УДК 621.99:004.89
DOI: 10.31471/1993-9965-2024-2(57)-86-95

ВИКОРИСТАННЯ ЕКОСИСТЕМИ SciPy ДЛЯ АВТОМАТИЗАЦІЇ ПРОЄКТУВАННЯ ТЕХНОЛОГІЧНИХ ПРОЦЕСІВ ВИГОТОВЛЕННЯ РІЗЬБ

В. Б. Копей*, О. Р. Онисько, І. І. Чудик, І. В. Пронюк

ІФНТУНГ; 76019, м. Івано-Франківськ, вул. Карпатська, 15;
e-mail: v k o p e y @ g m a i l . c o m

На основі мови Python та екосистеми її пакетів SciPy розроблено програмні компоненти, призначені для автоматизації проєктування технологічних процесів виготовлення різьб, зокрема для розрахунку режимів обробки різьб, розрахунку основного часу на обробку різьб, вибору оптимальних режимів і методів їх виготовлення. Їх можна використати як компоненти САПР ТП та PLM-систем різьбових деталей. Запропоновано принципи розроблення таких компонентів. Розроблено функції, що повертають значення основного часу виготовлення різьб різними методами, та функції, що перетворюють їх у SymPy-рівняння, а також розв'язують ці рівняння в символьному вигляді. Це додає в Python можливості декларативного програмування та інтелектуалізує розв'язування багатьох задач вибору оптимальних методів і режимів обробки. Розроблено програмні засоби для автоматизації формування документації функцій, які містять опис аргументів та LaTeX-формули, що описують ці функції. Показані способи формалізації довідкових табличних даних для вибору режимів виготовлення різьб, у тому числі з використанням пакету Pandas. Запропоновані способи орієнтовані на зручні запити до бази даних, аналіз та обробку цих даних. Показано способи інтерполяції та візуалізації цих табличних даних для більш точного розрахунку режимів. Показано приклади використання розроблених компонентів в Jupyter Notebook.

Ключові слова: автоматизоване проєктування, обробка різьб, технологічний процес, Python, SymPy.

Based on the Python language and its SciPy package ecosystem, software components have been developed that are intended for automating the design of technological processes for thread manufacturing, in particular for calculating thread processing modes, calculating the main time for thread processing, and selecting optimal modes and methods for thread manufacturing. They can be used as components of CAPP and PLM systems for threaded parts. The principles for designing such components are presented. Functions that yield the value of the primary time for thread manufacturing using various ways have been written, as have functions that transform these into SymPy equations and solve these equations symbolically. This adds declarative programming capabilities to Python and intellectualizes the solution of many problems of selecting optimal methods and processing modes. Software tools have been developed to automate the formation of documentation for functions that contain a description of arguments and LaTeX formulas describing these functions. Methods for formalizing reference tabular data for selecting thread manufacturing modes, including using the Pandas package, are shown. The proposed methods are focused on convenient database queries, analysis and processing of this data. Methods of interpolation and visualization of this tabular data for more accurate calculation of modes are shown. Examples of using the developed components in Jupyter Notebook are shown.

Keywords: automated design, thread processing, technological process, Python, SymPy.

Вступ

Системи автоматизованого проєктування технологічних процесів (САПР ТП), які в англомовній літературі називають Computer-Aided Process Planning (CAPP), є невід'ємною частиною концепції Industry 4.0. Сьогодні САПР ТП повинні бути інтелектуальними системами з високим рівнем автономності, адаптивності, інтерактивності та широкими аналітичними можливостями [1, 2, 3]. Складність їх створення та використання, а також можливість їхньої інтеграції в інформаційні системи вищого рівня (PLM-системи) є важливими компонентами їхньої якості. Створення таких систем вимагає спеціальних підходів, методів і засобів сучасної програмної інженерії. Одним із таких засобів є мова програмування загального призначення Python та множина її пакетів.

Аналіз джерел та формулювання проблеми

Проєктування технологічних процесів завжди вимагає прийняття оптимальних рішень, наприклад за критерієм мінімального основного часу. Для цього потрібна ефективна обробка та різносторонній аналіз даних в САПР ТП, інтерактивна робота з даними, їхня візуалізація. Зокрема нерідко виникає необхідність перетво-

рити дані у графіки чи номограми [4], інтерполювати та апроксимувати табличні дані [5]. Велика кількість можливих методів і режимів обробки ускладнює розроблення САПР ТП та пошук таких рішень. Для розв'язування таких задач доцільно застосувати SciPy – екосистему програмного забезпечення з відкритим кодом для математичних, наукових та інженерних обчислень на основі мови Python [6]. Основними її складовими є пакет для роботи з масивами NumPy [7], пакет для чисельних обчислень SciPy [8], пакет символьної математики SymPy [9], графічний пакет Matplotlib [10], пакет аналізу даних Pandas [11], а також засоби інтерактивних обчислень – IPython та Jupyter [12].

Python нерідко використовується для інноваційних розробок в галузі автоматизації підготовки виробництва. Це можна пояснити її орієнтованістю на зручне прикладне програмування та наявністю великої кількості пакетів для математичних та інженерних завдань. Python застосовують для роботи з CAD-моделями та задач автоматичного розпізнавання елементів, що обробляються [13, 14, 15, 16], для моделювання виробничих середовищ [17], для створення розподілених виробничих систем [18], для роботи з технологічними базами знань [16, 19] та штучним інтелектом [20], а також для задач оптимізації [21].

За допомогою Python більш ефективною є розробка інноваційних інтелектуальних систем із дотриманням принципів теорії систем та принципів програмування (принципу модульності, ієрархічності, адаптивності та інших) та на основі таких підходів, як декларативне програмування [22, 23], машинне навчання [19] та мультиагентні системи [24].

Важливо також спростити створення та підвищити якість вбудованої документації САПР ТП. Як правило, вбудована Python-документація містить звичайний неформатований текст, що значно обмежує її інформативність. Для автоматизованого формування документації в форматі HTML існують такі інструменти, як Sphinx [25] (найбільш популярний і з широкими можливостями), pdoc [26] (простий у використанні), Pycco [28] (генерує документацію в стилі Literate programming [27]), MkDocs [29] (спеціалізується на використанні Markdown), Jupyter Book [30] (інтерактивна документація на основі Jupyter Notebook). Багато з них використовують такі мови розмітки, як reStructuredText, Markdown або MyST та підтримується сервісом розміщення документацій readthedocs.com. Проте такі Python-засоби, як inspect та SymPy, дозволяють легко розробити свій власний спосіб автоматизації документування з можливістю відображення звичайних Python-функцій як LaTeX-формул.

Метою дослідження є розроблення принципів створення компонентів системи автоматизованого проєктування технологічних процесів різьбових деталей, які базуються на мові Python та екосистемі її пакетів SciPy, призначених для прийняття інтелектуальних та оптимальних рішень під час вибору методів і режимів обробки різьб та є зручними під час створення, удосконалення і використання.

Результати дослідження

Розроблене програмне забезпечення складається з Python-модулів ThreadingT.py, ThreadingV.py, example2.py, Interpolate Example.py, Jupyter-документа Threading.ipynb та є вільним ПЗ з відкритим кодом (ліцензія GPL-3.0). Початковий код доступний на GitHub [31].

Для установлення і запуску програми на локальному комп'ютері знадобиться Python 3.8x64 з пакетами numpy-1.24.4, scipy-1.10.1, matplotlib-3.7.5, sympy-1.13.3, pandas-2.0.3. Можна також установити пакет Jupyter. Останні версії цих пакетів можна установити за допомогою команди установлення пакетів (необхідне підключення до Інтернет):

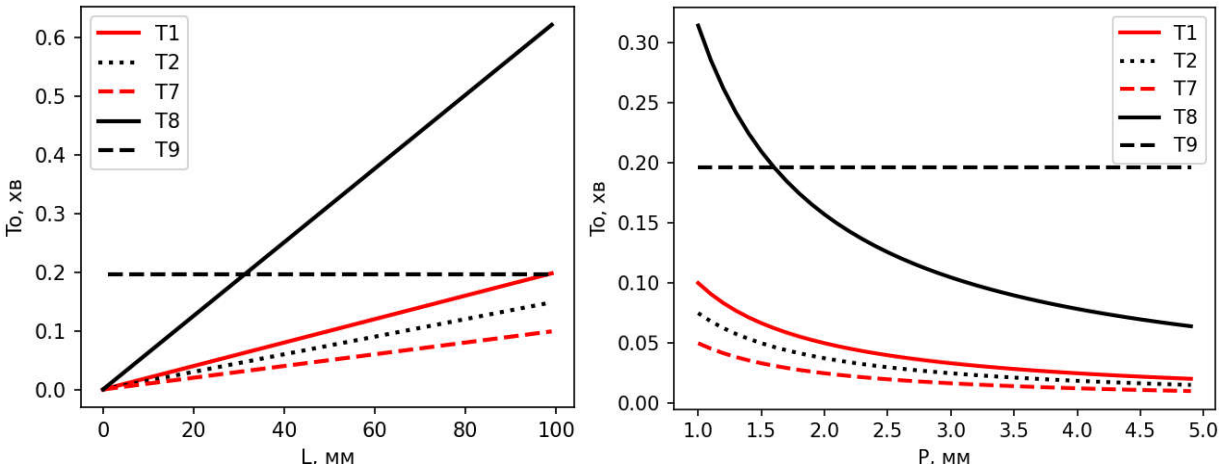
```
pip install numpy scipy matplotlib sympy pandas
```

Тепер можна ввести команду запуску модуля. Для прикладу:

```
python ThreadingT.py
```

Запуск програми також можливий через Google Colab або JupyterLite. Цей спосіб вимагає наявності сучасного веб-браузера та підключення до Інтернет. Для запуску документа Threading.ipynb потрібно відкрити в браузері сторінку https://jupyter.org/try-jupyter/lab, перетягнути у File Browser (ліворуч) файли ThreadingT.py, ThreadingV.py, example2.py, Threading.ipynb, запустити Threading.ipynb і виконати усі комірки (Run/Run All Cells).

Спочатку розглянемо модулі ThreadingT.py та example2.py. Python-модуль ThreadingT.py містить функції для розрахунку основного часу на обробку різьб, а також приклади їхнього використання. Python-модуль example2.py містить функції для перетворення Python-функцій у рівняння SymPy та формули LaTeX, а також засоби символьного розв'язування рівнянь, ліва частина яких є Python-функцією. Інтерактивний



а б

Рисунок 1 – Приклади залежностей основного часу To від довжини робочого ходу L (а) та кроку різьби P (б) для різних методів виготовлення різьби: T1 – мітчиком, T2 – мітчиком з прискореним реверсуванням, T7 – точінням, T8 – дисковою фрезою, T9 – гребінчастою фрезою

Jupyter-документ Threading.ipynb містить приклади використання усіх розроблених модулів.

В модулі ThreadingT.py [31] розроблено функції T1-T18, які повертають значення основного часу для різних методів виготовлення різьби. Для прикладу код функції T7() виглядає так:

```
def T7(L, P, n, g=1, i=1):
    """Повертає основний час для точіння різьби, хв"""
    return L*i*g/(P*n)
```

Аргументи функції вказані в круглих дужках в першому рядку. Другий рядок коду містить вбудовану документацію, що може бути автоматично розширена, як це буде показано нижче. В третьому рядку команда return повертає значення, яке є результатом обчислення аналітичного виразу.

Нерідко виникає задача візуально порівняти залежності Ti(X) для вибору оптимального методу обробки різьби за критерієм основного часу (рис. 1), де X може бути будь-яким аргументом цих функцій. Наприклад, щоб порівняти T7(L) і T9(L) – залежності основного часу від довжини робочого ходу L, якщо інші значення параметрів відомі, достатньо в оболонці Python виконати команди:

```
>>> from ThreadingT import *
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> L=np.arange(0.0, 100, 1) # масив
>>>  plt.plot(L, T7(L, P=1, n=1000), T9(d=10, S_z=0.5, Z=4, n_i=100)*L/L)
>>> plt.show()
```

Тут P – крок різьби (мм); d – діаметр різьби (мм); S_z – подача на зуб фрези (мм/зуб); Z – число зубів інструмента; n_i – частота обертання інструмента, хв$^{-1}$. Оскільки функція T7() отримує NumPy-масив значень L, то вона повертає масив значень часу. Ці масиви містять значення координат точок, за якими будуються криві. В даному випадку функція T9() не залежить від L і повертає скаляр, тому її необхідно помножити на масив L/L, щоб отримати масив з однаковими значеннями. З графіка видно, що метод обробки точінням (T7) забезпечує менший час, аніж фрезерування гребінчастою фрезою (T9), на інтервалі L від 0 до 100 мм. Також, видно, що в даному випадку і в заданих інтервалах L та P точіння є найкращим методом з усіх розглянутих за критерієм основного часу.

Розроблений модуль example2.py [31] містить засоби перетворення Python-функцій з аналітичними виразами, наприклад описаних вище функцій Ti(), у рівняння SymPy та формули LaTeX, а також засоби символьного розв'язування рівнянь, ліві частини яких є такими Python-функціями. Функція fn2sympy(f, **args) намагається перетворити функцію f в рівняння SymPy. Вона має один обов'язковий аргумент f (функція, що перетворюється) та необов'язкові іменовані аргументи args (аргументи функції f). Вона використовує getfullargspec() із стандартного модуля inspect для отримання списку аргументів функції f та перетворює їх у об'єкти-символи SymPy. Після чого функція повертає SymPy-рівняння, ліва частина якого є SymPy-символом з назвою f, а права –

ISSN 1993–9965 print
ISSN 2415–3524 online
Науковий вісник ІФНТУНГ
2024. № 2(57)

```
[5]: from ThreadingT import __doc__ # документація модуля
     # розширити документацію функцій Ti описом аргументів
     for f in [T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18]:
         f.__doc__+="\n\n$$"+fn2latex(f)+"$$\n"
         extdoc(f, doc2dict(__doc__))

[6]: # вивести розширену документацію функції T7
     from IPython.display import display, Markdown
     display(Markdown(T7.__doc__))
```

Повертає основний час для точіння різьби, хв

$$T_7 = \frac{Lgi}{Pn}$$

L - довжина робочого ходу, мм; P - крок різьби, мм; n - частота обертання заготовки або інструменту, хв g - число заходів різьби, що нарізується; i - число робочих ходів інструмента;

Рисунок 2 – Доповнення і візуалізація документації функції в Jupyter Notebook

SymPy-виразом, що використовується функцією `f` для обчислення значення. За допомогою `subs(args)` в цей вираз можна підставити будь-які значення його аргументів `args`.

Наприклад, перетворити в SymPy-рівняння можна функцію `T7`:

```
>>> from example2 import *
>>> eq1=fn2sympy(T7)
>>> eq1
Eq(T7, L*g*i/(P*n))
```

Або можна використати необов'язкові іменовані аргументи `args` і уточнити значення змінної `P`:

```
>>> fn2sympy(T7, P=2)
Eq(T7, L*g*i/(2*n))
```

Як буде показано нижче, отримані рівняння можуть бути використані для будь-яких символьних операцій з використанням SymPy, зокрема диференціювання, інтегрування, розв'язування рівнянь.

В Jupyter може знадобитись перетворити функцію `Ti()` у формулу LaTeX і відобразити її. Формули SymPy можна також візуалізувати за допомогою Unicode або MathML. Функція `fn2latex(f, **args)` використовує `fn2sympy()` та `sympy.latex()` для перетворення функції в формат LaTeX.

Ось приклад для `T7()`:

```
>>> fn2latex(T7)
'T_{7} = \\frac{L g i}{P n}'
```

Документація модуля `ThreadingT` (атрибут `__doc__`) містить перелік з описом величин, що використовується у функціях `Ti()`. Вона подана у вигляді рядків `<позначення> – <опис>`. Самі функції не містять цього опису, але його можна додати автоматично. Це запобігає дублюванню коду та відповідає відомому в програмній інженерії принципу DRY «Don't repeat yourself» [32]. Розроблена функція `doc2dict()` може перетворити такий текст у словник, в якому ключами є позначення, а значеннями є описи. А функція `extdoc(f, d)` доповнює документацію функції `f` інформацією про аргументи, якщо вона є в такому словнику `d`. Вона використовує функцію `getfullargspec()` стандартного модуля `inspect` для отримання списку аргументів функції.

Для прикладу, до застосування цих функцій документація `T7` містить тільки базову інформацію:

```
>>> T7.__doc__
'Повертає основний час для точіння різьби, хв'
```

Але можна доповнити документацію описом аргументів функції та формулою LaTeX. Тоді робота в Jupyter Notebook дозволяє бачити документацію як відформатований Markdown-текст з LaTeX-формулами та іншими елементами розмітки Markdown (рис. 2).

Функції `Ti()` дозволяють обчислення тільки основного часу. Для обчислення, наприклад, величини `n` з рівняння `T7=L*i*g/(P*n)` потрібно мати окрему функцію, що повертає `L*i*g/(P*T7)`. Для обчислення величин `L`, `i`, `g`, `P` теж потрібно мати окремі функції. Знову це суперечить принципу DRY. Але проблему можна вирішити шляхом застосування

засобів декларативного програмування [23]. Розроблена функція `fn_solve(f, v, **args)`, розв'язує рівняння `f(x, args)=v` та повертає список значень `x` з SymPy-виразами. Спочатку вона шляхом знаходження різниці множини аргументів функції `f` і своїх аргументів `args` визначає, яка змінна є коренем рівняння. Далі вона для знаходження кореня використовує `fn2sympy()` та функцію `solve()` пакету SymPy. Розв'язок містить список коренів у вигляді чисел, або символьних виразів.

Для прикладу знайдемо значення `n` з рівняння `T7(L=10, P=2, n, g=1, i=1) = 0.1`:

```
>>> fn_solve(T7, 0.1, L=10, P=2, g=1, i=1)
[50.0000000000000]
```

Або отримаємо вираз для обчислення `n`, якщо `P` є символом:

```
>>> fn_solve(T7, 0.1, L=10, P=sympy.S("P"), g=1, i=1)
[100.0/P]
```

Знайдемо вирази для обчислення значень `L` та `P`, за яких методи `T7` та `T9` забезпечують однаковий основний час. Такі вирази можуть бути використані для обґрунтування вибору кращого методу обробки (рис. 1). Спершу виведемо документацію про `T9` та її рівняння:

```
>>> T9.__doc__
'Повертає основний час для фрезерування гребінчастою фрезою, хв'
>>> eq2=fn2sympy(T9)
>>> eq2
Eq(T9, 3.925*d/(S_z*Z*n_i))
```

Тепер розв'яжемо рівняння `T7=T9` відносно `L` та `P`:

```
>>> sympy.solve(sympy.Eq(eq1.rhs, eq2.rhs), "L")
[3.925*P*d*n/(S_z*Z*g*i*n_i)]
>>>sympy.solve(sympy.Eq(eq1.rhs, eq2.rhs), "P")
[0.254777070063694*L*S_z*Z*g*i*n_i/(d*n)]
```

Python-модуль `ThreadingV` містить функції для розрахунку режимів обробки різьб з прикладами їхнього використання. Функція `V1()` повертає швидкість різання машинними мітчиками зі сталі Р6М5. Аргументами функції є `D` (діаметр різьби, мм), `P` (крок, мм), `m` (тип матеріалу), `k1` (підтип матеріалу), `k2` (ступінь точності різьби) та `HB` (твердість чавуну). Якщо не усі аргументи вказані, то функція повертає словники `V`, `K1`, `K2` з базою даних. Якщо значення усіх аргументів відомі, то функція повертає два значення швидкості різання (мінімальне і максимальне). Більше значення треба приймати для різьб з більшими діаметрами і меншими кроками. Оскільки ці дані є дискретними, то виникає проблема знаходження проміжних значень. Цієї проблеми би не було, якби алгоритми використовували теоретичні чи емпіричні формули, наприклад формули Тейлора [5]. Проте емпіричні формули теж базуються на табличних експериментальних даних, які потім апроксимують або інтерполюють.

Спочатку виконаємо імпорт усіх функцій з модуля:

```
>>> from ThreadingV import *
```

Виведемо документацію до функції `V1`:

```
>>> V1.__doc__
'Швидкість різання машинними мітчиками зі сталі Р6М5 [Якухин, т.38].\n Більше значення треба приймати для різьб з більшими діаметрами і меншими кроками\n    D - діаметр різьби, мм\n P - крок, мм,\n  m - матеріал\n  k1 - матеріал\n  k2 - ступінь точності\n HB - твердість чавуну\n  Приклад:\n V1(5, 1, "Сталь конструкційна", "Сталь нормалізована", 6)\n  V1()\n '
```

Розрахуємо максимальне і мінімальне значення швидкості різання:

```
>>> V1(D=5, P=1, m="Сталь конструкційна", k1="Сталь нормалізована", k2=6)
array([8., 5.])
```

Функція `V1pd()` є аналогом `V1()`, але використовує Pandas. Використання Pandas дозволяє працювати з цими табличними даними більш зручно. Зокрема можна легко додавати нові стовпчики, фільтрувати та аналізувати дані. А в Jupyter Notebook результати запитів виводяться у вигляді таблиці (Таблиця 1).

Границі швидкості різання для діаметра 5 мм і матеріалу "Сталь конструкційна" можна отримати так:

```
>>> V=V1pd()
>>> V["Сталь конструкційна"][ [i<5<j for i,j in V.D] ]
0    (8, 5)
Name: Сталь конструкційна, dtype: object
```


ISSN 1993–9965 print
ISSN 2415–3524 online
Науковий вісник ІФНТУНГ
2024. № 2(57)

Таблиця 1 – Частина інтервальних табличних даних Pandas для вибору швидкості різання (м/хв)

D, мм	P, мм	Сталь конструкт-ційна	Сталь корозо-стійка та жароміцна	Чавун	Кольорові сплави	Термо-реактивні пластмасси
(3, 6)	(0.5, 1)	(8, 5)	(5, 3)	(6, 3)	(10, 6)	(1, 5)
(8, 10)	(0.5, 1.5)	(12, 6)	(6, 4)	(8, 5)	(12, 8)	(1.5, 7)
(12, 16)	(0.5, 2)	(16, 8)	(8, 6)	(10, 6)	(16, 10)	(2, 9)
(18, 27)	(0.75, 3)	(18, 10)	(10, 7)	(11, 6)	(18, 12)	(2, 12)
(30, 39)	(1, 3)	(20, 10)	(0, 0)	(12, 7)	(20, 14)	(0, 0)
(42, 48)	(1, 3)	(24, 10)	(0, 0)	(0, 0)	(24, 15)	(0, 0)

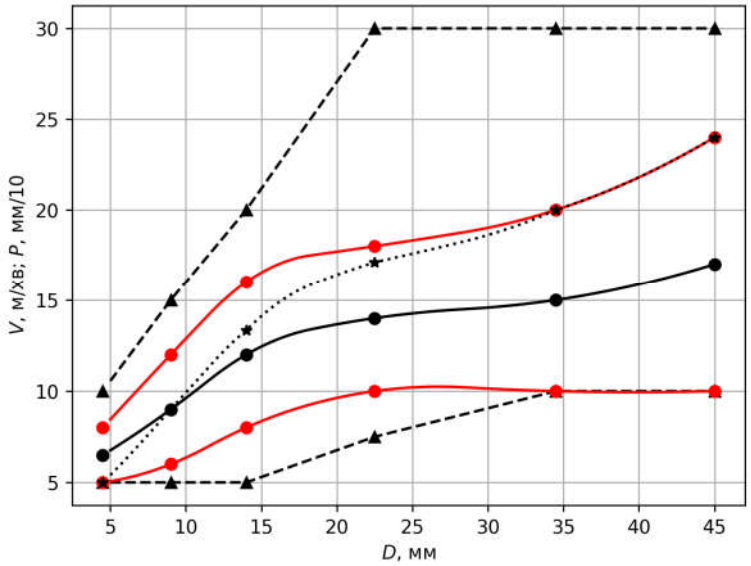


Рисунок 3 – Залежності швидкості різання машинними мітчиками зі сталі Р6М5 (*V*) і кроку різьби (*P*) в заготовках з конструкційної сталі від діаметра різьби (*D*): квадратична інтерполяція *Vmin* та *Vmax* (суцільна червона); квадратична інтерполяція середнього значення *V* (суцільна чорна); лінійна інтерполяція *Pmin* та *Pmax* (штрихова); квадратична інтерполяція *V* для *P*=1 мм (пунктирна)

На рисунку 3 показано результати одновимірної інтерполяції цих даних за допомогою `scipy.interpolate`. Табличні дані показані точками. Програмний код для створення цих графіків знаходиться у модулі `InterpolateExample` [31]. Використано функцію `interp1d()` для лінійної та квадратичної інтерполяції. Для прикладу, за допомогою масивів `D` та `Vmin` створимо функцію `vmin()`, яка повертає інтерпольоване значення мінімальної швидкості різання, та нарисуємо інтерпольовану криву з табличними точками:

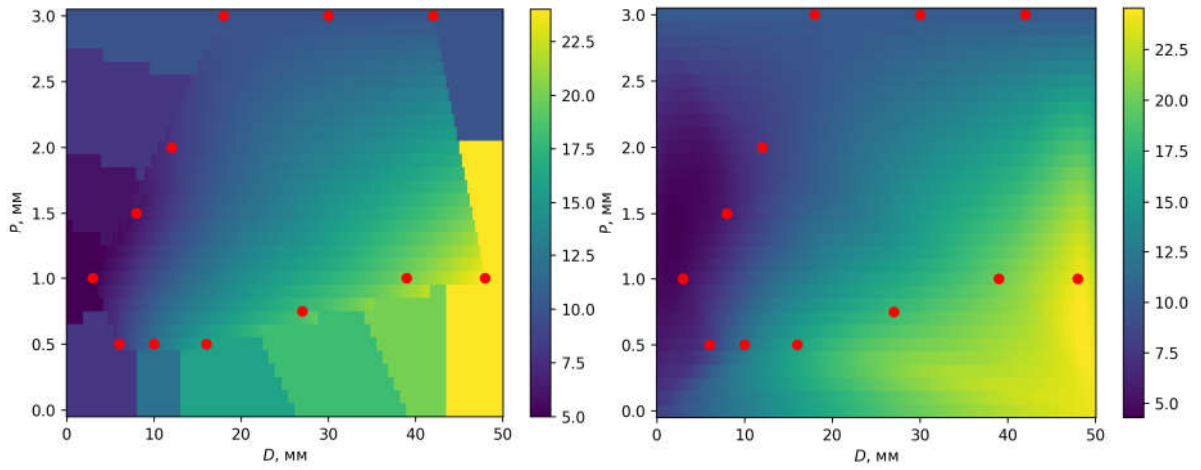
```
>>> vmin = interp1d(D, Vmin, kind='quadratic')
>>> X=np.arange(5,45,0.1)
>>> plt.plot(X, vmin(X), 'r'); plt.plot(D, Vmin, 'ro')
```

Щоб побудувати криву квадратичної інтерполяція `V` для `P`=1 мм (на рис. пунктирна), знадобиться функція `Vinterp(D, P)`, яка повертає інтерпольоване значення швидкості для діаметра `D` та кроку `P` за правилом «більше значення приймати для різьб меншими кроками»:

```
>>> plt.plot(X, [Vinterp(x, 1.0) for x in X], 'k:'); plt.plot(D, [Vinterp(x, 1.0) for x in D], 'k*')
```

На рис. 4 показано результати двовимірної інтерполяції цих даних. Табличні дані показано точками. Для двовимірної кубічної інтерполяції табличних даних `D`, `P`, `v` можна використати функцію `CloughTocher2DInterpolator()` зі `scipy.interpolate`.

```
>>> f = CloughTocher2DInterpolator(list(zip(D,P)), v, rescale=True)
```



а б

Рисунок 4 – Інтерпольовані залежності швидкості різання машинними мітчиками зі сталі Р6М5 (*V*, м/хв) в заготовках з конструкційної сталі від діаметра (*D*) і кроку (*P*) різьби: поєднання `CloughTocher2DInterpolator` та `NearestNDInterpolator` (а); використання додаткових кутових точок та `CloughTocher2DInterpolator` (б)

Проте функція `f()` повертає інтерпольовані значення швидкості тільки в межах плоского многокутника, утвореного точками (D, P) (рис. 4а). Якщо потрібно отримати інтерпольовані значення за межами цього многокутника, то можна скористатись інтерполятором найближчого сусіда `NearestNDInterpolator()` зі `scipy.interpolate`:

```
>>> f2 = NearestNDInterpolator(list(zip(D, P)), v, rescale=True)
```

Розроблена функція `COMBinterp(x, y, f, f2)` дозволяє об'єднати способи інтерполяції `f` і `f2` та побудувати рисунок 4а таким чином:

```
>>> X, Y = np.meshgrid(np.arange(0., 50.1, 0.1), np.arange(0., 3.1, 0.1))
>>> Z = COMBinterp(X, Y, f, f2)
>>> plt.plot(D, P, "ro")
>>> plt.pcolormesh(X, Y, Z)
```

За допомогою `COMBinterp()` можна отримати значення в чотирьох кутових точках (0, 0), (50, 0), (50, 3), (0, 3) та використати їх як додаткові точки для нової кубічної інтерполяції за допомогою `CloughTocher2DInterpolator()`, результат якої показано на рис. 4б.

Висновки

Запропоновані принципи розроблення програмних компонентів САПР ТП на основі Python та екосистеми її пакетів SciPy, які полегшують прийняття оптимальних рішень під час вибору методів і режимів обробки різьб. Показано приклади функцій для розрахунку основного часу та режимів обробки різьб, а також продемонстровано засоби інтелектуального використання цих функцій.

Перетворення звичайних Python-функцій, які повертають значення з аналітичних виразів, в SymPy-рівняння та їхнє розв'язування відносно будь-яких аргументів додає в Python можливості декларативного програмування, що запобігає дублюванню коду і сприяє зменшенню його об'єму, спрощує розроблення нових функцій, а також дозволяє будь-які операції символьної математики, наприклад диференціювання, інтегрування, розв'язування систем звичайних і диференціальних рівнянь. Це полегшує перетворення САПР ТП в інтелектуальну експертну систему.

Автоматизація формування документації Python-функцій шляхом знаходження їх аргументів засобами інтроспекції Python та перетворення в формат LaTeX дозволяє полегшити створення якісної документації, яка адаптована для використання в Jupyter Notebook.

Подання даних про режими різання за допомогою таблиць Pandas дозволяє операції з даними зробити більш зручними та гнучкими, а також легко інтегрувати ці дані в екосистему SciPy. Як приклад такої інтеграції, показано способи візуалізації аналітичних залежностей основного часу та табличних даних про режими різання з використанням Matplotlib, а також способи одновимірної та двовимірної інтерполяції даних за допомогою scipy.interpolate. Розроблені компоненти орієнтовані на використання в середовищі Jupyter Notebook, що покращує їхнє інтерактивне використання.

ISSN 1993–9965 print
ISSN 2415–3524 online
Науковий вісник ІФНТУНГ
2024. № 2(57)

Це дослідження було профінансоване Міністерством освіти і науки України, грант номер РК 0124U000654.

Література

1. Azab A., Osman H., Baki F. CAPP-GPT: A computer-aided process planning-generative pre-trained transformer framework for smart manufacturing. *Manufacturing Letters*. 2024. Vol. 41. Supplement P.51-62. URL: https://doi.org/10.1016/j.mfglet.2024.09.009.

2. Stavropoulos P., Tzimanis K., Souflas T. et al. Knowledge-based manufacturability assessment for optimization of additive manufacturing processes based on automated feature recognition from CAD models. *Int J Adv Manuf Technol.* 2022. No 122. P. 993–1007. URL: https://doi.org/10.1007/s00170-022-09948-w

3. Dannen T., Gey M., Weber S., Stauder L., Barth S., Bergs T. Approaches for automated and computer-aided work plan generation for adaptive manufacturing process chains in one-off manufacturing – a literature review. *Procedia CIRP*. 2023. Vol. 120. P. 1410-1415. URL: https://doi.org/10.1016/j.procir.2023.09.185.

4. Розрахунок найвигідніших режимів різання при точінні: навч. посіб. / А. І. Грабченко, М. Д. Узунян, Н. В. Зубкова та ін. Харків : НТУ «ХПІ», 2014. 88 с.

5. Astakhov V.P. Cutting Force Modeling: Genesis, State of the Art, and Development. *Mechanical and Industrial Engineering. Materials Forming, Machining and Tribology*. Davim, J.P. (eds). Cham : Springer, 2022. URL: https://doi.org/10.1007/978-3-030-90487-6_2

6. Scientific computing tools for Python - SciPy.org. URL: https://projects.scipy.org (accessed 31.12.24).

7. Harris C.R., Millman K.J., van der Walt S.J. et al. Array programming with NumPy. *Nature*. 2020. 585. P. 357–362. https://doi.org/10.1038/s41586-020-2649-2.

8. Virtanen P., Gommers R., et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*. 2020. No 17(3). P. 261-272. URL: https://doi.org/10.1038/s41592-019-0686-2.

9. Meurer A, Smith CP, et al. SymPy: symbolic computing in Python. *PeerJ Computer Science*. 2017. 3:e103. URL: https://doi.org/10.7717/peerj-cs.103.

10. Hunter J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. 2007. 9(3). P. 90-95. URL: https://doi.org/10.1109/MCSE.2007.55.

11. McKinney W. Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference*, Stefan van der Walt and Jarrod Millman (eds). 2010.Vol. 445. P. 56 – 61. URL: https://doi.org/10.25080/Majora-92bf1922-00a.

12. Kluyver T, Ragan-Kelley B, Fernando Perez, Granger B, Bussonnier M, Frederic J, et al. Jupyter Notebooks – a publishing format for reproducible computational workflows. *IOS Press, Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Loizides F, Schmidt B, editors. 2016. P. 87–90. URL: https://doi.org/10.3233/978-1-61499-649-1-87.

13. Tonejca L., Trautner T., Slimane E., Peso N., Zulehner J., Bleicher F. Automated Design of Experiments supporting Feature-based Optimisation of Manufacturing Processes. *Procedia CIRP*. 2024. Vol. 130. P. 1611-1616, URL: https://doi.org/10.1016/j.procir.2024.10.290.

14. Ince C.-V., Schönburg J., Raatz A. Approach of Automated Robot Arrangement in Manufacturing Cells. *Procedia CIRP*. 2024. Volume 128. P. 286-291. URL: https://doi.org/10.1016/j.procir.2024.06.023.

15. Wolf M., Elser A., Riedel O., Verl A. A software architecture for a multi-axis additive manufacturing path-planning tool. *Procedia CIRP*. 2020. Vol. 88. P. 433-438. URL: https://doi.org/10.1016/j.procir.2020.05.075.

16. Köhler T., Song B., Bergmann J. P., Peters D. Geometric feature extraction in manufacturing based on a knowledge graph. *Heliyon*. 2023. Vol. 9. Issue 9. e19694. URL: https://doi.org/10.1016/j.heliyon.2023.e19694.

17. Fuchs T., Enslin C., Samsonov V., Lütticke D., Schmitt R. H. ProdSim: An Open-source Python Package for Generating High-resolution Synthetic Manufacturing Data on Product, Machine and Shop-Floor Levels. *Procedia CIRP*. 2022. Vol. 107. P. 1343-1348. URL: https://doi.org/10.1016/j.procir.2022.05.155.

18. Katsigiannis M., Evans M., Osho J., Pantelidakis M., Bitencourt J., Mykoniatis K. Empowering decentralized production: A distributed manufacturing system for additive manufacturing processes. *Manufacturing Letters*. 2024 .Vol. 41. Supplement, P. 1507-1514. URL: https://doi.org/10.1016/j.mfglet.2024.09.177.

19. Копей В.Б. Розробка програмних компонентів мовою Python та їх використання в IPython Notebook. *Вісник Університету «Україна», серія «Інформатика, обчислювальна техніка та кібернетіка»*. 2017. №1(19). С.208-214. URL: https://visn-it.uu.edu.ua/old_site/files/2015/15.pdf.

20. Holland M., Chaudhari K. Large language model based agent for process planning of fiber composite structures. *Manufacturing Letters*. 2024. Vol. 40. P. 100-103. URL: https://doi.org/10.1016/j.mfglet.2024.03.010.

21. Esmaeili-Qeshlaqi M., Tavakkoli-Moghaddam R., Siadat A. Optimizing Dynamic Optimization of a Reconfigurable Manufacturing System under Risk and Human Factors. *IFAC-PapersOnLine*. 2024. Vol. 58. Issue 19. P. 361-366. URL: https://doi.org/10.1016/j.ifacol.2024.09.238.

22. Копей В.Б. Компонент для декларативного програмування в Python. Інформаційні та моделюючі технології. *Матеріали всеукраїнської науково-практичної конференції ІМТ-2015*. Черкаси : ПП Нечитайло О.Ф., 2015. С.23-23.

23. Копей В.Б., Біленко П.О. Реалізація декларативного програмування в Python за допомогою математичних бібліотек SymPy та SciPy. *Молодь: освіта, наука, духовність: тези доповідей XIII Всеукр. наук. конф.*, (м. Київ, 12–14 квітня 2016 р.) у II част. Ч. II. Київ : Університет «Україна», 2016. С. 227-229.

24. Kopei V., Onysko O., Barz C., Dašić P., Panchuk V. Designing a Multi-Agent PLM System for Threaded Connections Using the Principle of Isomorphism of Regularities of Complex Systems. Machines. 11(2). 2023. P.263. URL: https://doi.org/10.3390/machines11020263.

25. Sphinx. Sphinx documentation. URL: https://www.sphinx-doc.org (accessed 31.12.24).

26. pdoc – Generate API Documentation for Python Projects. URL: https://pdoc.dev (accessed 31.12.24).

27. Knuth D. E. Literate Programming. *The Computer Journal*. 1984. Volume 27. Issue 2. P. 97–111. URL: https://doi.org/10.1093/comjnl/27.2.97

28. Pycco. URL: https://pycco-docs.github.io/pycco (accessed 31.12.24).

29. MkDocs. URL: https://www.mkdocs.org (accessed 31.12.24).

30. Built with Jupyter Book. URL: https://jupyterbook.org (accessed 31.12.24).

31. Kopei V., Onysko O., Proniuk I. Components of the expert system of threaded connections: ThreadingT.py, ThreadingV.py, example2.py, Threading.ipynb. URL: https://github.com/vkopey/ThreadES (accessed 31.12.24).

32. Orthogonality and the DRY Principle. A Conversation with Andy Hunt and Dave Thomas, Part II by Bill Venners, March 10, 2003. URL: https://www.artima.com/articles/orthogonality-and-the-dry-principle (accessed 31.12.24).

References

1. Azab A., Osman H., Baki F. CAPP-GPT: A computer-aided process planning-generative pre-trained transformer framework for smart manufacturing. *Manufacturing Letters*. 2024. Vol. 41. Supplement P.51-62. URL: https://doi.org/10.1016/j.mfglet.2024.09.009.

2. Stavropoulos P., Tzimanis K., Souflas T. et al. Knowledge-based manufacturability assessment for optimization of additive manufacturing processes based on automated feature recognition from CAD models. *Int J Adv Manuf Technol.* 2022. No 122. P. 993–1007. URL: https://doi.org/10.1007/s00170-022-09948-w

3. Dannen T., Gey M., Weber S., Stauder L., Barth S., Bergs T. Approaches for automated and computer-aided work plan generation for adaptive manufacturing process chains in one-off manufacturing – a literature review. *Procedia CIRP*. 2023. Vol. 120. P. 1410-1415. URL: https://doi.org/10.1016/j.procir.2023.09.185.

4. Rozrakhunok naivyhidnishykh rezhymiv rizannia pry tochinni: navch. posib. / A. I. Hrabchenko, M. D. Uzunian, N. V. Zubkova ta in. Kharkiv : NTU «KhPI», 2014. 88 p. [in Ukrainian]

5. Astakhov V.P. Cutting Force Modeling: Genesis, State of the Art, and Development. *Mechanical and Industrial Engineering. Materials Forming, Machining and Tribology*. Davim, J.P. (eds). Cham : Springer, 2022. URL: https://doi.org/10.1007/978-3-030-90487-6_2

6. Scientific computing tools for Python - SciPy.org. URL: https://projects.scipy.org (accessed 31.12.24).

7. Harris C.R., Millman K.J., van der Walt S.J. et al. Array programming with NumPy. *Nature*. 2020. 585. P. 357–362. https://doi.org/10.1038/s41586-020-2649-2.

8. Virtanen P., Gommers R., et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods.* 2020. No 17(3). P. 261-272. URL: https://doi.org/10.1038/s41592-019-0686-2.

9. Meurer A, Smith CP, et al. SymPy: symbolic computing in Python. *PeerJ Computer Science*. 2017. 3:e103. URL: https://doi.org/10.7717/peerj-cs.103.

10. Hunter J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. 2007. 9(3). P. 90-95. URL: https://doi.org/10.1109/MCSE.2007.55.

11. McKinney W. Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference*, Stefan van der Walt and Jarrod Millman (eds). 2010.Vol. 445. P. 56 –

61. URL: https://doi.org/10.25080/Majora-92bf1922-00a.

12. Kluyver T, Ragan-Kelley B, Fernando Perez, Granger B, Bussonnier M, Frederic J, et al. Jupyter Notebooks – a publishing format for reproducible computational workflows. *IOS Press, Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Loizides F, Schmidt B, editors. 2016. P. 87–90. URL: https://doi.org/10.3233/978-1-61499-649-1-87.

13. Tonejca L., Trautner T., Slimane E., Peso N., Zulehner J., Bleicher F. Automated Design of Experiments supporting Feature-based Optimisation of Manufacturing Processes. *Procedia CIRP*. 2024. Vol. 130. P. 1611-1616, URL: https://doi.org/10.1016/j.procir.2024.10.290.

14. Ince C.-V., Schönburg J., Raatz A. Approach of Automated Robot Arrangement in Manufacturing Cells. *Procedia CIRP*. 2024. Volume 128. P. 286-291. URL: https://doi.org/10.1016/j.procir.2024.06.023.

15. Wolf M., Elser A., Riedel O., Verl A. A software architecture for a multi-axis additive manufacturing path-planning tool. *Procedia CIRP*. 2020. Vol. 88. P. 433-438. URL: https://doi.org/10.1016/j.procir.2020.05.075.

16. Köhler T., Song B., Bergmann J. P., Peters D. Geometric feature extraction in manufacturing based on a knowledge graph. *Heliyon*. 2023. Vol. 9. Issue 9. e19694. URL: https://doi.org/10.1016/j.heliyon.2023.e19694.

17. Fuchs T., Enslin C., Samsonov V., Lütticke D., Schmitt R. H. ProdSim: An Open-source Python Package for Generating High-resolution Synthetic Manufacturing Data on Product, Machine and Shop-Floor Levels. *Procedia CIRP*. 2022. Vol. 107. P. 1343-1348. URL: https://doi.org/10.1016/j.procir.2022.05.155.

18. Katsigiannis M., Evans M., Osho J., Pantelidakis M., Bitencourt J., Mykoniatis K. Empowering decentralized production: A distributed manufacturing system for additive manufacturing processes. *Manufacturing Letters*. 2024 .Vol. 41. Supplement, P. 1507-1514. URL: https://doi.org/10.1016/j.mfglet.2024.09.177.

19. Kopei V.B. Rozrobka prohramnykh komponentiv movoiu Python ta yikh vykorystannia v IPython Notebook. *Visnyk Universytetu «Ukraina», seriia «Informatyka, obchysliuvalna tekhnika ta kibernetika»*. 2017. No 1(19). P. 208-214. URL: https://visn-it.uu.edu.ua/old_site/files/2015/15.pdf. [in Ukrainian]

20. Holland M., Chaudhari K. Large language model based agent for process planning of fiber composite structures. *Manufacturing Letters*. 2024. Vol. 40. P. 100-103. URL: https://doi.org/10.1016/j.mfglet.2024.03.010.

21. Esmaeili-Qeshlaqi M., Tavakkoli-Moghaddam R., Siadat A. Optimizing Dynamic Optimization of a Reconfigurable Manufacturing System under Risk and Human Factors. *IFAC-PapersOnLine*. 2024. Vol. 58. Issue 19. P. 361-366. URL: https://doi.org/10.1016/j.ifacol.2024.09.238

22. Kopei V.B. Komponent dlia deklaratyvnoho prohramuvannia v Python. Informatsiini ta modeliuiuchi tekhnolohii. *Materialy vseukrainskoi naukovo-praktychnoi konferentsii IMT-2015*. Cherkasy : PP Nechytailo O.F., 2015. P.23-23. [in Ukrainian]

23. Kopei V.B., Bilenko P.O. Realizatsiia deklaratyvnoho prohramuvannia v Python za dopomohoiu matematychnykh bibliotek SymPy ta SciPy. *Molod: osvita, nauka, dukhovnist: tezy dopovidei KhIII Vseukr. nauk. konf.*, (m. Kyiv, 12–14 kvitnia 2016 r.) u II chast. Ch. II. Kyiv : Universytet «Ukraina», 2016. P. 227-229. [in Ukrainian]

24. Kopei V., Onysko O., Barz C., Dašić P., Panchuk V. Designing a Multi-Agent PLM System for Threaded Connections Using the Principle of Isomorphism of Regularities of Complex Systems. Machines. 11(2). 2023. P.263. URL: https://doi.org/10.3390/machines11020263.

25. Sphinx. Sphinx documentation. URL: https://www.sphinx-doc.org (accessed 31.12.24).

26. pdoc – Generate API Documentation for Python Projects. URL: https://pdoc.dev (accessed 31.12.24).

27. Knuth D. E. Literate Programming. *The Computer Journal*. 1984. Volume 27. Issue 2. P. 97–111. URL: https://doi.org/10.1093/comjnl/27.2.97

28. Pycco. URL: https://pycco-docs.github.io/pycco (accessed 31.12.24).

29. MkDocs. URL: https://www.mkdocs.org (accessed 31.12.24).

30. Built with Jupyter Book. URL: https://jupyterbook.org (accessed 31.12.24).

31. Kopei V., Onysko O., Proniuk I. Components of the expert system of threaded connections: ThreadingT.py, ThreadingV.py, example2.py, Threading.ipynb. URL: https://github.com/vkopey/ThreadES (accessed 31.12.24).

32. Orthogonality and the DRY Principle. A Conversation with Andy Hunt and Dave Thomas, Part II by Bill Venners, March 10, 2003. URL: https://www.artima.com/articles/orthogonality-and-the-dry-principle (accessed 31.12.24).