Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматизації, комп'ютерно-інтегрованих технологій та систем
(повна назва)

Кафедра Кафедра комп'ютерно-інтегрованих технологій, автоматизації, робототехніки та безпекової інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розроблення програмного забезпечення для автоматизації діалогових інтерфейсів розрахунків наукового призначення
(тема)

Виконав:

здобувач 4 року навчання,
групи АКТСІ-22-1
Олексій ЗАХАРОВ
(власне ім'я, прізвище)

Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва спеціальності)

Тип програми освітньо-професійна
Освітня програма Системна інженерія
(повна назва освітньої програми)

Керівник проф. Юрій РОМАШОВ
(посада, власне ім'я, прізвище)

Допускається до захисту
Зав. кафедри КІТАРБІ _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім'я, прізвище)

2026 р.

Я , Захаров Олексій Дмитрович, як здобувач вищої освіти ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

"20" червня 2026 р.  Олексій ЗАХАРОВ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет АКІТС
Кафедра КІТАРБІ
Рівень вищої освіти перший (бакалаврський)
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва)
Тип програми освітньо-професійна
Освітня програма Системна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри КІТАРБІ _____
(підпис)
« ___» ___червень____2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Захарову Олексію Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення програмного забезпечення для автоматизації діалогових інтерфейсів розрахунків наукового призначення
затверджена наказом по університету від «04» червня 2026 р. №473 Ст.
2. Термін подання здобувачем роботи до екзаменаційної комісії 06 липня 2026 р.
3. Вихідні дані до роботи
3.1 Програмне забезпечення CodeBlocks;
3.2 Файл casta.f90.
4. Перелік питань, що потрібно опрацювати в роботі
4.1 Основна ідея розробки програмного забезпечення для автоматизації діалогових інтерфейсів розрахунків наукового призначення;
4.2 Програми для наукових розрахунків та їх структура;
4.3 Розробка універсального інтерфейсу за допомогою Fortran;
4.4 Охорона праці.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)
Графічний матеріал у вигляді презентації – 20 арк. ф. А4.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Приміт-ка
1	Аналіз існуючих рішень	07 червня 2026	Виконано
2	Аналіз літератури по Fortran	10 червня 2026	Виконано
3	Основна ідея розробки програмного забезпечення для автоматизації діалогових інтерфейсів розрахунків наукового призначення	15 червня 2026	Виконано
4	Програми для наукових розрахунків та їх структура	17 червня 2026	Виконано
5	Розробка універсального інтерфейсу за допомогою Fortran	20 червня 2026	Виконано
6	Подання роботи на перевірку Інтернет-сервісом StrikePlagiarism	08 липня 2026	Виконано
7	Оформлення пояснювальної записки	08 липня 2026	Виконано
8	Подання роботи на рецензію	08 липня 2026	Виконано
9	Подання роботи на підпис зав. кафедри	08 липня 2026	Виконано
10	Подання кваліфікаційної роботи в ЕК	08 липня 2026	Виконано

Дата видачі завдання 04 червня 2026 р.

Здобувач  Олексій ЗАХАРОВ
(підпис) (власне ім'я, прізвище)

Керівник роботи _____ проф. Юрій РОМАШОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 44с., 7 табл., 16 рис., 2 додатків, 15 джерел.

ІНТЕРФЕЙС, CODEBLOCKS, ПРОГРАМНИЙ ЗАСІБ, FORTRAN

Об’єктом цієї роботи є процес розробки програмного забезпечення для розв’язування обчислювальних задач наукового призначення.

Предметом є автоматизація розробки програмних компонентів текстових діалогових інтерфейсів.

Метою роботи є підвищення ефективності взаємодії користувача з обчислювальними програмами шляхом проєктування та програмної реалізації текстового інтерфейсу користувача.

В першому розділі було проведено аналіз типів інтерфейсів та мов програмування для їх розробки.

В другому розділі було розглянута загальна структура програм для наукових розрахунків та описано зовнішній вигляд власної програми.

В третьому розділі розроблювався текстовий інтерфейс за допомогою Fortran

У результаті було розроблено текстовий інтерфейс який може використовуватись з великою кількістю різних математичних моделей які були розроблені за допомогою Fortran. Також, отримані результати роботи можна віднести до Цілі сталого розвитку 9 «Промисловість, інновації та інфраструктура», а саме п. 9.4 «Сприяти прискореному розвитку високо- та середньо високотехнологічних секторів переробної промисловості, які формуються на основі використання ланцюгів «освіта – наука – виробництво» та кластерного підходу за напрямами: розвиток інноваційної екосистеми».

ABSTRACT

Explanatory note: 44p., 7 tables, 16 figures, 2 appendices, 15 sources.

INTERFACE, CODEBLOCKS, SOFTWARE, FORTRAN

The object of this work is the process of developing software for solving computational problems of scientific purpose.

The subject is automation of the development of software components of text dialog interfaces.

The purpose of the work is to increase the efficiency of user interaction with computational programs by designing and software implementation of an interactive text user interface.

In the first section, an analysis of the types of interfaces and programming languages for their development was conducted.

In the second section, the general structure of programs for scientific calculations was considered and the appearance of the own program was described.

In the third section, a text interface was developed using Fortran

As a result, a text interface was developed that can be used with a large number of different mathematical models that were developed using Fortran. Also, the results of the work can be attributed to Sustainable Development Goal 9 "Industry, Innovation and Infrastructure", namely item 9.4 "Promote the accelerated development of high- and medium-high-tech sectors of the processing industry, which are formed on the basis of the use of the "education - science - production" chains and a cluster approach in the areas of: development of the innovation ecosystem."

ЗМІСТ

Перелік скорочень 9

Вступ 10

1 Основна ідея розробки програмного забезпечення для автоматизації діалогових інтерфейсів розрахунків наукового призначення 11

1.1 Інтерфейси наукових розрахунків 11

1.2 Мови програмування для створення інтерфейсу та мат-моделей 17

1.3 Актуальність використання Fortran 20

2 Програми для наукових розрахунків та їх структура 23

2.1 Загальна структура програм наукових розрахунків 23

2.2 Вигляд інтерфейсу власної програми для наукових розрахунків 25

2.3 Обґрунтування архітектури інтерфейсу з погляду теорії автоматичного управління 26

2.3.1 Об’єкт керування та основна концепція 27

2.3.2 Математичне визначення стану зависання через динамічні характеристики 27

3 Розробка універсального інтерфейсу за допомогою Fortran 29

3.1 Основна задача інтерфейсу 29

3.2 Аналіз створених файлів 29

3.2.1 Аналіз файлу casta.f90 модуля casta_scr 29

3.2.2 Аналіз файлу casta.f90 модуля casta_appl 35

3.2.4 Аналіз файлу casta.f90 модуля casta_task 41

3.2.5 Аналіз файлу casta.f90 модуля casta_action 42

3.2.6 Аналіз файлу casta.f90 модуля casta_execut 43

3.2.7 Аналіз файлу main.f90 програма запуску 44

4 Охорона праці 48

Висновки 49

Перелік джерел посилання 50

Додаток А Файл casta.f90 52

Додаток Б Файл main.f90 73

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення;

ШІ – штучний інтелект;

G – Git (система контролю версій);

GIL – Global Interpreter Lock (глобальне блокування інтерпретатора);

ipynb – Interactive Python Notebook (інтерактивний пайтон блокнот);

JSON – JavaScript Object Notation (нотація об’єктів джаваскрипт);

RAM – Random Access Memory (оперативна пам’ять).

ВСТУП

У сучасній інженерії та науково-дослідній практиці залишається великий пласт критично важливого програмного забезпечення створеного мовою Fortran. Завдяки своїй гарній продуктивності у математичних операціях, ця мова досі є стандартом для моделювання фізичних процесів, гідродинаміки, астрофізики та квантової хімії. Але більшість таких програм це legacy програми які функціонують через консоль у режимі командного рядка що може відштовхнути від вивчення цієї мови програмування.

Світова практика модернізації наукових програм розвивається в двох напрямках, а саме повне переписування коду сучасними мовами що дуже дорого та має величезні ризики через можливі неточності та помилки, або, створення надбудов для існуючих систем. По цій причині тема цієї роботи залишатиметься актуальною ще дуже довго оскільки вона намагається вирішити фундаментальну інженерну проблему при використанні legacy коду, а саме вдосконалення взаємодії таких програм з користувачем.

Тема цієї роботи буде актуальна ще дуже довго оскільки необхідно підвищувати ергономічність та надійність розрахункових програм. А розробка текстового інтерфейсу дозволить полегшити взаємодію між користувачем та мат-моделі та при цьому не погіршить швидкість розрахунку через те що цей тип інтерфейсу не ресурсномісткий.

Метою роботи є підвищення ефективності взаємодії між користувачем та математичної моделі за допомогою створення текстового інтерфейсу мовою програмування Fortran.

Пояснювальну записку кваліфікаційної роботи оформлено згідно з ДСТУ 3008:2015 [1] та з рекомендаціями з підготовки і оформлення кваліфікаційної роботи здобувачами першого (бакалаврського) рівня вищої освіти [2].

1 ОСНОВНА ІДЕЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ДІАЛОГОВИХ ІНТЕРФЕЙСІВ РОЗРАХУНКІВ НАУКОВОГО ПРИЗНАЧЕННЯ

1.1 Інтерфейси наукових розрахунків

Інтерфейси наукових розрахунків – це програмні та графічні середовища що дозволяють користувачам взаємодіяти з математичними моделями, опрацьовувати великі об’єми даних та візуалізувати ті чи інші результати виконання цих моделей.

Інтерфейси слугують для поліпшення розуміння користувача того що в даний проміжок часу відбувається, а також може візуалізувати деякі потрібні елементи. В залежності від інтерфейсу і того яку саме роль він бере на себе, також залежить те як сильно він буде навантажувати. Усі типи інтерфейсів будуть корисними якщо їх використовувати в розрахунках наукового призначення, але для початку потрібно зрозуміти які типи інтерфейсів існують взагалі і це можна побачити на рис. 1.1.

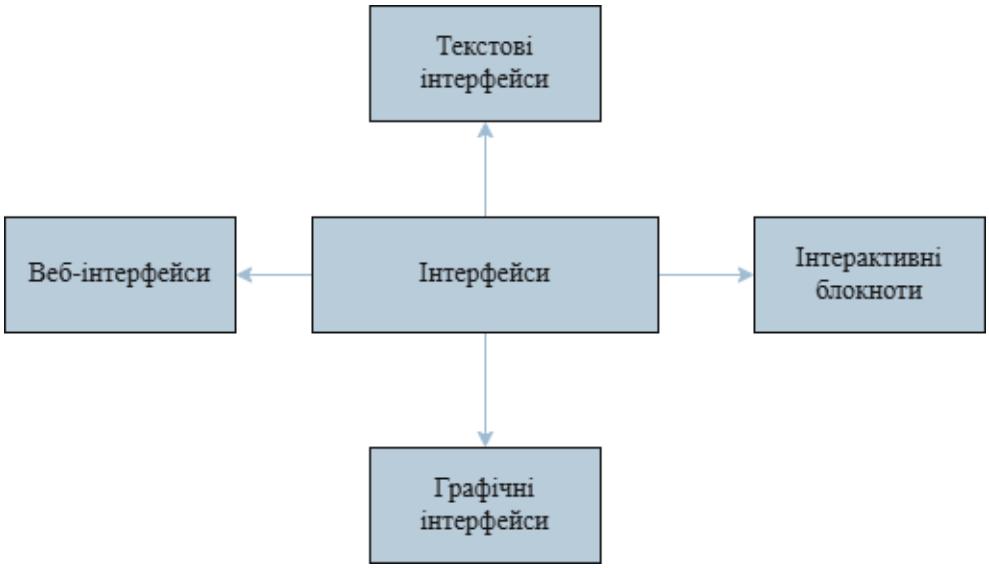


Рисунок 1.1 – Типи інтерфейсів для наукових розрахунків

Дивлячись на рисунок 1.1 можна побачити що було розглянуто лише чотири типи інтерфейсів. Нижче розписано більш докладно о всіх них:

– текстові інтерфейси – це базовий тип інтерфейсу який взаємодіє з користувачем лише завдяки тексту в якому можуть бути питання чи стан програми (рис. 1.2);

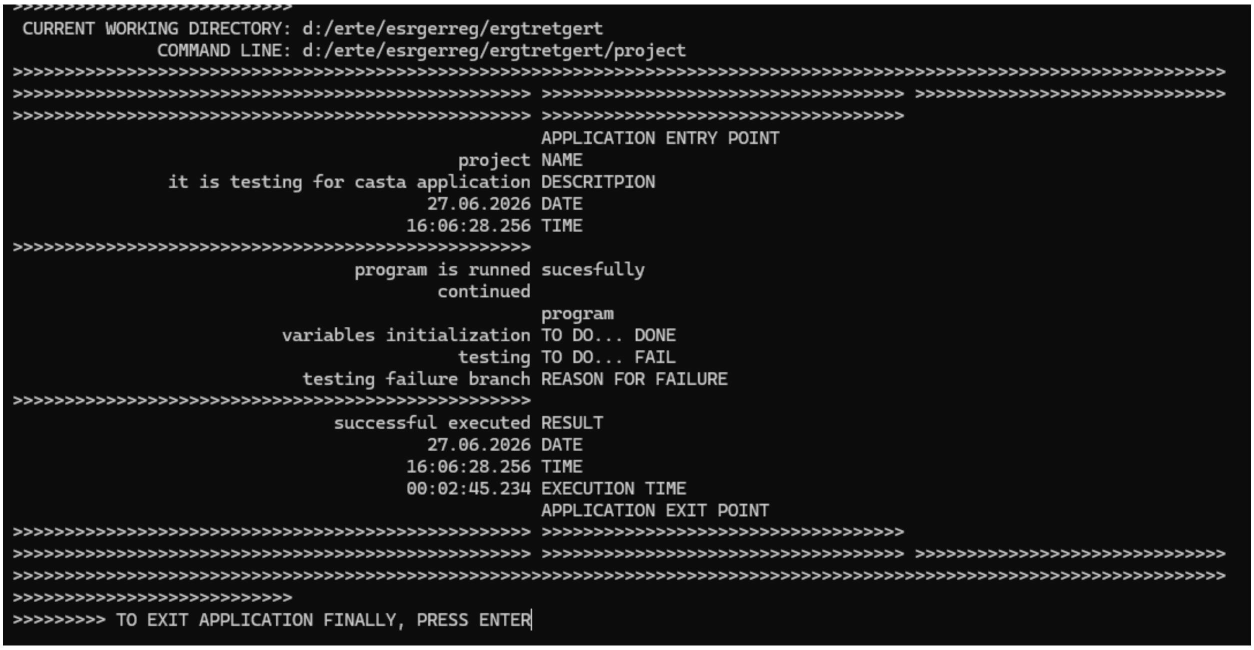


Рисунок 1.2 – Приклад вигляду текстового інтерфейсу

– графічні інтерфейси – це такий тип інтерфейсу який дозволяє більш легко взаємодіяти з користувачем оскільки використовує не лише текст, а і графічні елементи такі як кнопки, бігунки тощо (рис. 1.3);

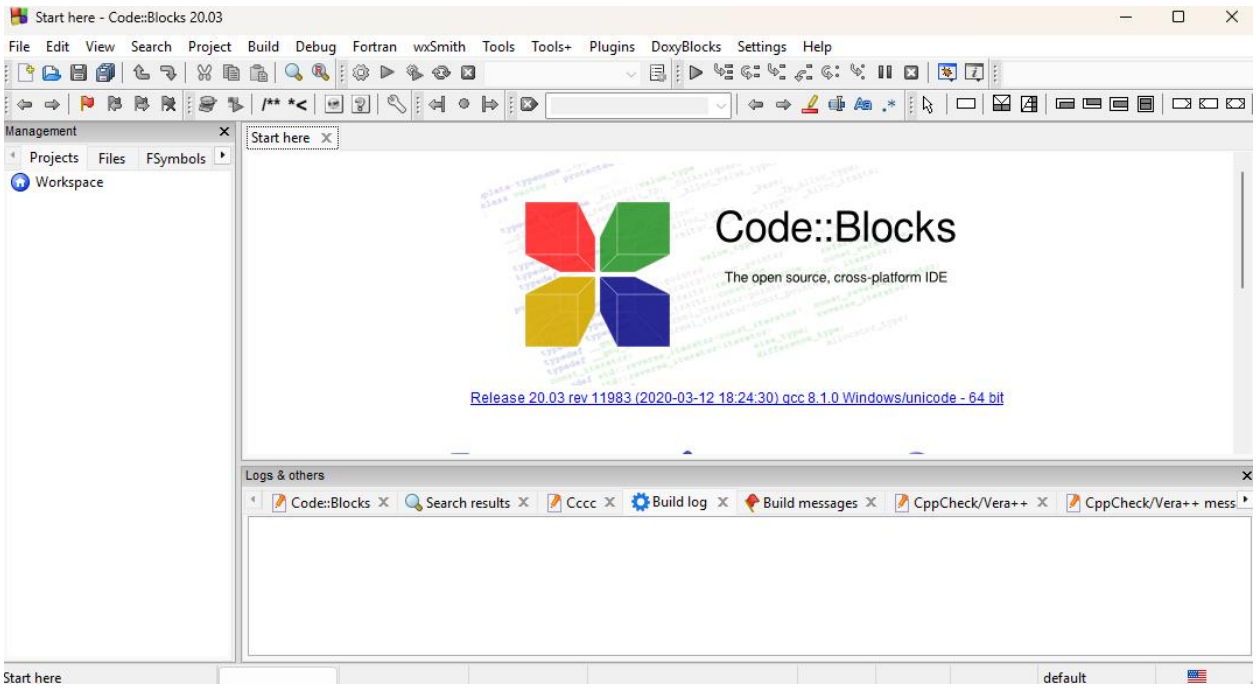


Рисунок 1.3 – Приклад зовнішнього вигляду графічного інтерфейсу

– веб-інтерфейси – цей тип інтерфейсу такий самий як графічний тип інтерфейсу, але який можливо використовувати лише в браузері і за наявності постійного та стабільного інтернет з’єднання (рис. 1.4);

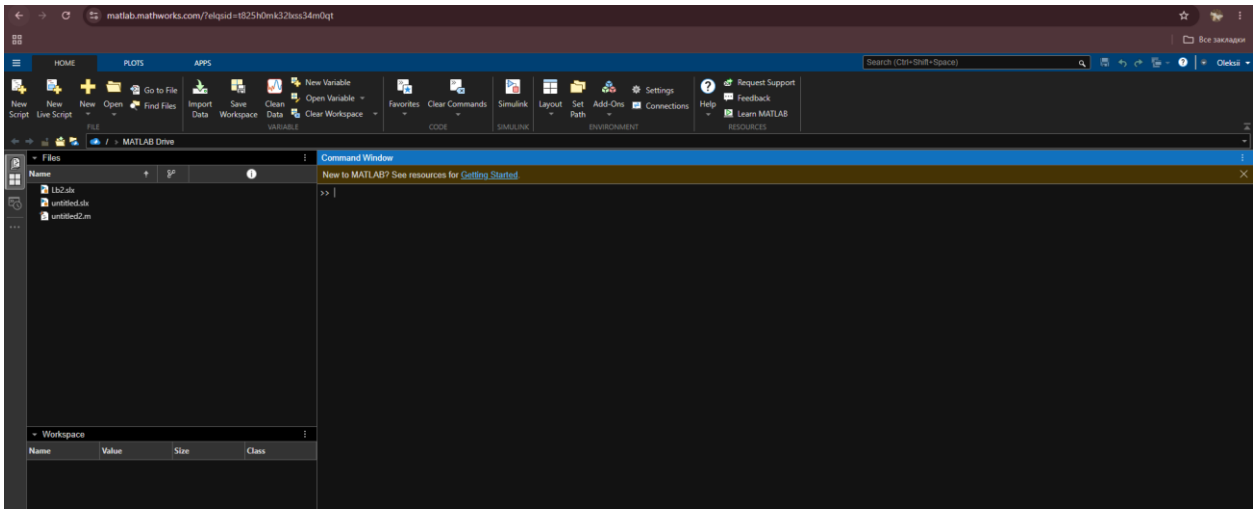


Рисунок 1.4 – Приклад зовнішнього вигляду веб-інтерфейсу [4]

– інтерактивні блокноти – це формат який лише набуває популярності завдяки тому цей тип інтерфейсу поєднує в собі властивості графічного

інтерфейсу та веб інтерфейсу, а також може використовувати ШІ, але на відміну від інший ШІ які можуть вигадувати інформацію чи плутати її в інтерактивному блокноту вони використовують інформацію лише з тих джерел що були додані (рис. 1.5).

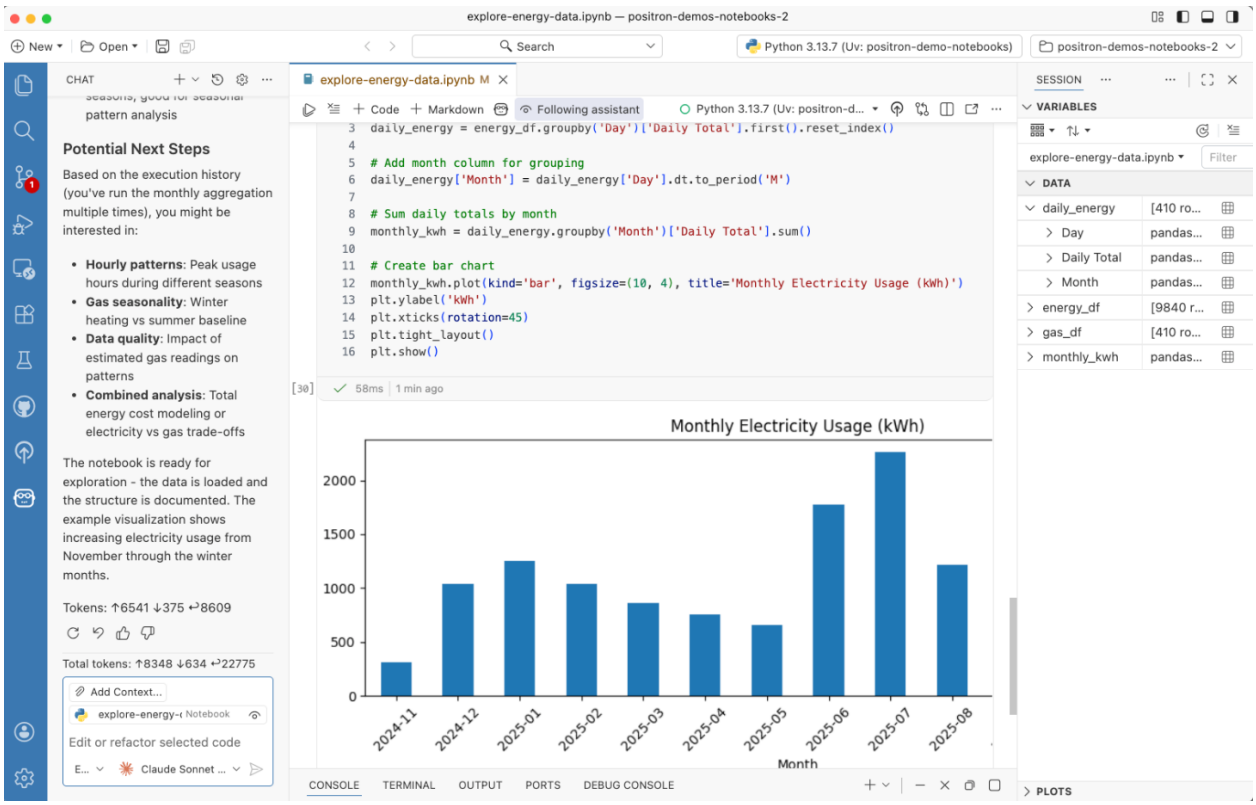


Рисунок 1.5 – Приклад зовнішнього вигляду інтерактивного блокноту [5]

Усі ці інтерфейси мають свої сильні та слабкі сторони о яких я хочу трішки розповісти. У таблиці 1.1 наведені деякі переваги та недоліки текстового інтерфейсу.

Таблиця 1.1 – Переваги та недоліки текстового інтерфейсу

Текстовий інтерфейс	
Переваги	Недоліки
Не навантажує систему та зазвичай не потребує інтернет з’єднання	При недоліку знань може бути занадто складно створити, а в деяких випадках навіть і використовувати

Продовження таблиці 1.1

Текстовий інтерфейс	
Переваги	Недоліки
Гнучкість при створенні скрипта (бо можливо створити унікальний скрипт під потрібну тобі задачу)	Обмеженість того що можливо реалізувати використовуючи лише текст
Можливість розвернути майже будь де	Порівняючи з іншими типами інтерфейсів текстовий майже завжди гірший в виконанні задачі по полегшенню розуміння того що відбувається у програмі

У таблиці 1.2 наведені деякі переваги та недоліки графічного інтерфейсу.

Таблиця 1.2 – Переваги та недоліки графічного інтерфейсу

Графічний інтерфейс	
Переваги	Недоліки
Більш легке розуміння того що відбувається	Вартість (подібні програми коштують дуже багато та зазвичай це підписка, а не разова покупка)
Можливість візуалізації деяких елементів	Вимога до заліза
Зазвичай велика кількість документації	В разі якщо розроблювати такі програми це дуже складно, а якщо брати вже готові то часто немає можливості додати потрібні тобі функції якщо розробник їх не зробив

Оскільки графічні інтерфейси завжди робляться великими компаніями вони дуже зручні та мають багато матеріалів по користуванню цими типами програм.

У таблиці 1.3 наведені деякі переваги та недоліки веб-інтерфейсу.

Таблиця 1.3 – Переваги та недоліки веб-інтерфейсу

Веб-інтерфейс	
Переваги	Недоліки
Не потрібно нічого встановлювати на свій пристрій (бо все працює в хмарі)	Дуже важливо мати гарне з’єднання з інтернетом

Продовження таблиці 1.3

Веб-інтерфейс	
Переваги	Недоліки
Доступність (через те що все працює в хмарі не важливо який в тебе пристрій)	Обмеження сесій в безкоштовній версії
Сумісна робота (можливо налаштувати сумісну працю з іншими користувачами над одним проєктом з загальним доступом)	Невелика гнучкість (бо ви використовуєте вже заготовлені команди)

У таблиці 1.4 наведені деякі переваги та недоліки інтерактивного блокноту

Таблиця 1.4 – Переваги та недоліки інтерактивного блокноту

Інтерактивний блокнот	
Переваги	Недоліки
Легкість в демонстрації	Високі вимоги до кількості та якості джерел
Швидкість користування	Через те що подібні блокноти працюють в хмарі їх праця дуже сильно залежить від RAM та інтернет з’єднання користувача
Наочність	Через те що .ipynb це складний JSON-файл може бути важко порівняти в G

Як можна побачити по таблиця 1.1 – 1.4 у кожного з цих інтерфейсів є свої сильні сторони і кожен із них набув своєї популярності. Якщо розглядати з точки зору розробника то створити текстовий інтерфейс буде найлегше, а веб-інтерфейс чи інтерактивний блокнот найважче, бо для останніх потрібні і графічні елементи які потрібно створювати та сервер на якому ці інтерфейси будуть знаходитись. А якщо говорити про графічний інтерфейс тоді основними проблемами буде створення усіх графічних елементів та можливість стабільної праці на різних платформах.

Нижче є рис. 1.6 на якому наведено які програми до якого типу інтерфейсу відносяться.

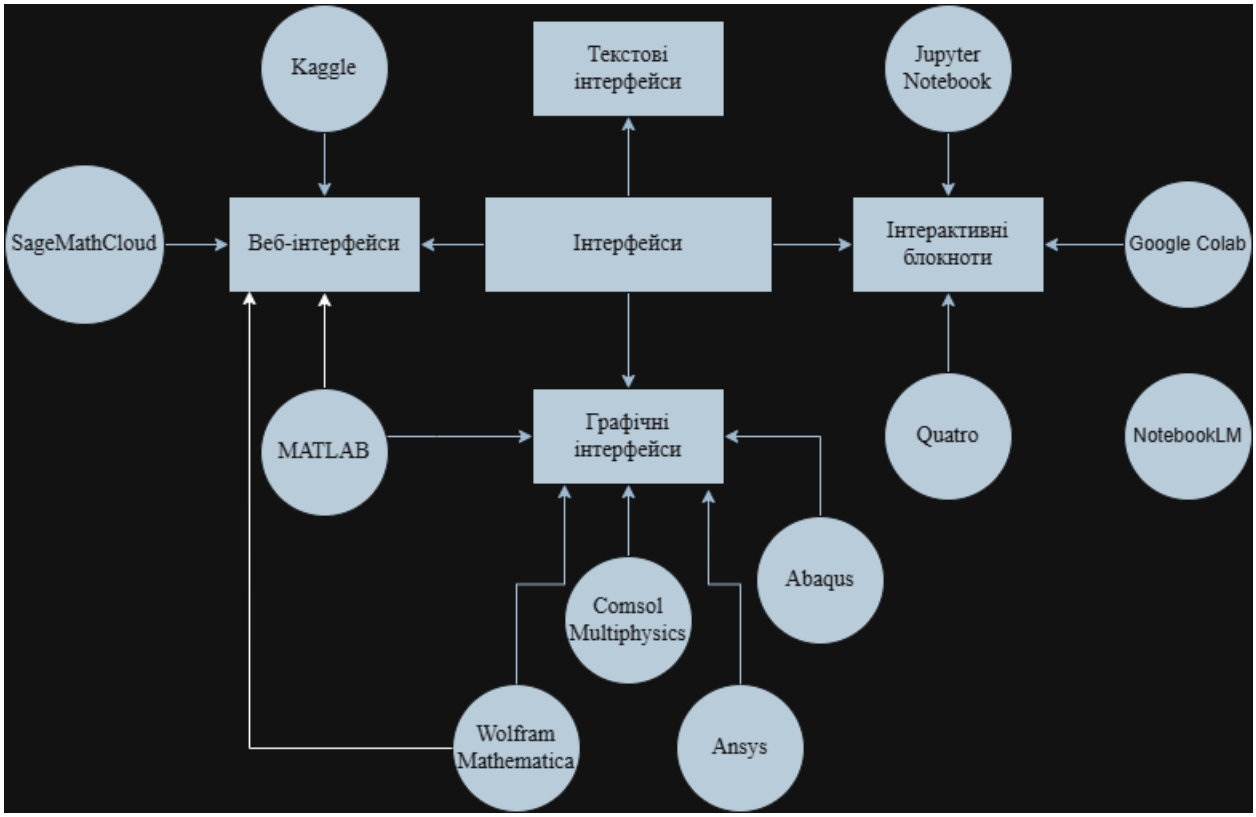


Рисунок 1.6 – Назви програм і до яких типів інтерфейсів вони відносяться

Усі програми які були наведені на схемі з рис. 1.6 можна використовувати для наукових розрахунків, але більшість з них платні. У MATLAB є можливість використовувати деякі його функції безкоштовно але лише двадцять годин в місяць [4]. Wolfram Mathematica також має безкоштовну веб версію але в ній ще більше обмежень [6]. Оскільки усі ці програми працюють по схемі підписки щоб отримати потрібний функціонал в разі якщо цей функціонал не надається безкоштовно може бути дорого та складно. Тому є сенс в тому щоб самому розробити мат-модель під свою задачу та інтерфейс до неї.

1.2 Мови програмування для створення інтерфейсу та мат-моделей

Насправді і для створення інтерфейсу, і для створення мат-моделі можна

використовувати майже будь яку мову програмування, але в залежності від обраної мови буде залежати швидкість, складність та вимоги до заліза. Усі програми написані на якісь мові програмування у MATLAB та Wolfram Mathematica свої мови які можливо використовувати лише в їх ПЗ тому вони не допоможуть в створенні свого інтерфейсу чи мат-моделі, а інші використовують наприклад, C++, Python чи Fortran. У таблицях 1.5-1.7 наведені деякі переваги, недоліки та швидкість розробки програм на цих мовах

Таблиця 1.5 – Переваги та недоліки C++

Переваги	Недоліки	Швидкість розробки
Має прямий доступ до управління пам’яттю та залізом пристрою завдяки чому може реалізувати його потенціал на повну	Потрібно дуже велика кількість знань щоб зробити щось нормальне	Низька
Велика кількість можливих поєднань з різними бібліотеками завдяки чому можливо досягнути потрібного результату	Через можливість управління власним залізом є ризик пошкодити свою систему	

C++ надає велику кількість можливостей але в разі якщо не знати як їх використовувати це майже нічого не дасть і може навпаки нашкодити.

Таблиця 1.6 – Переваги та недоліки Python

Переваги	Недоліки	Швидкість розробки
Легкість в використанні та вивченні	Не дуже швидкий коли йдуть важкі розрахунки	Висока

Продовження таблиці 1.6

Переваги	Недоліки	Швидкість розробки
Можливість легко виводити різні графіки чи діаграми	Через GIL [8] не використовує усі ядра в багатоядерних процесорах що робить його не дуже швидким в складних задачах	Висока

Python надає увесь потрібний інструментарій для створення різних програм і при цьому не дуже складний в синтаксисі та в вивченні, але якщо потрібно зробити дуже складні розрахунки краще використати іншу мову програмування.

Таблиця 1.7 – Переваги та недоліки Fortran[9]

Переваги	Недоліки	Швидкість розробки
Спеціально розроблена мова програмування для математичних операцій завдяки чому має дуже високу швидкість	Із-за того що мова спрямована лише на мат. операції в усьому іншому вона найгірша оскільки немає інструментів для цього	Висока
Легкість написання коду навіть якщо до цього ніколи не програмував	Відсутність можливості створення веб версії	

Не дивлячись на недоліки Fortran [3,7] його синтаксис дуже простий і якщо задача стоїть лише в тому щоб розробити мат-модель для власних розрахунків тоді вибір цієї мови програмування буде правильним.

1.3 Актуальність використання Fortran

Якщо дивитись на те наскільки Fortran в 2026 році актуальний то слідує подивитись на рейтинги TIOBE за останні місяці які показують те як часто використовують ту чи іншу мову програмування (рис. 1.7-1.8).

May 2026	May 2025	Change	Programming Language	Ratings	Change
1	1		Python	19.98%	-5.37%
2	3	↑	C	11.55%	+1.84%
3	4	↑	Java	7.94%	-1.37%
4	2	↓	C++	7.92%	-2.02%
5	5		C#	5.41%	+1.19%
6	6		JavaScript	3.08%	-0.60%
7	8	↑	Visual Basic	2.90%	+0.28%
8	12	⇈	R	1.77%	+0.31%
9	10	↑	SQL	1.57%	-0.33%
10	9	↓	Delphi/Object Pascal	1.44%	-0.85%
11	11		Fortran	1.22%	-0.55%
12	14	↑	Scratch	1.18%	-0.16%
13	16	↑	Perl	1.18%	-0.02%
14	15	↑	PHP	1.15%	-0.07%
15	19	⇈	Rust	1.14%	+0.21%
16	7	⇊	Go	1.12%	-1.58%
17	18	↑	Assembly language	1.02%	+0.05%
18	23	⇈	Swift	0.93%	+0.16%
19	13	⇊	Ada	0.92%	-0.50%
20	17	↓	MATLAB	0.89%	-0.13%

Рисунок 1.7 – Рейтинг мов програмування за травень [10]

Jun 2026	Jun 2025	Change	Programming Language	Ratings	Change
1	1		Python	18.96%	-6.91%
2	3	↑	C	10.77%	+1.30%
3	2	↓	C++	8.03%	-2.65%
4	4		Java	7.90%	-0.94%
5	5		C#	4.85%	+0.17%
6	6		JavaScript	3.04%	-0.17%
7	8	↑	Visual Basic	2.80%	+0.59%
8	12	⇈	SQL	1.77%	+0.23%
9	14	⇈	R	1.69%	+0.30%
10	9	↓	Delphi/Object Pascal	1.54%	-0.60%
11	16	⇈	Scratch	1.46%	+0.27%
12	18	⇈	Rust	1.26%	+0.30%
13	7	⇊	Go	1.20%	-1.08%
14	15	↑	PHP	1.17%	-0.08%
15	25	⇈	Swift	1.00%	+0.27%
16	11	⇊	Ada	0.97%	-0.74%
17	10	⇊	Fortran	0.95%	-0.91%
18	13	⇊	Perl	0.92%	-0.55%
19	19		Assembly language	0.87%	-0.04%
20	17	↓	MATLAB	0.85%	-0.28%

Рисунок 1.8 – Рейтинг мов програмування за червень [10]

Як можна побачити з цих двох рейтингів Fortran стабільно знаходиться в топ-20 мов програмування світу оскільки не дивлячись на те коли його створили він все ще залишається актуальною та потрібною мовою програмування в місцях де потрібна велика кількість математичних розрахунків (наприклад для прогнозування погоди чи симуляціях польоту ракет в космос).

Основна причина по якій Fortran [3] все ще використовується полягає в тому що досі не має мови програмування яка обходить Fortran у швидкості підрахунку і яка при цьому не помиляється в своїх розрахунках і що теж не мало важливо формули в якій будуть максимально наближені до таких самих на папері що спрощує розуміння написаного. Також не мало важну роль відіграє те що багата кількість ключових речей в енергетиці чи астронавтиці

працюють на програмах які написані саме на Fortran і замінить які майже неможливо оскільки вже існуючі програми повністю виконують поставлені перед ними задачі на високому рівні, а програми які можна було би створити наприклад на Python з високою часткою ймовірності впоралась гірше із-за чого величезна кількість грошей на розробку програми були би потрачені впусту, а ціною помилки були б не лише гроші, а й життя людей оскільки Fortran використовується в дуже небезпечних містах.

По цим причинам на мій погляд створення більш менш універсального інтерфейсу для мат-моделі яку створювали на Fortran буде корисно оскільки в відкритому доступі готових програм чи мат-моделей побудованих на Fortran немає, а тому є сенс створити власний інтерфейс який піді йде к будь якій мат-моделі.

2 ПРОГРАМИ ДЛЯ НАУКОВИХ РОЗРАХУНКІВ ТА ЇХ СТРУКТУРА

2.1 Загальна структура програм наукових розрахунків

Загальна структура програм для наукових розрахунків зазвичай не дуже сильно чи зовсім не концентрується на графічній частині інтерфейсу та взаємодії з користувачем, зміщена в сторону лінійної або паралельної обробки великих масивів даних, точності обчислень та ефективному використанні пам’яті. Це не залежить від мови програмування яку використовують для створення мат-моделі та інтерфейсу к ній і це пов’язано с тим що під час роботи таких програм користувачу не потрібно бачити усю цю інформацію тому що для того щоб її відобразити (особливо якщо графічно відобразити) потрібно знизити швидкість розрахунку які і так можуть проходити декілька годин чи днів.

Також слід враховувати що зазвичай подібні програми працюють на суперкомп’ютерах які розраховані на те щоб на них проводили дуже складні розрахунки. На рис. 2.1 можна побачити приблизну структуру роботи подібних програм.



Рисунок 2.1 – Структура роботи програм для наукових розрахунків

Як можна побачити з рис. 2.1 першим ділом програма ініціалізує усі свої елементи та те що їй потрібно зробити, також вона може виділяти потрібну кількість пам’яті оскільки наукові розрахунки зазвичай заздалегідь виділяють потрібну кількість пам’яті щоб уникнути затримок під час розрахунку. Після ініціалізації починає працювати математичка модель яку ще називають обчислювальним ядром, це буквально центр усієї програми і майже увесь час роботи програми займає саме цей етап. Наступний етап це створення контрольних точок і він потрібен на той випадок якщо під час виконання програми відбудеться якась проблема тому потрібно щоб математична модель періодично робила збереження щоб після усунення проблеми була можливість продовжити з контрольної точки, а не починати спочатку це дозволяє уникнути моментів коли розрахунок йшов наприклад двадцять годин після чого через апаратний збій зупинився. При умові що програма не робить контрольні точки усі двадцять годин розрахунку були потрачені в нікуди, а якщо вони робились тоді деяка частина розрахунку все ж таки залишилась.

Останній етап це збереження результатів розрахунку для подальшого аналізу, а також очищення пам’яті пристрою після завершення роботи програми. Оскільки зазвичай подібні програми працюють з великим обсягом даних їх результати еквівалентні тобто також великі за обсягом, а тому вони зберігаються в окремих файлах після чого вчені та спеціаліст аналізують їх та якщо потрібно завдяки іншій програми роблять графіки.

2.2 Вигляд інтерфейсу власної програми для наукових розрахунків

В цій роботі буде створено лише інтерфейс який буде можливо додати до будь якого обчислювального ядра розробленого на Fortran і для запобігання проблем з різницею версій чи налаштування взаємодій різних мов програмування для створення інтерфейсу буде використовуватись Fortran, а як вже говорилось в попередньому розділі на етапі розрахунку не потрібен графічний інтерфейс оскільки після розрахунку математичною моделлю усіх даних буде створено файл з результатами, а враховуючи те що в Fortran немає інструментів для створення графічного інтерфейсу буде створено текстовий інтерфейс для базового розуміння того що відбувається.

Цей інтерфейс можна буде налаштувати перед запуском програми, а для запобігання проблем з відображенням тексту будуть прописані обмеження по кількості символів та прописані базові значення та текст для комірок та тексту який слугує для комунікації з користувачем.

Для зручності інтерфейс буде розділено на три поля де кожне з цих полів буде переважно відображати конкретну інформацію що дозволить більш легко та швидко розуміти що відбувається і куди дивитись. У першому полі буде інформація зв’язана з тим що зараз робить інтерфейс, у другому полі загальна інформація, а в третьому полі інформація зв’язана с математичною моделлю. Схематичне зображення цього інтерфейсу можна побачити на рис. 2.2

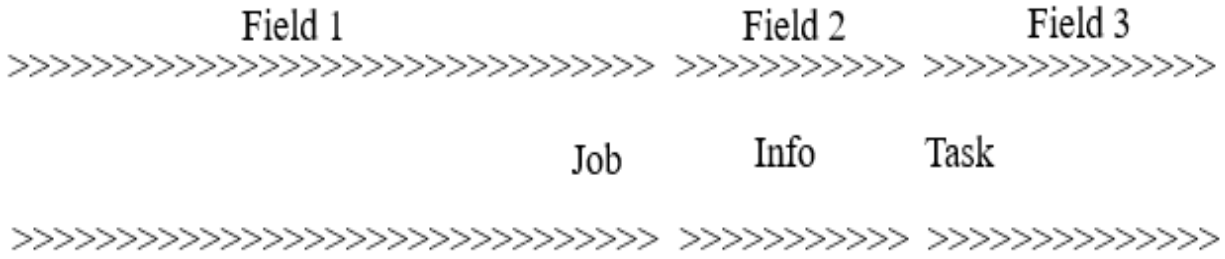


Рисунок 2.2 – Схематичне зображення інтерфейсу

На рис. 2.2 можна побачити схематичне зображення основного вигляду інтерфейсу. Для позначення границь полів будуть використовуватись різні символьні знаки на кшталт >, =, : що вирішить одразу декілька проблем. А саме те що буде зрозуміло де границя одного поля і початок іншого, а завдяки тому що границі позначаються різними символами буде легше орієнтуватись в усьому інтерфейсі. Оскільки текст в полях може відрізнятись по кількості символів, а в деяких випадках те що написано в одному полі буде продовжуватись в іншому буде зроблено вирівнювання для кожного поля. Це також можна побачити на рис. 2.2 де вирівнювання для першого поля по правому краю. Для другого поля по центру, а для третього поля по лівому.

2.3 Обґрунтування архітектури інтерфейсу з погляду теорії автоматичного управління

Цей інтерфейс можна віднести до розімкненої системи автоматичного контролю. Це обумовлено тим що розроблювана програма, а саме інтерфейс функціонує в пасивному режимі та не передбачає введення керуючих директив або параметрів користувачем для початку чи під час розрахунку. І з погляду теорії автоматичного управління таку систему класифікують як розімкнену систему автоматичного контролю та моніторингу

2.3.1 Об’єкт керування та основна концепція

У описаній системі обчислювальне ядро, що повинно також бути реалізовано за допомогою мови програмування Fortran, буде виступати як об’єкт керування. В кожен момент часу внутрішній стан цього об’єкту *t* описується вектором стану *X(t)* в який може входити масиви даних, проміжні зміні та лічильники ітерацій. Цей вектор не є явним оскільки знаходиться в RAM пристрою і він недоступний для прямого візуального аналізу користувачем.

Тому розроблюваний інтерфейс виконує роль детермінованого пасивного спостерігачу стану. Його математична функція зводиться до відображення простору станів *X(t)* у простір вихідних сигналів *Y(t)* які може сприйняти користувач формула 2.1 – 2.2:

$$Y(t) = F\big(X(t)\big), \tag{2.1}$$

де *Y(t)* – це тривимірний вектор вихідних даних інтерфейсу що відповідає запроєктованій структурі (Field1, Field2, Field3)

$$Y(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \end{bmatrix} = \begin{bmatrix} Field1 \\ Field2 \\ Field3 \end{bmatrix}. \tag{2.2}$$

2.3.2 Математичне визначення стану зависання через динамічні характеристики

Основною ідеєю розімкнутої системи моніторингу це надати користувачу можливість ідентифікувати коли програма зависла. Це можна реалізувати через аналіз похідної вихідного сигналу за часом формула 2.3:

$$\frac{dY}{dt}, \tag{2.3}$$

Поки математична модель виконує розрахунки вектор стану *X(t)* постійно змінюється. Відповідно, вихідні сигнали в полях інтерфейсу також можуть оновлюватись. Тобто математичною умовою нормального функціонування є нерівність похідної нулю формула 2.4:

$$\frac{dY}{dt} \neq 0, \tag{2.4}$$

Для користувача який в цій системі є зовнішнім елементом управління наявність динаміки $\Delta Y = 0$ відіграє роль візуального підтвердження стійкості процесу.

В разі якщо математична модель потрапляє в нескінченний цикл тобто зависає через будь яку проблему (наприклад переповнення пам’яті) тоді вектор стану також зависає $X(t) = const$ в наслідок чого текстові поля інтерфейсу перестають оновлюватись формула 2.5 :

$$\frac{dY}{dt} = 0, \tag{2.5}$$

Оскільки в цій системі не передбачено зворотного зв’язку, користувач коли бачить цю умови втручається в роботу системи зовні і перезапускає систему.

3 РОЗРОБКА УНІВЕРСАЛЬНОГО ІНТЕРФЕЙСУ ЗА ДОПОМОГОЮ FORTRAN

3.1 Основна задача інтерфейсу

Основними вимогами до нього буде те щоб його можливо було легко пере налаштувати, а його зовнішній вигляд був зрозумілий та демонстрував необхідну інформацію.

Для виконання поставлених задач буде створено два файли, а саме casta.f90 в якому буде вся логіка та базові налаштування інтерфейсу та main.f90 завдяки якому будуть викликатись потрібні функції з файлу casta.f90 і запускатись сам інтерфейс.

3.2 Аналіз створених файлів

В файлі casta.f90 всього шість модулів і кожен модуль потрібен для реалізації тієї чи іншої функції, а також має свої базові налаштування на той випадок якщо користувач вирішить нічого не змінювати. А файл main.f90 як вже було сказано потрібен для виклику потрібних функцій з файлу casta.f90 та виклику математичної моделі за її наявності.

3.2.1 Аналіз файлу casta.f90 модуля casta_scr

Лістинг коду – 3.1 згідно додатку А

```
module casta_scr

  implicit none
  integer, parameter, private :: f1_def=100, f2_def=35, f3_def=30
  integer, parameter, private :: f1_min=30,  f2_min=25, f3_min=16
  integer, parameter, private :: f1_max=165, f2_max=50, f3_max=40

  type casta_scr_topt
     integer :: field1=f1_def,field2=f2_def,field3=f3_def
  end type casta_scr_topt
```

```
type(casta_scr_topt), protected :: casta_scr_opt

public casta_scr_setfields,&
      casta_scr_linestart,casta_scr_linereturn,&
      casta_scr_textwof,casta_scr_textfar,casta_scr_textfal,&
      casta_scr_text,&
      casta_scr_textfar1,casta_scr_textfar2,casta_scr_textfar3,&
      casta_scr_textfal1,casta_scr_textfal2,casta_scr_textfal3,&
      casta_scr_linesymbolfields,casta_scr_textblancfields
contains
```

Після оголошення початку модуля я встановлюю правило завдяки якому кожна зміна повинна бути явно оголошеною (implicit none). Для зручності інтерфейс буде поділений на три комірки тому потрібно прописати значення які будуть за замовченням та рамки в яких можна встановлювати ці значення після чого створюється користувацький тип даних який містить усі значення за замовченням та прописується рядок який не дає іншим модулям змінювати значення за замовченням завдяки атрибуту protected. Далі перелічується список підпрограм цього модуля.

Лістинг коду – 3.2 згідно додатку А

```
subroutine casta_scr_setfields(field1,field2,field3)
      integer, intent(in), optional :: field1,field2,field3
      logical :: f1,f2,f3
      f1=present(field1)
      f2=present(field2)
      f3=present(field3)
      if ((.not.f1).and.(.not.f2).and.(.not.f3)) then
         casta_scr_opt%field1=f1_def
         casta_scr_opt%field2=f2_def
         casta_scr_opt%field3=f3_def
      else
         if (f1) call setfield(field1,f1_min,f1_max,casta_scr_opt%field1)
         if (f2) call setfield(field2,f2_min,f2_max,casta_scr_opt%field2)
         if (f3) call setfield(field3,f3_min,f3_max,casta_scr_opt%field3)
      end if
      contains
      subroutine setfield(fv,fmin,fmax,f)
```

```
        integer, intent(in) :: fv,fmin,fmax
        integer, intent(out) :: f
        if (fv<fmin) then
           f=fmin
        elseif (fv>fmax) then
           f=fmax
        else
           f=fv
        endif
    end subroutine setfield
end subroutine casta_scr_setfields
```

Підпрограма яку можна побачити у лістингу 3.2 визначає ширину комірок і якщо її не встановили вона встановлює значення за замовченням (100, 35, 30), а якщо все ж таки було передано хоча б один параметр тоді викликається підпрограма setfield яка перевіряє щоб задане значення було в встановлених межах мінімальних (30, 25, 16) та максимальних (165, 50, 40) значень, а в разі якщо встановлене значення менше мінімуму чи більше максимуму воно примусово зміниться на найближче дозволене значення.

Лістинг коду – 3.3 згідно додатку А

```
subroutine casta_scr_linestart()
    write(*,'(1X)',advance='no')
end subroutine casta_scr_linestart
subroutine casta_scr_linereturn()
    write(*,*)
end subroutine casta_scr_linereturn
subroutine casta_scr_textwof(text,rep)
    character(len=*), intent(in) :: text
    integer,intent(in),optional :: rep
    integer:: i
    if (present(rep)) then
        do i=1,rep
```

```
            write(*,'(A)',advance='no') text
        end do
    else
        write(*,'(A)',advance='no') text(1:len_trim(text))
    end if
end subroutine casta_scr_textwof
```

В лістингу 3.3 можна побачити три підпрограми, а саме linestart, linereturn та textwof. Linestart потрібна для того щоб починати новий рядок та залишати курсор на цьому рядку, а linereturn переводить курсор на новий рядок. Textwof виводить простий текст приймаючи рядок (text) з опціональною кількістю повторень (rep), а якщо кількість повторень не задана тоді виводить текст без пробілів в кінці (для цього використовується len_trim(text))і без переведення рядка.

Лістинг коду – 3.4 згідно додатку А

```
subroutine casta_scr_textfar(text,field,blanc)
    character(len=*), intent(in) :: text
    character(len=1), intent(in),optional :: blanc
    integer,intent(in) :: field
    character(len=1) :: bl
    integer :: lentext
    if (present(blanc)) then
        bl=blanc
    else
        bl=' '
    end if
    lentext=len_trim(text)
    if (field <= lentext) then
        call casta_scr_textwof(text(1:field),rep=1)
```

```
    else
        call casta_scr_textwof(bl,rep=field-lentext)
        call casta_scr_textwof(text(1:lentext),rep=1)
    end if
end subroutine casta_scr_textfar
subroutine casta_scr_textfal(text,field,blanc)
    character(len=*), intent(in) :: text
    character(len=1), intent(in),optional :: blanc
    integer,intent(in) :: field
    character(len=1) :: bl
    integer :: lentext
    if (present(blanc)) then
        bl=blanc
    else
        bl=' '
    end if
    lentext=len_trim(text)
    if (field <= lentext) then
        call casta_scr_textwof(text(1:field),rep=1)
    else
        call casta_scr_textwof(text(1:lentext),rep=1)
        call casta_scr_textwof(bl,rep=field-lentext)
    end if
end subroutine casta_scr_textfal
```

Обидві підпрограми які можна побачити в лістингу 3.4 слугують для вирівнювання тексту але з урахуванням на те що textfar працює на першу комірку та вирівнює по правому краю, а textfal на другу та по лівому. Обидві ці підпрограми працюють майже однаково, а саме що якщо був передано заповнювач тоді виконує його, а якщо ні то символ за замовчуванням (пробіл),

а також воно обчислює фактичну довжину тексту без кінцевих пробілів також в разі якщо текст більший чи такий самий як розмір поля він його обрізає до розміру поля.

Також є велика підпрограма яка має назву casta_scr_text і вона об’єднує в собі майже всі інші підпрограми в цьому модулі, завдяки чому стає легше використовувати усі ці підпрограми. В самій підпрограмі text можна задати текст для будь якої комірки, а також вирівнювання чи заповнювач. Також в цій підпрограмі прописано встановлення нового рядка (виклик підпрограм linestart та linereturn), э можливість задати кількість повторень тексту (завдяки textwof). Якщо коротко ця підпрограма говорить коли і як виконуватись іншим підпрограмам та спрощує це.

Група підпрограм textfar1, textfar2, textfar3, textfal1, textfal2, textfal3 працюють за одним й тим самим принципом але з поправкою на те що кожна з них працює лише на конкретну комірку та визиває іншу підпрограму. Наприклад textfal2 працює на першу комірку та визиває підпрограму textfal.

Лістинг коду – 3.5 згідно додатку А

```
subroutine casta_scr_linesymbolfields(symb1,symb2,symb3)
    character(len=1),intent(in),optional :: symb1,symb2,symb3
    call casta_scr_linestart()
    if (present(symb1)) then
       call casta_scr_textwof(text=symb1,rep=casta_scr_opt%field1)
    else
       call casta_scr_textwof(text=’ ‘,rep=casta_scr_opt%field1)
    end if
    call casta_scr_textwof(text=’ ‘,rep=1)
    if (present(symb2)) then
       call casta_scr_textwof(text=symb2,rep=casta_scr_opt%field2)
    else
       call casta_scr_textwof(text=’ ‘,rep=casta_scr_opt%field2)
```

```
    end if
    call casta_scr_textwof(text=' ',rep=1)
    if (present(symb3)) then
       call casta_scr_textwof(text=symb3,rep=casta_scr_opt%field3)
    else
       call casta_scr_textwof(text=' ',rep=casta_scr_opt%field3)
    end if
    call casta_scr_linereturn()
  end subroutine casta_scr_linesymbolfields
```

Підпрограма яку можна побачити на лістингу 3.5 використовується для створення горизонтальних ліній для того щоб відділити шапку таблиці від даних. Вона приймає різні символи (наприклад -, =, *, <, >) при умові якщо символ не передано заповнює поле пробілами. Також ця підпрограма визиває підпрограми linestart, textwof, linereturn.

Остання підпрограма в цьому модулі має назву textblancfields. Саме завдяки цій підпрограмі в консолі усі комірки мають заповнення. Як і майже усі підпрограми до цього ця підпрограма викликає інші підпрограми при певних умовах завдяки чому цими умовами можна маніпулювати змінюючи зовнішній вигляд програми.

3.2.2 Аналіз файлу casta.f90 модуля casta_appl

Якщо перший модуль (casta_scr) був націлен на вирівнювання тексту, колонок то другий модуль (casta_appl) потрібен для відображення цього при запуску та підсумків при завершені роботи.

Лістинг коду – 3.6 згідно додатку А

```
module casta_appl

  integer,parameter,private :: width=27
```

```
   contains

   subroutine casta_appl_entry(name,descr,curdate,curtime,comline,curwdir)
      use casta_scr,only: opt => casta_scr_opt,&
                  linestart => casta_scr_linestart,&
                  linereturn => casta_scr_linereturn,&
                  addtext => casta_scr_text
      character(len=*),intent(in),optional                                    ::
name,curdate,curtime,comline,curwdir,descr
      call addtext(text='>',rep=width,newline='yes')
      if (present(curwdir)) call insmessage(name='CURRENT WORKING
DIRECTORY:',message=curwdir)
      if (present(comline)) call insmessage(name='COMMAND
LINE:',message=comline)
      call
addtext(text='>',rep=opt%field1+opt%field2+opt%field3+2,newline='yes')
      call
addtext(text1='',blanc1='>',text2='',blanc2='>',text3='',blanc3='>',newline='yes
')
      call          addtext(text1='',blanc1='>',text2='',blanc2='>',text3='',blanc3='
',newline='yes')
      call addtext(text3='',blanc1=' ',&
            text2='APPLICATION ENTRY POINT',align1='right',&
            text1='',align2='left',&
            newline='yes')
      if (present(name)) call addtext(text1=name,blanc1=' ',align1='right',&
                        text2='NAME',align2='left',&
                        text3='',align3='right',&
                        newline='yes')
      if (present(descr)) call addtext(text3='',blanc3=' ',&

text1=descr,text2='DESCRITPION',align1='r',align2='l',newline='y')
      if (present(curdate)) call addtext(text3='',blanc3=' ',&
                        text2='DATE',align2='left',&
                        text1=curdate,align1='right',&
                        newline='yes')
      if (present(curtime)) call addtext(text3='',blanc3=' ',&
                        text2='TIME',align2='left',&
                        text1=curtime,align1='right',&
                        newline='yes')
      call      addtext(text1='',blanc1='>',text2='',blanc2='      ',text3='',blanc3='
',newline='yes')
      contains
      subroutine insmessage(name,message)
```

```
        character(len=*),intent(in) :: name,message
        call linestart()
        call addtext(text=name,align='r',field=width)
        call addtext(text=' ',rep=1)
        call addtext(text=message)
        call linereturn()
    end subroutine insmessage
end subroutine casta_appl_entry
```

Як можна побачити у лістингу 3.6 першою підпрограмою цього модулю є casta_appl_entry і вона потрібна для виведення стартової інформації. Спочатку програма виводить задані данні (на даному етапі вони не перевіряються в реальному часі) і вони всі optional. Далі імпортуються підпрограми з першого модуля і завдяки можливостям Fortran замість повної назви їм можливо дати коротке прізвище (наприклад замість casta_scr_text у цьому модулі буде використовуватись addtext). При відображенні в консолі першим буде лінія заданих символів (>) довжиною 27 та виводиться передана назва директорії (задано до початку) після чого на ширину усіх комірок виводиться заданий символ (>) далі за допомогою підпрограми addtext виводиться уся ця інформація. Наступна підпрограма з назвою exit робить майже те саме але в іншому порядку (приймає результат роботи res, час виконання exetime та задану дату, та час)

Лістинг коду – 3.7 згідно додатку А

```
subroutine casta_appl_message(mes1,mes2)
    use casta_scr, only: addtext => casta_scr_text
    character(len=*),intent(in),optional :: mes1,mes2
    character(len=255) :: m1,m2
    if (present(mes1)) then
        m1=mes1
    else
        m1=''
    end if
    if (present(mes2)) then
        m2=mes2
    else
```

```
        m2=''
    end if
    call addtext(text1=m1,text2=m2,align1='r',align2='l',newline='y')
  end subroutine casta_appl_message

  subroutine casta_appl_finalpause()
    use casta_scr,only: linestart => casta_scr_linestart,&
                addtext => casta_scr_text
    call linestart()
    call addtext(text1=' TO EXIT APPLICATION FINALLY, PRESS
ENTER',align1='r',blanc1='>')
    read(*,*)
  end subroutine casta_appl_finalpause
```

Наступні дві підпрограми це message та finalpause. Перша потрібна для швидкого виведення повідомлення у дві колонки, а друга потрібна для того щоб коли програма закінчила виконуватись вона не закривалась одразу, а чекала поки користувач не натисне Enter. Це зроблено для того щоб якщо користувач відійшов від пристрою програма не закрилась поки його немає і він міг все побачити (реалізовано завдяки read(*, *)).

Лістинг – 3.8 згідно додатку А

```
subroutine casta_appl_todo_entry(descr)
    use casta_scr,only: linestart => casta_scr_linestart,&
                addtext => casta_scr_text
    character(len=*),intent(in) :: descr
    character(len=255) :: text
    call linestart()
    call addtext(text1=descr,align1='r')
    call addtext(text=' ',rep=1)
    call addtext(text='TO DO… ',rep=1)
  end subroutine casta_appl_todo_entry

  subroutine casta_appl_todo_exit(failure)
    use casta_scr,only: linereturn => casta_scr_linereturn,&
                addtext => casta_scr_text
    character(len=*),intent(in),optional :: failure
    if (present(failure)) then
      call addtext(text='FAIL')
      call linereturn()
```

```
        if (failure.ne.'') call casta_appl_message(mes1=failure,mes2='REASON
FOR FAILURE')
      else
        call addtext(text='DONE')
        call linereturn()
      end if
   end subroutine casta_appl_todo_exit

end module casta_appl
```

Останні дві підпрограми мають назви todo_entry та todo_exit і вони потрібні для того щоб користувач бачив що програма працює для цього перед початком завдання вона виводить опис та надпис «TO DO…», а підпрограма todo_exit викликається після завершення завдання і якщо сталась помилка вона пише «FAIL», а якщо все добре «DONE».

3.2.3 Аналіз файлу casta.f90 модуля casta_job

Лістинг – 3.9 згідно додатку А

```
module casta_job

   contains

   subroutine casta_job_entry(name,curdate,curtime,descr)
      use casta_scr,only: addtext => casta_scr_text
      character(len=*),intent(in),optional :: name,curdate,curtime,descr
      call
addtext(text1='',blanc1='=',text2='',blanc2='=',text3='',blanc3='=',newline='yes')
      call addtext(text1='',blanc2=' ',&
                text3='',text2='JOB                                    ENTRY
POINT',align2='r',align1='r',newline='yes')
      if (present(name)) call addtext(text1=name,blanc2='.',align1='r',&
                              text2='NAME',text3='',align2='r',align3='l',newline='yes')
      if (present(descr)) call addtext(text3='',blanc2='.',&

text1=descr,text2='DESCRITPION',align1='r',align2='r',newline='yes')
      if (present(curdate)) call addtext(text1=curdate,blanc2='.',&

text2='DATE',text3='',align2='r',align1='r',newline='yes')
      if (present(curtime)) call addtext(text1=curtime,blanc2='.',&
                               text2='TIME',text3='',align2='r',align1='r',newline='yes')
```

```
      call addtext(text1='',blanc1='=',text2='',blanc2='=',text3='',blanc3='
',newline='yes')
   end subroutine casta_job_entry

subroutine casta_job_exit(res,curdate,curtime,exetime)

      use casta_scr,only: addtext => casta_scr_text
      character(len=*),intent(in),optional :: res,curdate,curtime,exetime
      call addtext(text1='',blanc1='=',text2='',blanc2='=',text3='',blanc3='
',newline='yes')
      if (present(res)) call addtext(text3='',blanc2='.',&

text1=res,text2='RESULT',align1='r',align2='r',newline='yes')
      if (present(curdate)) call addtext(text1=curdate,blanc2='.',&

text2='DATE',text3='',align2='r',align1='r',newline='yes')
      if (present(curtime)) call addtext(text1=curtime,blanc2='.',&
                              text2='TIME',text3='',align2='r',align1='r',newline='yes')
      if (present(exetime)) call addtext(text1=exetime,blanc2='.',&
                              text2='EXECUTION
TIME',text3='',align2='r',align1='r',newline='yes')
      call addtext(text1='',blanc1=' ',&
               text2='JOB EXIT POINT',text3='',align2='r',align3='l',newline='yes')
      call
addtext(text1='',blanc1='=',text2='',blanc2='=',text3='',blanc3='=',newline='yes')

   end subroutine casta_job_exit
```

Як можна побачити по цьому лістингу цей модуль використовує символ дорівнює (=) в якості візуальної відміни від минулого модулю що дозволяє по одному лише погляду зрозуміти який модуль зараз виконується. Перші дві підпрограми які виконуються в цьому модулі це casta_job_entry та casta_job_exit де перший відображає інформацію початку завдання, а друга кінця завдання.

Наступна підпрограма яка має назву casta_job_message потрібно для того щоб була змогу при необхідності додати коментар к тому що зараз буде відбуватись. А останні дві підпрограми casta_job_todo_entry та casta_job_todo_exit потрібні для того що відобразити чи виконався цей етап без помилок чи ні. Для цього casta_job_todo_entry створюю зовнішній вигляд поля,

а casta_job_todo_exit виводить помилку при умові що опціональний аргумент failure передано, а якщо ні виводить стандартне повідомлення «FAIL».

3.2.4 Аналіз файлу casta.f90 модуля casta_task

Лістинг – 3.10 згідно додатку А

```
module casta_task

   private insert_line,insert_par

   contains

   subroutine insert_line(extborder)
      use casta_scr,only: opt => casta_scr_opt,&
                          linestart => casta_scr_linestart,&
                          linereturn => casta_scr_linereturn,&
                          addtext => casta_scr_text
      character(len=*),intent(in),optional :: extborder
      call linestart()
      call addtext(text=':',rep=opt%field1+opt%field2+1)
      if (present(extborder)) then
         if ((extborder(1:1)=='y').or.(extborder(1:1)=='Y')) then
            call addtext(text=' ',rep=1)
            call addtext(text=':',rep=opt%field3)
         end if
      end if
      call linereturn()
   end subroutine insert_line

   subroutine insert_par(parname,parvalue)
      use casta_scr,only: opt => casta_scr_opt,&
                          linestart => casta_scr_linestart,&
                          linereturn => casta_scr_linereturn,&
                          addtext => casta_scr_text
      character(len=*),intent(in) :: parname,parvalue
      call linestart()
      call addtext(text=parvalue,field=opt%field1+opt%field2+1,align='r')
      call addtext(text=' ',rep=1)
      call addtext(text=parname,field=opt%field3,align='l')
      call linereturn()
   end subroutine insert_par
```

На відміну від модуля casta_job цей модуль зосереджується на третьому

стовпці, а також робить дві свої підпрограми приватними, а саме insert_line та insert_par завдяки чому вони доступні лише в цьому модулі. Це пов’язано з тим що цей модуль об’єднує Field1 та Field2 в одне велике поле, а тому було легше створити для цього модуля власні налаштування виведення границь (в цьому модулі для відокремлення границь використовується символ двокрапки)

Інші підпрограми більш менш стандартні це casta_task_entry, casta_task_exit та casta_task_message з невеликою різницею яка полягає в тому де і як відображається текст.

3.2.5 Аналіз файлу casta.f90 модуля casta_action

Лістинг – 3.11 згідно додатку А

```
module casta_action

   private insert_line,insert_par

   contains

   subroutine insert_line(extborder)
      use casta_scr,only: opt => casta_scr_opt,&
                          linestart => casta_scr_linestart,&
                          linereturn => casta_scr_linereturn,&
                          addtext => casta_scr_text
      character(len=*),intent(in),optional :: extborder
      call linestart()
      call addtext(text='*',rep=opt%field1+opt%field2+1)
      if (present(extborder)) then
         if ((extborder(1:1)=='y').or.(extborder(1:1)=='Y')) then
            call addtext(text=' ',rep=1)
            call addtext(text='*',rep=opt%field3)
         end if
      end if
      call linereturn()
   end subroutine insert_line

   subroutine insert_par(parname,parvalue,blanc)
      use casta_scr,only: opt => casta_scr_opt,&
                          linestart => casta_scr_linestart,&
                          linereturn => casta_scr_linereturn,&
```

```
                addtext => casta_scr_text
    character(len=*),intent(in) :: parname,parvalue
    character(len=*),intent(in),optional :: blanc
    call linestart()
    call addtext(text=parvalue,field=opt%field1+opt%field2+1,align='r')
    call addtext(text=' ',rep=1)
    if (present(blanc)) then
       call addtext(text=parname,field=opt%field3,align='r',blanc=blanc)
    else
       call addtext(text=parname,field=opt%field3,align='r')
    end if
    call linereturn()
  end subroutine insert_par
```

Цей модуль також концентрується на заповненні третього поля, а тому також має власні налаштування заповнення (в якості символа для заповнення використовує зірочку), а також на відміну від модулю casta_task підпрограма має необов’язковий аргумент blanc в який передається крапка під час виклику casta_action_entry що дозволяє заповнити зайве місце яка з’явилось через то що в цьому модулі потрібно вирівнювати текст в третьому полі по правому краю.

3.2.6 Аналіз файлу casta.f90 модуля casta_execut

Лістинг – 3.12 згідно додатку А

```
subroutine casta_execut_percent(percent)

    real,intent(in) :: percent
    write(unit=*,fmt=10,advance='no') char(13),percent
    10 format(A,1X,F7.3,1X,'%')

  end subroutine casta_execut_percent
```

Основною задачею цього модуля є підпрограма яку можна побачити в лістингу 3.12 і вона дозволяє створити відсоткову завантаження на одному рядку завдяки чому консоль не буде кожного разу перескакувати. Інші підпрограми в цьому розділі так само відповідають за формування інтерфейсу щоб він виглядав добре.

3.2.7 Аналіз файлу main.f90 програма запуску

У файлі casta.f90 є велика кількість різних функцій і майже всі з них мають значення за замовченням завдяки тому щоб не встановлювати їх кожного разу якщо це не потрібно, але в разі якщо це потрібно їх можна змінити саме в цьому файлі викликав потрібну функцію.

Лістинг коду – 3.13 згідно додатку Б

```
program launch

   use casta_scr
   use casta_appl
   use casta_job
   use casta_task
   use casta_action
   use casta_execut

   implicit none

   integer :: dt_values(8)
   character(len=10) :: sys_date
   character(len=12) :: sys_time

   integer :: i
   character(len=255) :: inp
   character(len=255),dimension(10) :: menuitems

   call casta_scr_setfields(field1=50)

   call date_and_time(VALUES=dt_values)
   write(sys_time, '(I2.2, ":", I2.2, ":", I2.2, ".", I3.3)') dt_values(5), dt_values(6), dt_values(7), dt_values(8)
   write(sys_date, '(I2.2, ".", I2.2, ".", I4.4)') dt_values(3), dt_values(2), dt_values(1)

   call casta_appl_entry(name='project',curdate=sys_date,curtime=sys_time,&
                         comline='',&
                         curwdir='',&
                         descr='it is testing for casta application')
   call casta_appl_message(mes1='mes1',mes2='mes2')
```

```
call casta_appl_todo_entry('testing')
call sleep(3)
call casta_appl_todo_exit()
call casta_appl_todo_entry('testing')
```

Як видно з цього лістингу майже всі функції взяті з файлу casta.f90 але тут є унікальні наприклад sys_date та sys_time які потрібні для того щоб передавати в потрібні місця дату та час системи. Зовнішній вигляд кожного модулю з файлу casta.f90 та загальний вигляд можна побачити на рис. 3.1 – 3.6

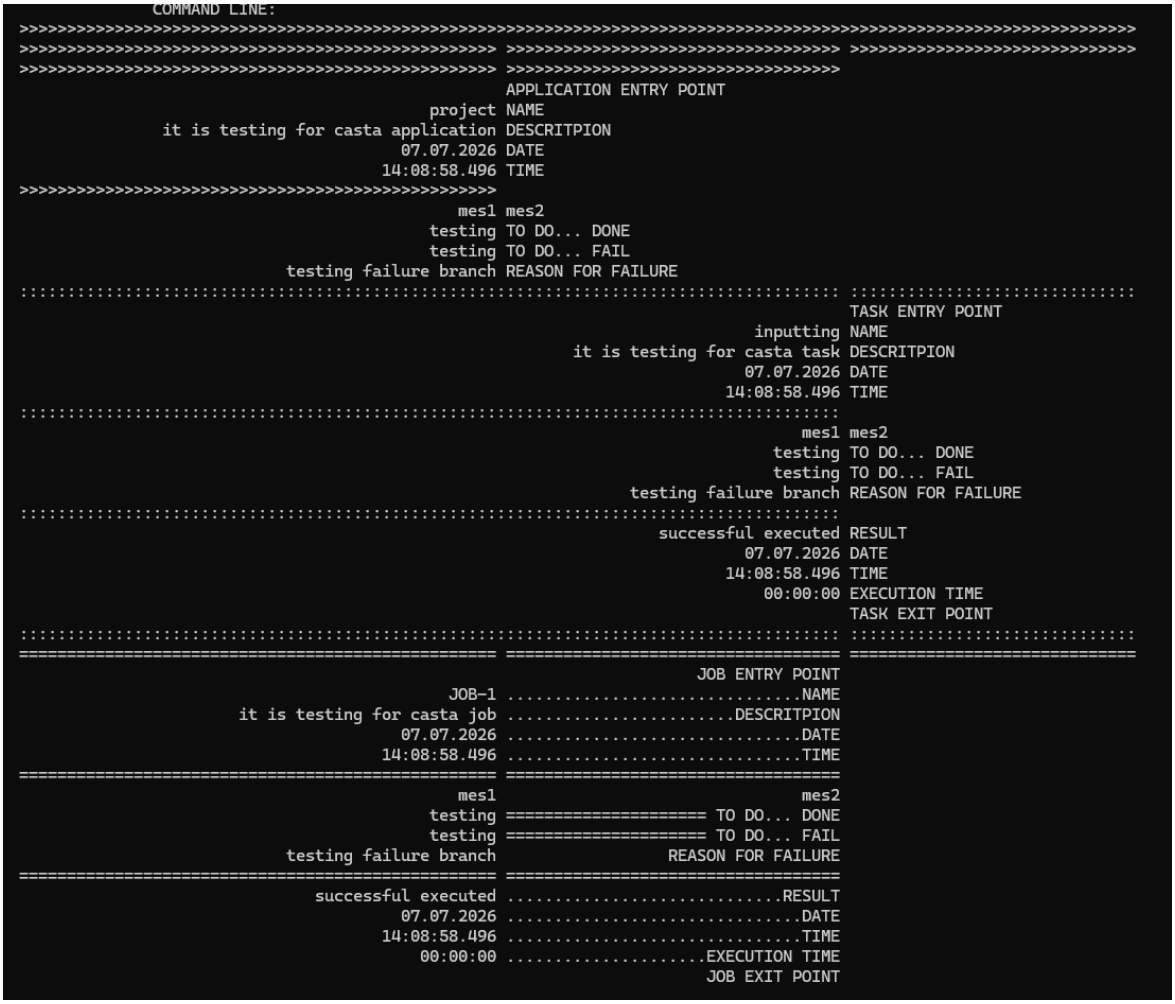


Рисунок 3.1 – Загальний вигляд інтерфейсу

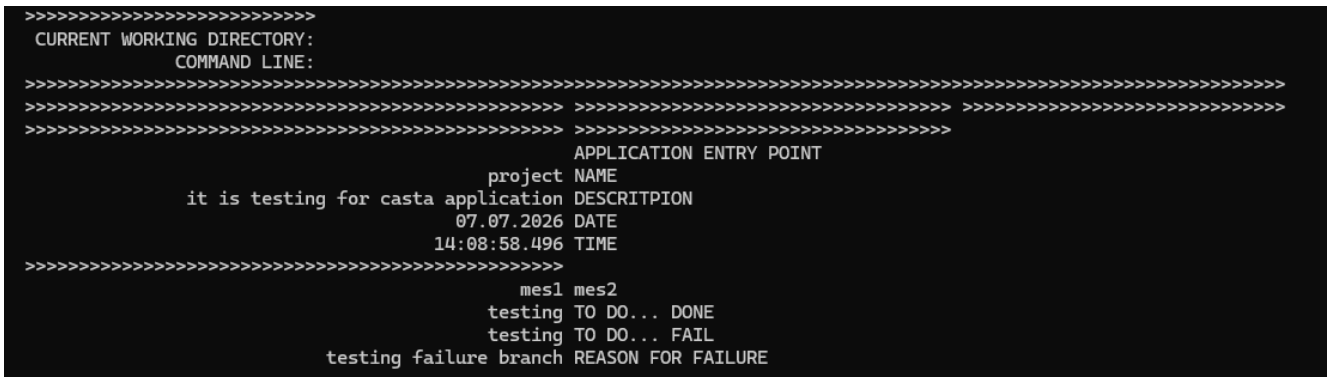


Рисунок 3.2 – Вигляд модуля casta_appl

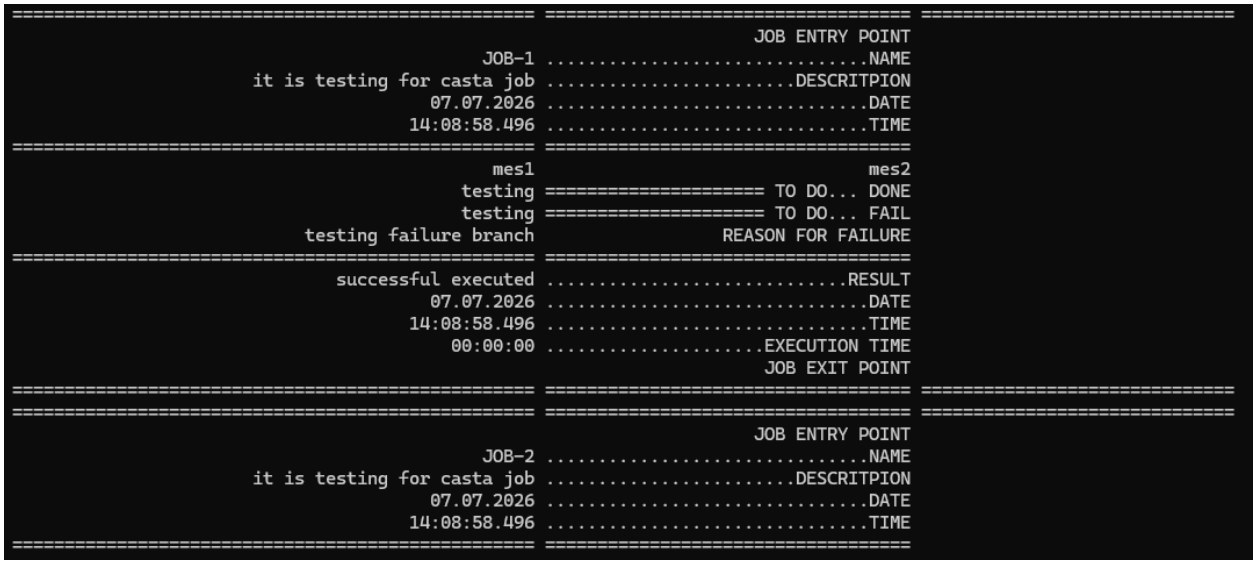


Рисунок 3.3 – Вигляд модуля casta_job

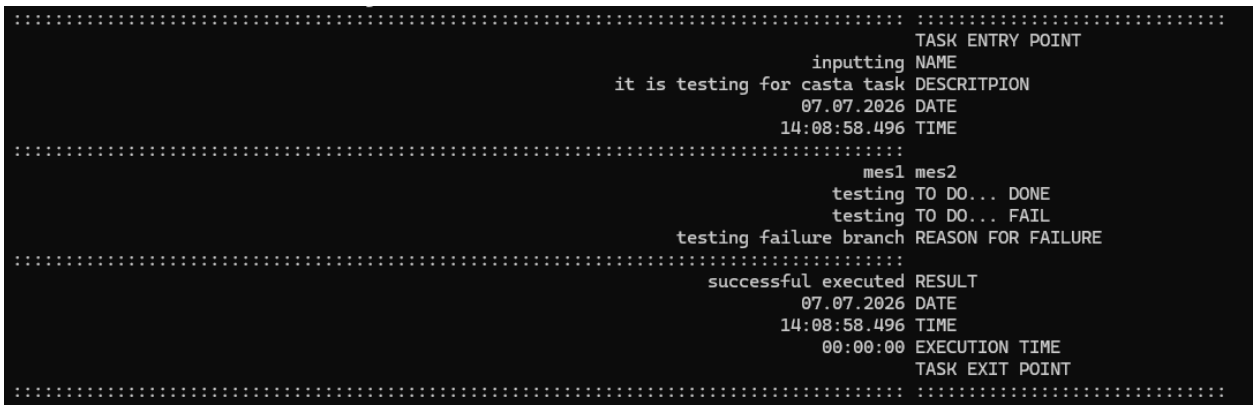


Рисунок 3.4 – Вигляд модуля casta_task

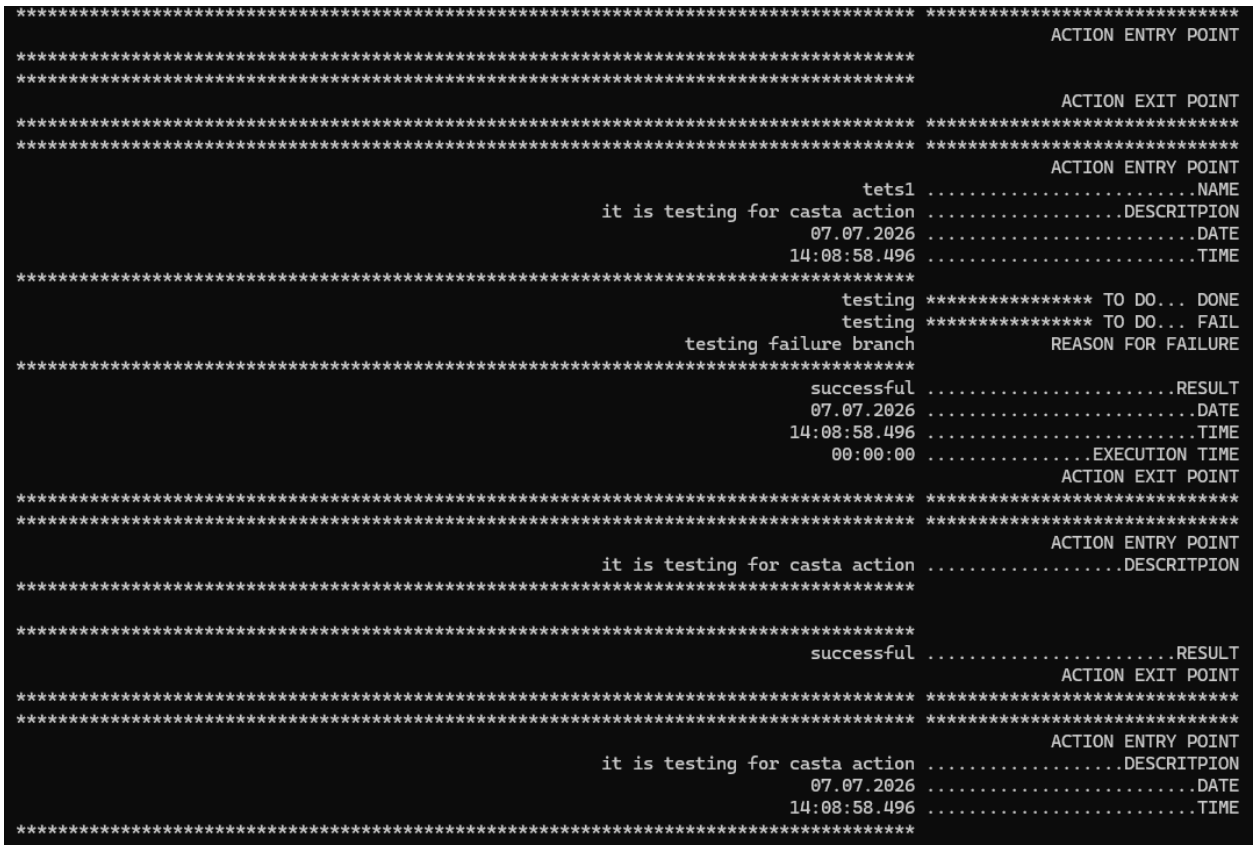


Рисунок 3.5 – Вигляд модуля casta_action

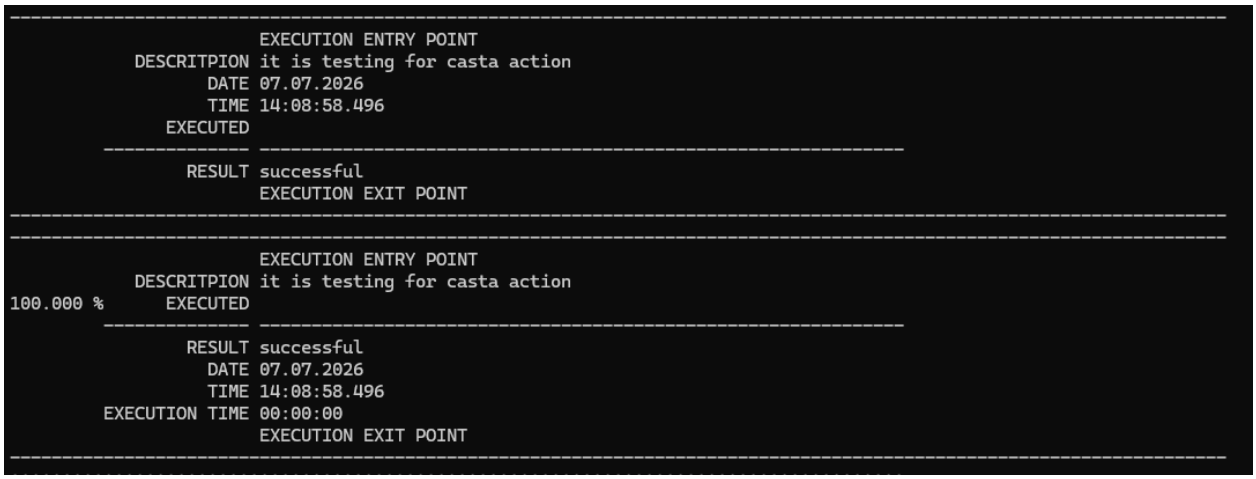


Рисунок 3.6 – Вигляд модуля casta_execut

В програму з цим інтерфейсом можна додати будь яку мат-модель розроблену завдяки Fortran завдяки чому під час запуску можливо буде відстежити час початку та закінчення розрахунку, а також виявити деякі проблеми в мат-моделі якщо вони є.

4 ОХОРОНА ПРАЦІ

Процес розробки програм та написання коду здійснюється на робочому місці і для такої категорії робіт, а саме легкої фізичної праці, що виконується сидячи [13]. Для такої роботи характерні такі психофізіологічні чинники як зорове напруження (через те що потрібно постійно зчитувати велику кількість тексту на екрані), нервово-психічне та розумове перенапруження (через те що під час роботи потрібна висока концентрація уваги під час аналізу тексту та текстових команд, синтаксису, а також тривала інтелектуальна праця викликають втому, зниження швидкості реакції та психоемоційний стрес), гіподинамія та статичні навантаження (через те що під час майже усього робочого дня потрібно перебувати у вимушеній робочій позі за столом це спричиняє статичне напруження м'язів шийно-плечової зона та хребта, а постійні монотонні рухи пальців при роботі з клавіатурою можуть слугувати причиною синдрому зап'ястного каналу).

Також дуже важливим є властивості приміщення в якому працюють оскільки можуть бути порушення мікроклімату, освітлення чи небезпека ураження електричним струмом. Згідно з ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» [14] температура повітря, вологість та швидкість руху повітря в приміщенні в теплий період року повинна бути в діапазоні 22-28°С, 55% вологості та швидкість не більше 0.2 м/сек., а у холодний період року 20-25°С, 70% вологості та швидкість не більше 0.1 м/сек. Що стосується освітлення то згідно з ДБН В.2.5-28-2018 «Природне і штучне освітлення» [12] при роботі з текстовими документами та дисплеями рівень освітленості на робочому столів зоні розміщення документів повинно бути не менше 300-500 люкс, а у самому приміщенні повинна бути застосовуватись комбінована система освітлення (загальна та місцева), а як джерела світла використовуються люмінесцентні або світлодіодні LED лампи з колірною температурою 3500-4500 Кельвінів.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було створено текстовий інтерфейс за допомогою Fortran який буде корисним при використані з більшістю математичних моделей розроблений на цій мові програмування.

В ході виконання роботи аналіз сучасних рішень по створенню інтерфейсів для математичних моделей за допомогою Fortran показав що у відкритому доступі їх майже немає як і самих моделей, що дозволило обґрунтувати доцільність створення власного текстового інтерфейсу який можна буде використовувати та вдосконалювати в подальшому.

Для створення інтерфейсу використовувалось крос-платформне середовище розробки CodeBlocks через його можливість створення програм мовою Fortran що дозволило більш легко відстежувати баги та стабільність створеної програми.

Також, отримані результати можна віднести до Цілі сталого розвитку 9 «Промисловість, інновації та інфраструктура», а саме до пунктів п 9.4 «Сприяти прискореному розвитку високо- та середньо високотехнологічних секторів переробної промисловості, які формуються на основі використання ланцюгів «освіта – наука – виробництво» та кластерного підходу за напрямами: розвиток інноваційної екосистеми; розвиток інформаційно телекомунікаційних технологій (ІКТ); застосування ІКТ в АПК, енергетиці, транспорті та промисловості; високотехнологічне машинобудування; створення нових матеріалів; розвиток фармацевтичної та біоінженерної галузей» та п 9.7 «Забезпечити збільшення участі молоді у наукових дослідженнях».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. Структура та правила оформлення. Введ. 2015-06-22. К. Держстандарт України, 2017. – 29 с.

2. Методичні вказівки з підготовки кваліфікаційної роботи для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології освітньої програми «Системна інженерія» / Упоряд.: І.Ш. Невлюдов, О.М. Цимбал, О.В. Токарєва, А.І. Бронніков. Харків: ХНУРЕ, 2023. 65 с.

3. Інформація про використання різних інтерфейсів від професіоналів Режим доступу: https://labs.quansight.org/blog/building-scipy-with-flang Дата використання 19.04.2026

4. MATLAB : official product page. MathWorks. URL: https://www.mathworks.com/products/matlab.html (accessed: 03.07.2026).

5. Positron : Jupyter Notebooks integration page. URL: https://positron.posit.co/jupyter-notebooks.html (accessed: 03.07.2026).

6. Wolfram Research : official website. URL: https://www.wolfram.com/?source=nav (accessed: 03.07.2026).

7. Відгуки на кшталт використання Fortran Режим доступу: https://www.reddit.com/r/fortran/comments/1bmn0xj/most_popular_programming_languages_listing/?tl= Дата використання 19.04.2026

8. Про GIL Режим доступу: https://uk.wikipedia.org/wiki/Глобальне_блокування_інтерпретатора Дата використання 19.04.2026

9. Фортран : матеріали з Вікіпедії — вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Фортран (дата звернення: 03.07.2026).

10. TIOBE Index : TIOBE Index for July 2026. URL: https://www.tiobe.com/tiobe-index/ (accessed: 03.07.2026).

11. Fortran Packages : web portal. URL: https://fortran-lang.org/packages/ (accessed: 03.07.2026).

12. Jorgensen E. Introduction to Programming using Fortran 95/2003/2008. Open Textbook Library, 2018. 498 p.

13. Державні будівельні норми України. Природне і штучне освітлення : ДБН В.2.5-28:2018. URL: https://e-construction.gov.ua/laws_detail/3074958732556240833?doc_type=2 (дата звернення: 03.07.2026).

14. Про охорону праці : Закон України від 14.10.1992 № 2694-XII. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення: 03.07.2026).

15. Санітарні правила і норми по оптимізації світлового середовища і утриманню освітлювальних установок промислових підприємств : СанПіН № 2282-99 / Наказ Міністерства охорони здоров'я України від 08.06.1999. URL: https://zakon.rada.gov.ua/rada/show/va042282-99#Text (дата звернення: 03.07.2026).