

## **ПРОГРАММНЫЙ КОД**

Системный анализ процессов химической технологии

Лабораторная работа №8

«Идентификация кинетических параметров при  
моделировании химических реакций»

# Листинги

1	Описание кинетики с помощью матрицы (файл <code>kinetic.py</code> ) . . . . .	3
2	Подбор кинетических параметров с помощью методов оптимизации функций нескольких переменных (файл <code>main.py</code> ) . . . . .	5
3	Исходные данные (файл <code>data.txt</code> ) . . . . .	6

# Листинг 1 – Описание кинетики с помощью матрицы (файл kinetic.py)

```

1 import numpy as np
2 from scipy.integrate import solve_ivp
3
4
5 kinetic_matrix = np.array(
6     [
7         [-1, 1, 0, 1],
8         [0, -1, 1, 1]
9     ]
10 )
11
12
13 def kinetic_scheme(
14     time: float,
15     c: np.ndarray,
16     k: np.ndarray,
17     stoich_matrix: np.ndarray = kinetic_matrix
18 ) -> np.ndarray:
19
20     mask = stoich_matrix < 0
21     p = (c ** -(stoich_matrix * mask)).prod(axis=1)
22     reaction_rates = p * k
23
24     right_parts = (
25         (stoich_matrix.T * reaction_rates).sum(axis=1)
26     )
27
28     return right_parts
29
30
31 def main() -> None:
32     """
33     Функция для тестирования и проверки
34     работоспособности кода.
35     Выводит значения правых частей
36     дифференциальных уравнений.
37     """
38     c = np.array([1, 0, 0, 0])
39     k = np.array([.85, .5])
40     time = .1
41     print(kinetic_scheme(time, c, k))
42     return
43
44
45 def test_kinetic_constants() -> None:
46     """
47     Функция для проверки констант скоростей.
48     Выводит значения концентраций компонентов
49     в каждый момент времени.
50     """
51     time = np.arange(0, 1.1, .1)
52     k = np.array([1.7939239513204417, 1.0262965367591952])
53     solution = solve_ivp(
54         fun=kinetic_scheme,
55         t_span=(time[0], time[-1]),
56         y0=[1, 0, 0, 0],
57         dense_output=True,
58         args=(k, )
59     )
60     c9h20, c9h18, c9h16, h2 = solution.sol(time)
61     for line in zip(c9h20, c9h18, c9h16, h2):
62         print('{:>8.4f} {:>8.4f} {:>8.4f} {:>8.4f}'.format(*line))
63
64     return
65
66

```

```
67 if __name__ == '__main__':  
68     #main()  
69     test_kinetic_constants()
```

Листинг 2 – Подбор кинетических параметров с помощью методов оптимизации функций нескольких переменных (файл main.py)

```

1 import numpy as np
2 from scipy.optimize import minimize
3 from scipy.integrate import solve_ivp
4 from kinetic import kinetic_scheme
5 import genetic_algorithm as ga
6
7
8 def obj_func(
9     k: np.ndarray, c: np.ndarray,
10    kinetic_scheme: callable,
11    time: np.ndarray,
12    c0: np.ndarray,
13 ) -> float:
14     """
15     Целевая функция для минимизации
16     """
17     solution = solve_ivp(
18         fun=kinetic_scheme,
19         t_span=(0, time[-1]),
20         y0=c0,
21         dense_output=True,
22         args=(k, )
23     )
24     c_calc = solution.sol(time)
25     return ((c - c_calc[:, 1:].T) ** 2).sum()
26
27
28 if __name__ == '__main__':
29     data = np.loadtxt('data.txt')
30     h = .1
31     c0 = np.array([1, 0, 0, 0])
32     time = np.arange(0, 1+h, h)
33
34     #С использованием метода Нелдера-Мида
35     res_nm = minimize(
36         obj_func,
37         x0=[.5, .5],
38         args=(data, kinetic_scheme, time, c0),
39         method='Nelder-Mead'
40     )
41     print(f'НелдерМид-: fun={res_nm.fun} x={res_nm.x}')
42
43     #С использованием генетического алгоритма
44     bounds = (.5, 2.), (.5, 2.)
45     res_ga = ga.genetic_algorithm(
46         bounds=bounds,
47         func=obj_func,
48         popsize=100,
49         selection_size=20,
50         generations_count=10,
51         args=(data, kinetic_scheme, time, c0)
52     )[0]
53     fun = obj_func(res_ga, data, kinetic_scheme, time, c0)
54     print(f'Генетический алгоритм: fun={fun} x={res_ga}')

```

Листинг 3 – Исходные данные (файл data.txt)

0.8353	0.1563	0.0084	0.1732
0.6977	0.2715	0.0308	0.3331
0.5827	0.354	0.0633	0.4805
0.4868	0.4104	0.1028	0.6161
0.4066	0.4463	0.1471	0.7405
0.3396	0.4662	0.1942	0.8546
0.2837	0.4736	0.2427	0.9591
0.2369	0.4716	0.2915	1.0545
0.1979	0.4625	0.3396	1.1417
0.1653	0.4481	0.3866	1.2213