14	6.316	0.105	0.477	0.023	0.992	1.472	0.038	0.547
15	6.842	0.092	0.483	0.017	0.981	1.504	0.032	0.568
16	7.368	0.081	0.487	0.013	0.972	1.532	0.026	0.587
17	7.895	0.071	0.490	0.010	0.963	1.556	0.022	0.602
18	8.421	0.063	0.492	0.008	0.955	1.576	0.019	0.616
19	8.947	0.055	0.494	0.006	0.948	1.594	0.016	0.628
20	9.474	0.048	0.496	0.004	0.942	1.609	0.014	0.638
21	10.000	0.042	0.497	0.003	0.936	1.621	0.012	0.646

Data saved in Excel

Figure 3. Concentration plot and output excel file created by the code for combustion reaction.

5. Model validation and comparison with analytical results

To validate the accuracy of the proposed Python model, its results were compared against analytical solutions for a simple first-order irreversible reaction:

$$A \rightarrow B \tag{9}$$

governed by the rate law:

$$\frac{d[A]}{dt} = -k[A] \Rightarrow [A](t) = [A]_0 e^{-kt} \tag{10}$$

Using the mass_action_solver(), the numerical solution was computed with $k = 1.0$ and $[A]_0 = 1.0$. The concentrations of A over time matched the analytical expression closely, with relative error under 0.5% across all time points. This agreement confirms that the solver accurately captures the kinetics for systems where analytical benchmarks are available. Additional confidence is gained from consistency with trends reported in literature studies on combustion and biochemical systems[27,28]. For better understanding one can also view **Figure 4**.

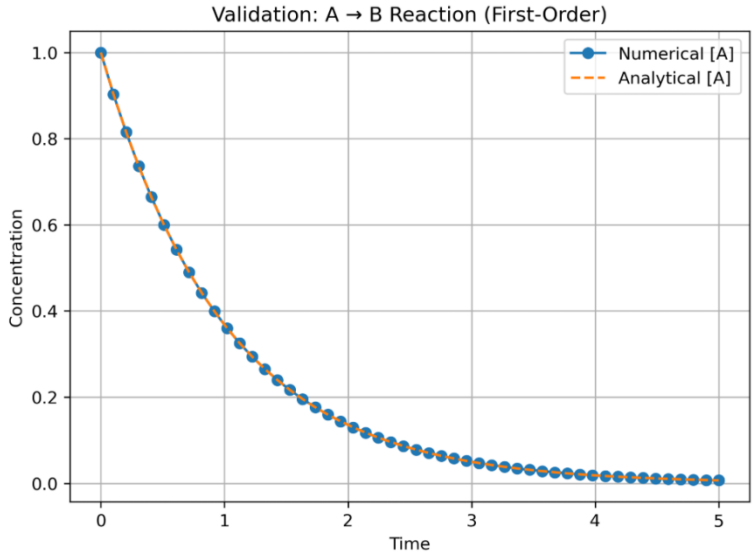


Figure 4. Validation of the python model using a first-order irreversible reaction A→B.

6. Conclusion

This article presents a comprehensive methodology for modelling chemical reaction kinetics using the "Law of Mass Action". By deriving the governing equations and implementing them in Python, a flexible and generalized structure was developed. This structure simplifies the computational modelling of chemical systems thereby allowing the users to solve the reaction networks of varying complexity with slight changes. The two example problems demonstrated the solver's flexibility while solving both simple and complex reaction systems. Also, providing the insights into the temporal behaviour of species concentrations. The approach not only emphasizes the precision of computational tools in chemical engineering but also emphasizes the importance of integrating mathematical modelling with the modern programming techniques. The generalized model developed in this study can be practically applied across a wide spectrum of chemical processes. In combustion systems, it can be used to simulate fuel oxidation mechanisms involving multiple intermediate species. In the pharmaceutical industry, it is useful for modelling drug synthesis reactions, enzyme kinetics, and metabolic pathways. Environmental engineers can employ this model to understand pollutant degradation or atmospheric chemistry. Additionally, it can serve as a pedagogical tool in academic settings, helping students grasp the dynamic behaviour of multi-species reactions through code-based simulations. The flexibility to customize reaction networks, rate constants, and reaction orders makes this solver a valuable asset for both industrial applications and educational curricula. Future work could expand this Python function to include additional complexities, such as temperature-dependent rate constants, equilibrium constraints, and stochastic simulations. This study will serve as a foundational reference for researchers and engineers aiming to influence the computational methods for chemical reaction modelling.

Conflict of interest

The authors declare no conflict of interest.

References

1. F. Horn, R. Jackson, General mass action kinetics, Arch. Ration. Mech. Anal. 47 (1972) 81–116. https://doi.org/10.1007/BF00251225.
2. Vasudevan, A., Aanisha, A. C., Mohammad, S. I., Manoharan, R., Raja, N., Oqilat, O., & Alshurideh, M. T. (2025). Divided square divisor cordial and Fibonacci prime labeling of theta graphs in Python. Applied Mathematics and Information Sciences, 19(1), 149–159. https://doi.org/10.18576/amis/190113.
3. M. Järvinen, V.V. Visuri, E.P. Heikkinen, A. Kärnä, P. Sulasalmi, C. De Blasio, T. Fabritius, Law of mass action based kinetic approach for the modelling of parallel mass transfer limited reactions: Application to metallurgical systems, ISIJ Int. 56 (2016) 1543–1552. https://doi.org/10.2355/isijinternational.ISIJINT-2016-241.
4. L.P. De Oliveira, D. Hudebine, D. Guillaume, J.J. Verstraete, A Review of Kinetic Modeling Methodologies for Complex Processes, Oil Gas Sci. Technol. 71 (2016). https://doi.org/10.2516/ogst/2016011.
5. W. Ji, F. Richter, M.J. Gollner, S. Deng, Autonomous kinetic modeling of biomass pyrolysis using chemical reaction neural networks, Combust. Flame 240 (2022) 111992. https://doi.org/https://doi.org/10.1016/j.combustflame.2022.111992.
6. J. Bauermann, S. Laha, P.M. McCall, F. Jülicher, C.A. Weber, Chemical Kinetics and Mass Action in Coexisting Phases, J. Am. Chem. Soc. 144 (2022) 19294–19304. https://doi.org/10.1021/jacs.2c06265.
7. J. Clow, G. Tzimpragos, D. Dangwal, S. Guo, J. McMahan, T. Sherwood, A pythonic approach for rapid hardware prototyping and instrumentation, 2017 27th Int. Conf. F. Program. Log. Appl. FPL 2017 (2017). https://doi.org/10.23919/FPL.2017.8056860.
8. G. Van Rossum, others, Python Programming Language., in: USENIX Annu. Tech. Conf., 2007: pp. 1–36.
9. Y.C. Huei, Benefits and introduction to python programming for freshmore students using inexpensive robots, in: Proc. IEEE Int. Conf. Teaching, Assess. Learn. Eng. Learn. Futur. Now, TALE 2014, 2015: pp. 12–17. https://doi.org/10.1109/TALE.2014.7062611.
10. A. Holkner, J. Harland, Evaluating the dynamic behaviour of Python applications, Conf. Res. Pract. Inf. Technol. Ser. 91 (2009) 19–27.
11. C. Bauckhage, NumPy / SciPy Recipes for Data Science: Subset-Constrained Vector Quantization via Mean Discrepancy Minimization, (2020) 1–4.
12. S. Van Der Walt, S.C. Colbert, G. Varoquaux, The NumPy array: A structure for efficient numerical computation, Comput. Sci. Eng. 13 (2011) 22–30. https://doi.org/10.1109/MCSE.2011.37.

13. K. Gajula, V. Sharma, B. Sharma, D.R. Mishra, P. Dumka, Modelling of Energy in Transit Using Python, Int. J. Innov. Sci. Res. Technol. 7 (2022) 1152–1156.
14. R. Johansson, Numerical python: Scientific computing and data science applications with numpy, SciPy and matplotlib, Second edition, Apress, Berkeley, CA, 2018. https://doi.org/10.1007/978-1-4842-4246-9.
15. C. Fuhrer, O. Verdier, J.E. Solem, C. Führer, O. Verdier, J.E. Solem, Scientific Computing with Python. High-performance scientific computing with NumPy, SciPy, and pandas, Packt Publishing Ltd, 2021.
16. J. Ranjani, A. Sheela, K. Pandi Meena, Combination of NumPy, SciPy and Matplotlib/Pylab-A good alternative methodology to MATLAB-A Comparative analysis, in: Proc. 1st Int. Conf. Innov. Inf. Commun. Technol. ICIICT 2019, 2019: pp. 1–5. https://doi.org/10.1109/ICIICT1.2019.8741475.
17. G.R. Kanagachidambaresan, G. Manohar Vinoothna, Visualizations, in: K.B. Prakash, G.R. Kanagachidambaresan (Eds.), EAI/Springer Innov. Commun. Comput., Springer International Publishing, Cham, 2021: pp. 15–21. https://doi.org/10.1007/978-3-030-57077-4_3.
18. V. Porcu, Matplotlib, in: Python Data Min. Quick Syntax Ref., Apress, Berkeley, CA, 2018: pp. 201–234. https://doi.org/10.1007/978-1-4842-4113-4_10.
19. E. Bisong, Matplotlib and Seaborn, in: Build. Mach. Learn. Deep Learn. Model. Google Cloud Platf., Apress, Berkeley, CA, 2019: pp. 151–165. https://doi.org/10.1007/978-1-4842-4470-8_12.
20. J.D. Hunter, Matplotlib: A 2D graphics environment Computing in Science & Engineering 9 (3): 90-95, (2007).
21. W. McKinney, Python for data analysis: Data wrangling with Pandas, NumPy, and IPython, " O'Reilly Media, Inc.," 2012.
22. Patel, V., Judal, K. B., Panchal, H., Singh, B., Jomde, A., Kumar, A., Patel, A., Jain, R., & Sadasivuni, K. K. (2023). Investigation on drying kinetics analysis of gooseberry slices dried under open sun. Environmental Challenges, 13, 100778. https://doi.org/10.1016/j.envc.2023.100778
23. Kumar, M., Sahdev, R. K., Tiwari, S., Manchanda, H., Chhabra, D., Panchal, H., & Sadasivuni, K. K. (2021). Thermal performance and kinetic analysis of vermicelli drying inside a greenhouse for sustainable development. Sustainable Energy Technologies and Assessments, 44, 101082. https://doi.org/10.1016/j.seta.2021.101082
24. L. Adleman, M. Gopalkrishnan, M.D. Huang, P. Moisset, D. Reishus, On the mathematics of the law of mass action, A Syst. Theor. Approach to Syst. Synth. Biol. I Model. Syst. Charact. (2014) 3–46. https://doi.org/10.1007/978-94-017-9041-3_1.
25. E.O. Voit, H.A. Martens, S.W. Omholt, 150 Years of the Mass Action Law, PLoS Comput. Biol. 11 (2015) 1–7. https://doi.org/10.1371/journal.pcbi.1004012.
26. A. Van Der Schaft, S. Rao, B. Jayawardhana, On the mathematical structure of balanced chemical reaction networks governed by mass action kinetics, SIAM J. Appl. Math. 73 (2013) 953–973. https://doi.org/10.1137/11085431X.
27. M. Balat, Biomass Energy and Biochemical Conversion Processing for Fuels and Chemicals, Energy Sources, Part A Recover. Util. Environ. Eff. 28 (2006) 517–525. https://doi.org/10.1080/009083190927994.
28. N. Adi Sasongko, N. Gunadi Putra, M.L. Donna Wardani, Review of types of biomass as a fuel-combustion feedstock and their characteristics, Adv. Food Sci. Sustain. Agric. Agroindustrial Eng. 6 (2023) 170–184. https://doi.org/10.21776/ub.afssaae.2023.006.02.8.