DOI https://doi.org/10.30525/978-9934-26-506-8-203

THERMAL ENGINEERING CALCULATIONS AND RESULTS VISUALIZATION USING MATLAB AND PYTHON

ТЕПЛОТЕХНІЧНІ РОЗРАХУНКИ ТА ВІЗУАЛІЗАЦІЯ РЕЗУЛЬТАТІВ З ВИКОРИСТАННЯМ MATLAB І PYTHON

Miroshnychenko S.O.,
Senior Teacher,
LLC "Technical university "Metinvest polytechnic",
Zaporizhzhia, Ukraine

Мірошниченко С.О.,
старший викладач,
ТОВ «Технічний університет «Метінвест політехніка»,
м. Запоріжжя, Україна

Сучасні методи дослідження процесів передачі теплової енергії потребують ефективних інструментів для математичного моделювання та візуалізації результатів, які б забезпечували необхідну доступність та гнучкість при використанні. Натепер достатньо широке розповсюдження для вказаних задач отримали мови програмування Matlab [1] та Python [2]. У зв'язку з цим проведено порівняльний аналіз вказаних засобів на прикладі реалізації моделі процесу конвективного теплообміну, спрощений математичний опис якої за законом Ньютона-Ріхмана має вигляд:

$$\frac{dT}{dt} = -\alpha(T - T_0),$$

де T – температура тіла, К; T_0 – температура навколишнього середовища, К; t – час, хв.

Для розв'язання диференціального рівняння конвективного теплообміну в Python використано метод *odeint* [3], що міститься в бібліотеці *scipy.integrate* [4], тоді як у Matlab застосовано вбудований метод *ode45* [5].

При порівняльному аналізі процесу та результатів моделювання виявлено особливості за наступними ознаками. З точки зору ергономічності мови програмування Matlab має більш компактний спеціалізований синтаксис для роботи з масивами та матрицями (рис.1, а), в той час як Python потребує імпорту додаткових бібліотек, але надає можливість налаштувати реалізацію під конкретні потреби завдання моделювання (рис. 1, б).

Зокрема, метод *odeint* для розв'язання диференціальних рівнянь використовує алгоритм LSODA, який має можливість автоматично перемикатися між методами для жорстких систем та нежорстких систем, тоді як *ode45* базується на методі Рунге-Кутта-Фельберга 4-5 порядку, який краще працює з нежорсткими системами. При контролі точності розрахунків *odeint* контролює локальну похибку усічення, в той час як *ode45* за вибором користувача може здійснювати контроль як абсолютної, так і відносної похибки обчислень.

Newtons_Law_of_Cooling.m

```
% Визначення температури об'єкта з часом при охолодженні від 400 С до нормальних умов (20 °C)
% Вхідні дані:
t_start = 0; t_stop = 100; t10 = 10;  % Час початку, закінчення та для визначення температури відповідно, хв.
T0 = 400; step_number = 1000; % Початкова температура, °C та кількість кроків.
tRange = linspace(t_start, t_stop, step_number); % змінна для інтервалу часу t=0 до 100 хвилин
[tSol, TSol] = ode45(@sinterODEfun, tRange, T0); % Виклик ode45, щоб розв'язати диференціальне рівняння
[~, idx] = min(abs(tSol - t10)); T_10min = TSol(idx); % Пошук температури через 10 хвилин
plot(tSol, TSol, "r", 'DisplayName', 'зміна температури від часу') % Графік охолодження від 400 °C до нормальних умов
hold on; scatter(t10, T_10min, 'r', 'filled');
text(t10+2, T_10min, sprintf('%.1f °C - температура через 10 хв охолодження', T_10min), ...
    'Color', 'red', 'HorizontalAlignment', 'left');
plot([t_start t10], [T_10min T_10min], 'r--', 'LineWidth', 0.7);
plot([t10 t10], [t_start T_10min], 'r--', 'LineWidth', 0.7);
title("Охолодження об'єкта (Matlab)") % Опис та налаштування графіка
legend("зміна температури від часу"); xlabel("час (хвилини)"); ylabel("температура (°C)")
xlim([t_start t_stop]); ylim([t_start T0]); hold off;
function dTdt = sinterODEfun(t,T) % Локальна функція що імплементує Закон Ньютона – Ріхмана
    a = 0.058;  % Коефіцієнт тепловіддачі, Вт/м²/К.
    Tn = 20;  % Температура при нормальних умовах, °C.
    dTdt = - a * (T - Tn);
end
```

а)

Newtons_Law_of_Cooling.py

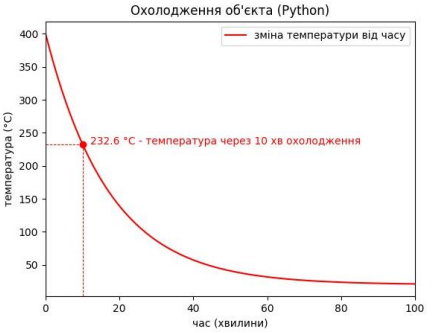
```
# Визначення температури об'єкта з часом при охолодженні від 400 °C до нормальних умов (20 °C)
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
# Вхідні дані:
t_start, t_stop, t10 = 0, 100, 10  # Час початку, закінчення та для визначення температури, хв.
T0, Tn = 400, 20  # Початкова температура та температура при нормальних умовах °C.
a, step_number = 0.058, 1000  # Коефіцієнт тепловіддачі, Вт/м²/К та кількість кроків.

def sinter_ode_fun(T, t):  # Локальна функція що імплементує Закон Ньютона – Ріхмана  1 usage  new *
    return - a * (T - Tn)

t_range = np.linspace(t_start, t_stop, step_number)  # Змінна для інтервалу часу t=0 до 100 хвилин
T_sol = odeint(sinter_ode_fun, T0, t_range)  # Виклик odeint, щоб розв'язати диференціальне рівняння
T_10min = T_sol[np.abs(t_range - t10).argmin()][0]  # Пошук температури через 10 хвилин
plt.plot( *args: t_range, T_sol, 'r', label='зміна температури від часу')  # Графік охолодження
plt.scatter(t10, T_10min, color='red')
plt.text(t10 + 2, T_10min, s: f'{T_10min:.1f} °C - температура через 10 хв охолодження', color='red', ha='left')
plt.axhline(y=T_10min, xmin=t_start, xmax=t10 / t_stop, color='red', linestyle='--', linewidth=0.7)
plt.axvline(x=t10, ymin=t_start, ymax=(T_10min - Tn) / (T0 - Tn), color='red', linestyle='--', linewidth=0.7)
plt.title("Охолодження об'єкта (Python)")  # Опис та налаштування графіка
plt.xlabel('час (хвилини)'); plt.ylabel('температура (°C)'); plt.xlim(left=t_start, right=t_stop); plt.show()
```

б)

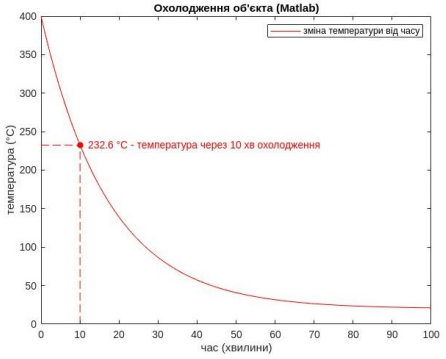
Рис. 1. Програмні реалізації спрощеної моделі конвективного теплообміну мовами програмування Matlab (а) і Python (б)

При візуалізації результатів досліджень обидва засоби надають потужні інструменти для побудови графіків: середовище Matlab має більш зручний інтерфейс для інтерактивного налаштування графіків, тоді як бібліотека Python Matplotlib [6] забезпечує більше можливостей для програмного налаштування візуалізації.

При моделюванні охолодження об’єкту впродовж 100 хвилин від початкової температури 400°C обидва засоби моделювання надали ідентичні результати (рис. 2), що підтверджує коректність реалізації математичної моделі та точність обчислень в обох випадках.



а)



б)

Рис. 2. Візуалізація результатів моделювання конвективного теплообміну засобами Python (а) і Matlab (б)

Проведений аналіз демонструє, що обидві мови програмування є ефективними інструментами для моделювання теплових процесів. Середовище Matlab надає більш розвинений математичний інструментарій із зручним інтерфейсом, пропонує простоту використання та готові рішення, на кшталт Simulink та Simscape, що робить його ідеальним для інженерів та практиків. Мова програмування Python, яка відрізняється більшою гнучкістю, відкритістю та можливістю безкоштовного використання, є відмінним вибором для дослідників, які потребують відкритого та розширювального коду. Вибір конкретного засобу моделювання може залежати від специфіки задачі та наявних ресурсів.

Перелік використаних джерел

1. MATLAB. *MathWorks – Maker of MATLAB and Simulink – MATLAB & Simulink*. URL: https://www.mathworks.com/products/matlab.html (date of access: 07.10.2024)

2. Welcome to Python.org. *Python.org*. URL: https://www.python.org/ (date of access: 17.10.2024).

3. odeint – SciPy v1.14.1 Manual. *Numpy and Scipy Documentation – Numpy and Scipy documentation*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html#scipy.integrate.odeint (date of access: 17.10.2024).

4. Integration and ODEs (scipy.integrate) – SciPy v1.14.1 Manual. *Numpy and Scipy Documentation – Numpy and Scipy documentation*. URL: https://docs.scipy.org/doc/scipy/reference/integrate.html (date of access: 17.10.2024).

5. ode45 – Solve nonstiff differential equations – mediumorder method – MATLAB. *MathWorks – Maker of MATLAB and Simulink – MATLAB & Simulink*. URL: https://www.mathworks.com/help/matlab/ref/ode45.html?s_tid=srchtitle_site_search_1_ode45 (date of access: 10.10.2024).

6. Matplotlib – Visualization with Python. *Matplotlib – Visualization with Python*. URL: https://matplotlib.org/ (date of access: 17.10.2024).