Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Інститут інженерної механіки
Кафедра комп'ютеризованого машинобудування

Вірстюк Владислав Михайлович
(прізвище, ім'я, по батькові)

УДК 621.88:[004.89+004.94]
(індекс)

МАГІСТЕРСЬКА РОБОТА

Мультиагентна система для дослідження та проектування різьбових з'єднань
(назва роботи)

Комп'ютеризовані та роботизовані технології машинобудування
(назва освітньої програми)

131 - Прикладна механіка
(шифр і назва спеціальності)

В. М. Вірстюк
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник **Копей Володимир Богданович, д-р техн. наук, доцент**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту
Завідувач кафедри

докт. техн. наук, професор **В. Г. Панчук**
(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2021

РЕФЕРАТ

кваліфікаційної магістерської роботи " Мультиагентна система для дослідження та проектування різьбових з’єднань "

Розрахунково-пояснювальна записка: 79 с., 17 рис., 47 джерел.

Графічна частина: 6 аркушів формату А1.

Об’єкт дослідження – система інформаційної підтримки життєвого циклу різьбового з'єднання.

Предмет дослідження – методи і засоби побудови компонентів мультиагентної системи для інформаційної підтримки життєвого циклу різьбових з'єднань, у першу чергу, для етапів проектування конструкції і технологічного процесу.

Мета роботи – розробити принципи побудови і компоненти мультиагентної PLM-системи для інформаційної підтримки життєвого циклу різьбових з'єднань на основі Python-пакету Ray.

Виконано огляд концепцій життєвого циклу виробів та інформаційних систем для підтримки життєвого циклу виробів. Виявлено, що успішна така система повинна бути ізоморфною до інших складних систем, тобто володіти усіма загальносистемними закономірностями. Одним із способів забезпечити це є організація мультиагентної системи, паралелізм якої забезпечується за допомогою моделі акторів. Виконано огляд технології різьбофрезерування на верстатах з ЧПК. Розроблено гнучкий CAM-модуль для різьбофрезерування різьби з довільним профілем. За допомогою пакету Pycalculix розроблено FEA-модуль для симуляції напружено-деформованого стану різьбових з’єднань. Запропоновано принципи реалізації і показано приклад мультиагентної PLM-системи для спеціальних різьбових з’єднань пластмасових труб.

Ключові слова: *інформаційна підтримка виробу, різьбове з’єднання, мультиагентна система, різбофрезерування, скінченно-елементний аналіз*.

Студент Вірстюк В.М.

ABSTRACT

of the master's work " Multi-agent system for research and design of threaded connections "

Paper: 79 pages, 17 figures, 47 references.

The graphical part: 6 sheets of A1 format.

The object of research is the product life cycle management system of the threaded connection.

The subject of research is methods and means of construction of components of multiagent system for information support of life cycle of threaded connections, first of all, for stages of thread design and technological process creation.

The purpose of the work is to develop the principles of construction and components of multi-agent PLM-system for information support of the life cycle of threaded connections based on Python-package Ray.

A review of product life cycle concepts and information systems for product life cycle management is performed. It was found that a successful such system must be isomorphic to other complex systems, i.e. have all the system-wide properties. One way to ensure this is to organize a multi-agent system, the parallelism of which is ensured by the actor model. A review of the technology of thread milling on CNC machines is performed. A flexible CAM module for thread milling with an arbitrary profile has been developed. Using the Pycalculix package, an FEA module has been developed to simulate the stress-strain state of threaded connections. The principles of implementation are offered and an example of a multi-agent PLM system for special threaded connections of plastic pipes is shown.

Keywords: *product information support, threaded connection, multi-agent system, thread milling, finite element analysis.*

Student V. Virstyuk

ЗМІСТ

ВСТУП

Актуальність теми. Застосування систем інформаційної підтримки життєвого циклу виробів (PLM-систем) може значно скоротити витрати часу і коштів на проектування конструкції, проектування технологічного процесу, виробництво і підтримку експлуатації виробу. Незважаючи на те, що PLM-концепція давно відома в машинобудуванні, розроблення таких систем залишається прерогативою тільки великих корпорацій, які диктують споживачам свої не завжди адекватні умови. Однак, розвиток сучасного open-source програмного забезпечення та технологій програмування дозволяє сьогодні створювати та супроводжувати такі системи навіть окремим спеціалістам.

Мета і задачі досліджень. Метою роботи є розроблення принципів побудови і компонентів мультиагентної PLM-системи для інформаційної підтримки життєвого циклу різьбових з'єднань на основі Python-пакету Ray.

Для досягнення мети необхідно розв’язати наступні задачі:

1. Виконати огляд концепцій життєвого циклу виробів та інформаційних систем для підтримки життєвого циклу виробів.

2. Виконати огляд побудови мультиагентних систем на основі моделі акторів і пакету Ray.

3. Виконати огляд технології різбофрезерування на верстатах з ЧПК.

4. Розробити гнучкий CAM-модуль для різьбофрезерування різьби з довільним профілем. Продемонструвати роботу модуля.

5. Розробити FEA-модуль для симуляції напружено-деформованого стану різьбових з’єднань. Продемонструвати роботу модуля.

6. Запропонувати принципи реалізації і розробити приклад мультиагентної PLM-системи для спеціальних різьбових з’єднань пластмасових труб.

Об’єкт дослідження – система інформаційної підтримки життєвого циклу різьбового з'єднання.

Предмет дослідження - методи і засоби побудови компонентів мультиагентної системи для інформаційної підтримки життєвого циклу різьбових з'єднань, у першу чергу, для етапів проектування конструкції і технологічного процесу.

Методи дослідження. Для створення мультиагентної PLM-системи застосовували об’єктно-орієнтоване програмування та модель акторів. Для симуляції напружено-деформованого стану застосовували метод скінченних елементів для задач теорії пружності. Для реалізації експертної системи застосовували логіку предикатів та метод прямого виведення. Для створення мультиагентної системи використовували закономірності теорії систем.

Інноваційне та практичне значення результатів.

1. Розроблено простий і гнучкий FEA-модуль для симуляції напружено-деформованого стану різьбових з’єднань пластикових труб.

2. Розроблено простий і гнучкий CAM-модуль для різьбофрезерування на трьохосьових фрезерних верстатах з ЧПК з можливістю фрезерування різьб з довільним профілем.

3. В Jupyter Notebook розроблено приклад мультиагентної PLM-системи для спеціальних різьбових з’єднань пластмасових труб з базою знань, FEA-агентом і CAM-агентом.

РОЗДІЛ 1. СТАН ПРОБЛЕМИ І ЗАДАЧІ ДОСЛІДЖЕНЬ

1.1. Життєвий цикл виробів і мультиагентні PLM-системи

Системи інформаційної підтримки життєвого циклу виробів (PLM-системи) призначені для інформаційної підтримки таких етапів життєвого циклу виробу як етапу проектування конструкції, етапу проектування технологічного процесу виробництва, етапу виготовлення та етапу експлуатації [1]. Інформаційна підтримка в PLM-системі можлива за допомогою відповідних інформаційних ресурсів, до яких можуть належати різноманітні алгоритми, програмні компоненти, бази даних і знань, комп'ютерні системи моніторингу технічного стану, які вимірюють значення величин в реальному часі, та інші інформаційні ресурси (рис. 1.1) [2].

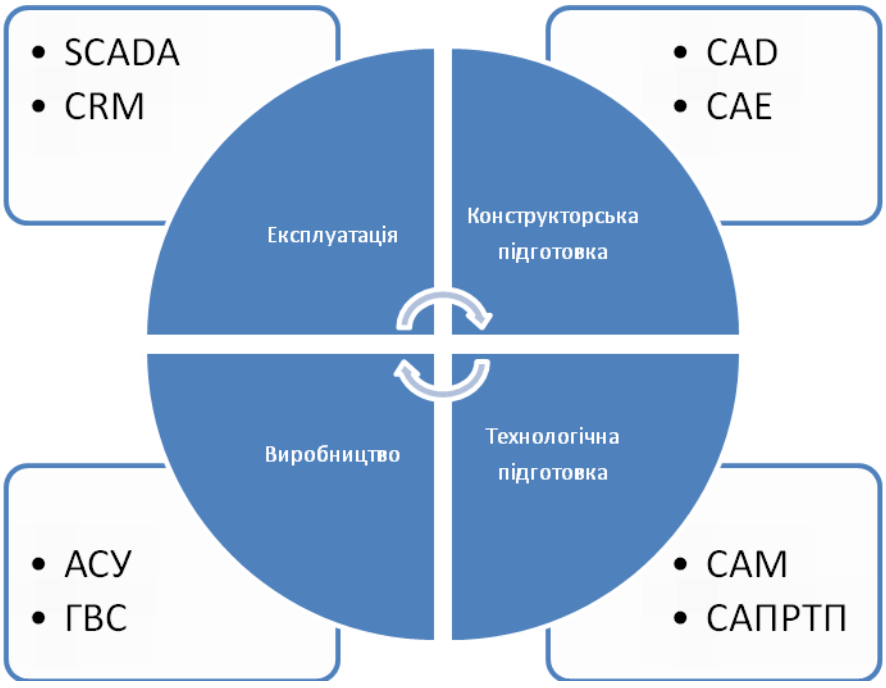


Рисунок 1.1 - Життєвий цикл виробу та відповідні компоненти PLM-системи

На етапі конструкторської підготовки виробництва найчастіше застосовують системи автоматизованого проектування CAD, які дозволяють створювати геометричні моделі, а на наступних етапах - системи

інженерного аналізу CAE, які дозволяють моделювати роботу виробу під час експлуатації, зокрема за допомогою чисельних методів, таких як метод скінченних елементів. В машинобудуванні найбільш відомими такими системами є FEA-системи для моделювання напружено-деформованого стану виробу за допомогою методу скінченних елементів. Як правило застосовують комерційні FEA-системи, але останнім часом відомими стають і вільні та open-source FEA-системи. Зокрема Calculix [3]. На етапі інженерного аналізу конструкції виробу також можуть застосовані системи мултидоменного моделювання фізичних явищ, які основані на мові моделювання Modelica [4].

На етапі технологічної підготовки виробництва застосовується CAM-системи та різноманітні САПР ТП, які дозволяють полегшити проектування складних технологічних процесів, зокрема автоматизувати генерацію технологічної документації, обчислення режимів різання, обчислення штучного часу, генерацію програм для верстатів з ЧПК. Останнім часом такі системи інтегрують в CAD-системи (як SOLIDWORKS [5]). А також існує велика кількість open-source проектів, наприклад CAM-модуль FreeCAD [6].

На етапі виробництва можуть бути застосовані різноманітні автоматизовані системи керування (АСУ) виробничими процесами, системи керування гнучкими виробничими системами (ГВС), верстатами з ЧПК та промисловими роботами, а також інші системи для контролю та оптимізації виробничих процесів (SCADA).

На етапі експлуатації виробу можуть бути застосовані системи комп'ютерного моніторингу технічного стану виробу в реальному часі SCADA, бази даних, де зберігається інформація про технічний стан виробу, а також системи, які забезпечують зв'язок споживачів з розробниками.

Сучасні вироби машинобудівного виробництва часто основані на великій кількості знань і технологій. Цю систему знань можна розглядати як складну систему. Теорія систем [7, 8] стверджує, що усі складні системи подібні загальносистемними закономірностями, до яких належить закономірність цілісності, закономірність історичності, закономірність ієрархічності та інші. Зокрема закономірність [8] цілісності вказує на те, що

успішна і життєздатна система містить емерджентні елементи, які доповнюють один одного до цілісності. Закономірність історичності вказує на те, що система має життєвий цикл, в якому етапи послідовно змінюють і доповнюють одне одного з переходом на вищий рівень розвитку [9]. Закономірність ієрархічності [8, 9] тісно пов'язана з іншими закономірностями і вказує на те, що система має ієрархічну структуру з елементами фрактальності (самоподібності). Зокрема кожний етап життєвого циклу ізоморфний (тобто подібний за структурою) на сам життєвий цикл.

Принцип ізоморфізму стверджує, що усі складні системи подібні цими закономірностями . Отже інформаційну систему підтримки життєвого циклу виробу слід розглядати як складну систему, яка володіє вказаними закономірностями. Розроблення таких інформаційних систем повинно бути основане на сучасній теорії систем та системному аналізі .

Звичайно, для того щоб об'єднати усі компоненти в єдину систему потрібний спеціальний підхід та уніфікована єдине інформаційне середовище. В практиці побудови програмних систем досить давно відомий мультиагентний підхід [10], який полягає в тому, що децентралізована система складається з великої кількості автономних агентів, які можуть обмінюватися інформацією. Кожний агент - це автономний програмний компонент, який може виконуватись паралельно з іншими агентами навіть на різних машинах, які об'єднані мережею. Агент може отримувати інформацію від інших агентів, обробляти її за певним алгоритмом і повертати цю інформацію для інших агентів. Мультиагентні системи володіють закономірностями цілісності, історичності і ієрархічності, які мають інші складні системи, зокрема, біологічні, соціальні, економічні та інші. Таким чином, мультиагентна PLM-система є ізоморфною до інших складних систем і може бути настільки ж ефективною. На жаль під час розробки сучасних PLM-систем дуже мало наголошують на цьому. Хоча сам мультиагентний підхід відомий досить давно. Існує велика кількість бібліотек, які полегшують розробку таких мультиагентних систем на різних мовах програмування [11], проте високорівневі мови програмування, такі як в

Python, можуть бути застосовані для розроблення таких систем і без всіляких додаткових компонентів.

В даній роботі на простому прикладі показано принципи розроблення, функціонування та використання такої системи для підтримки життєвого циклу різьбового з'єднання пластикових труб. Для побудови системи, яка може працювати на кластері, застосовували відому Python-бібліотеку Ray [12] та модель акторів [13]. Агенти системи, які можуть працювати паралельно, програмували, в основному, за допомогою мови програмування Python [14]. Розроблено агенти для: збереження фактів бази знань, правил логічного виведення, машини логічного виведення, скінченно-елементного аналізу напружено-деформованого стану різьбового з'єднання, генерації програмного G-коду для фрезерування різьби на фрезерному верстаті з ЧПК. Система може бути легко розширена іншими агентами, наприклад, для підтримки етапу експлуатації виробу. Показано приклад інтерактивної роботи з PLM-системою в середовищі Jupyter.. Детальніше з проектом можна ознайомитись на GitHub [15].

Передусім потрібно проаналізувати особливості функціонування агентів на етапах конструювання виробу, його інженерного аналізу та проектування технології, зокрема на фрезерних верстатах, які мають важливу перевагу - здатність до високої концентрації операції. Іншими словами, вони дозволяють зробити більшість роботи на одному робочому місці, що значно економить час кошти. Така здатність до концентрації операцій надзвичайно важлива в умовах одиничного та дрібно-серійного виробництва. А також потрібно проаналізувати особливості створення багатоагентної системи на основі моделі акторів та бібліотеки Ray. Це зроблено в наступних оглядових розділах.

1.2. Основні команди G-коду

G-код - це умовна назва мови для програмування пристроїв з ЧПК (CNC) (Числове програмне керування). Була створена компанією Electronic Industries Alliance на початку 1960-х [16]. Фінальна доробка була схвалена в

лютому 1980 року як RS274D стандарт. Комітет ISO затвердив G-код як стандарт ISO 6983-1:1982, Держкомітет стандартів СРСР - як ГОСТ 20999-83. У радянській технічній літературі G-code позначається як код ISO-7 біт. Виробники систем ЧПК використовують G-code як базову підмножину мов програмування, розширюючи його на власний розсуд.

Програма, написана за допомогою G-code, має жорстку структуру. Усі команди керування об'єднуються у кадри — групи, які складаються з однієї або більше команд. Кадр завершується символом переведення рядка (ПС/LF) і має номер, окрім першого кадру програми. Перший кадр містить лише один символ "%". Завершується програма командою M02 чи M30.

G-коди залежно від можливості збереження їх у пам'яті системи числового програмного управління ділять на дві групи – модальні команди та немодальні команди [17]. Модальні G-коди зберігаються в пам'яті системи ЧПК і діють до їх прямого скасування протягом усього часу виконання керуючої програми системи ЧПК. Немодальні G-коди діють лише в межах одного кадру керуючої програми, в якому вони знаходяться.

У зв'язку з тим, що команда G01 модальна, програму можна записати, не вказуючи в кадрах N002, N003 явно цю команду:

```
N01 G01 X10 Y20
N02 X5 Y10
N03 X2 Y6
N04 G00 X0 Y0
```

Основні команди (у стандарті називаються підготовчими) мови починаються з літери G: переміщення робочих органів обладнання із заданою швидкістю (лінійне та кругове; виконання типових послідовностей (таких, як обробка отворів та різьб); керування параметрами інструменту, системами координат, та робочих площин.

Список деяких основних команд G-коду для фрезерних верстатів з ЧПК [18, 19]:

G00 - прискорене переміщення інструменту (холостий перебіг). Приклад: G0 X0 Y0 Z100;

G01 - лінійна інтерполяція. Приклад: G01 X0 Y0 Z100 F200;

G02 (або G03) - кругова інтерполяція за (або проти) годинниковою стрілкою. Приклад: G02 X15 Y15 R5 F200;

G17 - вибір площини XY в якості площини обробки;

G21 - режим роботи у метричній системі. Приклад: G90 G21;

G28 - повернутись на референтну точку. Приклад: G28 G91 Z0 Y0;

G41 (або G42) - компенсувати радіус інструменту зліва (або справа) . Приклад: G41 X15 Y15 D1 F100;

G43 - компенсувати висоту інструменту позитивно. Приклад: G43 X15 Y15 Z100 H1 S1000 M3;

G54 - перемикання на задану оператором систему координат;

G90 - абсолютна система координат. Приклад: G90 G21;

G94 F (подача) - подача у форматі мм/хв. Приклад: G94 G80 Z100.

Технологічні команди мови починаються з літери M. Включають такі дії, як: змінити інструмент, увімкнути/вимкнути шпиндель, увімкнути/вимкнути охолодження, викликати/закінчити підпрограму. Основні допоміжні (технологічні) команди:

M02 - кінець програми. Приклад: M02;

M03/M04 - почати обертання шпинделя за/ проти годинниковою стрілкою. Приклад: M3 S2000;

M05 - зупинити обертання шпинделя. Приклад: M5;

M06 - змінити інструмент. Приклад: M6 T15.

Параметри команд задаються літерами латинського алфавіту:

X - координата точки траєкторії по осі X;

Y - координата точки траєкторії по осі Y;

Z - координата точки траєкторії по осі Z;

F - швидкість робочої подачі;

S - швидкість обертання шпинделя;

D - параметр корекції вибраного інструменту;

I, J, K - параметри дуги при круговій інтерполяції.

Для прикладу наступна програма:

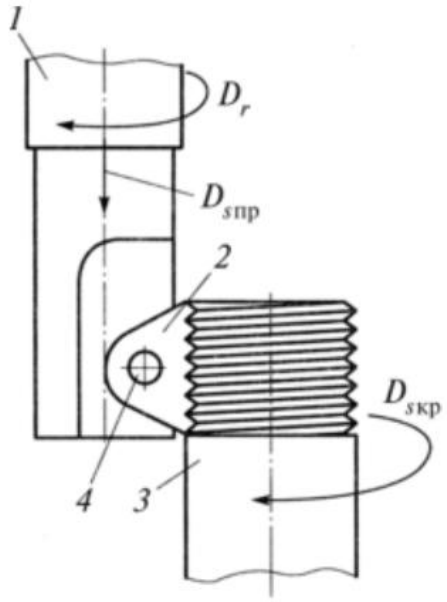
```
N10 G1 G54 G17 G21 G90 G94 M05 T0 F100 S1000
N20 M03
N30 G1 X30.000 Y0.000 Z0.000
N40 M05
```

у першому кадрі здійснює перемикання на задану оператором систему координат, вибирає площину XY в якості площини обробки, установлює режим роботи у метричній системі, вибирає абсолютну систему координат, установлює подачу 100 у форматі мм/хв, зупиняє обертання шпинделя, вибирає інструмент 0 та установлює швидкість обертання шпинделя на 1000 об/хв. У другому кадрі вмикається шпиндель. У третьому кадрі здійснюється переміщення у точку X=30, Y=0, Z=0. У четвертому кадрі виключається обертання шпинделя.

1.3 Різьбофрезерування на верстатах з ЧПК

Різьбофрезерування [20] виконується за допомогою фрез, показаних на рис. 1.2. На державці 1 гвинтом 4 кріпиться пластина 2 у вигляді гребінки з профілем, що відповідає профілю різьблення заготовки 3.

Ріжуча пластина може бути виготовлена з кераміки, нітриду бору та інших сучасних інструментальних матеріалів, яким можна давати швидкості різання більші за 1000 м/хв. На верстатах з ЧПК за допомогою таких різьбових фрез можна обробляти зовнішні та внутрішні різьби на дуже великих деталях [20]; на деталях, які важко чи неможливо закріпити на токарному верстаті для нарізування різьблення; на несиметричних деталях,



1 - державка; 2 - ріжуча пластина; 3 - заготовка; 4 - кріпильний гвинт

Рисунок 1.2 - Різьбофрезерування на верстатах з ЧПУ зовнішньої різьби

що мають при обертанні значний дисбаланс; у глухих отворах без канавок для виходу різьбового інструменту; при дуже великих діаметрах різьблення. Додатковою перевагою є скорочення машинного часу завдяки високим швидкостям різання та подачам, можливості обробки за один прохід повного по глибині профілю різьблення. Завдяки отриманню короткої стружки, що легко видаляється, скорочується час на її видалення. Можна зменшити комплект інструментів за рахунок застосування одного інструменту для внутрішнього та зовнішнього (лівого та правого) різьблення. Досить однієї державки для зовнішньої та внутрішньої різьби, у тому числі різного профілю та кроку. Зменшується час різання за рахунок наявності на кожній пластині кількох ріжучих кромок. Підвищується період стійкості ріжучих пластин через доступність нанесення на них зносостійкого покриття.

Переваги процесу фрезерування різьб на ЧПУ в порівнянні з нарізкою внутрішнього різьблення мітчиком [21]: більш надійна технологія обробки, коротка стружка, отримання різьби повного профілю на дні отвору, можливості налаштування на обробку різьблення з різними допусками, великої стійкості інструменту, можливості обробки різних матеріалів, універсальність, різьблення може бути використана для обробки різьблення

різних діаметрів з одним кроком, один і той же інструмент може бути використаний для обробки правих та лівих внутрішніх різьблень, для дюймового різьблення G, для внутрішніх та зовнішніх різьблень, можливості обробки без МОР, можливості обробки фаски при фрезеруванні метричних різьблень, при цьому якість фрезерованих конічних різьблень істотно вища, ніж нарізаних мітчиком.

Попутне фрезерування підвищує стійкість, запобігає подрібненню, але підвищується конусність [22]. Зустрічне фрезерування застосовують для оброблення загартованих матеріалів або для усунення конусності.

При нарізанні різьблення фрезами можна використовувати наступні три методи врізування [22]:

1. Тангенціально-дугове врізання.

2. Радіальне врізання.

3. Тангенціально лінійне врізання.

Операції різьбофрезерування рекомендується виконувати без використання СОЖ, щоб уникнути появи термічних тріщин [22]. Щоб мінімізувати можливі відхилення профілю, діаметр різьбофрези не повинен перевищувати 70% від діаметра різьбового отвору [22].

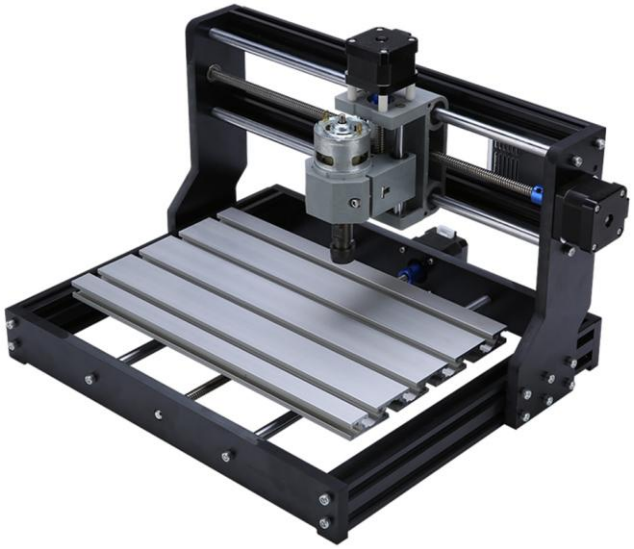
Рисунок 1.3 - Різьбофреза [23]

1.4. Огляд верстату CNC 3018 Pro та вільного програмного забезпечення для нього

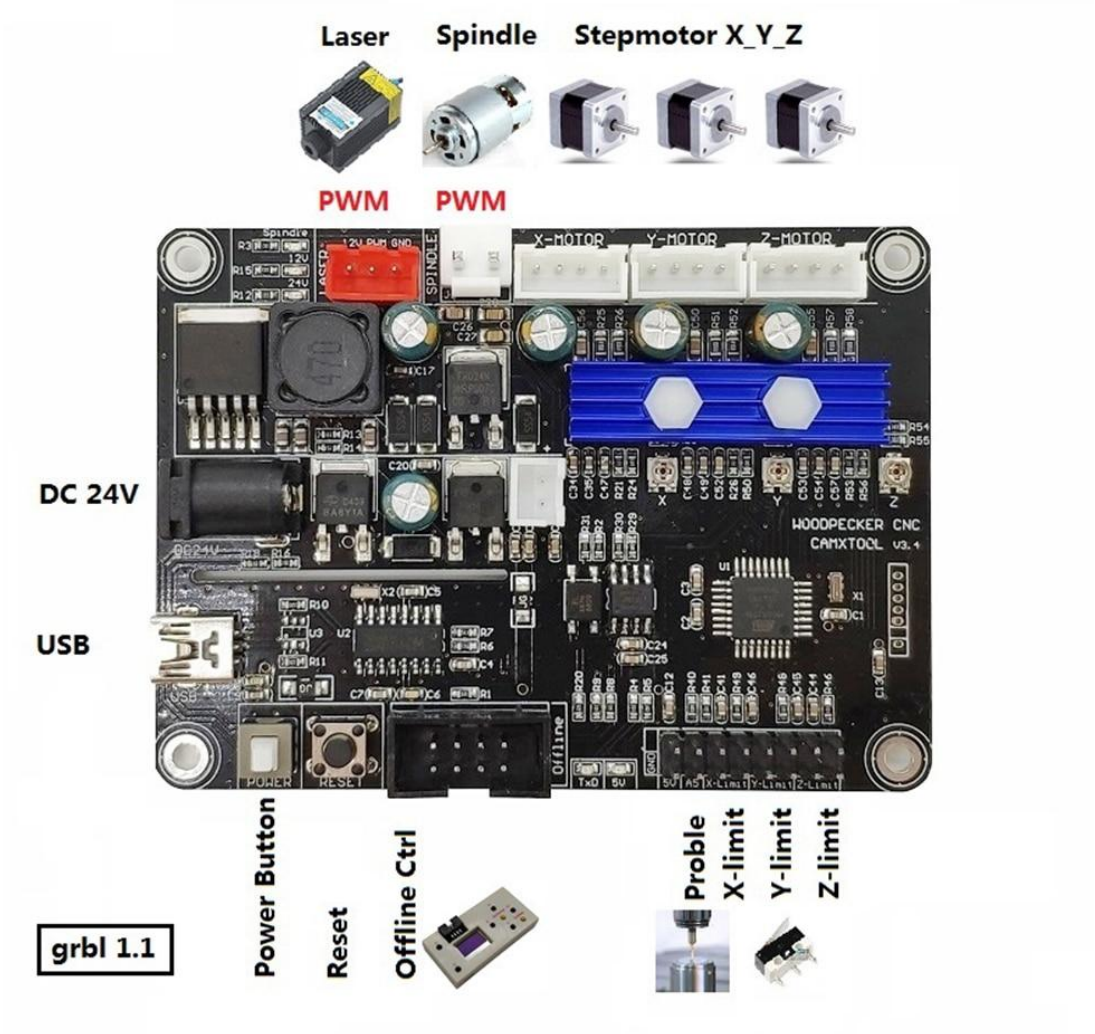
Верстат CNC 3018 Pro. Верстат CNC 3018 Pro [24] - це мініатюрний і недорогий (175$) 3-осьовий верстат з ЧПК з можливістю фрезерування або лазерного гравірування (рис. 1.4). У першу чергу використовується для навчання. Характеристики:

- Плата керування: CNC-CAMTOOL-V3.4;
- Вбудована GRBL 1.1f прошивка;
- Робоча площа: 300x180x45 мм;
- Площа алюмінієвого стола: 300x180 мм;
- Двигун шпинделя: DC 24 В, 96 Вт;
- Крокові двигуни: DC 12 В, 1.3 А, 0.3 Нм крутний момент;
- Джерело живлення: Input AC100-240 В, 50/60 Гц, 1.3 А, Output DC 24 В 5А;
- Програмне забезпечення: GRBL, LaserGRBL;
- Використовується для ПК: Windows XP/7/8/10 (не підтримується Mac OS).

GRBL. GRBL [25] - це безкоштовне високопродуктивне програмне забезпечення з відкритим вихідним кодом для керування верстатами з ЧПК, що працює безпосередньо на платі Arduino. GRBL підходить для легких виробництв. Його використовують для фрезерування, запускаючи його з ноутбуків або Raspberry Pi, використовуючи графічні інтерфейси, написані для GRBL для потокової передачі G-коду. GRBL написаний на високооптимізованій мові C з використанням функцій чіпів Atmega328p Arduino. Він здатний підтримувати частоту кроків понад 30 кГц і забезпечує чистий потік керуючих імпульсів без тремтіння.



а



б

Рисунок 1.4 - Верстат CNC 3018 Pro (а) та його плата керування (б), що підтримує ПЗ GRBL

GRBL призначений для трьохосьових машин. Осей обертання поки що немає – лише X, Y і Z. Інтерпретатор G-коду реалізує підмножину стандарту LinuxCNC і без проблем підтримується більшістю CAM-інструментів.

У версії v1.1 підтримуються наступні G-коди: G0, G1, G2, G3, G4, G10 L2, G10 L20, G17, G18, G19, G20, G21, G28, G30, G28.1, G30.1, G38.2, G38.3, G38.4, G38.5, G40, G43.1, G49, G53, G54, G55, G56, G57, G58, G59, G61, G80, G90, G91, G91.1, G92, G92.1, G93, G94, M0, M2, M30, M3, M4, M5, M7 , M8, M9, M56.

Щоб підключити крокові двигуни до GRBL, вам знадобляться деякі драйвери крокових двигунів для живлення крокових двигунів та підключення входів драйвера до контактів контролера Arduino (рис. 1.5). Існує ряд драйверів, які можуть це зробити: EasyDriver V4.4 [26], grblshield, stepper shield [27], Arduino CNC Shield або Raspberry Pi CNC Board/Hat.

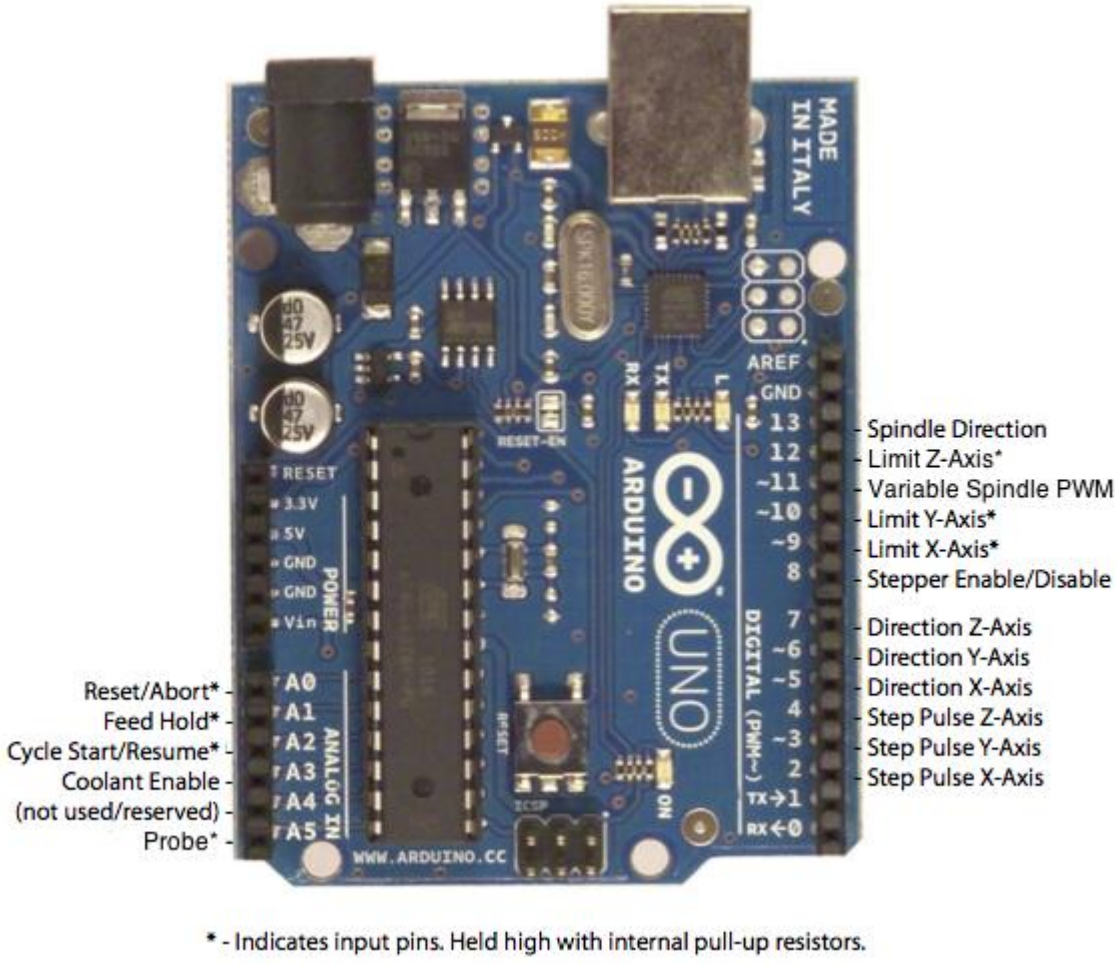


Рисунок 1.5 - Схема під’єднання пінів Arduino Uno до драйвера [28]

Після перепрошивки Grbl на Arduino підключення до Grbl досить просте. Ви можете використовувати Arduino IDE для підключення до Grbl. Інші програми для комунікації через послідовний порту, такі як CoolTerm або PuTTY, також чудово працюють. Інструкції майже однакові.

1. Відкрийте Arduino IDE і переконайтеся, що ваш Arduino з Grbl підключено до вашого USB-порту.

2. Виберіть послідовний порт Arduino в меню «Інструменти», як зазвичай у Arduino.

3. Відкрийте вікно «Serial Monitor» у меню «Інструменти».

4. Якщо ви використовуєте Grbl версії 0.9 або новішої, не забудьте змінити швидкість передачі даних з 9600 на 115200 бод.

5. Після відкриття ви побачите вітальне повідомлення Grbl, наприклад Grbl v0.Xx ['$' for help]. Це означає, що все добре! Ви підключені!

6. Переконайтеся, що ви змінили спадне меню «No line ending» на «Carriage return». Якщо ви використовуєте будь-яку іншу програму послідовного порту, ви повинні зробити те ж саме.

7. Якщо ви не отримали вітальне повідомлення або отримали деякі випадкові символи, переконайтеся, що швидкість передачі даних встановлена на рівні 9600 (або 115200 для версії 0.9+).

Тепер ви можете просто почати надсилати Grbl деякі команди G-коду, і він виконає їх за вас. Або ви можете ввести $, щоб отримати довідку про деякі спеціальні команди Grbl або про те, як записати деякі параметри вашої машини в пам'ять EEPROM Grbl.

Коли ви будете відчувати себе комфортно з G-code/CNC і готові запустити цілу програму з G-кодом, ми рекомендуємо вам використовувати один із багатьох графічних інтерфейсів, наприклад Candle.

Програма Candle. Програма Candle 1.1.7 [29] є контролером GRBL з візуалізатором G-коду, створена за допомогою графічної бібліотеки Qt. Це програмне забезпечення з відкритим вихідним кодом. Працює на Windows/Linux x86, потрібна підтримка процесором інструкцій SSE2 та

наявність графічної карти з підтримкою OpenGL 2.0. Основні функції програми:

- Управління верстатом з ЧПК на базі GRBL за допомогою консольних команд, кнопок на вікні, цифрової панелі.
- Контроль стану верстату.
- Завантаження, редагування, збереження та відправка файлів G-коду на верстат з ЧПУ.
- Візуалізація файлів G-коду.

Candle працює з ЧПК, керованим прошивкою GRBL 1.1 [30]. Багато проблем можна вирішити за допомогою правильної версії GRBL, використовуючи належну конфігурацію.

Python Streaming Scripts. Для Python-програм можна використовувати Python Streaming Scripts [31]. Не повністю підтримуються всі функції Grbl. Для потокових сценаріїв потрібен встановлений модуль pySerial. Встановіть модуль pySerial. Завантажте скрипт simple_stream.py Python. Відкрийте сценарій у звичайному текстовому редакторі та змініть наступний рядок, щоб він відповідав вашій системі:

```
s = serial.Serial('/dev/tty.usbmodem1811', 9600)
```

Замість /dev/tty.usbmodem1811(Mac) ви повинні вказати назву пристрою послідовного порту вашого Arduino. Це буде різним для кожної машини та ОС. Наприклад, у системі Linux це виглядатиме як /dev/ttyACM0, а на комп’ютері з Windows це може виглядати як COM3.

Сценарій шукає і зчитує gcode з файлу з іменем grbl.gcode, вам слід створити цей файл і помістити в нього G-код, який ви хочете виконати. Відкрийте вікно терміналу та змініть каталоги на розташування сценарію Python та виконайте сценарій Python за допомогою такої команди:

```
./simple_stream.py (Mac/Linux)
```

```
python simple_stream.py (Windows)
```

Тепер ви повинні побачити, як gcode передається в grbl разом із повідомленнями «OK», і ваш верстат повинен почати працювати.

Інший, більш просунутий потоковий сценарій stream.py має аргументи командного рядка і не вимагає модифікації самого сценарію, на відміну від simple_stream.py.

Цей спосіб дозволяє використовувати grbl як компонент багатоагентної інформаційної системи управління життєвим циклом різьбових з’єднань, що створена за допомогою Python. Зокрема з’являється можливість створити CNC-агент, який працює паралельно з іншими агентами і виконує посилання команд G-коду на верстат з ЧПК до їхнього завершення. Виникає також можливість організації простої гнучкої виробничої системи, яка складається з промислового робота та верстата з ЧПК. Роботом управляє один агент, а верстатом з ЧПК - інший.

1.5. Огляд можливостей бібліотеки Ray для побудови розподіленої МАС

Ray - Python-бібліотека з відкритим програмним кодом, яка дозволяє легко масштабувати будь-які Python програми, які вимагають інтенсивних обчислень [12]. Ray Core надає прості примітиви для створення розподілених додатків Python. Це підходить для розпаралелювання програм Python з мінімальними змінами коду. З Ray ваш код буде працювати на одній машині і його можна легко масштабувати до великого кластера. Кластер - це група комп'ютерів, об'єднаних високошвидкісними каналами зв'язку, що представляє з погляду користувача єдиний апаратний ресурс [32]. Нижче наведено приклади використання ray. Спочатку імпортуємо потрібні модулі:

```
import ray, time
```

Запуск Ray. Якщо ви підключаєтеся до існуючого кластера, слід використати ray.init(адреса=<адреса кластера>). Тоді Ray зможе використовувати всі ядра вашої машини. Дізнайтеся, як налаштувати кількість ядер, які використовуватиме Ray, у розділі документації Налаштування Ray. Щоб запустити багатовузловий кластер Ray, перегляньте сторінку документації «Налаштування кластера».

```
ray.init()
```

Звичайна функція, що виконується синхронно:

```
def f():
    return 1

f()
f()
f()
```

Ray дозволяє асинхронно виконувати довільні функції. Ці асинхронні функції Ray називаються «віддаленими функціями». Додавши декоратор `@ray.remote`, звичайна функція Python з аргументом стає функцією віддаленого керування Ray. Ось приклад:

```
@ray.remote
def f():
    return 1
```

Щоб викликати цю віддалену функцію, скористайтеся методом `remote`. Це негайно поверне об'єкт ref (a future), а потім створить завдання, яке буде виконуватися в робочому процесі.

```
obj_ref1 = f.remote()
```

Результат можна отримати за допомогою `ray.get`.

```
assert ray.get(obj_ref1) == 1
```

Віддалена функція Python з аргументом

```
@ray.remote
def f2(x):
    return x+10

obj_ref2 = f2.remote(1) # Ви можете передати аргумент
віддаленій функції Ray.
assert ray.get(obj_ref2) == 11
```

Посилання на об’єкт також можна передати у віддалені функції. Коли функція справді виконується, аргумент буде отримано як звичайний об’єкт. Для прикладу візьмемо:

```
obj_ref3 = f2.remote(obj_ref1) # Ви можете передати об’єкт
ref як аргумент іншій віддаленій функції Ray.
assert ray.get(obj_ref3) == 11
```

Друге завдання не буде виконано, доки не завершиться виконання першого, оскільки друге завдання залежить від результату першого завдання. Якщо два завдання заплановано на різних машинах, вихідні дані першого завдання буде надіслано по мережі на машину, де заплановано друге завдання. Віддалена функція, яка довго виконується:

```
@ray.remote
```

```
def f3():
    time.sleep(10) # чекати 10 секунд
    return 1
```

Виклики віддалених функцій Ray відбуваються паралельно. Усі обчислення виконуються у фоновому режимі, керуючись внутрішнім циклом подій Ray.

```
for _ in range(8):
    f3.remote() # тут немає блокування
```

Віддалені функції можуть бути відмінені: ray.cancel(obj_ref).

У Ray ми можемо створювати і обчислювати об’єкти [12]. Ми називаємо ці об’єкти віддаленими об’єктами, а для посилань на них використовуємо посилання на об’єкти. Віддалені об’єкти зберігаються в сховищах об’єктів спільної пам’яті, і є одне сховище об’єктів для кожного вузла в кластері. Посилання на об’єкт, по суті, є унікальним ідентифікатором, який можна використовувати для посилання на віддалений об’єкт. Якщо ви знайомі з Python futures, наші посилання на об’єкти концептуально схожі. Посилання на об’єкт можна створити кількома способами:

1.Вони повертаються віддаленими викликами функцій.

2.Їх повертає put.

```
y = 1
object_ref = ray.put(y) # Помістити об’єкт у сховище
об’єктів ray.
```

Віддалені об’єкти незмінні. Тобто їх значення не можуть бути змінені після створення. Це дозволяє реплікувати віддалені об’єкти в кількох сховищах об’єктів без необхідності синхронізації копій. Ви можете використовувати метод get, щоб отримати результат віддаленого об’єкта з

об’єкта ref. Якщо сховище об’єктів поточного вузла не містить об’єкта, об’єкт завантажується.

Отримайте значення кількох посилань на об’єкт паралельно:

```
assert ray.get([ray.put(i) for i in range(3)]) == [0, 1, 2]
```

Після запуску ряду завдань вам може знадобитися знати, які з них завершили виконання. Це можна зробити за допомогою wait (ray.wait). Функція працює наступним чином:

```
ready_refs, remaining_refs = ray.wait([obj_ref1, obj_ref2,
obj_ref3], num_returns=3, timeout=None)
print(ray.get(ready_refs), ray.get(remaining_refs))
ray.shutdown()
```

1.6. Огляд моделі акторів

Модель акторов [33, 34] - математична модель паралельних обчислень, що будується навколо поняття актора (англ. actor «актор; діючий суб'єкт»), що вважається універсальним примітивом паралельного виконання. Актор у цій моделі взаємодіє шляхом передачі повідомлень з іншими акторами, у відповідь отриманих повідомлень може приймати локальні рішення, створювати нові актори, посилати свої повідомлення, визначати, як слід реагувати на наступні повідомлення. Наприклад, мова програмування SmallTalk [35] побудована виключно на взаємодії об'єктів у вигляді надсилання повідомлень один одному. Код кожного об'єкта при цьому виконується ізольовано від сусідів у паралельному середовищі. Основні ідеї та база для моделі акторів закладені в 1973 році публікації Хьюїтта, Бішопа та Штайгера [36]. Основною мотивацією створення моделі стало завдання побудови розподілених обчислювальних систем з урахуванням сотень і тисяч незалежних комп'ютерів, оснащених своєї локальної пам'яттю і комунікаційними інтерфейсами [37].

Розроблено бібліотеки та табличні структури з акторами для забезпечення актороподібного стилю програмування мовами, які не мають вбудованих акторів [33]. Наприклад для Python це: Pulsar, Pykka, Thespian, Ray та інші.

Багато ідей, запроваджені в моделі акторів, нині знаходять також застосування у мультиагентних системах з цих причин. Ключовою відмінністю є те, що агент системи (здебільшого) накладає додаткові обмеження на акторів, як правило, вимагаючи, щоб вони використовували зобов'язання та цілі.

За аналогією з філософією об'єктно-орієнтованого програмування, де кожен примітив розглядається як об'єкт, модель акторів виділяє як універсальну сутність поняття «актора» [38]. Актор, по суті, є виконавцем (або службою), що зберігає стан. Коли створюється екземпляр нового актора, створюється новий обробник, і методи актора плануються для цього конкретного виконавця, і вони можуть отримати доступ до стану цього виконавця та змінити його [12]. Приклад:

```
import ray

@ray.remote
class C(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value
```

Створити актора з цього класу:

```
c = C.remote()
```

Здійснюється виклик актора:

```
obj_ref = c.inc.remote()
assert ray.get(obj_ref) == 1
```

Створити десять акторів:

```
cs = [C.remote() for _ in range(10)]
```

Викликати inc кожного актора один раз і отримати результати. Усі ці задачі будуть виконані паралельно.

```
results = ray.get([c.inc.remote() for c in cs])
print(results) # надрукує [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

Викликати increment першого актора 5 раз. Ці задачі виконуються послідовно і діляться станом.

```
results = ray.get([cs[0].inc.remote() for _ in range(5)])
print(results) # друкує [2, 3, 4, 5, 6]
```

Ще один приклад використання акторів, який розроблено автором, показано нижче. В ньому створюються два класи акторів, які мають конструктори і функцію rule. Створюється список акторів db. Відбувається запуск функцій rule, які можуть виконуватись паралельно. Після цього ми чекаємо завершення роботи одного актора за допомогою ray.wait().

```
import ray,time
ray.init()

@ray.remote
```

```
class A(object):
    def __init__(self,a):
        self.a = a
    def rule(self):
        self.a += 1
        time.sleep(20)
        return self.a

@ray.remote
class B(object):
    def __init__(self,b):
        self.b = b
    def rule(self):
        self.b += 1
        time.sleep(10)
        return self.b

db = [A.remote(1), B.remote(2)]

obj_refs = [a.rule.remote() for a in db]

#print(ray.get(obj_refs))

ready_refs, remaining_refs = ray.wait(obj_refs,
num_returns=1, timeout=1)
print(ray.get(ready_refs))#, ray.get(remaining_refs))
ray.shutdown()
```

В наступному прикладі демонструється можливість передачі актору класу A іншого актора класу B. Функція rule актора “a” отримує актора “b”, виконує його функцію rule і чекає результату. Таким чином можна

організувати взаємодію між різними акторами. Функція time.sleep використовується тільки для імітації тривалої роботи акторів.

```
import ray,time
ray.init()

@ray.remote
class A(object):
    def __init__(self,a):
        self.a = a
    def rule(self, obj):
        r=obj.rule.remote()
        self.a += ray.get(r)
        time.sleep(1)
        return self.a

@ray.remote
class B(object):
    def __init__(self,b):
        self.b = b
    def rule(self):
        self.b += 1
        time.sleep(2)
        return self.b

a=A.remote(1)
b=B.remote(2)
a_ref=a.rule.remote(b)
print(ray.get(a_ref))

ray.shutdown()
```

РОЗДІЛ 2. FEA-МОДУЛЬ ДЛЯ СИМУЛЯЦІЇ НАПРУЖЕНО-ДЕФОРМОВАНОГО СТАНУ РІЗЬБОВОГО З'ЄДНАННЯ

Нижче описано FEA-модуль для симуляції напружено-деформованого стану різьбового з'єднання пластмасових трубчатих деталей (рис. 2.1) Модуль розроблено автором у співавторстві з науковим керівником за допомогою Python-пакету pycalculix [39], що використовує FEA Calculix.

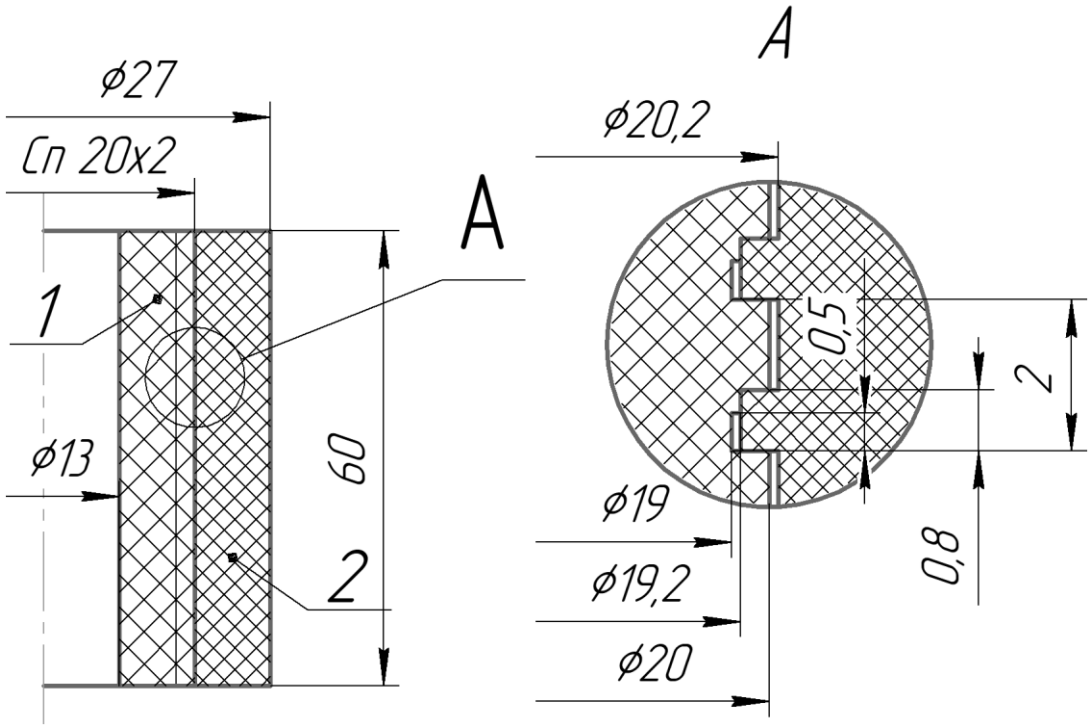


Рисунок 2.1 - Різьбове з'єднання зі спеціальною різьбою

Передусім імпортуються необхідні модулі:

```
# -*- coding: utf-8 -*-
import sys
import pycalculix as pyc
```

Вказуємо шлях до генератора скінченно-елементної сітки Gmsh [40] gmsh.exe, розв'язувача Calculix [3] ccx.exe, переглядача результатів Calculix [3] cgx.exe:

```
pyc.environment.GMSH=r"d:\Portable\gmsh-4.8.4-
Windows64\gmsh.exe"
pyc.environment.CCX=r"d:\Portable\cae_20200725_windows\bin\c
cx.exe"
pyc.environment.CGX=r"d:\Portable\cae_20200725_windows\bin\c
gx.exe"
```

Створюємо модель та вказуємо одиниці вимірювання:

```
model_name = 'thread'
model = pyc.FeaModel(model_name) # модель
model.set_units('mm') # одиниці вимірювання
```

Далі створюється геометрична модель різьбового з'єднання. Геометричні параметри з'єднання:

```
show_gui = True # показувати графіки
l = 30
t1 = 3.5
t2 = 3.5
r0 = 6.5
hh=0.1
```

Створюємо об'єкт-деталь `part` та рисуємо контури з'єднання, які складаються з ліній `l1-l6`:

```
part = pyc.Part(model) # деталь
part.goto(r0, 0) # перейти в точку
l1=part.draw_line_ax(l) # рисувати лінію вздовж
l2=part.draw_line_rad(t1) # рисувати лінію поперек
```

```
l3=part.draw_line_rad(t2)
l4=part.draw_line_ax(-l)
l5=part.draw_line_rad(-t2)
l6=part.draw_line_rad(-t1)
```

Зазори в різьбі створюються як прямокутні отвори за допомогою ліній, які мають знаходитися в межах контуру з’єднання. Потрібно створити 29 таких отворів за допомогою конструкції циклу for:

```
for i in range(15): # 15 отворів
    d=i*2.0 # осьова координата
    w = 1.2 # ширина
    h = hh # висота
    part.goto(10, 0.3+d, holemode=True) # перейти в точку,
режим отворів
    part.draw_line_rad(h) # рисувати лінію поперек
    part.draw_line_ax(w) # рисувати лінію вздовж
    part.draw_line_rad(-h)
    part.draw_line_ax(-w)

for i in range(14): # 14 отворів
    d=i*2.0 # осьова координата
    w = 0.5 # ширина
    h = hh # висота
    part.goto(9.5, 1.5+d, holemode=True) # перейти в точку,
режим отворів
    part.draw_line_rad(h) # рисувати лінію поперек
    part.draw_line_ax(w) # рисувати лінію вздовж
    part.draw_line_rad(-h)
    part.draw_line_ax(-w)
```

Тепер можна показати геометрію і подивитись на результати створення геометричної моделі (рис. 2.2):

```
model.plot_geometry(model_name + '_pre', display=show_gui)
model.view.print_summary() # текстова інформація про
геометрію
```

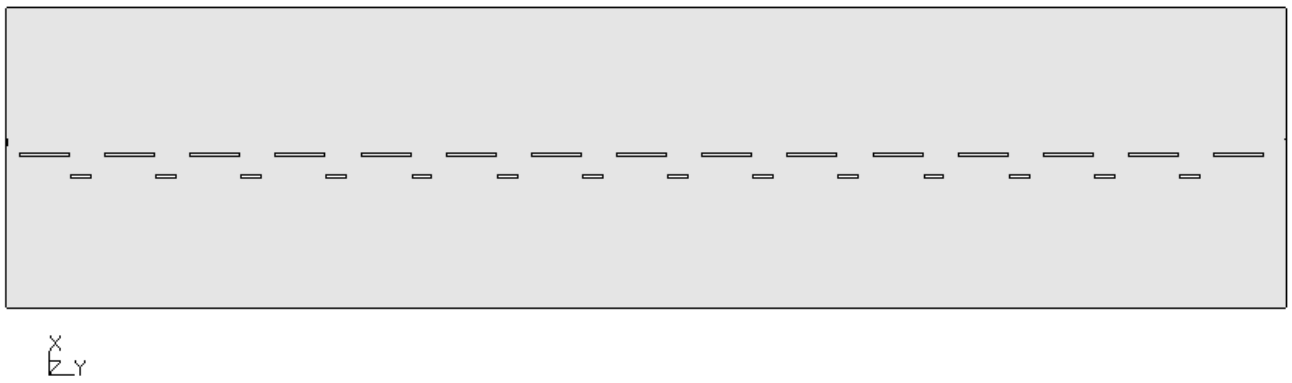


Рисунок 2.2 - Геометрична модель

Наступний етап - створення сітки скінченних елементів. Для цього потрібно вказати тип і форму елементів. На комп'ютері також має бути установлена вільна програма для створення сітки Gmsh [40]. Перший параметр функції model.mesh визначає якість сітки. З його збільшенням погіршується якість. Результати створення сітки показані на рис. 2.3.

```
model.set_etype('axisym', part) # тип елементів -
осесиметричні
model.set_eshape('tri', 2) # форма елементів - трикутні
model.mesh(0.1, 'gmsh') # створити сітку за допомогою gmsh
model.plot_elements(model_name+'_elements',
display=show_gui) # показати сітку
model.view.print_summary()
model.print_summary()
```

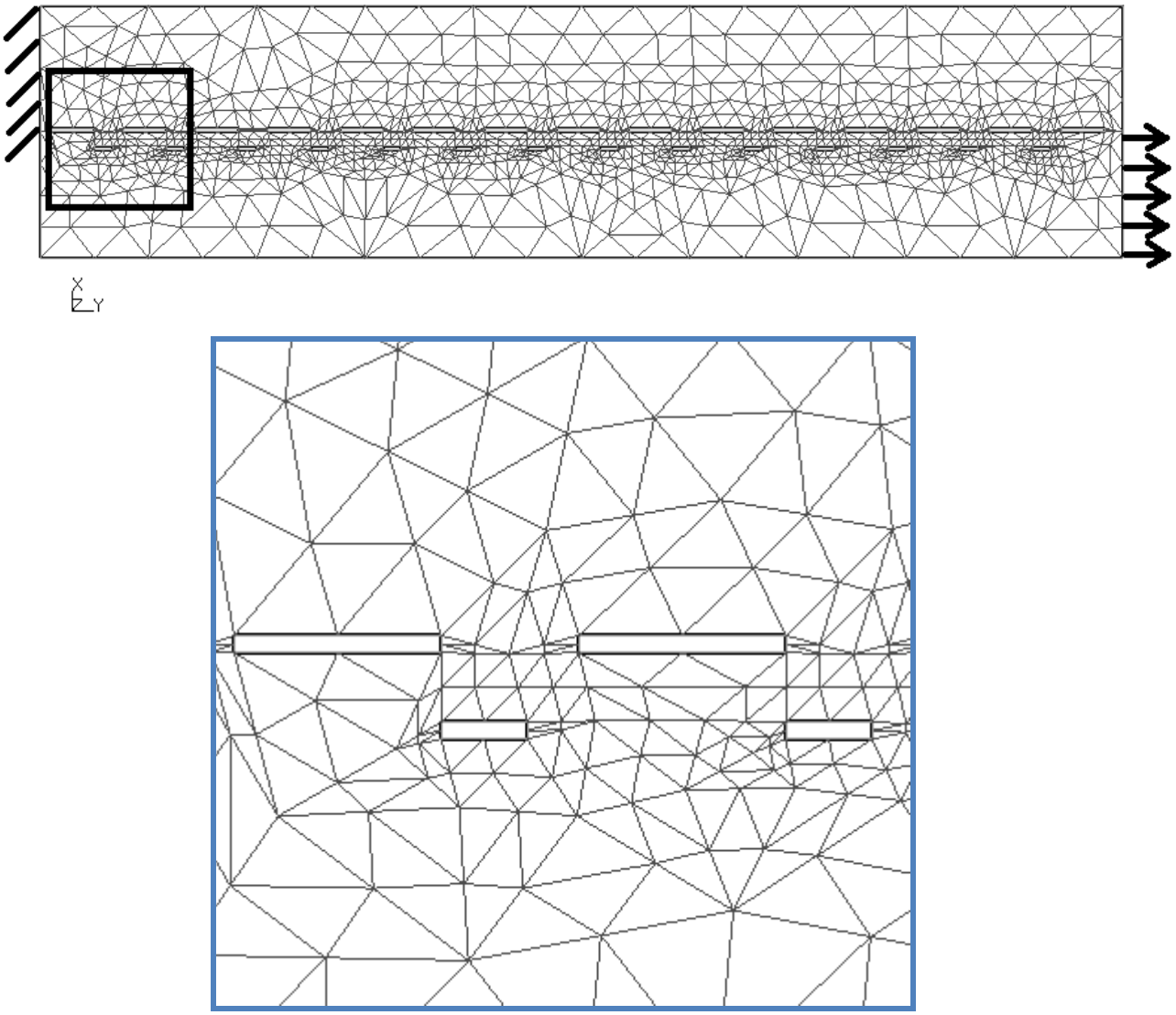


Рисунок 2.3 - Граничні умови і сітка скінченних елементів

Наступник крок - визначення матеріалу деталей і його механічних характеристик:

```
mat = pyc.Material('steel')
mat.set_mech_props(7800, 210*(10**9), 0.3) # власт.матеріалу
model.set_matl(mat, part)
```

Далі визначаються навантаження на різьбове з’єднання і граничні умови. Ліва (нижня) частина деталі з внутрішньою різьбою позбавлена переміщень по x і y (рис. 2.3). Права (верхня) частина деталі з зовнішньою

різьбою отримує навантаження розтягу у вигляді тиску величиною -10 МПа (рис. 2.3).

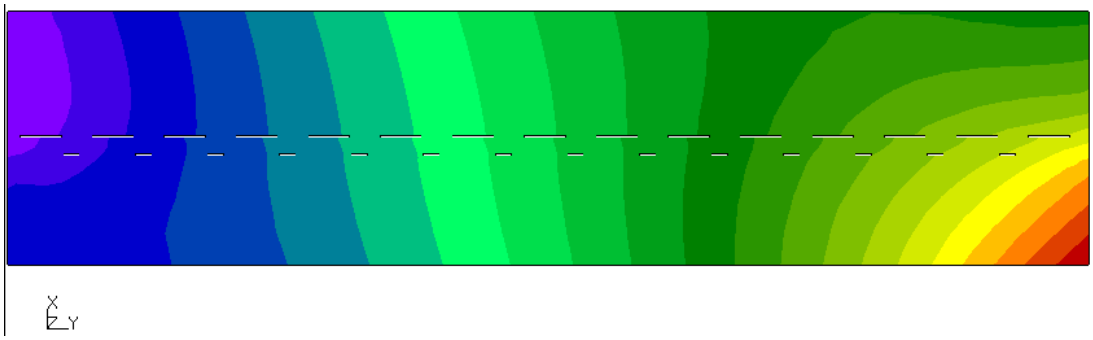
```
# низ нерухомий
model.set_constr('fix',[l5[0]],'x')
model.set_constr('fix',[l5[0]],'y')
model.set_load("press", [l2[0]], -10e6) # навантаження
```

Далі потрібно створити завдання і виконати його. Викликати функцію solve потрібно в блоці try-except, що перехоплює помилки виконання:

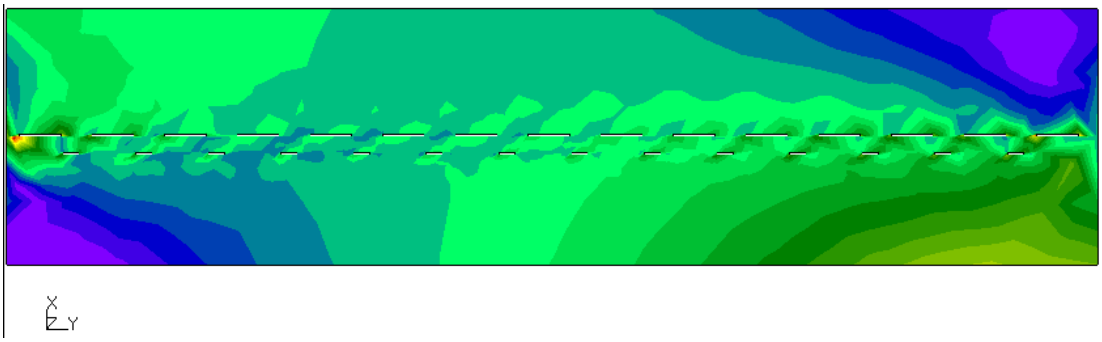
```
prob = pyc.Problem(model, 'struct')
try:
    prob.solve() # розв'язати
except:
    pass
```

Після розв’язання потрібно визначити крок і момент часу симуляції, в який потрібно отримати результати. В даному випадку задача статична, тому вказуємо кінцевий час симуляції 1,0. Після цього можна отримати рисунки з розподілами величин Sx, Sy (напруження по x і y), S1, S2, S3 (головні напруження), Seqv (еквівалентне напруження за Мізесом-Губером (рис. 2.4б)), ux, uy (деформація по x і y), utot (сумарна деформація (рис. 2.4а)).

```
prob.rfile.set_time(1.0) # результати в заданий момент часу
# рисунки результатів
fields = 'Sx,Sy,S1,S2,S3,Seqv,ux,uy,utot' #величини для рез.
fields = fields.split(',')
for field in fields: # для кожної величини
    fname = model_name+'_'+field
    prob.rfile.nplot(field, fname, display=False) # рисунок
```



а



б

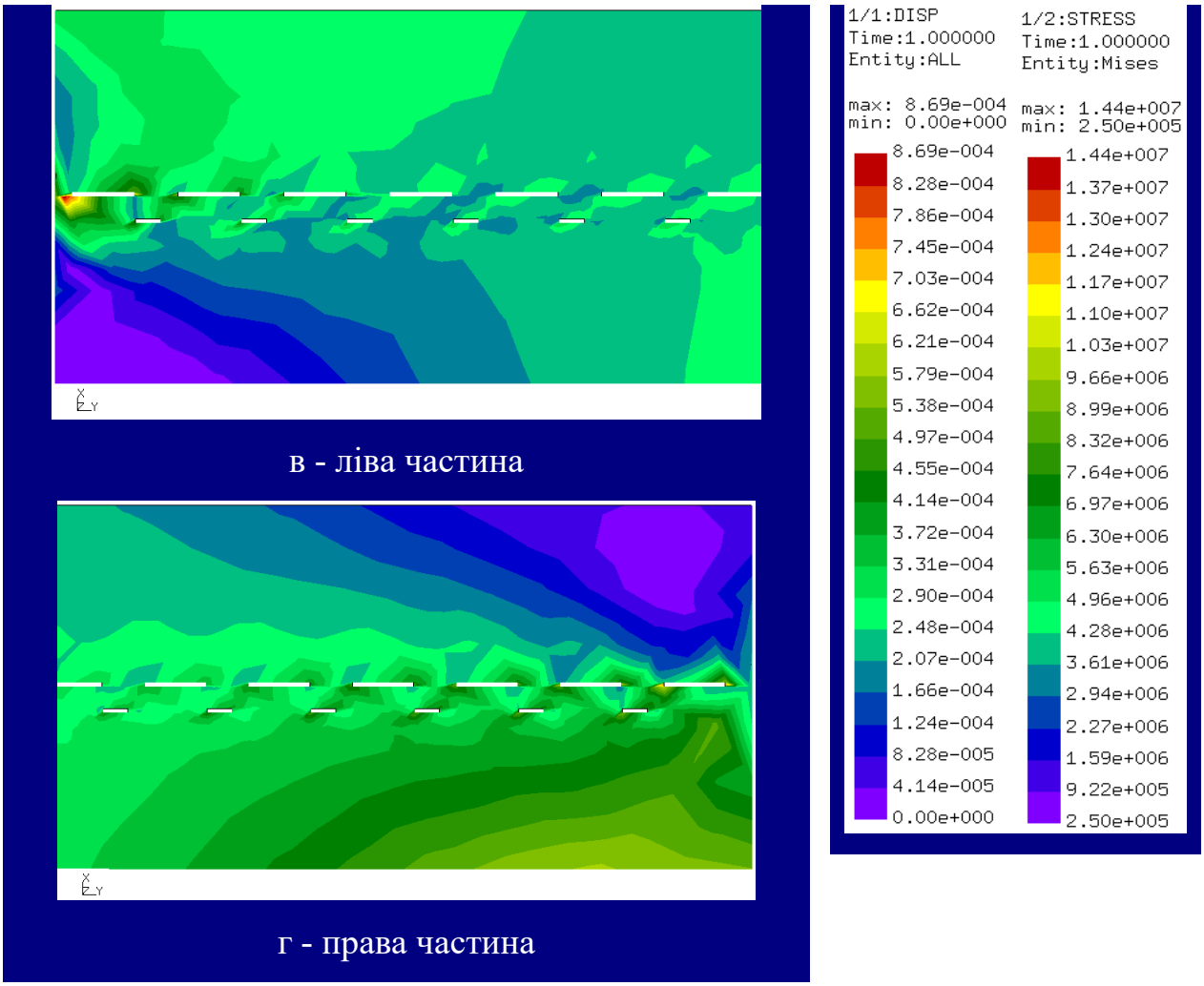


Рисунок 2.4 - Деформації (мм) (а) і еквівалентні напруження за Мізесом (Па) (б, в, г) в різьбовому з’єднанні

Можна також отримати максимальне значення еквівалентного напруження Seqv таким чином:

```
print(prob.rfile.get_nmax('Seqv'))
```

Отримаємо максимальне значення: 14367274 Па.

Наведений вище код для зручності його використання в мультиагентній системі, слід розмістити в функції run(). Тоді цю функцію слід викликати так:

```
if __name__=="__main__":
    show_gui = False
    hh=0.1
    print(run())
```

РОЗДІЛ 3. CAM-МОДУЛЬ ДЛЯ ФРЕЗЕРУВАННЯ РІЗЬБИ З ДОВІЛЬНИМ ПРОФІЛЕМ

3.1. Розроблення CAM-модуля мовою Python

Існує чимало CAM систем для підготовки програм для фрезерування різьб. Наприклад калькулятор [41]. Проте їх важко інтегрувати в мультиагентну PLM-систему. Для фрезерування різьби з довільним профілем на фрезерному верстаті з ЧПК автором у співавторстві з науковим керівником розроблено CAM-модуль [15] за допомогою мови програмування Python. Модуль має можливість генерувати програму для фрезерного трьохосьового верстата з системою ЧПК, яка підтримує лінійну інтерполяцію G01 та не має підтримки кругової (G02, G03). Це дещо знижує точність гвинтової лінії, або призводить до збільшення G-коду. Модуль також може бути використаний для систем ЧПК, які не мають підтримки корекції на радіус інструмента (G41 і G42). Підтримується фрезерування конічних зовнішніх та внутрішніх різьб з кількома проходами і довільним профілем різьби, який заданий координатами точок в площині X-Z. Цей модуль може бути використаний в якості компоненту мультиагентної системи підтримки життєвого циклу різьбових з'єднань.

Нижче наведено опис модуля. Кодування символів модуля - utf-8. Для зручних операцій над масивами, в яких зберігатимуться координати опорних точок, використаємо бібліотеку Numpy [42]:

```
# -*- coding: utf-8 -*-
import numpy as np
pi=np.pi
```

Функція helixPoints(r, h, p, fi, n, z0=0) повертає список точок гвинтової лінії [43]. Аргументи функції: r - мінімальний радіус, h - висота, p - крок, fi -

кут нахилу, n - кількість точок на одному витку, z0 - осьове зміщення (за замовчуванням 0). Координати точок визначаються з рівняння гвинтової лінії [43]:

$$x = (r + at)\cos(-t),$$
$$y = (r + at)\sin(-t),$$
$$z = bt + z0.$$

де $a = p \cdot \frac{tg(\varphi)}{2\pi}$, $b = \frac{p}{2\pi}$. Код функції наведено нижче:

```
def helixPoints(r, h, p, fi, n, z0=0):
    a = np.tan(fi)*p/(2*pi)
    b = p/(2*pi)
    k = h/p # кількість витків
    N = int(n*k) # загальна кількість точок
    t = np.linspace(0, h/b, N) # параметр
    x = (r + a*t)*np.cos(-t)
    y = (r + a*t)*np.sin(-t)
    z = b*t + z0 # вісь гвинтової лінії
    return zip(x,y,z)
```

Функція TreadMulti(R, Z, h, p, fi, n) повертає список точок для фрезерування різьби в кілька проходів. R, Z - списки з радіусами (x-координата) і z-координатою перших точок гвинтових ліній. Іншими словами це x,z-координати точок профілю. Для одного проходу R=[r] Z=[0], де r - мінімальний радіус різьби. h - висота, p - крок, fi - кут нахилу, n - кількість точок на одному витку. Довжина різьби h повинна бути кратна кроку p. Алгоритм функції показано на рис. 3.1.

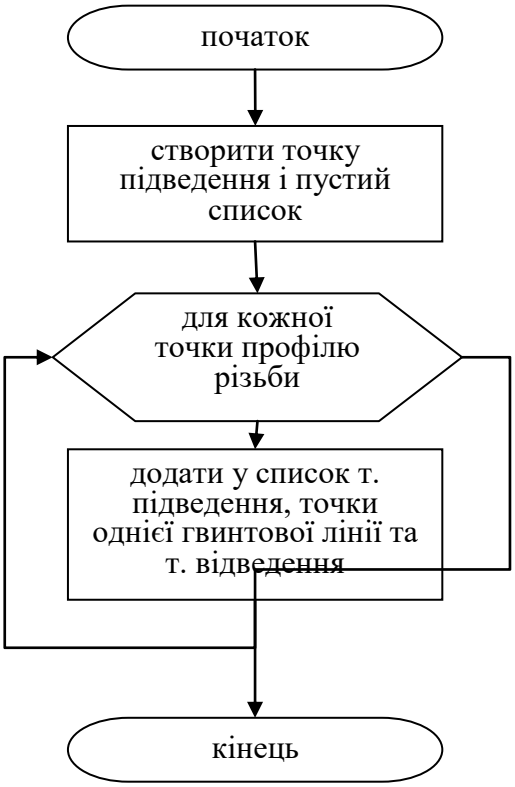


Рисунок 3.1 - Блок-схема алгоритму функції TreadMulti

Код функції:

```
def TreadMulti(R, Z, h=10, p=2, fi=0, n=64):
    fp=max(R)+10,0,0 # перша точка віддалена по x на 10 мм
    P=[] # список усіх точок
    for r, z in zip(R, Z): # для кожної точки профілю
        P.append(fp) # додати першу
        # точки 1 гвинтової лінії
        H=helixPoints(r=r, h=h, p=p, fi=fi, n=n, z0=z)
        P+=H # додати їх
        a=H[-1] # остання точка на гвинтовій лінії
        # додати точку відведення
        P.append((a[0]+10, a[1], a[2]))
    return P
```

Функція Gcode(P) повертає G-код за точками P. Алгоритм функції показано на рис. 3.2.

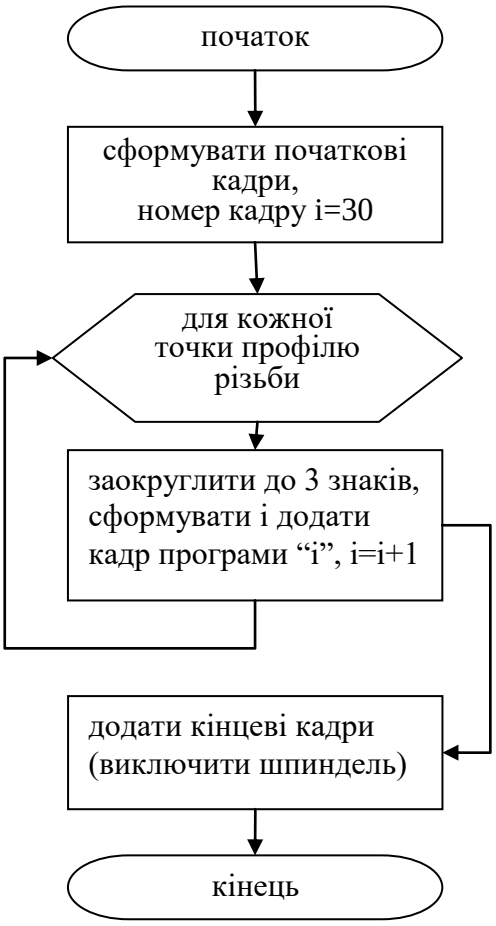


Рисунок 3.2 - Блок-схема алгоритму функції Gcode

Код функції:

```
def Gcode(P):
    S=["N10 G1 G54 G17 G21 G90 G94 M05 T0 F100 S1000", "N20
M03"] # поч. код
    i=30 # номер рядка
    for p in P: # для кожної точки
        x, y, z=p[0], p[1], p[2]
        x, y, z=(round(k, 3) for k in (x, y, z)) #
заокруглити до 3 знаків
        s="N%d G1 X%5.3f Y%5.3f Z%5.3f"%(i, x, y, z) # кадр
програми
```

```
        S.append(s)
        i+=10 # наступний кадр
    S.append("N%d M05"%i) # виключити шпиндель
    return "\n".join(S)
```

Нижче наведено приклад генерації G-коду для фрезерування спеціальної різьби довжиною 10 мм, профіль якої показано на рис. 3.3.

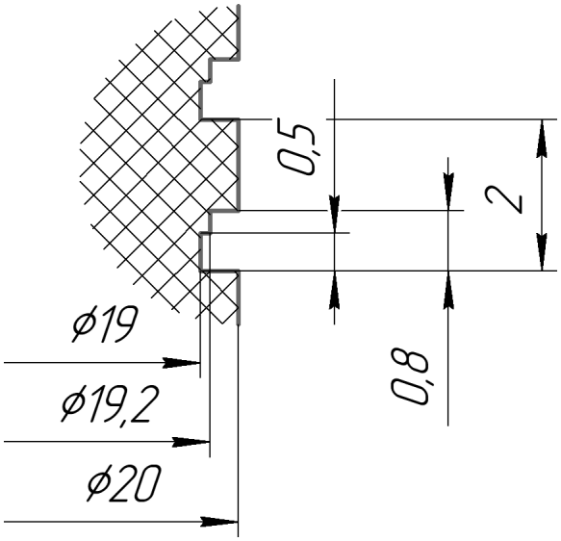


Рисунок 3.3 - Профіль спеціальної різьби

Проблемою є те, що нульова точка інструменту не збігається з опорними точками гвинтової лінії різьби. Ця проблема може бути вирішена шляхом впровадження у пристрій ЧПУ функції еквідистантного зміщення контуру на задану відстань, яка отримала назву корекції на радіус інструменту [44]. Програмісту достатньо запрограмувати траєкторію руху центру фрези, а при відпрацюванні її на верстаті вказати системі ЧПУ, яку величину потрібно зробити зміщення. Для включення функції корекції радіусу використовуються підготовчі команди G41 та G42, для зміщення вліво та вправо по ходу руху від вихідного контуру відповідно. Це дозволяє легко програмувати обробку того самого контуру фрезами різного діаметра. Для цього величина зсуву вводиться в спеціальну комірку таблиці

інструментів ЧПК, а в програмі позначається адресою D та номером комірки. Формат кадру програми має вигляд:

```
Nn G1 G41/G42 Xn.n Yn.n Dn Fn.n
```

де: G41/G42 – включення корекції на радіус ліворуч/праворуч;
Dn – номер комірки системи ЧПУ із заданим значенням зміщення;

Після проходження фрези відносно заданого контуру зі зміщенням корекцію необхідно вимкнути, для цього використовується функція G40. Для того, щоб активувати корекцію на радіус, необхідна деяка відстань, на якій траєкторія буде еквідистантно зміщена. Така відстань називається підведенням до контуру. Величина підведення, як правило, не повинна бути меншою за радіус фрези. Проте команди G41 та G42 є не обов'язковими, якщо застосовується CAM-система, а більш актуальні для створення G-коду «вручну», а також для введення корекції на спрацювання інструмента.

Спочатку вводимо радіус дискової фрези `rf` (0, якщо система ЧПК має команди корекції). Товщина фрези - 0,5 мм. Нуль інструмента знаходиться на нижньому торці на її осі. І генеруємо програму для перших двох проходів - примірочного r=10 і остаточного для першої точки профілю (r=9,5 мм, z0=0).

```
rf=10
P=TreadMulti(R=[10+rf, 9.5+rf], Z=[0, 0],  h=14, p=2, fi=0,
n=64)
code=Gcode(P) # код
```

Після цього зберігаємо цей код у файл:

```
f=open("cnc.txt", 'w')
f.write(code)
```

```
f.close()
```

В результаті отримаємо текстовий файл cnc.txt з наступним змістом (показана частина файлу):

```
N10 G1 G54 G17 G21 G90 G94 M05 T0 F100 S1000
N20 M03
N30 G1 X30.000 Y0.000 Z0.000
N40 G1 X20.000 Y-0.000 Z0.000
N50 G1 X19.903 Y-1.965 Z0.031
N60 G1 X19.614 Y-3.910 Z0.063
N70 G1 X19.135 Y-5.818 Z0.094
...
N9030 M05
```

Щоб згенерувати програму для наступних двох проходів - примірочного r=10 і остаточного для 2-ї точки профілю (r=9,6 мм, z0=0,3 мм) потрібно ввести:

```
P=TreadMulti(R=[10+rf, 9.6+rf], Z=[0.3, 0.3],  h=14, p=2,
fi=0, n=64)
```

Цього разу вміст файлу буде наступним (показана частина файлу):

```
N10 G1 G54 G17 G21 G90 G94 M05 T0 F100 S1000
N20 M03
N30 G1 X30.000 Y0.000 Z0.000
N40 G1 X20.000 Y-0.000 Z0.300
N50 G1 X19.903 Y-1.965 Z0.331
N60 G1 X19.614 Y-3.910 Z0.363
N70 G1 X19.135 Y-5.818 Z0.394
```

```
...
N9030 M05
```

Можна також додатково згенерувати ще два чистових проходи з малою подачею 50 мм/хв.

Візуалізація опорних точок відбувається за допомогою графічної бібліотеки Matplotlib [45]:

```
import matplotlib.pyplot as plt
# координати
X=[p[0] for p in P]
Y=[p[1] for p in P]
Z=[p[2] for p in P]
# тривимірні графіки
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure() # рисунок
ax = Axes3D(fig) # система координат
ax.plot3D(X,Y,Z, 'ko-') # лінії
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()
```

Результати для останніх двох проходів показані на рис. 3.4.

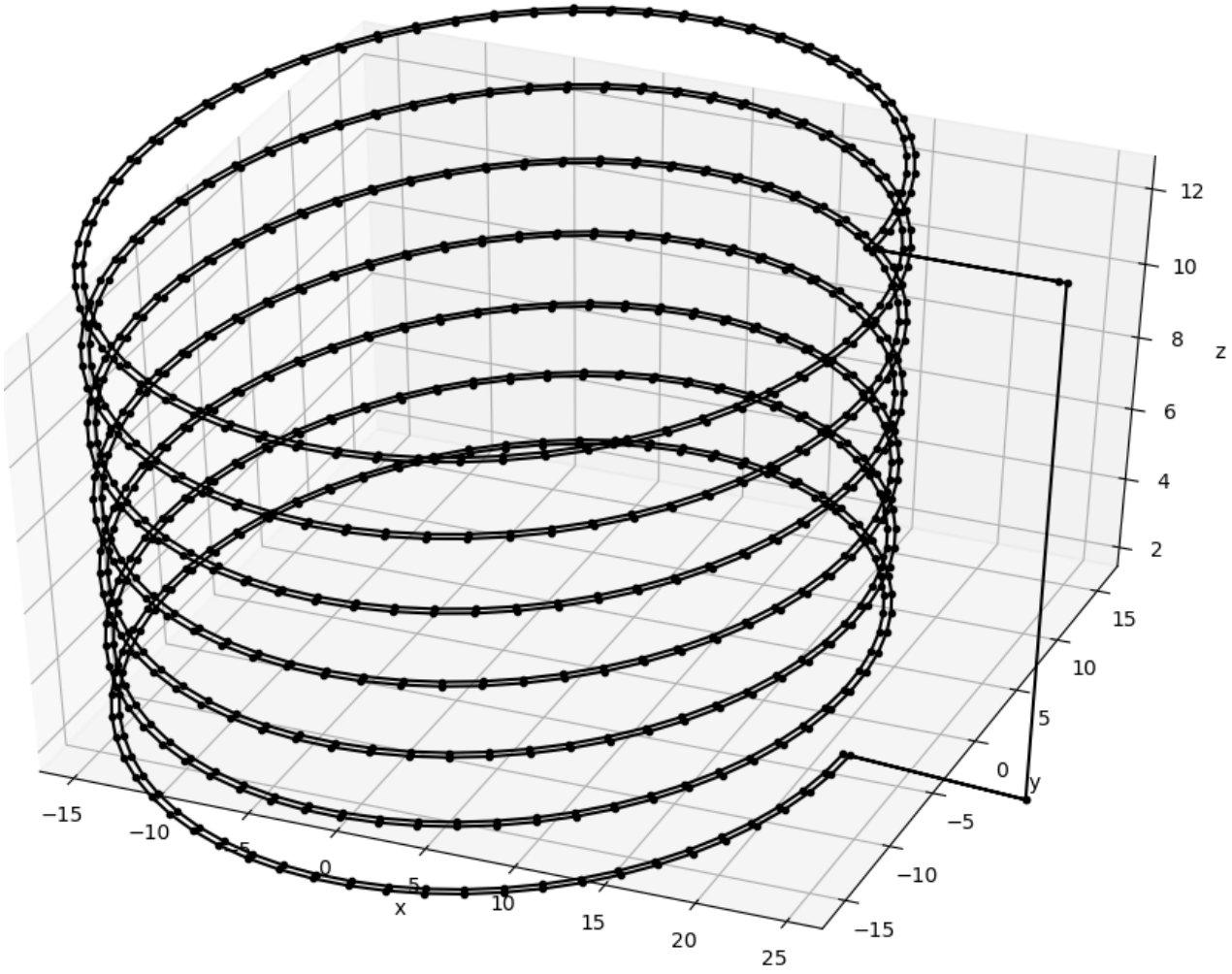


Рисунок 3.4 - Візуалізація траєкторії переміщення т.0 інструмента за допомогою Matplotlib

3.2. Фрезерування різьби за згенерованою CAM-модулем програмою

Виконаємо фрезерування різьби на CNC-роутері CNC 3018 Pro. Заготовка - труба поліпропіленова зовнішнім діаметром 20 мм. Матеріал заготовки - поліпропілен Pn-20. Діаметр фрези D=20 мм, кількість зубів z=1. Режими різання: швидкість різання v=60 м/хв, прийнята частота обертання шпинделя n=1000 об/хв, рекомендована подача Sхв=100 мм/хв (чорновий прохід), Sхв=50 мм/хв (чистовий прохід). Дійсна швидкість різання:

$$v = \frac{\pi D n}{1000} = \frac{3{,}14 * 20 * 1000}{1000} = 62{,}8 \frac{\text{м}}{\text{хв}}.$$

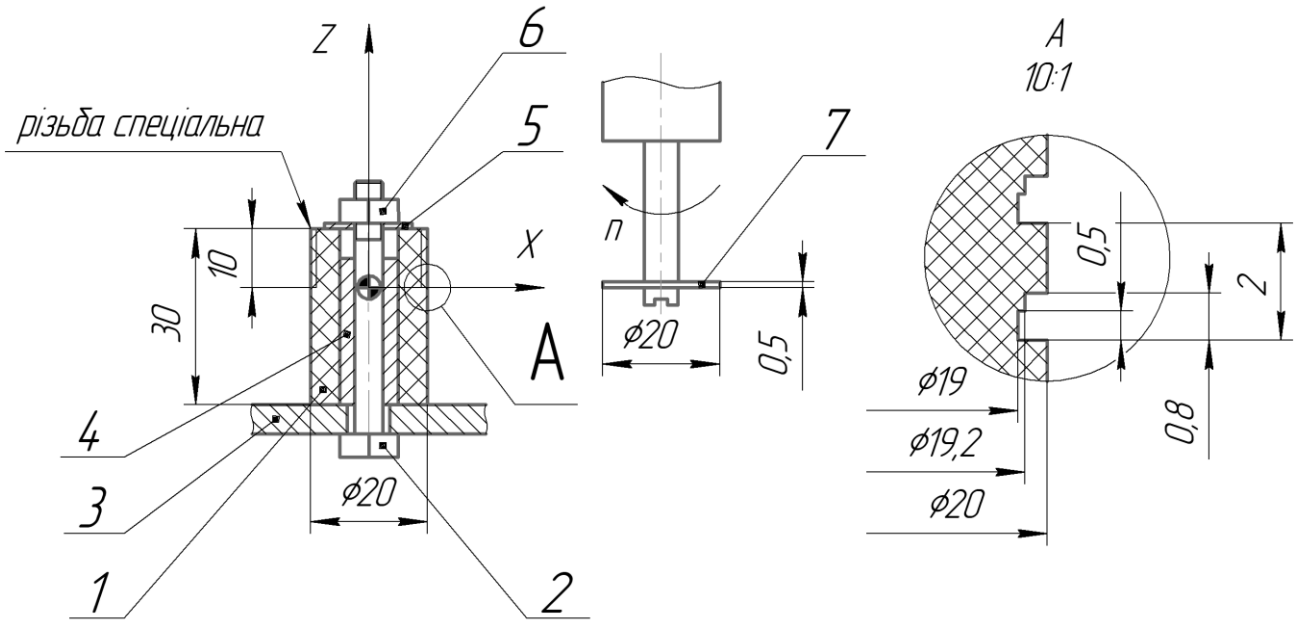
Подача на зуб для чорнової обробки:

$$Sz = \frac{S}{nz} = \frac{100}{1000 * 1} = 0{,}1 \frac{\text{мм}}{\text{зуб}}.$$

Подача на зуб для чорнової обробки:

$$Sz = \frac{S}{nz} = \frac{50}{1000 * 1} = 0{,}05 \frac{\text{мм}}{\text{зуб}}.$$

Технологічний ескіз показаний на рис. 3.5.



1 - заготовка, 2 - болт, 3 - стіл верстата з Т-подібними пазами, 4 - втулка, 5 - шайба, 6 - гайка, 7 - дискова фреза

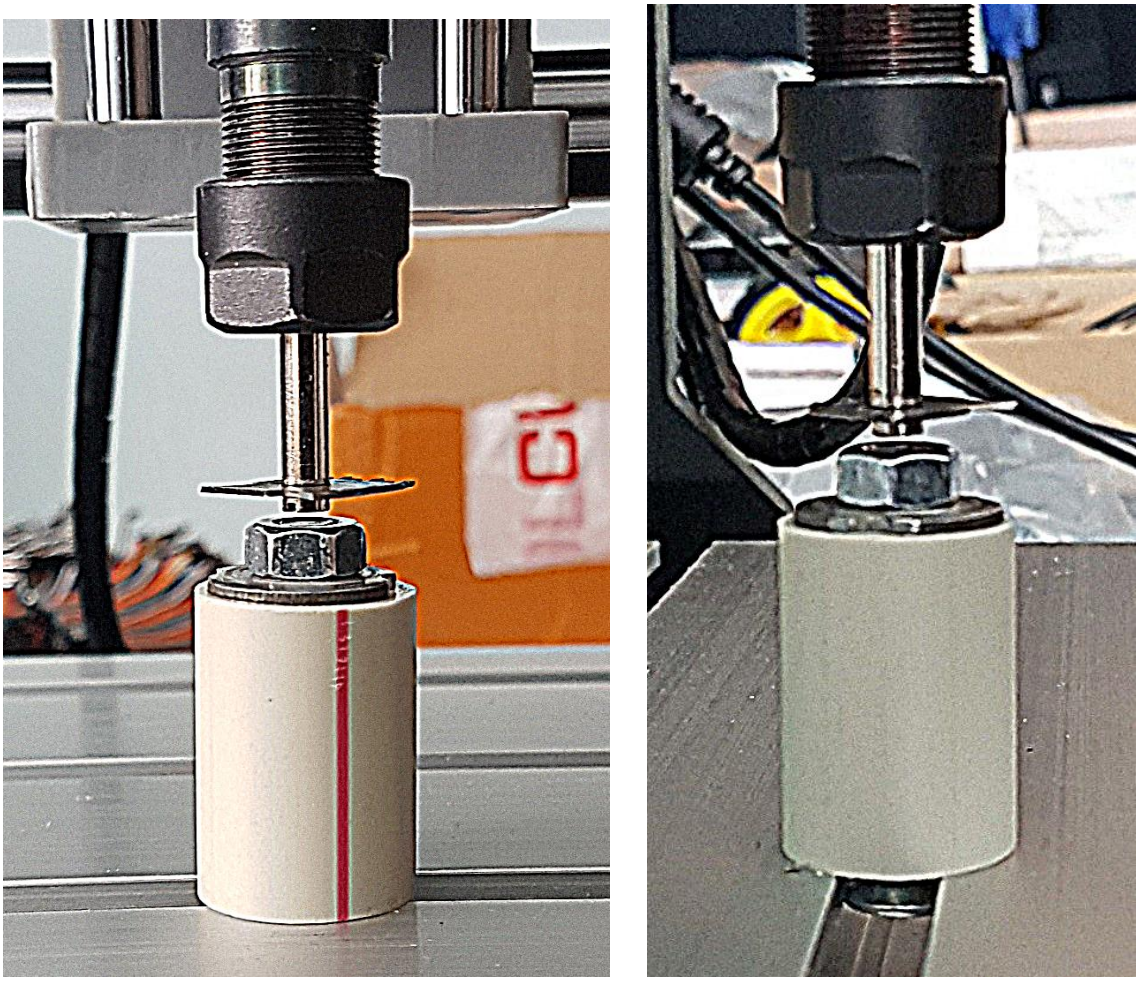
Рисунок 3.5 - Технологічний ескіз фрезерування спеціальної різьби

У першу чергу потрібно закріпити заготовку 1 і виставити фрезу в задане положення. Для закріплення використаємо базування на циліндричний палець 4 з мінімальним зазором (рис. 3.5, 3.6).

Рисунок 3.6 - Заготовка, закріплена на верстаті CNC 3018 Pro

Робочий центр координат (0, 0, 0) повинен знаходитись внизу гвинтової лінії, в її центрі (рис. 3.5). Нуль інструмента знаходиться на нижньому торці дискової фрези на її осі. Спочатку потрібно виставити вісь

фрези так, щоб вона точно збігалась з віссю заготовки. Для виставлення фрези в положення використовуємо кнопки Jog і крок 1..0.01 мм. Використано штангенциркуль для вимірювання відстані від фрези до циліндричної поверхні заготовки у двох взаємоперпендикулярних площинах (рис. 3.7, 3.8). Якщо осі збіглися, натискаємо  , щоб позначити X=0, Y=0.



спереду зліва

Рисунок 3.7 - Виставлення нуля інструмента в точку X=0, Y=0

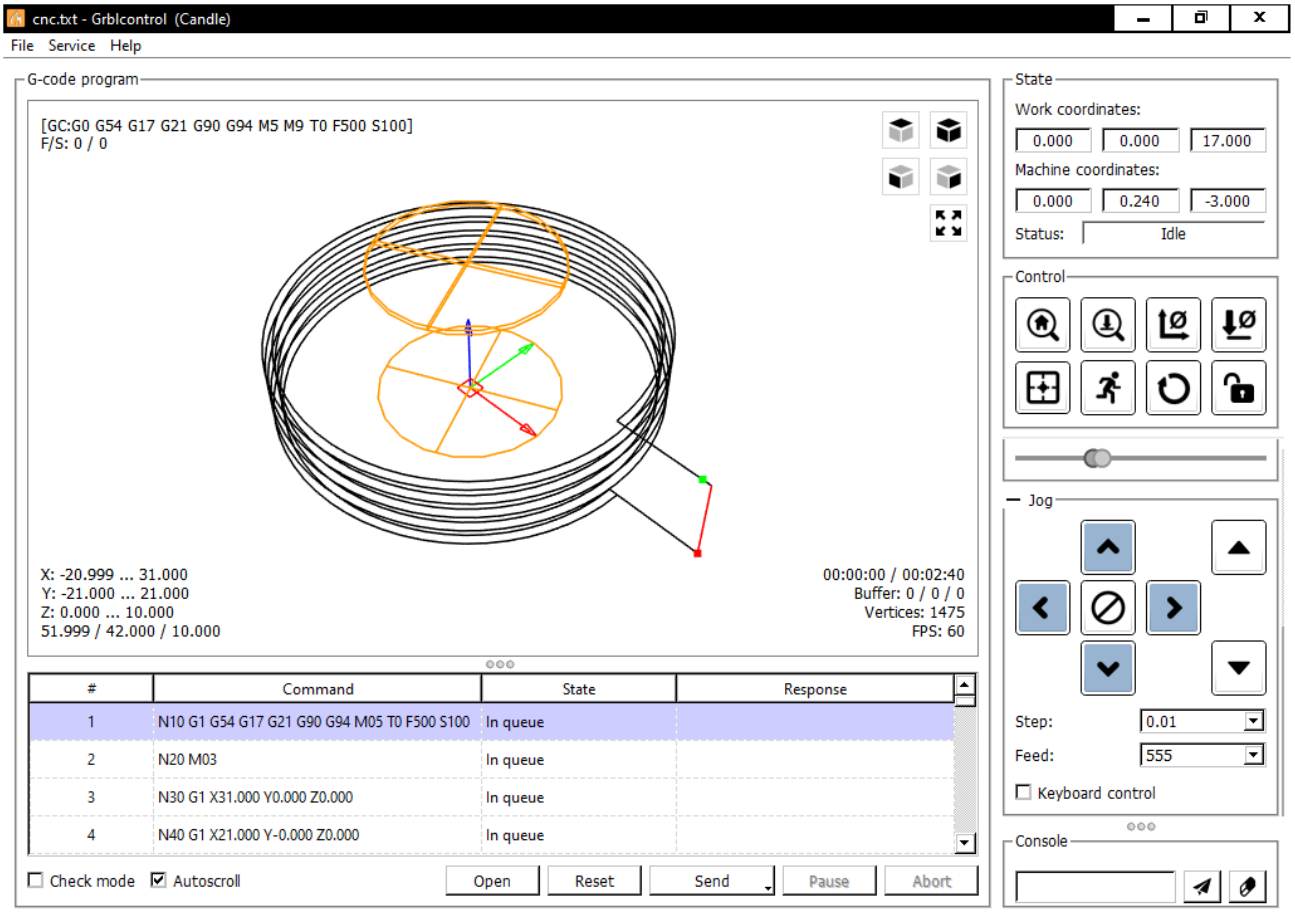


Рисунок 3.8 - Вікно програми Candle - виставлення нуля інструмента в точку X=0, Y=0

Після цього відводимо фрезу в сторону (рис. 3.9) і опускаємо її нижче верхнього торця заготовки на довжину різьби - 10 мм. Натискаємо  , щоб позначити Z=0 (рис. 3.10). Таким чином робочий центр координат (0, 0, 0) задано.

Завантажуємо згенеровану CAM-модулем програму (Open) і виконуємо її (Send). Чекаємо виконання фрезерування в два проходи. Основний час фрезерування To=18 хв.

Якщо пробний прохід не успішний (фреза не точно виставлена в 0), натискаємо Abort і відвівши фрезу в сторону повертаємо її в X=0, Y=0 за допомогою команди G0 X0 Y0, яку потрібно ввести в полі Console.

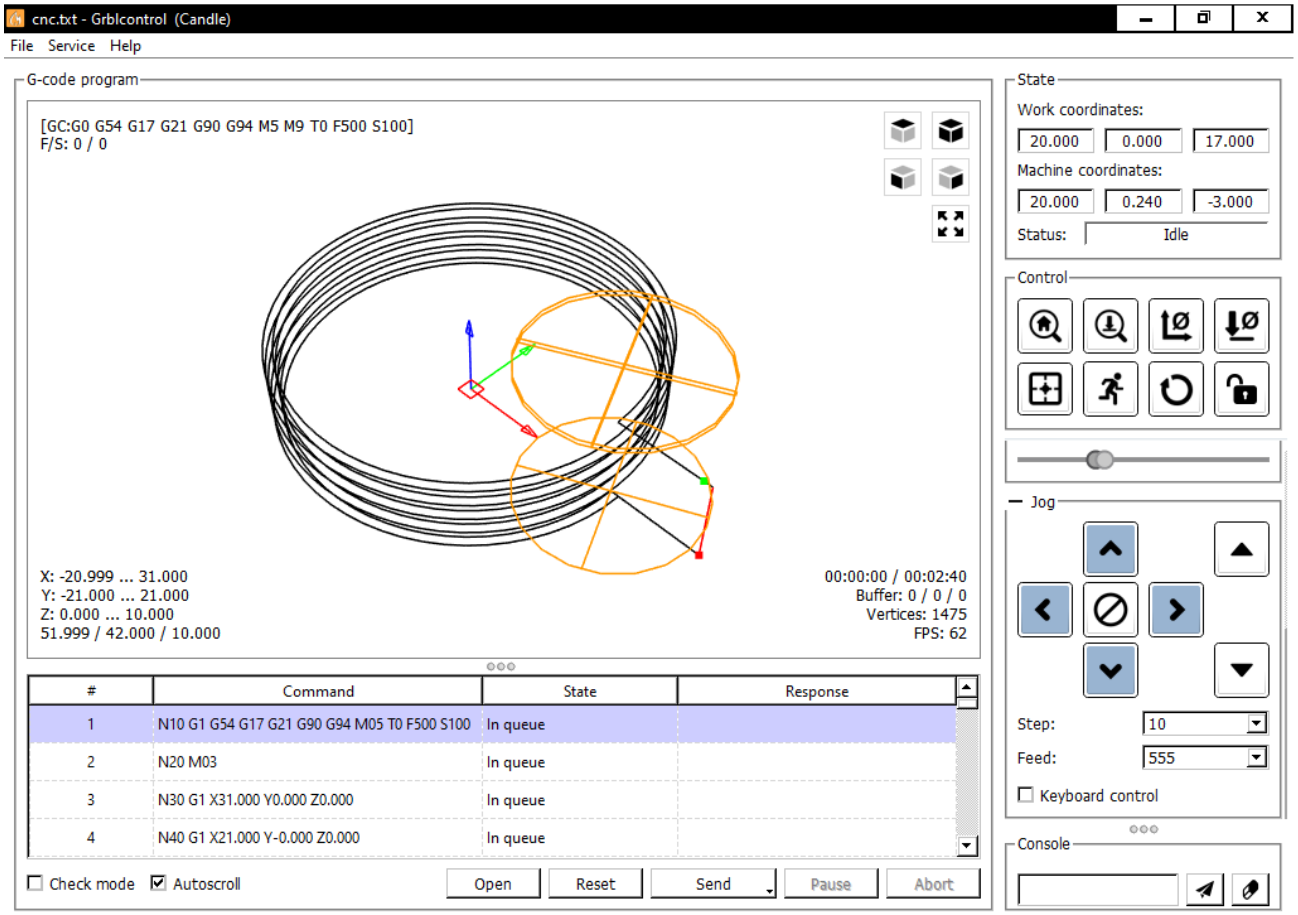


Рисунок 3.9 - Вікно програми Candle - відведення фрези вправо

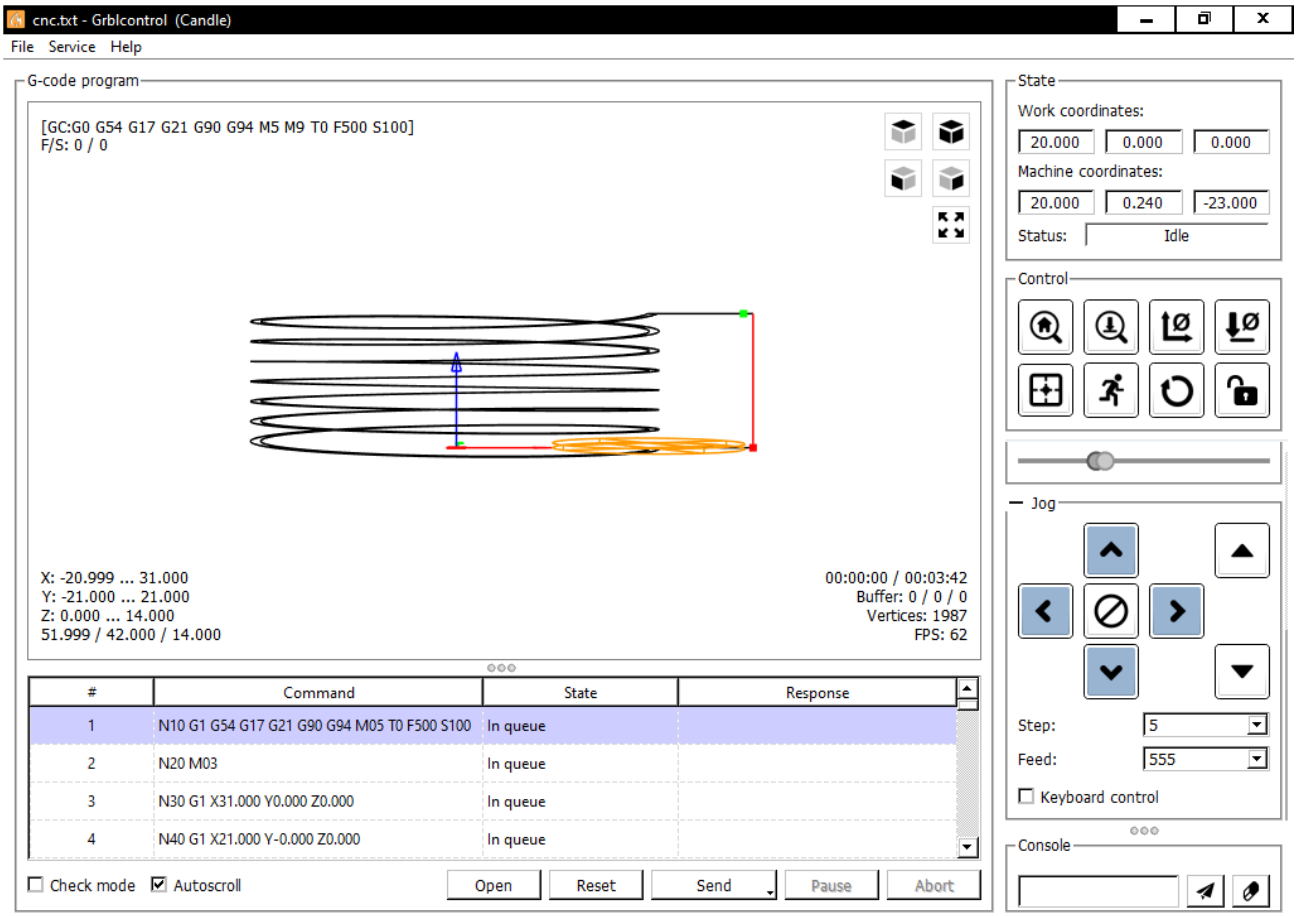
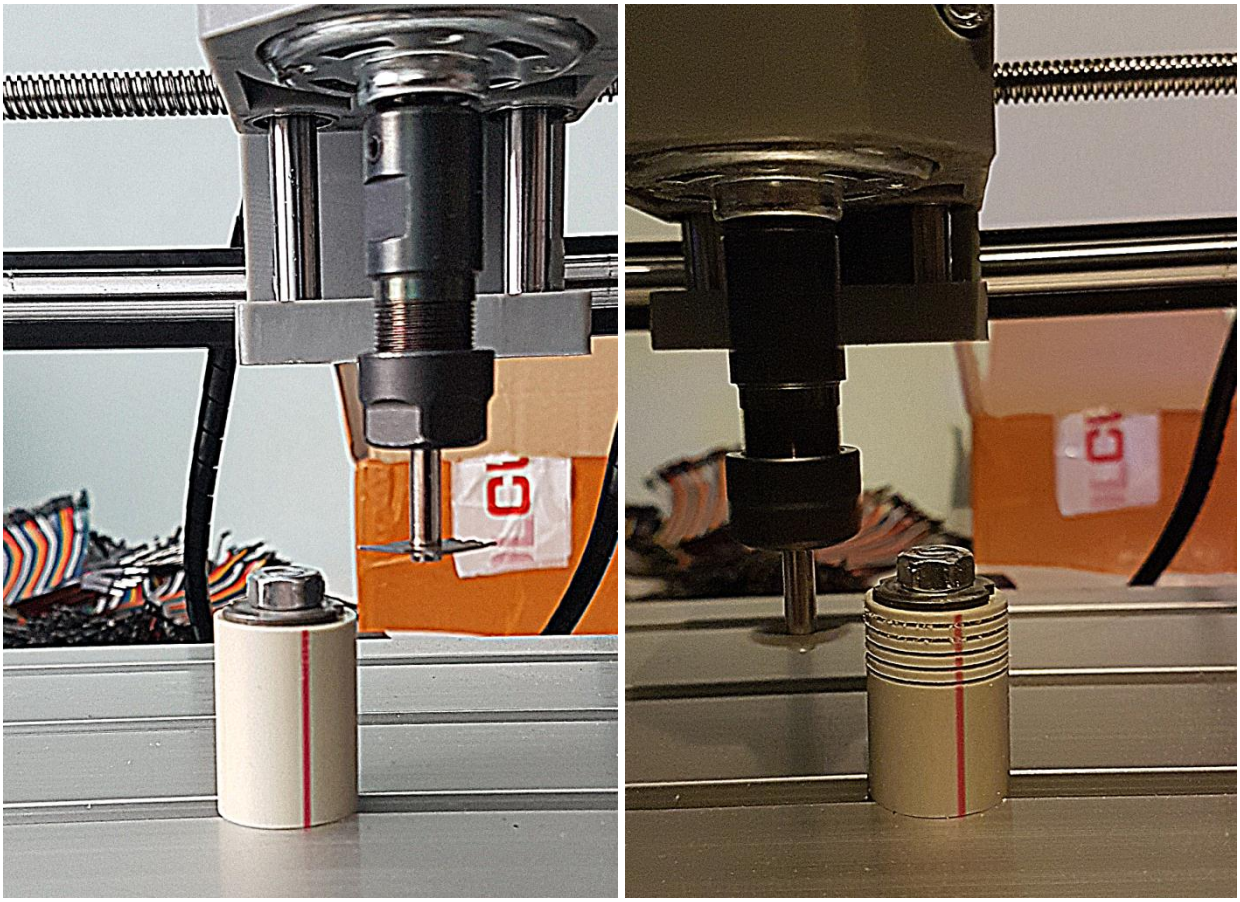
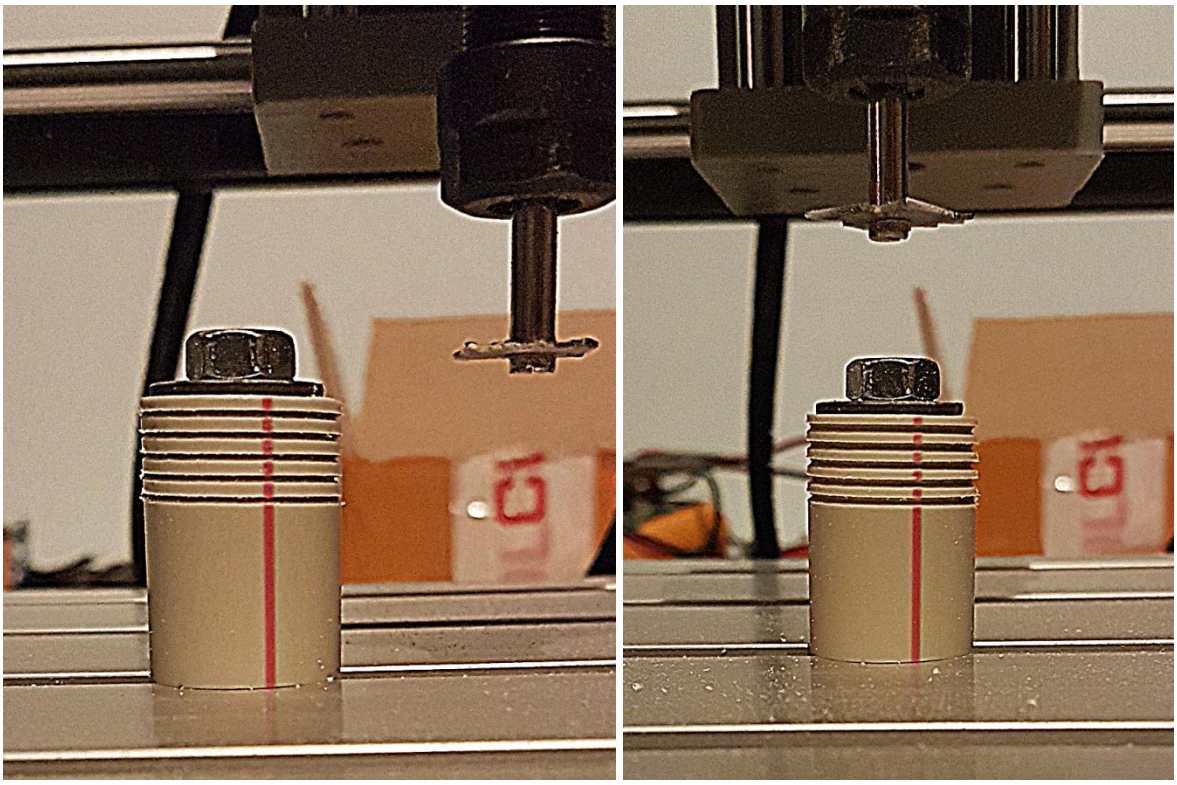


Рисунок 3.10 - Вікно програми Candle - виставлення фрези в координату Z=0



а б



в г

Рисунок 3.11 - Демонстрація етапів фрезерування спеціальної різьби: а - заготовка, б - фрезерування першої точки профілю, в - фрезерування другої точки профілю, г - готова деталь

Робимо потрібні зміни щоб осі збіглися і натискаємо  , щоб позначити X=0, Y=0. Тепер, щоб повернутись в Z=0 достатньо відвести фрезу в сторону і ввести в консолі G0 Z0.

Генеруємо програму для наступної точки профілю і виконуємо фрезерування в два проходи. Можна також разом виконати фрезерування усіх проходів за одним кодом. Для здійснення чистових проходів (за потреби) достатньо змінити значення подачі в програмі з F100 на F50. На рис. 3.11 показані етапи фрезерування даної різьби.

РОЗДІЛ 4. ПРИКЛАД СТВОРЕННЯ І ВИКОРИСТАННЯ МУЛЬТИАГЕНТНОЇ PLM-СИСТЕМИ РІЗЬБОВИХ З'ЄДНАНЬ

4.1. Підготовчі дії

Нижче наведено приклад інтерактивного документа Jupyter Notebook [46], який містить у відповідних комірках відформатований за допомогою мови розмітки Markdown [47] текст та програмний код мовою програмування Python. Цей документ можна використовувати як мультиагентну PLM-систему для підтримки життєвого циклу різьбового з'єднання, зокрема етапів проектування конструкції та проектування технології (рис. 4.1). Існує можливість додати агенти для підтримки етапів виробництва і експлуатації.

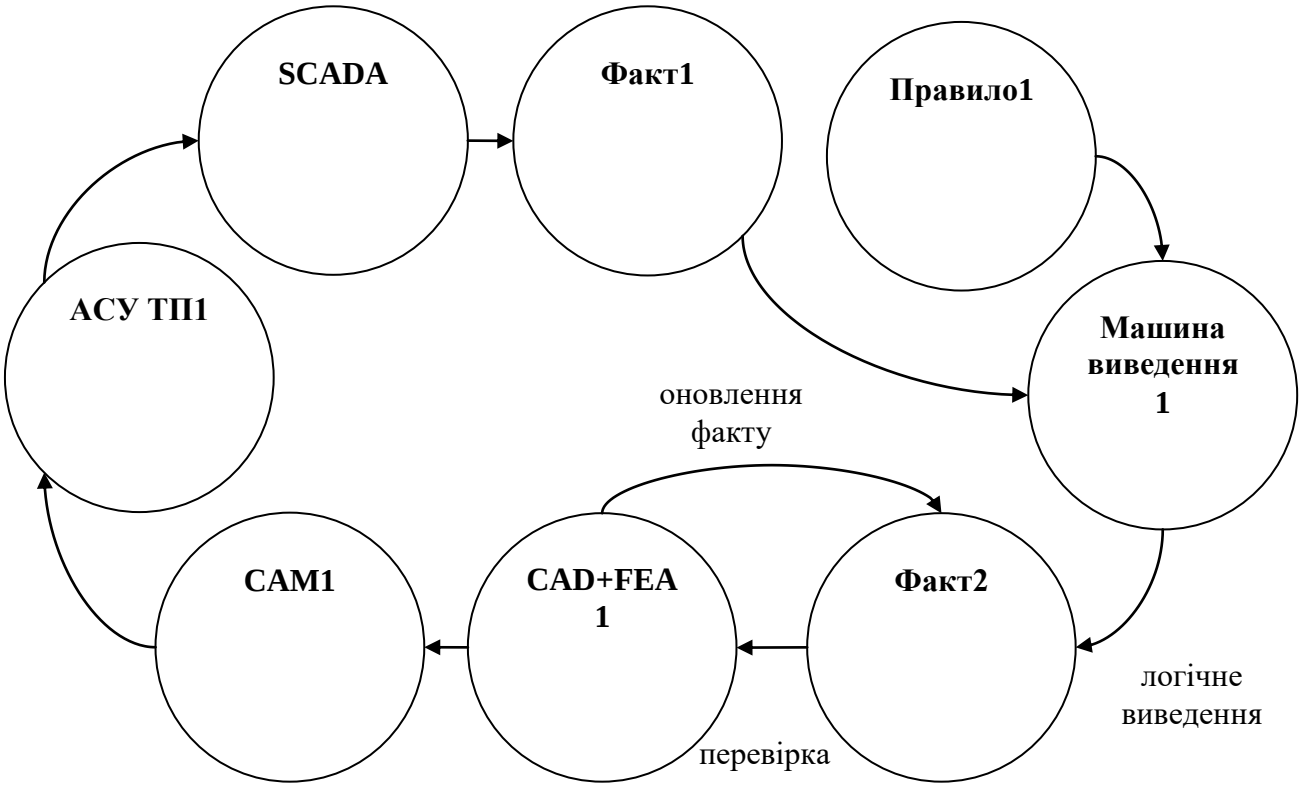


Рисунок 4.1 - Приклад мультиагентної PLM-системи

Розподілена МАС створена на основі концепції акторів і Python Ray. Кожен агент (актор) може виконуватись паралельно разом з іншими агентами. Можна організувати автоматичну роботу системи за допомогою простого алгоритму: виконувати функцію rule кожного агента, поки ці функції повертають якісь значення, що не рівні None. В даному прикладі це

реалізовано для логічного виведення нових фактів з бази існуючих фактів і правил логічного виведення.

Інші Python-модулі слід розмістити в одному каталозі з документом ThreadPLM_ray.ipynb. Для запуску документа введіть в командному рядку:

```
"d:\Portable\Portable Python-3.8.9 x64\App\Python\Scripts\jupyter-notebook.exe" ThreadPLM_ray.ipynb
```

Передусім потрібно ініціалізувати систему ray. В даному прикладі використовуються виключно ресурси одного комп'ютера, але можна використати і кластер.

Приклад документу:

In [1]:
```
import ray
ray.init()
```
```
D:\Portable\Portable Python-3.8.9 x64\App\Python\lib\site-packages\redis\connection.py:72: UserWarning: redis-py works best with hiredis. Please consider installing
  warnings.warn(msg)
```
Out[1]:
```
{'node_ip_address': '127.0.0.1',
 'raylet_ip_address': '127.0.0.1',
 'redis_address': '127.0.0.1:6379',
 'object_store_address': 'tcp://127.0.0.1:64265',
 'raylet_socket_name': 'tcp://127.0.0.1:53842',
 'webui_url': None,
 'session_dir': 'D:\\Temp\\ray\\session_2021-12-18_10-57-53_554289_1072',
 'metrics_export_port': 64337,
 'node_id': '8e1d59da19d0edad89b06146e60c9d199cf217529e16e73b7baba675'}
```

4.2. Клас акторів Fact

Клас описує поняття факту у вигляді триплету (суб'єкт, предикат, об'єкт). Функція rule повертає кортеж триплету.

In [2]:
```
@ray.remote
class Fact(object):
    def __init__(self,s,p,o):
        self.s, self.p, self.o = s, p, o
    def rule(self):
        return self.s, self.p, self.o
```

4.3. Клас акторів Rule1

Клас описує поняття правила логічного виведення для симетричної властивості p: (1, p, 3)->(3, p, 1). Для пришвидшення алгоритму використовується техніка мемоїзації - в атрибуті ready зберігаються оброблені триплети. Однократно обробляється множина триплетів ts. Повертається множина нових триплетів.

In [3]:
```
@ray.remote
class Rule1(object):
    def __init__(self,p):
        self.p = p
        self.ready=set()
    def rule(self, ts=None):
        """Симетрична властивість 2: (1, 2, 3)->(3,
2, 1)"""
        tn=set()
        for s,p,o in ts:
            if self.p!=p: continue
            if (s,p,o) in self.ready: continue
            self.ready.add((s,p,o))
            self.ready.add((o,p,s))
            tn.add((o,p,s))
        return tn
```

4.4. Клас акторів Rule2

Клас описує поняття правила логічного виведення для транзитивної властивості p: (1, p, 3)&(3, p, 4)->(1, p, 4). Однократно обробляється множина триплетів ts. Повертається множина нових триплетів. Для пришвидшення алгоритму можна використати техніку мемоїзації і зберігати множину оброблених триплетів в ready. З множини ts можна буде вилучити ready.

In [4]:

```
@ray.remote
class Rule2(object):
    def __init__(self,p):
        self.p = p
        self.ready=set()
    def rule(self, ts=None):
        """Транзитивна властивість 2: (1, 2, 3)&(3,
2, 4)->(1, 2, 4)"""
        tn=set()
        X=set()
        for s,p,o in ts:
            if self.p!=p: continue
            X.add((s,p,o))
            for sx,px,ox in X:
                if (ox==s and sx!=o):
                    if (sx,p,o) in self.ready:
continue
                    tn.add((sx,p,o))
                    self.ready.add((sx,p,o))
                elif (sx==o and ox!=s):
                    if (s,p,ox) in self.ready:
continue
                    tn.add((s,p,ox))
                    self.ready.add((s,p,ox))
        return tn
```

4.5. Клас акторів Reasoner

Клас описує поняття машини логічного виведення. Функція rule запускає ОДНОКРАТНО паралельно правила агентів rs, яким передаються

триплети з фактами ts. Функція повертає множину триплетів з новими фактами. Аналогічно є можливість використати техніку мемоїзації, але це в класі не реалізовано.

In [5]:
```
@ray.remote
class Reasoner(object):
    def __init__(self):
        self.ready=set()
    def triplets(self, fs):
        refs=[f.rule.remote() for f in fs]
        ts=ray.get(refs)
        return ts
    def rule(self, rs, ts):
        refs=[r.rule.remote(ts) for r in rs] #
запустити паралельно
        T=ray.get(refs) # список множин триплетів
        r=set()
        for t in T:
            r.update(t)
        return r
```

4.6. Клас акторів FEA

Клас описує поняття FEA системи для скінченно-елементної симуляції моделі різьбового з'єднання. Для моделювання використовується pycalculix. Функція rule отримує параметр з'єднання x і повертає максимальне напруження. Для мемоїзації використовується словник ready.

In [6]:
```
@ray.remote
class FEA(object):
    def __init__(self):
        self.ready=dict()
    def rule(self, x=0.1):
        #if x in self.ready: return None
        import mypycalculix
        mypycalculix.show_gui = False
        mypycalculix.hh=x
```

```
        s=mypycalculix.run()
        self.ready[x]=s
        return s
```

4.7. Клас акторів CAM

Клас описує поняття CAM системи для генерації G-коду для фрезерування різьби ніпеля на 3-осьовому фрезерному верстаті з ЧПК. Функція rule отримує параметр з'єднання x і повертає відповідний список опорних точок G-коду. Для мемоїзації використовується атрибут ready.

In [7]:

```
@ray.remote
class CAM(object):
    def __init__(self):
        self.ready=dict()
    def rule(self, x=0.1):
        #if x in self.ready: return None
        import CNC_thread_mill
        rf=CNC_thread_mill.rf=10 # радіус фрези (0,
якщо система ЧПК має команди корекції)
        # товщина фрези 0.5 мм
        # перші два проходи - примірочний r=10 і
остаточний для 1-ї точки профілю
        P=CNC_thread_mill.TreadMulti(R=[10+rf,
9.5+x+rf], Z=[0, 0],  h=14, p=2, fi=0, n=64)
        code=CNC_thread_mill.Gcode(P, "cnc.txt") #
код, зберегти у файл
        self.ready[x]=P
        return P
```

4.8. Створення агентів системи (акторів)

Кожен агент може працювати паралельно з іншими. Проте це залежить від кількості процесорів на кластері. В даному випадку введені факти, які містять відповідність значень вхідних і вихідних параметрів моделі різьбового з'єднання: геометричного параметра h (мм), максимального еквівалентного напруження в різьбі Seqv (МПа), максимального сумарного

переміщення utot (мм). Ці факти можуть бути отримані в результаті моделювання, експериментів, промислових даних на попередніх етапах ЖЦ.

In [8]:
```
f1=Fact.remote("Seqv=14", "відповідає", "h=0.1")
f2=Fact.remote("h=0.1", "відповідає", "utot=869e-6")
f3=Fact.remote("h=0.2", "відповідає", "Seqv=16")
r1=Rule1.remote("відповідає")
r2=Rule2.remote("відповідає")
r=Reasoner.remote()
fea1=FEA.remote()
cam1=CAM.remote()
```

4.9. Логічне виведення фактів

Функція rule агентів, які містять правила, викликається до тих пір, поки є результати (нові факти). Виводяться триплети з новими фактами. Ці тріплети можна використати для створення нових агентів-фактів.

In [9]:
```
fs=[f1, f2, f3]
rs=[r1, r2]
T=set()
re=1
while re:
    re=0
    ts=ray.get(r.triplets.remote(fs))
    tn=ray.get(r.rule.remote(rs, ts))
    if tn:
        T.update(tn)
        re+=1
print(T)
{('Seqv=16', 'відповідає', 'h=0.2'), ('utot=869e-6',
'відповідає', 'h=0.1'), ('Seqv=14', 'відповідає',
'utot=869e-6'), ('h=0.1', 'відповідає', 'Seqv=14')}
```

4.9. FE-симуляція

За допомогою агента fea1 виконується симуляція різьбового з'єднання з параметром h=0,1 мм (зазор в різьбі). Відразу очікуємо результату. Але є

можливість запустити симуляцію асинхронно ro=fea1.rule.remote(0.1) і отримати результат в комірках нижче y1=ray.get(ro).

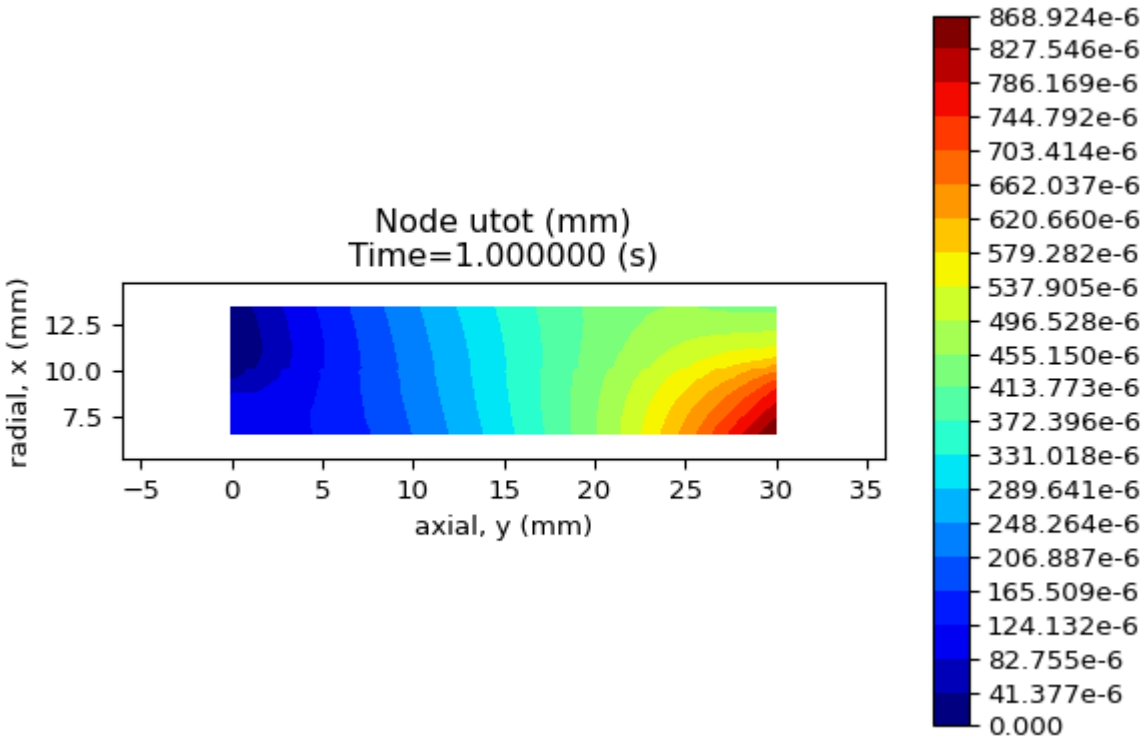
In [10]:
```
%%capture
y1=ray.get(fea1.rule.remote(0.1))
```

Виведення результатів симуляції. Максимальне еквівалентне напруження за Мізесом (Па):

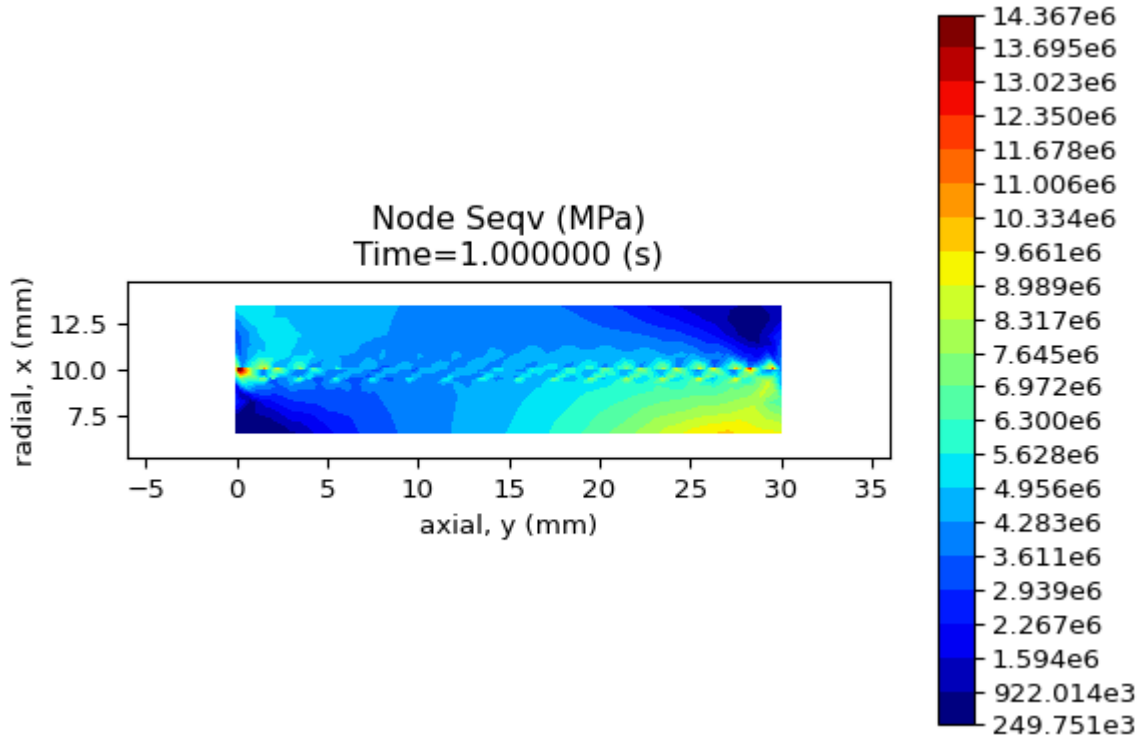
In [11]:
```
y1
```
Out[11]:
```
14367274.322702967
```

Агент fea1 (за допомогою pycalculix) зберігає результати в робочому каталозі в рисунках png. Ці рисунки відображаються тут за допомогою Markdown розмітки `![name.png](name.png)`.

Сумарні деформації (мм):



Еквівалентні напруження Мізеса (Па):



4.10. Генерація G-коду для фрезерування різьби

Генерація G-коду для фрезерування зовнішньої різьби з параметром h=0,1 мм відбувається за допомогою агента cam1. Відразу очікуємо результату. Але є можливість запустити симуляцію асинхронно ro=cam1.rule.remote(0.1) і отримати результат в комірках нижче y1=ray.get(ro).

In [12]:
```
y2=ray.get(cam1.rule.remote(0.1))
 pid=2384) File thread_utot.png was saved.
```

Перші опорні точки згенерованої траєкторії:

In [13]:
```
y2[0:5]
```

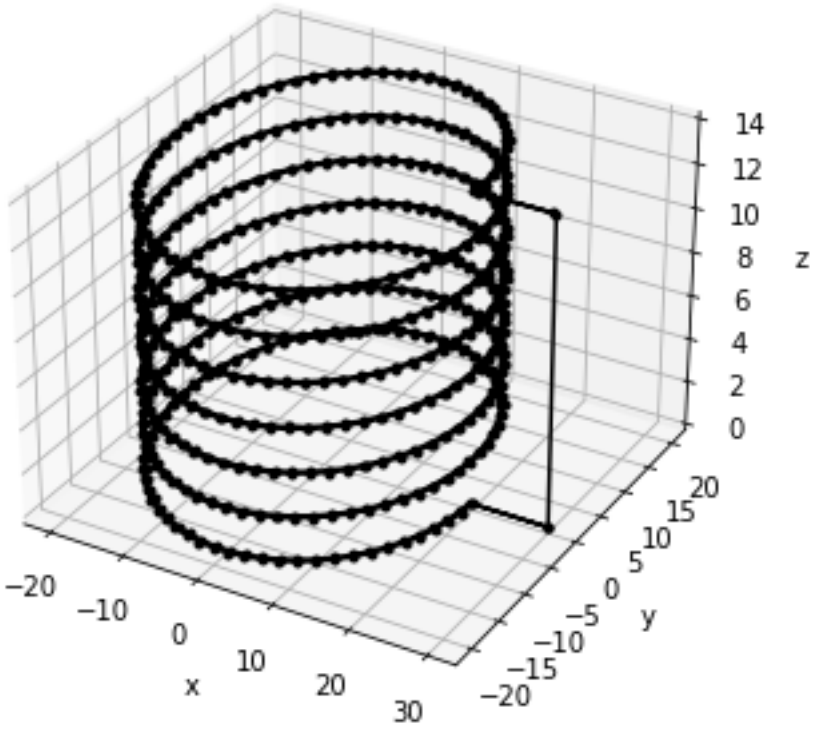
Out[13]:
```
[(30, 0, 0),
 (20.0, -0.0, 0.0),
 (19.90326350260106, -1.9647142148487098,
0.03131991051454139),
 (19.613989805397143, -3.9104224725439822,
0.06263982102908278),
```

```
 (19.13497724081396, -5.818302672904855,
0.09395973154362416)]
```

Візуалізація опорних точок траєкторії за допомогою Matplotlib:

In [14]:

```
import CNC_thread_mill
CNC_thread_mill.plotCNC(y2)
```



Робота з документом може бути ітеративною, тобто на будь-якому етапі можна вернутись до потрібних комірок, змінити дані, виконати код і отримати нові результати. Якщо потрібно завершити роботу з ray, введіть:

In [15]:

```
#ray.shutdown()
```

ВИСНОВКИ

1. Виконано огляд концепцій життєвого циклу виробів та інформаційних систем для підтримки життєвого циклу виробів. Виявлено, що така система повинна бути ізоморфною до інших складних систем, тобто володіти загальносистемними закономірностями. Одним із способів забезпечити це є організування мультиагентної системи, паралелізм якої забезпечується за допомогою моделі акторів.

2. Виконано огляд побудови мультиагентних систем на основі моделі акторів і пакету Ray. Виявлено, що пакет Ray дозволяє шляхом простої модифікації коду перетворити існуючу послідовну об’єктно-орієнтовану систему у високопродуктивну систему з паралельними агентами.

3. Виконано огляд технології різбофрезерування на верстатах з ЧПК. Основною перевагою цього методу є здатність цих верстатів до високої концентрації операцій.

4. Розроблено гнучкий CAM-модуль для різьбофрезерування різьби з довільним профілем. Модуль підтримує нарізування конічних різьб на системах ЧПК, які не мають корекції радіуса інструмента та кругової інтерполяції. Використання модуля для різьбофрезерування зовнішньої спеціальної різьби на верстаті CNC 3018 Pro показало його ефективність.

5. Розроблено FEA-модуль для симуляції напружено-деформованого стану різьбових з’єднань. Модуль може бути використаний для автоматичної оптимізації параметрів з’єднання. Модуль показав свою роботоздатність для аналізу напружено-деформованого стану різьбового з’єднання пластикових труб.

6. Запропоновано принципи реалізації та розроблено приклад мультиагентної PLM-системи для спеціальних різьбових з’єднань пластмасових труб. PLM-система працює в середовищі Jupyter Notebook за допомогою пакету Ray, інтегрує базу знань, машину прямого логічного виведення, FEA- та CAM-агенти. На прикладі показано, що використання системи може значно зменшити витрати часу на проектування конструкції та технологічного процесу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Saaksvuori A., Immonen A. Product lifecycle management. Third Edition. Berlin, Heidelberg : Springer-Verlag, 2008. 268 p.
2. Информационная поддержка жизненного цикла изделий машиностроения : принципы, системы и технологии CALS/ИПИ : учеб. пособие для студ. высш, учеб. заведений / А. Н. Ковшов, Ю. Ф. Назаров, И. М. Ибрагимов, А. Д. Никифоров. Москва : Издательский центр «Академия», 2007. 304 с.
3. Dhondt, G. The Finite Element Method for Three-Dimensional Thermomechanical Applications, Wiley, 2004.
4. Michael M. Tiller Modelica by example. URL: https://mbe.modelica.university/
5. SOLIDWORKS. URL: https://www.solidworks.com/ru
6. FreeCAD: Your own 3D parametric modeler. URL: https://www.freecadweb.org/
7. Берталанфи Л. фон. История и статус общей теории систем. В кн.: Системные исследования. Методологические проблемы. Ежегодник. Москва : «Наука», 1973, С. 20-37.
8. Системный анализ и принятие решений: Словарь-справочник: Учеб. пособие для вузов / Под. Ред. В. Н. Волковой, В. Н. Козлова. Москва : «Высшая школа», 2004. 616 с.
9. Копей В. Б. Абстрактна модель інформаційної системи підтримки життєвого циклу виробу // Прикарпатський вісник НТШ. Число. 2017. №2(38). С. 71-96.
10. Smart Python Agent Development Environment. URL: https://github.com/javipalanca/spade (дата звернення 16.09.2021).
11. Журавлев А.М. Обзор библиотек для построения мультиагентных систем в среде Python. DOI: 10.18698/2541-8009-2018-10-385
12. Scaling Python made simple, for any workload. URL: https://www.ray.io/

13.Карл Хьюитт, Питер Бишоп, Ричард Штайгер: Универсальный модульный формализм акторов для искусственного интеллекта. IJCAI, 1973.
14.Копей В. Б. Мова програмування Python для інженерів і науковців: Навчальний посібник. Івано-Франківськ : ІФНТУНГ, 2019. 275 с.
15.ThreadsPLM-MAS- URL: https://github.com/vkopey/ThreadsPLM-MAS-
16.https://shopstanki.ru/blog/podrobnoe-opisanie-g-i-m-kodov-dlya-programmirovaniya-chpu-cnc-stankov/
17.https://www.intuwiz.ru/articles/modal.html
18.Сосонкин В. Л., Мартинов Г. М. Программирование систем числового программного управления. Изд. Логос, Университетская книга, 2008, 344 с.
19.https://wiki.nikiforov.ru/index.php/%D0%9A%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D1%8F:G_%D0%BA%D0%BE%D0%B4%D1%8B
20.Вереина, Л. И. Металлообрабатывающие станки : учебник. Москва : ИНФРА-М, 2016. 440 с. DOI: www.dx.doi.org/10.12737/14542. ISBN 978-5-16-010887-2.
21.Руководство DORMER 2008. Обработка металлов резанием на металлорежущих станках.
22.https://cncmagazine.ru/polezno-znat/narezanie-rezby-korpusnymi-rezbovymi-frezami-cncm.-rekomendacii-raschety-primery-programmy-chpu/
23.Каждой резьбе по резьбофрезе! URL: https://mysku.ru/blog/diy/73346.html
24.CNC 3018 Pro. URL: https://aliexpress.ru/item/1005002347086216.html?spm=a2g39.orderlist.0.0.79364aa6MEiNCL&_ga=2.157818554.1090655133.1639072221-114830656.1637341130
25.Grbl. URL: https://github.com/gnea/grbl/wiki

26. Easy Driver Stepper Motor Driver. URL: http://www.schmalzhaus.com/EasyDriver/
27. Stepper Driver Arduino Shield. URL: http://www.buildlog.net/blog/2011/08/stepper-driver-arduino-shield/
28. Grbl's Pins. URL: https://github.com/gnea/grbl/wiki/Connecting-Grbl#grbls-pins
29. Candle. URL: https://github.com/Denvi/Candle/
30. grbl. URL: https://github.com/gnea/grbl/wiki
31. Python Streaming Scripts. URL: https://github.com/gnea/grbl/wiki/Using-Grbl#python-streaming-scripts-officially-supported-by-grbl-cross-platform
32. Кластер (группа компьютеров). URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80_(%D0%B3%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BE%D0%B2)
33. Модель акторов. URL: https://ru.wikipedia.org/wiki/Модель_акторов
34. Федотов И. Модели параллельного программирования. Москва : Солон-Пресс, 2012. С. 384. ISBN 978-5-91359-102-9.
35. Среда исполнения SmallTalk как современный пример реализации (сайт проекта Pharo). URL: https://pharo.org/
36. Карл Хьюитт, Питер Бишоп, Ричард Штайгер: Универсальный модульный формализм акторов для искусственного интеллекта. IJCAI, 1973 (англ.).
37. Уильям Клингер, Основы семантики акторов. MIT, докторская диссертация по математике, июнь 1981 (англ.).
38. John C. Mitchell. Concepts in programming languages. Cambridge University Press, 2003. 529 p. ISBN 978-0-521-78098-8.
39. pycalculix. URL: https://github.com/spacether/pycalculix
40. C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering 79(11), pp. 1309-1331, 2009.

41.https://mysku.ru/blog/diy/73346.html

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

43.Kopei V. B. et al. IOP Conf. Ser.: Mater. Sci. Eng. 477. 2019. 012032.

44.Программирование ЧПУ. Коррекция на радиус. URL: http://postprocessor.su/radius_compensation.html

45.Matplotlib: Visualization with Python. URL: https://matplotlib.org/

46.Jupyter. URL: https://jupyter.org/

47.Markdown Guide. URL: https://www.markdownguide.org/basic-syntax

ДОДАТКИ

Додаток А - Код FEA-модуля

```
# -*- coding: utf-8 -*-
import sys

import pycalculix as pyc
# шлях до gmsh, ccx, cgx
pyc.environment.GMSH=r"d:\Portable\gmsh-4.8.4-
Windows64\gmsh.exe"
pyc.environment.CCX=r"d:\Portable\cae_20200725_windows\
bin\ccx.exe"
pyc.environment.CGX=r"d:\Portable\cae_20200725_windows\
bin\cgx.exe"
model_name = 'thread'
model = pyc.FeaModel(model_name) # модель
model.set_units('mm') # одиниці вимірювання

show_gui = True # показувати графіки

# геометричні параметри з'єднання
l = 30
t1 = 3.5
t2 = 3.5
r0 = 6.5
hh=0.1

part = pyc.Part(model) # деталь

def run():
    # рисувати контури з'єднання
    part.goto(r0, 0) # перейти в точку
```

```
    l1=part.draw_line_ax(l) # рисувати лінію вздовж
    l2=part.draw_line_rad(t1) # рисувати лінію поперек
    l3=part.draw_line_rad(t2)
    l4=part.draw_line_ax(-l)
    l5=part.draw_line_rad(-t2)
    l6=part.draw_line_rad(-t1)

    # зазори
    for i in range(15): # 15 отворів
        d=i*2.0 # осьова координата
        w = 1.2 # ширина
        h = hh # висота
        part.goto(10, 0.3+d, holemode=True) # перейти в
точку, режим отворів
        part.draw_line_rad(h) # рисувати лінію поперек
        part.draw_line_ax(w) # рисувати лінію вздовж
        part.draw_line_rad(-h)
        part.draw_line_ax(-w)

    for i in range(14): # 14 отворів
        d=i*2.0 # осьова координата
        w = 0.5 # ширина
        h = hh # висота
        part.goto(9.5, 1.5+d, holemode=True) # перейти
в точку, режим отворів
        part.draw_line_rad(h) # рисувати лінію поперек
        part.draw_line_ax(w) # рисувати лінію вздовж
        part.draw_line_rad(-h)
        part.draw_line_ax(-w)

    # показати геометрію
```

```
    model.plot_geometry(model_name + '_pre',
display=show_gui)
    model.view.print_summary() # текстова інформація
про геометрію

    model.set_etype('axisym', part) # тип елементів -
осесиметричні
    model.set_eshape('tri', 2) # форма елементів -
трикутні
    model.mesh(0.1, 'gmsh') # створити сітку за
допомогою gmsh
    model.plot_elements(model_name+'_elements',
display=show_gui) # показати сітку
    model.view.print_summary()
    model.print_summary()

    # установити матеріал деталей
    mat = pyc.Material('steel')
    mat.set_mech_props(7800, 210*(10**9), 0.3) #
властивості матеріалу
    model.set_matl(mat, part)

    # установити навантаження і граничні умови
    #model.set_constr('fix',part.bottom,'x')
    # низ нерухомий
    model.set_constr('fix',[l5[0]],'x')
    model.set_constr('fix',[l5[0]],'y')
    model.set_load("press", [l2[0]], -10e6) #
навантаження у верхній частині

    # створити завдання
    prob = pyc.Problem(model, 'struct')
```

```
    try:
        prob.solve() # розв'язати
    except:
        pass

    prob.rfile.set_time(1.0) # результати в заданий
момент часу

    # рисунки результатів
    fields = 'Sx,Sy,S1,S2,S3,Seqv,ux,uy,utot' #
величини для результатів
    fields = fields.split(',')
    for field in fields: # для кожної величини
        fname = model_name+'_'+field
        prob.rfile.nplot(field, fname, display=False) #
рисунок

    return prob.rfile.get_nmax('Seqv') # максимальне
значення екв. напруження

if __name__=="__main__":
    show_gui = False
    hh=0.1
    print(run())
```

Додаток Б - Код CAE-модуля

```
# -*- coding: utf-8 -*-
"""CAM-модуль для фрезерування різьби з довільним
профілем на фрезерному верстаті з ЧПК. Python 2.7-3"""

import numpy as np
```

```
pi=np.pi
import matplotlib.pyplot as plt

def helixPoints(r, h, p, fi, n, z0=0):
    """Повертає список точок гвинтової лінії. r - мінімальний радіус, h - висота, p - крок, fi - кут нахилу, n - кількість точок на одному витку, z0 - осьове зміщення"""
    a=np.tan(fi)*p/(2*pi)
    b=p/(2*pi)
    k=h/p # кількість витків
    N=int(n*k) # загальна кількість точок
    t=np.linspace(0,h/b,N) # параметр
    x=(r+a*t)*np.cos(-t)
    y=(r+a*t)*np.sin(-t)
    z=b*t+z0 # вісь гвинтової лінії
    return list(zip(x,y,z)) # тут list() потрібен тільки для python 3

def TreadMulti(R, Z, h=10, p=2, fi=0, n=64):
    """Повертає список точок для фрезерування різьби в кілька проходів. R, Z - списки з радіусами (x-координата) і z-коодинатою перших точок гвинтових ліній. Іншими словами це x,z-координати точок профілю. Для одного проходу R=[r] Z=[0], де r - мінімальний радіус різьби. h - висота, p - крок, fi - кут нахилу, n - кількість точок на одному витку. Довжина різьби h повинна бути кратна кроку p"""

    fp=max(R)+10,0,0 # перша точка віддалена по x на 10 мм
    P=[] # список усіх точок
```

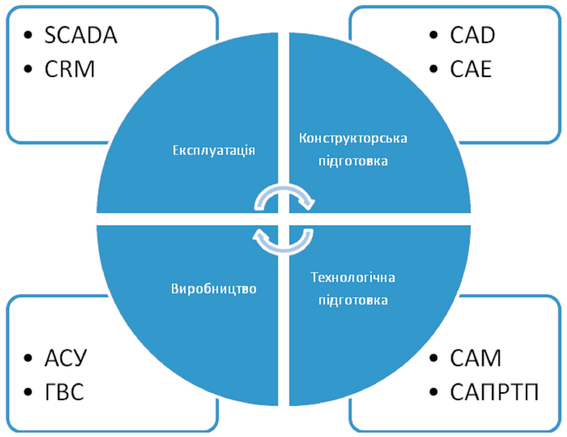
```
    for r, z in zip(R, Z): # для кожної точки профілю
        P.append(fp) # додати першу
        H=helixPoints(r=r, h=h, p=p, fi=fi, n=n, z0=z)
# точки 1 гвинтової лінії
        P+=H # додати їх
        a=H[-1] # остання точка на гвинтовій лінії
        P.append((a[0]+10, a[1], a[2])) # додати точку
відведення
    return P

def Gcode(P, fileName=None):
    """Повертає G-код за точками P"""
    S=["N10 G1 G54 G17 G21 G90 G94 M05 T0 F100 S1000",
"N20 M03"] # поч. код
    i=30 # номер рядка
    for p in P: # для кожної точки
        x, y, z=p[0], p[1], p[2]
        x, y, z=(round(k, 3) for k in (x, y, z)) #
заокруглити до 3 знаків
        s="N%d G1 X%5.3f Y%5.3f Z%5.3f"%(i, x, y, z) #
кадр програми
        S.append(s)
        i+=10 # наступний кадр
    S.append("N%d M05"%i) # виключити шпиндель
    code="\n".join(S)
    if fileName:
        f=open(fileName, 'w') # зберегти файл
        f.write(code)
        f.close()
    return code

def plotCNC(P):
```

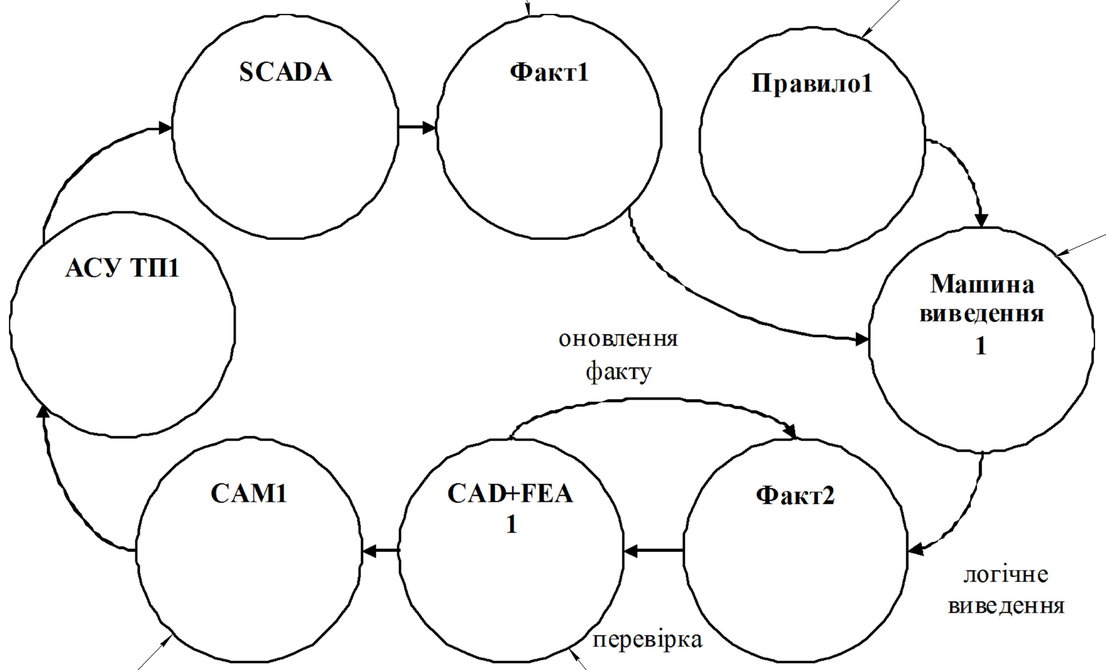
```
    """візуалізація точок за допомогою Matplotlib"""
    # координати
    X=[p[0] for p in P]
    Y=[p[1] for p in P]
    Z=[p[2] for p in P]
    # тривимірні графіки
    from mpl_toolkits.mplot3d import Axes3D
    fig = plt.figure() # рисунок
    ax = Axes3D(fig) # система координат
    ax.plot3D(X,Y,Z, 'k.-') # лінії
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    plt.show()

## приклад генерації коду
if __name__=="__main__":
    rf=10 # радіус фрези (0, якщо система ЧПК має
команди корекції)
    # товщина фрези 0.5 мм
    # перші два проходи - примірочний r=10 і остаточний
для 1-ї точки профілю
    P=TreadMulti(R=[10+rf, 9.5+rf], Z=[0, 0],  h=14,
p=2, fi=0, n=64)
    # наступні два проходи - примірочний r=10 і
остаточний для 2-ї точки профілю
    #P=TreadMulti(R=[10+rf, 9.6+rf], Z=[0.3, 0.3],
h=14, p=2, fi=0, n=64)
    # додатково ще два чистових проходи з малою подачею
50 мм/хв
    code=Gcode(P, "cnc.txt") # код, зберегти у файл
    plotCNC(P)
```




```
@ray.remote
class Fact(object):
    def __init__(self,s,p,o):
        self.s, self.p, self.o = s, p, o
    def rule(self):
        return self.s, self.p, self.o
```

```
@ray.remote
class Rule1(object):
    def __init__(self,p):
        self.p = p
        self.ready=set()
    def rule(self, ts=None):
        """Симетрична властивість 2: (1, 2, 3)->(3, 2, 1)"""
        tn=set()
        for s,p,o in ts:
            if self.p!=p: continue
            if (s,p,o) in self.ready: continue
            self.ready.add((s,p,o))
            self.ready.add((o,p,s))
            tn.add((o,p,s))
        return tn
```




```
@ray.remote
class Reasoner(object):
    def __init__(self):
        self.ready=set()
    def triplets(self, fs):
        refs=[f.rule.remote() for f in fs]
        ts=ray.get(refs)
        return ts
    def rule(self, rs, ts):
        refs=[r.rule.remote(ts) for r in rs] # запустити паралельно
        T=ray.get(refs) # список множин триплетів
        r=set()
        for t in T:
            r.update(t)
        return r
```

```
@ray.remote
class CAM(object):
    def __init__(self):
        self.ready=dict()
    def rule(self, x=0.1):
        #if x in self.ready: return None
        import CNC_thread_mill
        rf=CNC_thread_mill.rf=10 # радіус фрези (0, якщо система ЧПК має команди корекції)
        # товщина фрези 0.5 мм
        # перші два проходи – примірочний r=10 і остаточний для 1-ї точки профілю
        P=CNC_thread_mill.TreadMulti(R=[10+rf, 9.5+x+rf], Z=[0, 0],  h=14, p=2, fi=0, n=64)
        code=CNC_thread_mill.Gcode(P, "cnc.txt") # код, зберегти у файл
        self.ready[x]=P
        return P
```

```
@ray.remote
class FEA(object):
    def __init__(self):
        self.ready=dict()
    def rule(self, x=0.1):
        #if x in self.ready: return None
        import mypycalculix
        mypycalculix.show_gui = False
        mypycalculix.hh=x
        s=mypycalculix.run()
        self.ready[x]=s
        return s
```

					МР-125.00.001	
Изм.	Лист	№ докум.	Подп.	Дата	Мультиагентна PLM-система	Лит. / Масса / Масштаб: 1:1
Разраб.		Вірстюк В.М.				
Пров.		Копей В.Б.				Лист / Листов: 1
Т.контр.		Копей В.Б.				
Н.контр.		Копей В.Б.			ПМКм-20-1 ІФНТУНГ	
Утв.		Панчук В.Г.				

Копировал

Формат А1

Перв. примен.

Справ. №

Подп. и дата

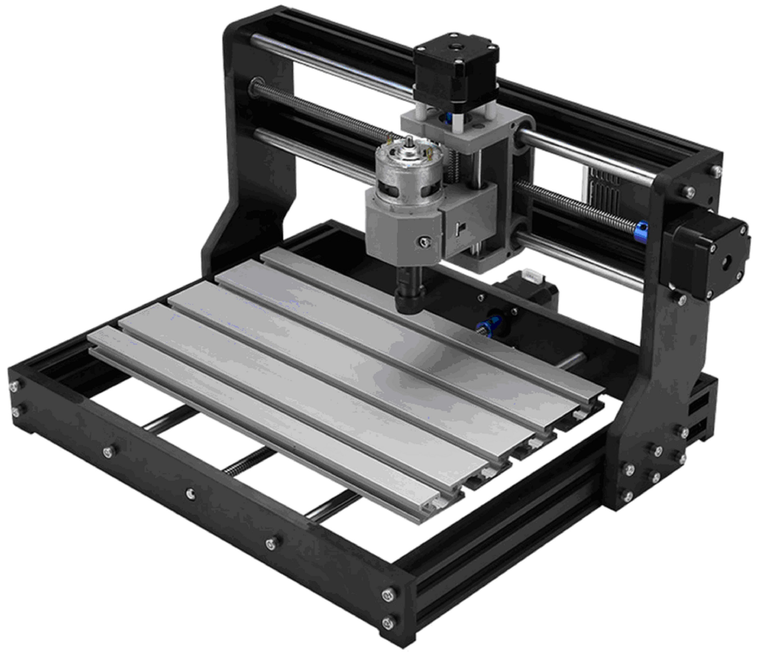
Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

Верстат CNC 3018 Pro



Заготовка, закріплена на верстаті CNC 3018 Pro

Плата керування, що підтримує ПЗ GRBL

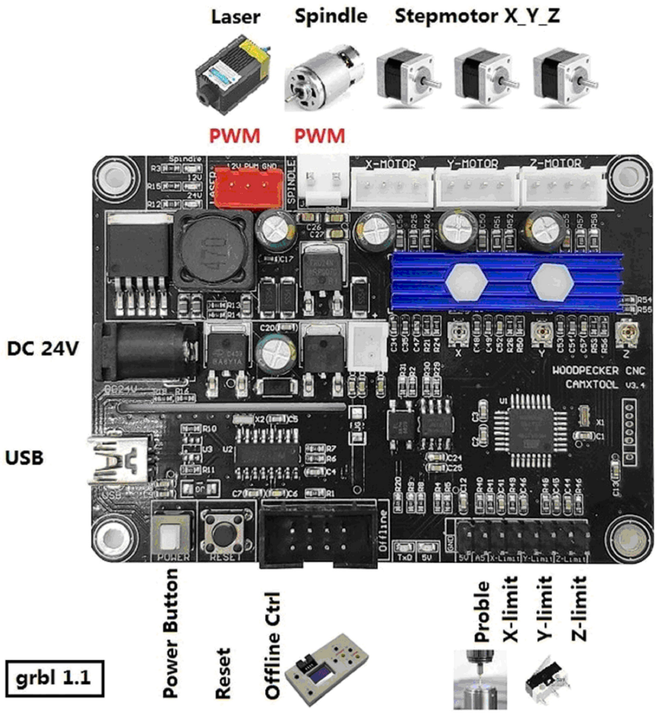
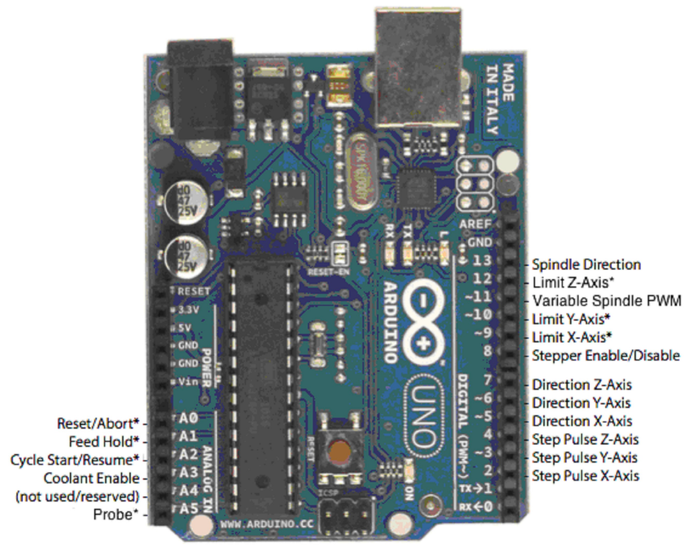


Схема під'єднання пінів Arduino Uno до драйвера



* - Indicates input pins. Held high with internal pull-up resistors.

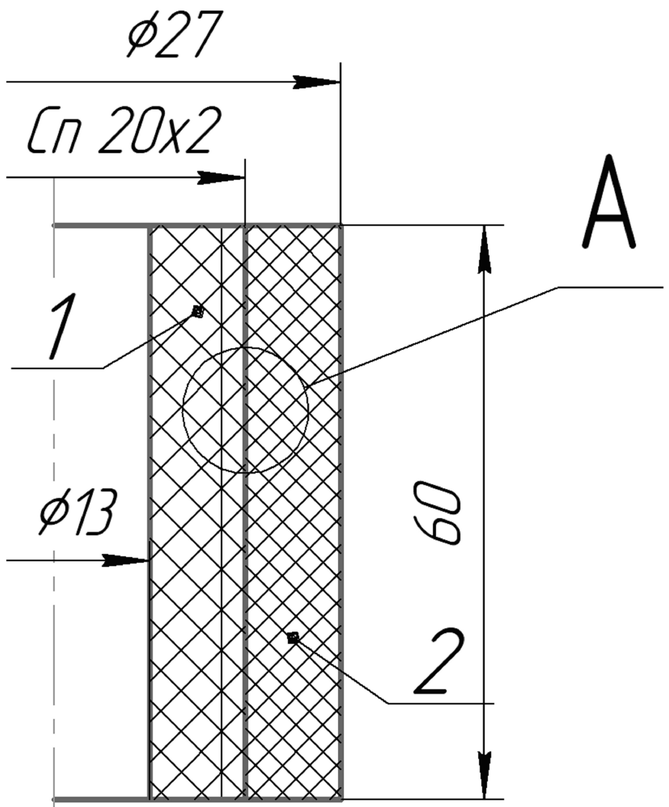
					МР-125.00.002	
Изм.	Лист	№ докум.	Подп.	Дата	Верстат CNC 3018 Pro	Лит. / Масса / Масштаб: 1:1
Разраб.		Верстюк В.М.				Лист / Листов: 1 / 1
Пров.		Копей В.Б.				ПМКм-20-1 ІФНТУНГ
Т.контр.		Копей В.Б.				
Н.контр.		Копей В.Б.				
Утв.		Панчук В.Г.				

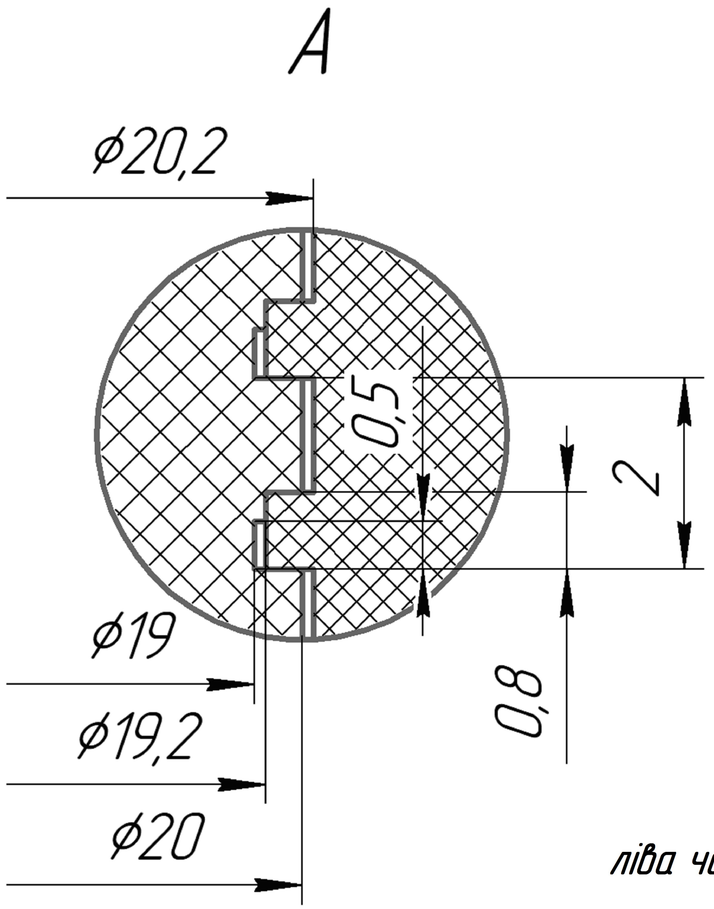
Копировал — Формат А1

Перв. примен. | Справ. № | Подп. и дата | Инв. № дубл. | Взам. инв. № | Подп. и дата | Инв. № подл.

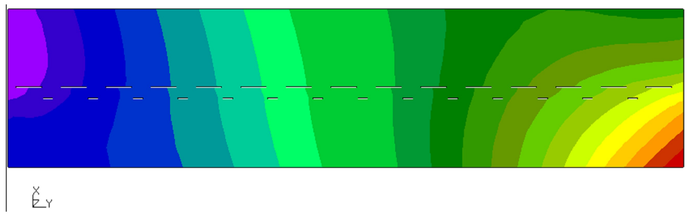
МР-125.00.003

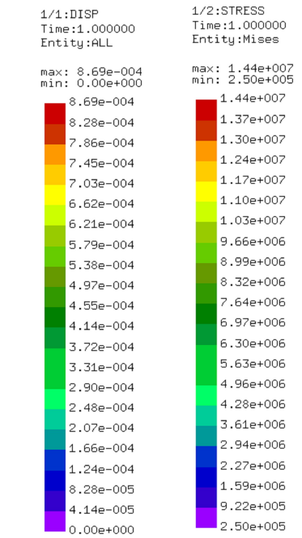
Різьбове з'єднання зі спеціальною різьбою



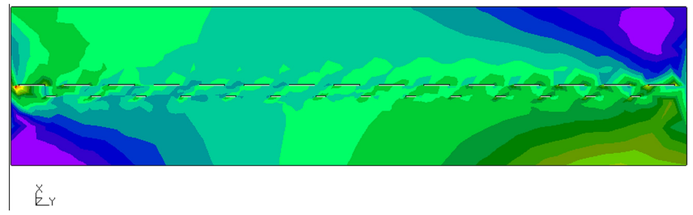


Деформації (мм) в різьбовому з'єднанні



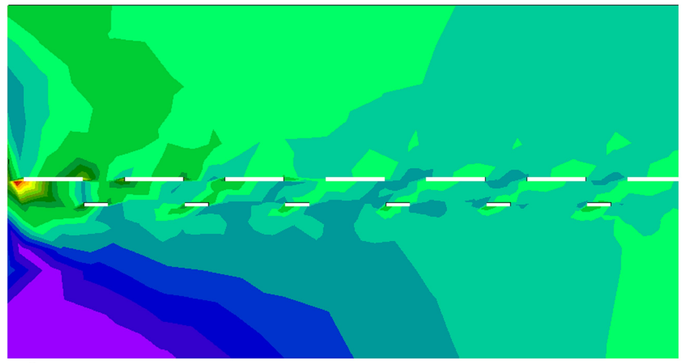


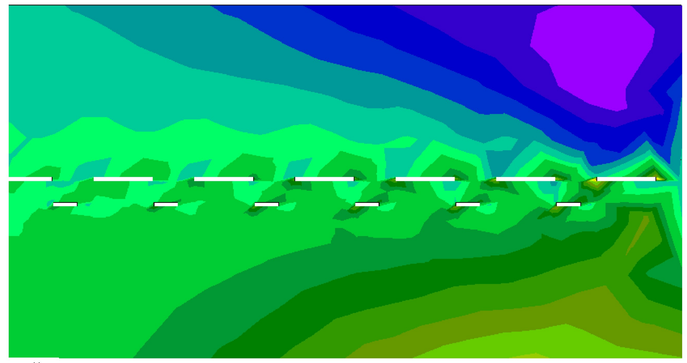
Еквівалентні напруження за Мізесом (Па) в різьбовому з'єднанні



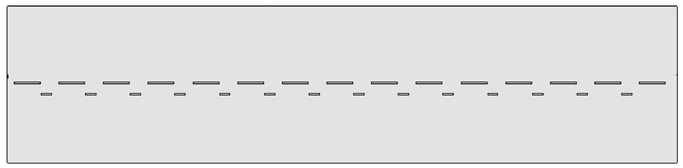
ліва частина

права частина

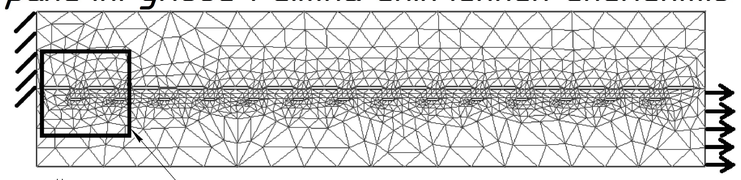


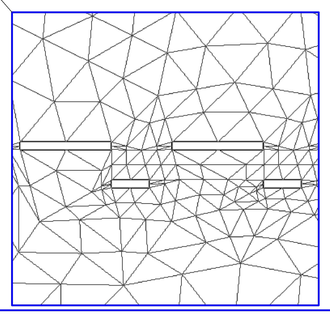


Геометрична модель



Граничні умови і сітка скінченних елементів





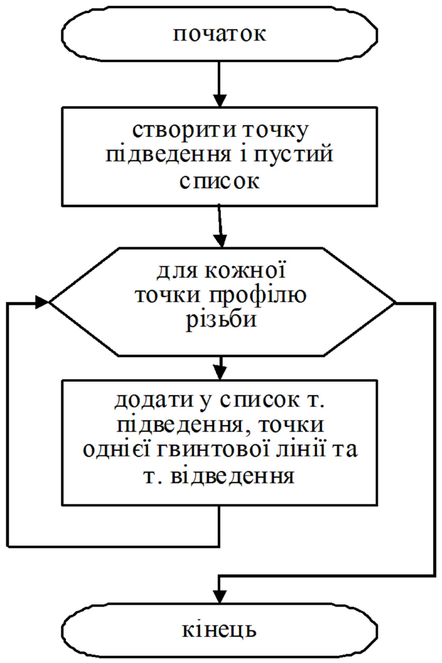
					МР-125.00.003		
Изм.	Лист	№ докум.	Подп.	Дата	Скінченно-елементна модель Р3	Лит.	Масса / Масштаб
Разраб.		Вірстюк В.М.					1:1
Пров.		Копей В.Б.				Лист	Листов 1
Т.контр.		Копей В.Б.					
Н.контр.		Копей В.Б.			ПМКм-20-1 ІФНТУНГ		
Утв.		Панчук В.Г.					

Копировал

Формат А1

Перв. примен. | Справ. № | Подп. и дата | Инв. № дубл. | Взам. инв. № | Подп. и дата | Инв. № подл.

Блок-схема алгоритму функції TreadMulti




```
def TreadMulti(R, Z, h=10, p=2, fi=0, n=64):
    fp=max(R)+10,0,0 # перша точка віддалена по x на 10 мм
    P=[] # список усіх точок
    for r, z in zip(R, Z): # для кожної точки профілю
        P.append(fp) # додати першу
        # точки 1 гвинтової лінії
        H=helixPoints(r=r, h=h, p=p, fi=fi, n=n, z0=z)
        P+=H # додати їх
        a=H[-1] # остання точка на гвинтовій лінії
        # додати точку відведення
        P.append((a[0]+10, a[1], a[2]))
    return P
```

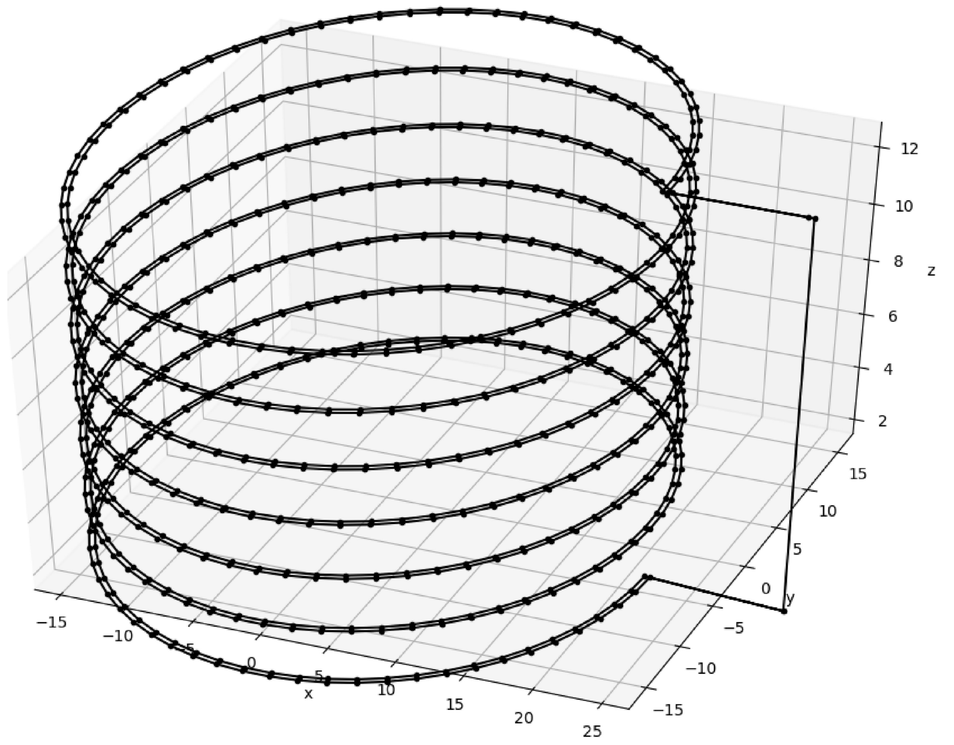
Рівняння гвинтової лінії

$$x = (r + at)\cos(-t),$$
$$y = (r + at)\sin(-t),$$
$$z = bt + z0.$$

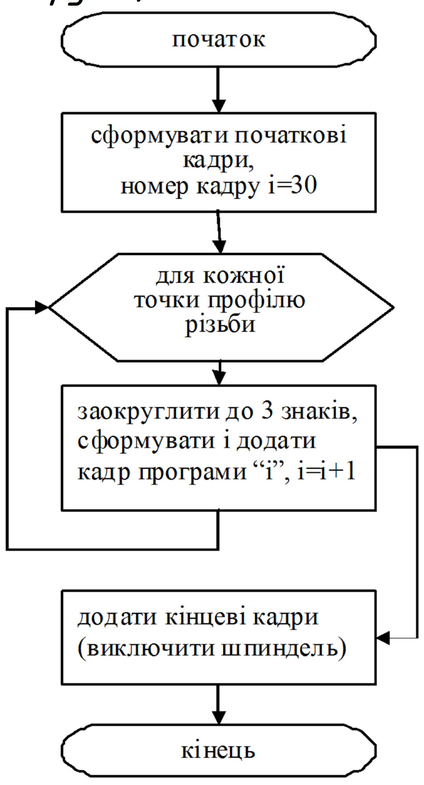
Функція, що повертає список точок гвинтової лінії

```
def helixPoints(r, h, p, fi, n, z0=0):
    a = np.tan(fi)*p/(2*pi)
    b = p/(2*pi)
    k = h/p # кількість витків
    N = int(n*k) # загальна кількість точок
    t = np.linspace(0, h/b, N) # параметр
    x = (r + a*t)*np.cos(-t)
    y = (r + a*t)*np.sin(-t)
    z = b*t + z0 # вісь гвинтової лінії
    return zip(x,y,z)
```

Візуалізація траєкторії переміщення т.0 інструмента за допомогою Matplotlib

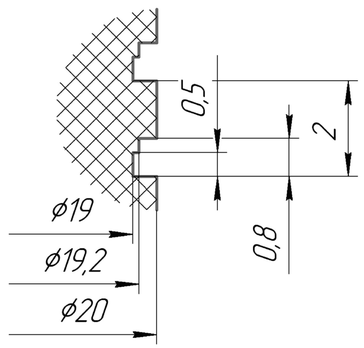


Блок-схема алгоритму функції Gcode




```
def Gcode(P):
    S=["N10 G1 G54 G17 G21 G90 G94 M05 T0 F100 S1000", "N20 M03"] # поч. код
    i=30 # номер рядка
    for p in P: # для кожної точки
        x, y, z=p[0], p[1], p[2]
        x, y, z=(round(k, 3) for k in (x, y, z)) # заокруглити до 3 знаків
        s="N%d G1 X%5.3f Y%5.3f Z%5.3f"%(i, x, y, z) # кадр програми
        S.append(s)
        i+=10 # наступний кадр
    S.append("N%d M05"%i) # виключити шпиндель
    return "\n".join(S)
```

Профіль спеціальної різьби



Фрагмент згенерованого G-коду

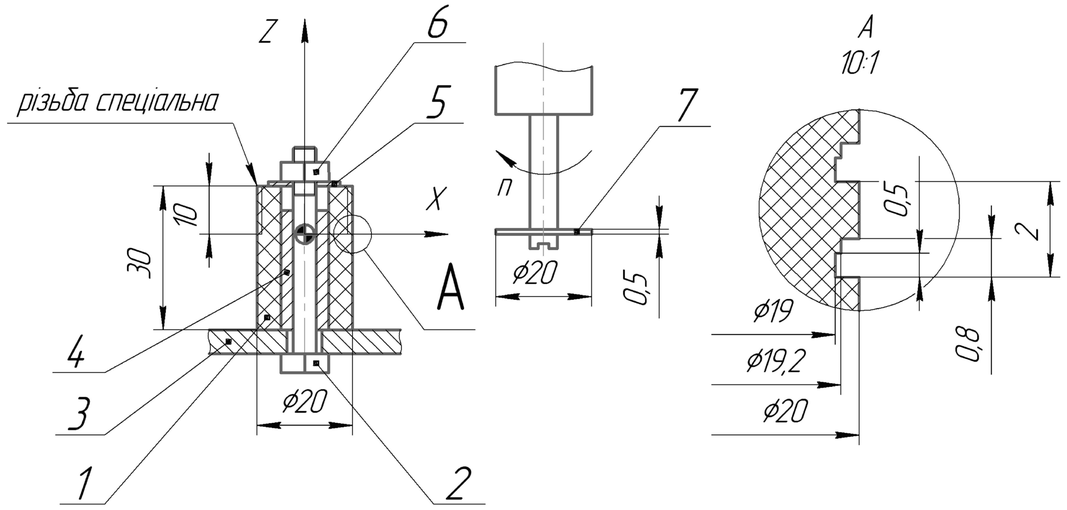
```
N10 G1 G54 G17 G21 G90 G94 M05 T0 F100 S1000
N20 M03
N30 G1 X30.000 Y0.000 Z0.000
N40 G1 X20.000 Y-0.000 Z0.000
N50 G1 X19.903 Y-1.965 Z0.031
N60 G1 X19.614 Y-3.910 Z0.063
N70 G1 X19.135 Y-5.818 Z0.094
...
N9030 M05
```

					МР-125.00.004		
Изм.	Лист	№ докум.	Подп.	Дата	САМ-модуль	Лит.	Масса / Масштаб
Разраб.		Вірстюк В.М.					1:1
Пров.		Копей В.Б.				Лист	Листов 1
Т.контр.		Копей В.Б.				ПМКм-20-1 ІФНТУНГ	
Н.контр.		Копей В.Б.					
Утв.		Панчук В.Г.					

Копировал — Формат А1

Перв. примен. | Справ. № | Подп. и дата | Инв. № дубл. | Взам. инв. № | Подп. и дата | Инв. № подл.

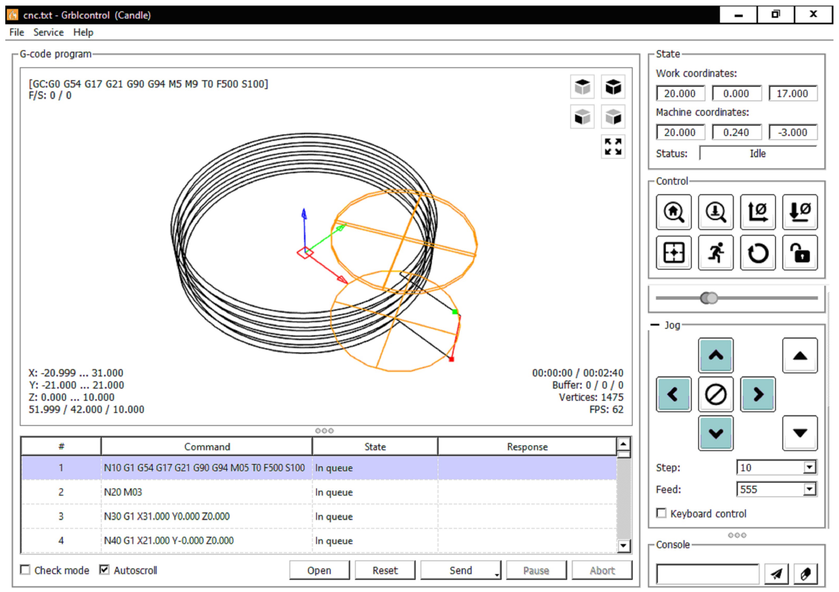
Технологічний ескіз фрезерування спеціальної різьби



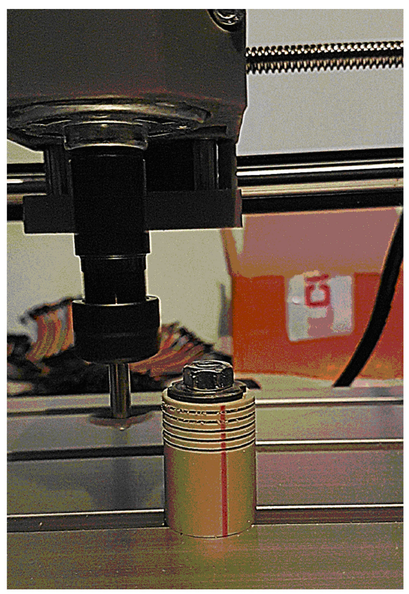
Виставлення нуля інструмента в точку X=0, Y=0

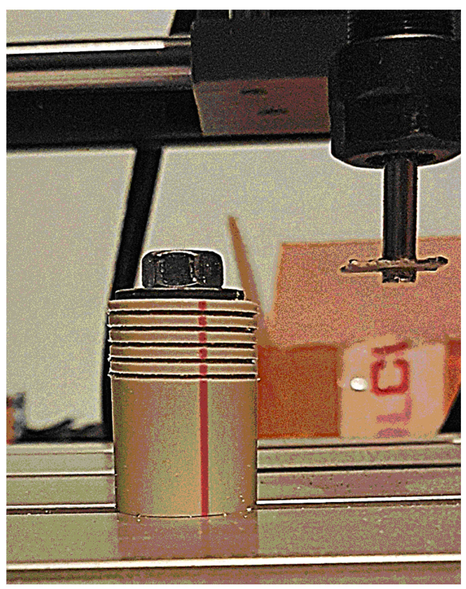
Відведення фрези вправо

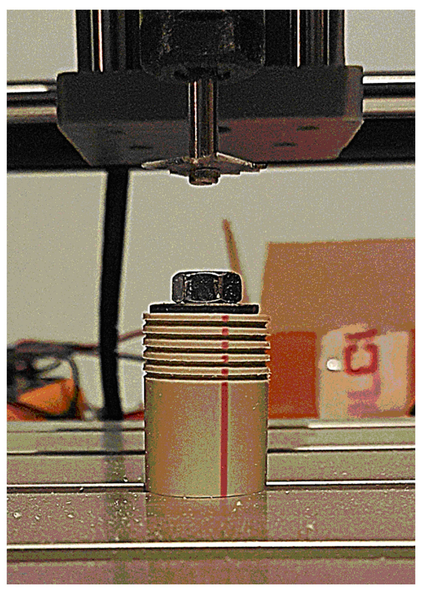
Вікно програми Candle – відведення фрези вправо



Етапи фрезерування різьби







					МР-125.00.005		
Изм.	Лист	№ докум.	Подп.	Дата	Фрезерування спеціальної різьби	Лит.	Масса / Масштаб
Разраб.		Вірстюк В.М.					1:1
Пров.		Копей В.Б.				Лист	Листов 1
Т.контр.		Копей В.Б.				ПМКм-20-1 ІФНТУНГ	
Н.контр.		Копей В.Б.					
Утв.		Панчук В.Г.					

Копировал

Формат А1

Перв. примен. | Справ. № | Подп. и дата | Инв. № дубл. | Взам. инв. № | Подп. и дата | Инв. № подл.

jupyter ThreadPLM_ray

Logout

File Edit View Insert Cell Kernel Help Доверенный Python 3 (ipykernel)

Запуск Код

Приклад використання мультиагентної PLM-системи різьбових з'єднань

Розподілена МАС створена на основі концепції акторів і Python Ray. Кожен агент (актор) може виконуватись паралельно разом з іншими агентами. Можна організувати автоматичну роботу системи за допомогою простого алгоритму: виконувати функцію rule кожного агента, поки ці функції повертають якісь значення, що не рівні None. В даному прикладі це реалізовано для логічного виведення нових фактів з бази існуючих фактів і правил логічного виведення.

Передусім потрібно ініціалізувати систему ray. В даному прикладі використовуються виключно ресурси одного комп'ютера, але можна використати і кластер.

Ввод [1]:
```
import ray
ray.init()
```

```
D:\Portable\Portable Python-3.8.9 x64\App\Python\lib\site-packages\redis\connection.py:72: UserWarning: redis-py works best with hiredis. Please consider installing
  warnings.warn(msg)
```

Out[1]:
```
{'node_ip_address': '127.0.0.1',
 'raylet_ip_address': '127.0.0.1',
 'redis_address': '127.0.0.1:6379',
 'object_store_address': 'tcp://127.0.0.1:64265',
 'raylet_socket_name': 'tcp://127.0.0.1:53842',
 'webui_url': None,
 'session_dir': 'D:\\Temp\\ray\\session_2021-12-18_10-57-53_554289_1072',
 'metrics_export_port': 64337,
 'node_id': '8e1d59da19d0edad89b06146e60c9d199cf217529e16e7
```

Клас акторів Fact

Клас описує поняття факту у вигляді триплету (суб'єкт, предикат, об'єкт). Функція rule повертає кортеж триплету.

Ввод [2]:
```
@ray.remote
class Fact(object):
    def __init__(self,s,p,o):
        self.s, self.p, self.o = s, p, o
    def rule(self):
        return self.s, self.p, self.o
```

Клас акторів Rule1

Клас описує поняття правила логічного виведення для симетричної властивості p: (1, p, 3)->(3, p, 1). Для пришвидшення алгоритму використовується техніка мемоїзації - в атрибуті ready зберігаються оброблені триплети. Однократно обробляється множина триплетів ts. Повертається множина нових триплетів.

Клас акторів Reasoner

Клас описує поняття машини логічного виведення. Функція rule запускає ОДНОКРАТНО паралельно правила агентів rs, яким передаються триплети з фактами ts. Функція повертає множину триплетів з новими фактами. Аналогічно є можливість використати техніку мемоїзації, але це в класі не реалізовано.

Клас акторів FEA

Клас описує поняття FEA системи для скінченно-елементної симуляції моделі різьбового з'єднання. Для моделювання використовується pycalculix. Функція rule отримує параметр з'єднання x і повертає максимальне напруження. Для мемоїзації використовується словник ready.

Ввод [6]:
```
@ray.remote
class FEA(object):
    def __init__(self):
        self.ready=dict()
    def rule(self, x=0.1):
        #if x in self.ready: return None
        import mypycalculix
        mypycalculix.show_gui = False
        mypycalculix.hh=x
        s=mypycalculix.run()
        self.ready[x]=s
        return s
```

Клас акторів CAM

Клас описує поняття CAM системи для генерації G-коду для фрезерування різьби ніпеля на 3-осьовому фрезерному верстаті з ЧПК. Функція rule отримує параметр з'єднання x і повертає відповідний список опорних точок G-коду. Для мемоїзації використовується атрибут ready.

Ввод [7]:
```
@ray.remote
class CAM(object):
    def __init__(self):
        self.ready=dict()
    def rule(self, x=0.1):
        #if x in self.ready: return None
        import CNC_thread_mill
        rf=CNC_thread_mill.rf=10 # радіус фрези (0, якщо сис
        # товщина фрези 0.5 мм
        # перші два проходи - примірочний r=10 і остаточний
        P=CNC_thread_mill.TreadMulti(R=[10+rf, 9.5+x+rf], Z=
        code=CNC_thread_mill.Gcode(P, "cnc.txt") # код, збер
        self.ready[x]=P
        return P
```

Створення агентів системи (акторів)

Кожен агент може працювати паралельно з іншими. Проте це залежить від кількості процесорів на кластері. В даному випадку введені факти, які містять відповідність значень вхідних і вихідних параметрів моделі різьбового з'єднання: геометричного параметра h (мм), максимального еквівалентного напруження в різьбі Seqv (МПа), максимального сумарного переміщення utot (мм). Ці факти можуть бути отримані в результаті моделювання, експериментів, промислових даних на попередніх етапах ЖЦ.

Ввод [8]:
```
f1=Fact.remote("Seqv=14", "відповідає", "h=0.1")
f2=Fact.remote("h=0.1", "відповідає", "utot=869e-6")
f3=Fact.remote("h=0.2", "відповідає", "Seqv=16")
r1=Rule1.remote("відповідає")
r2=Rule2.remote("відповідає")
r=Reasoner.remote()
fea1=FEA.remote()
cam1=CAM.remote()
```

Логічне виведення фактів

Функція rule агентів, які містять правила, викликається до тих пір, поки є результати (нові факти). Виводяться триплети з новими фактами. Ці триплети можна використати для створення нових агентів-фактів.

Ввод [9]:
```
fs=[f1, f2, f3]
rs=[r1, r2]
T=set()
re=1
while re:
    re=0
    ts=ray.get(r.triplets.remote(fs))
    tn=ray.get(r.rule.remote(rs, ts))
    if tn:
        T.update(tn)
        re+=1
print(T)
```

```
{('Seqv=16', 'відповідає', 'h=0.2'), ('utot=869e-6', 'відповідає', 'h=0.1'), ('Seqv=14', 'відповідає', 'utot=869e-6'), ('h=0.1', 'відповідає', 'Seqv=14')}
```

FE-симуляція

За допомогою агента fea1 виконується симуляція різьбового з'єднання з параметром h=0,1 мм (зазор в різьбі). Відразу очікуємо результату. Але є можливість запустити симуляцію асинхронно ro=fea1.rule.remote(0.1) і отримати результат в комірках нижче y1=ray.get(ro).

Ввод [10]:
```
%%capture
y1=ray.get(fea1.rule.remote(0.1))
```

Виведення результатів симуляції. Максимальне еквівалентне напруження за Мізесом (Па):

Ввод [11]:
```
y1
```

Out[11]: 14367274.322702967

Агент fea1 (за допомогою pycalculix) зберігає результати в робочому каталозі в рисунках png. Ці рисунки відображаються тут за допомогою Markdown розмітки `![name.png](name.png)`.

Генерація G-коду для фрезерування різьби

Генерація G-коду для фрезерування зовнішньої різьби з параметром h=0,1 мм відбувається за допомогою агента cam1. Відразу очікуємо результату. Але є можливість запустити симуляцію асинхронно ro=cam1.rule.remote(0.1) і отримати результат в комірках нижче y1=ray.get(ro).

Ввод [12]:
```
y2=ray.get(cam1.rule.remote(0.1))
```

```
pid=2384) File thread_utot.png was saved.
```

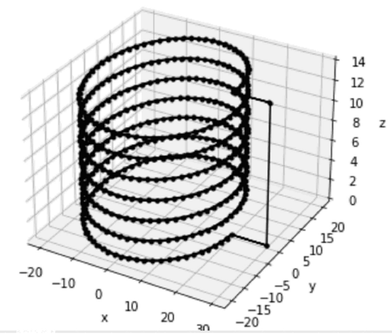
Перші опорні точки згенерованої траєкторії:

Ввод [13]:
```
y2[0:5]
```

Out[13]:
```
[(30, 0, 0),
 (20.0, -0.0, 0.0),
 (19.90326350260106, -1.9647142148487098, 0.031319910514541
39),
 (19.613989805397143, -3.9104224725439822, 0.06263982102908
278),
```

Візуалізація опорних точок траєкторії за допомогою Matplotlib:

Ввод [14]:
```
import CNC_thread_mill
CNC_thread_mill.plotCNC(y2)
```



Робота з документом може бути ітеративною, тобто на будь-якому етапі можна вернутись до потрібних комірок, змінити дані, виконати код і отримати нові результати. Якщо потрібно завершити роботу з ray, введіть:

Ввод [15]:
```
#ray.shutdown()
```

					МР-125.00.006	
Изм.	Лист	№ докум.	Подп.	Дата	Документ Jupyter Notebook	Лит. / Масса / Масштаб: 1:1
Разраб.		Вірстюк В.М.				
Пров.		Копей В.Б.				Лист / Листов: 1
Т.контр.		Копей В.Б.				
Н.контр.		Копей В.Б.			ПМКм-20-1 ІФНТУНГ	
Утв.		Панчук В.Г.				

Копировал Формат А1

Перв. примен. | Справ. № | Подп. и дата | Инв. № подл. | Взам. инв. № | Инв. № дубл. | Подп. и дата