Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня вищої освіти магістра
за спеціальністю 121 – Інженерія програмного забезпечення

На тему: Дослідження сучасних програмних методів вдосконалення та модифікацій класичного ПІД-регулятора

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-23-2м _____ /П. А. Ткач /

Керівник кваліфікаційної роботи _____ / _____ /

Економіко-організаційна частина _____ / _____ /

Нормоконтроль _____ / _____ /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг
2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____ А. М. Стрюк
«___» _____ 20__ р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-23-2м Ткачу Павлу Андрійовичу

1. Тема: Дослідження сучасних програмних методів вдосконалення та модифікацій класичного ПІД-регулятора затверджено наказом по КНУ № __ від «__»_____20__ р.
2. Термін подання студентом закінченої роботи: «__» ____20___ р.
3. Вихідні дані по роботі: розроблювана програмна модель системи автоматичного регулювання повинна мати коректне відтворення всіх фізичних процесів протягом комп'ютерного моделювання та мати відповідні автоматизовані налаштовувачі регулятора як елемента системи.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): виконати аналіз існуючих та потенціально можливих методів розв'язання задачі, розробити математичну модель системи автоматичного регулювання, спроектувати програмну реалізацію математичної моделі, розробити налаштовувачі регулятора системи різниз за методами, поставити експерименти, провести оцінку результатів експериментів, виконати аналіз економічної ефективності методів використовуваних налаштовувачами регулятора, зробити висновки.
5. Перелік ілюстративного матеріалу: структурні схеми систем автоматичного регулювання та регулятора, UML-схеми розроблених ієрархій класів та їх взаємодій, таблиці результатів оцінки експериментальних даних та рейтингування методів налаштовувачів за ефективністю застосувань.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури за тематикою роботи	24.02.2024 – 01.03.2024
2	Проведення аналізу існуючих теоретичних методів дослідження і вирішення проблеми	02.03.2024 – 05.03.2024
3	Проведення критичного порівняльного аналізу існуючих аналогів програмних методів та пошук додаткових методів.	06.03.2024 – 10.04.2024
4	Формулювання актуальності і завдань роботи	11.04.2024 – 15.05.2024
5	Оформлення матеріалів першого розділу роботи	16.05.2024 – 20.05.2024
6	Розробка математичної моделі системи автоматичного регулювання на базі електроприводу.	21.05.2024 – 16.06.2024
7	Розробка програмної моделі, що відтворяє вже розроблену математичну модель, розробка основних класів для відтворення роботи системи.	17.06.2024 – 14.07.2024
8	Розробка класа-фасаду та службових класів для інкапсюлювання алгоритма запуску моделювання роботи системи та алгоритмів фіксації результатів моделювання.	15.07.2024 – 05.08.2024
9	Розробка шаблоних класів для реалізації генетичного алгоритму та градієнтного методу для пошукку відповіді будь якої розмірності.	06.08.2024 – 10.09.2024
10	Розробка класа-адаптера налаштовувача регулятора, що використовує алгоритми, реалізації котрих попередньо були визначені в шаблонних класах.	11.09.2024 – 20.09.2024
11	Розробка класа, що інкапсулює алгоритм аналізу коливань регульованої величини та розширення налаштовувачів регулятора на ще один тип, що використовує алгоритм аналізу коливань аналізованої величини протягом застосування метода Циглера-Ніколса.	19.09.2024 – 29.09.2024

12	Оформлення матеріалів другого розділу роботи	30.09.2024 – 14.10.2024
13	Постановка експериментів з налаштовування регулятора системи за різних конфігурацій налаштовувачів та розробка підходу до оцінювання чисельно отриманих експериментальних даних через певні метрики.	15.10.2024 – 01.11.2024
14	Проведення експериментів, отримання експериментальних даних, таких як налаштувань регулятора на кожен проведений експеримент.	02.11.2024 – 05.11.2024
15	Проведення багаточисельних запусків моделі з експериментально отриманими налаштуваннями регулятора за різних умов роботи системи. Та складання рейтингів за отримуваними метриками на кожен випадок експериментально ориманих налаштувань.	06.11.2024 – 15.11.2024
16	Аналіз економічної ефективності та перспектив подібних інновацій та складання рейтингу налаштовувачів за єфективністю проведення налаштування регулятора.	16.11.2024 – 17.11.2024
17	Оформлення матеріалів третього розділу роботи.	18.11.2024 – 21.11.2024
18	Підведення висновків по виконаній роботі.	22.11.2024 – 24.11.2024
19	Остаточне оформлення пояснювальної записки	24.11.2024 – 03.12.2024

РЕФЕРАТ

ПРОПОРЦІЙНО-ІНТЕГРАЛЬНО-ДИФЕРЕНЦІАЛЬНИЙ РЕГУЛЯТОР, СИСТЕМА АВТОМАТИЗОВАНОГО РЕГУЛЮВАННЯ, НАЛАШТУВАННЯ, НАЛАШТОВУВАЧ РЕГУЛЯТОРА, АЛГОРИТМ НАЛАШТОВУВАЧА, МЕТОД НАЛАШТУВАННЯ, ЕКСПЕРИМЕНТ, ПРОГРАМНЕ МОДЕЛЮВАННЯ

Пояснювальна записка: 75 стр., 27 табл., 7 рис., 3 дод., 15 джерел.

Об’єктом дослідження цієї роботи – автоматизація налаштування пропорційно-інтегрально-диференціального регулятора (далі ПІД-регулятора). Предмет дослідження – можливі програмні методи вдосконалення ПІД-регулятора.

В ході виконання кваліфікаційної роботи задля досягнення вичерпних результатів дослідження було поглиблено вивчено регулювання за алгоритмами пропорційно-інтегрально-диференціального регулятора ПІД-регулятора, системи автоматизованого регулювання (далі САР) за участю цього регулятора, було вивчено тонкощі, що виникають під час його експлуатації не тільки з теоретичної точки зору а й з практичного сторони, було вивчено проблеми його експлуатації та все вище зазначене детально було звітовано в пояснювальній записці.

В процесі пошуку рішення було розглянуто чи мало можливих рішень в просторі можливостей програмного забезпечення. Пошук починався посеред традиційних методів застосовуваних в реаліях експлуатації для зменшення впливу негативних властивостей регулятора та закінчився пошуком у просторі популярних методів використовуваних для вирішення проблем машинного навчання як потенційне рішення або як конкурентоспроможна альтернатива з огляду її відносної новітньості. Потенційне рішення могло мати будь яку основу для себе, чи буть то чисто традиційне, чи методи застосовувані в машинному навчанні, може їх поєднання, чи їх рейтингування, або виведення

закономірностей вигідності їх застосування у порівнянні одне іншим. Тобто поле для пошуку рішень було дійсно широким.

В процесі побудови математичної моделі САР на базі абстракцій реального обладнання, було вивчено яким чином виконати моделювання електродвигуна постійного струму з обмоткою незалежного збудження та реалізовано два різних за ступенем точності розрахунків чисельних методів вирішення системи диференціальних рівнянь для коректного моделювання поведінки процесу визначеного в контексті САР як регульованого. Розроблена математична модель САР цілком була задовільнена в своїх потребах за допомогою абстракції двигуна у ролі об'єкта керування.

Виконано проектування програмного забезпечення за парадигмою об'єктно-орієнтовного програмування на мові C++, що реалізувало програмну реалізацію математичної моделі, засоби фіксації результатів моделювання.

Як додаткові програмні модулі було розроблено налаштовувачі регулятора, що використовували принципіально різні методи налаштовування регулятора в автоматичному режимі призначені для рішення проблеми визначеної в цій роботі.

Оцінка проводилася шляхом багаторазового проведення експериментів з різними умовами пусків, та отримані результати налаштувань перевірялись також за допомогою багаторазових пусків в різних умовах моделі САР з відповідними налаштуваннями та визначенням фітнес-функції кожного пуску, що слугувала як певна метрика якості налаштування.

В результаті проведених досліджень було виведено цілу низку стверджень про потенціал кожного з досліджуваних напрямків вирішення проблеми, про можливості подальших досліджень та зроблене вичерпне заключення щодо способу вирішення розглядуваної проблеми.

ABSTRACT

PROPORTIONAL–INTEGRAL–DERIVATIVE CONTROLLER, AUTOMATIC CONTROL SYSTEM, ADJUSTMENT, REGULATOR ADJUSTER, ADJUSTER ALGORITHM, ADJUSTMENT METHOD, EXPERIMENT, SOFTWARE SIMULATION

Explanatory note: 80 pages, 27 tables, 7 figures, 3 appendices, 15 sources.

The object of research of this work is the automation of the setting of the proportional-integral-differential controller (hereinafter PID controller). The subject of research is possible software methods for improving the PID controller.

In the course of the qualification work, in order to achieve comprehensive research results, the regulation according to the algorithms of the proportional-integral-differential regulator of the PID controller, the automatic control system (hereinafter ACS) with the participation of this regulator was studied in depth, the subtleties that arise during its operation were studied not only from a theoretical point of view and from a practical point of view, the problems of its operation were studied and all of the above was reported in detail in the explanatory note.

In the process of finding a solution, it was considered whether there were few possible solutions in the space of software capabilities. The search began in the midst of traditional methods used in the realities of operation to reduce the influence of the negative properties of the regulator and ended with a search in the space of popular methods used to solve machine learning problems as a potential solution or as a competitive alternative in view of its relative novelty. A potential solution could have any basis for itself, be it purely traditional, or methods used in machine learning, or a combination of them, or their ranking, or derivation of laws of the benefit of their application in comparison with each other. That is, the field for finding solutions was really wide.

In the process of building a mathematical model of the ACS based on abstractions of real equipment, it was studied how to model a direct current electric motor with an independent excitation winding, and two different numerical methods

of solving the system of differential equations with different degrees of accuracy were implemented for the correct modeling of the behavior of the process defined in the context of the ACS as regulated . The developed mathematical model of the ACS was fully satisfied in its needs by abstracting the engine as a control object.

The software was designed according to the paradigm of object-oriented programming in the C++ language, which implemented the software implementation of the mathematical model, the means of fixing the modeling results.

Regulator adjusters were developed as additional software modules that used fundamentally different methods of adjusting the regulator in automatic mode, designed to solve the problem defined in this work.

The evaluation was carried out by repeatedly conducting experiments with different start-up conditions, and the obtained settings results were also checked using multiple starts in different conditions of the ACS model with appropriate settings and determination of the fitness function of each start-up, which served as a certain metric of the setting quality.

As a result of the conducted research, a number of statements were made about the potential of each of the researched directions for solving the problem, about the possibilities of further research, and an exhaustive conclusion was made about the way to solve the problem under consideration

ЗМІСТ

ВСТУП

Пропорційно-інтегрально-диференціальний регулятор (далі ПІД-регулятор) був винайдений ще за часів першої половини 20 століття, а саме у 1910. Найперший метод його налаштування у 1942 році Джоном Зіглером і Натаніелем Ніколсом та отримав назву “метод Ціглера-Нікольса”. Частота публікацій наукових робіт про регулятор яскраво демонструє нам зріст його популярності:

— 14 публікацій – 1973-1982 рр. (9 років);

— 111 публікацій – 1983-1992 рр. (9 років);

— 225 публікацій – 1992-2002 рр. (лише 4 роки).

Невелика вартість мікропроцесорів та дешевих аналого-цифрових перетворювачів в ПІД-регуляторах спростили впровадження цей тип регулятора, що у свою чергу спричинили стрибок у розробці різноманітних алгоритмів налаштування його параметрів від адаптивних алгоритмів, нейронних мереж, методів нечіткої логіки до генетичних алгоритмів, а також спричинило появу різноманітних ускладнень самого регулятора, з’явилися поняття регуляторі двох ступенів свобод, принципів розімкненого керування та в результаті ПІД-регуляторів багатоконтурних на сьогодні 34%, та 64% одноконтурих [1].

Стрімкий зріст ускладнень як алгоритмів налаштування та алгоритмів власне самого регулятора вимагають певної кваліфікації від робочого персоналу, інженерів автоматизації систем керування технологічними процесами (далі АСКТП), та чи малої їх кількості задля задоволення попиту в персоналі, що має відповідні навички для виконання налаштувань того чи іншого ПІД-регулятора або системи ПІД-регуляторів у багаточисельних окремих випадках особливих умов системи керування та динамічних характеристиках керованого процесу або об’єкту регулювання. Подібне положення справ наштовхує на припущення, що подібні потреби збільшують економічне навантаження на будь яку особу, що планує якісне використання

цієї та їй суміжних технологій задля того щоб залишатися конкурентоспроможним у сьогоденних реаліях. Проте для конкурентоспроможності будь який виробник має не тільки досягати високого рівня якості, але й низької собівартості створюваного продукту. Отже, після впровадження регуляторів для підвищення якості технологічного процесу та утворюваного продукту було б доцільним знайти рішення для вирішення проблеми збільшення витрат на персонал, та в наслідок зменшити собівартість продукції. Існує вже традиційний варіант вирішення такої проблеми через впровадження автоматизації в певні сфери праці людини. У даному випадку цей метод є також доречним. Отже, головне питання полягає у вирішенні більш фундаментальної проблеми на рівні самої технології, що полягає у складнощах налаштування простого за конструкцією регулятора та у складнощах автоматизування цього процесу.

Вирішення цієї проблеми надаватиме можливість вирішити проблему зростання економічного навантаження. Отже, ця робота буде направлена на вирішення фундаментальної проблеми на рівні технології, що потенційно може принести можливість заощаджувати.

Об’єктом дослідження цієї роботи – автоматизація налаштування регулятора, а предмет дослідження – можливі програмні методи вдосконалення ПІД-регулятора.

Головним методом дослідження сучасних програмних методів вдосконалення та модифікацій класичного ПІД-регулятора виступатиме проведення багаторазових експериментів на попередньо розробленій математичній моделі.

ПІД-регулятор є регулятором, котрий завдяки своїм універсальним обчислювальним алгоритмам може бути використовуваним у будь якому технологічному процесі, що потребує автоматизації регулювання. Тобто регулятор є універсальним та простим за своєю реалізацію та впровадженням, що є беззаперечною його перевагою. Проте також існує фундаментальна проблема, що пов'язана з експлуатацією, та полягає у складному процесі його

налаштування, а саме в підборі коефіцієнтів для кожної складової обчислювального алгоритму для оптимального співвідношення їх впливу на керуючу величину для отримання оптимального перехідного процесу за часом, точністю та стійкістю. Ця проблема і породжує потребу у додатковому персоналі кваліфікованих для виконання подібних налаштувань, тому вона як першопричина і вимагає уваги до свого розв’язання.

Проблема виникає з протиріччя ідеальних умов, за яких мало б виконуватися налаштування регулятора та умов, що переважно є занадто далекими від ідеального випадку. В ідеальному випадку налаштування регулятора має виконуватися за умови коли є відомими динамічні характеристики об'єкта керування або керованого процесу чи коли є відомим діапазон їх змін, проте в реальному випадку його застосування це неможливо через те, що навіть у випадку якщо динамічні характеристики керованого об'єкта або процесу визначити, то вони не будуть залишатися постійними протягом часу і навіть діапазон зміни динамічних характеристик об'єкта або процесу може бути змінним у часі. Така властивість об'єктів та процесів керування спричиняє складнощі в підтримці керування на задовільному рівні згідно певних характеристик.

Тобто через протиріччя умов ідеальних і реальних (не ідеальних) умов експлуатації виникає проблема експлуатації, що полягає в налаштуванні регулятора[13].

Головним методом дослідження – багаторазові експерименти під час комп’ютерного моделювання на базі програмної реалізації математичної моделі та застосовуваних до неї регуляторів. Попередньо була проведена розробка математичної моделі системи автоматичного регулювання (далі САР) та розроблено кардинально різні за своїми алгоритмами та властивостями налаштовувачі регулятора, котрий виступатиме частиною тієї самої розробленої математичної моделі САР. Порівняння результатів різних алгоритмів налаштовувачів дозволяють нам порівняти перспективність різних напрямків можливого розвитку налаштовувачів та в цілому оцінити

перспективність власне цього напряму досліджень через визначення чи є можливим взагалі замінити спеціаліста в цій справі відповідним програмним забезпеченням.

РОЗДІЛ 1 РЕГУЛЯТОР ЯК ЧАСТИНА МАТЕМАТИЧНОЇ МОДЕЛІ САР

1.1 Принцип роботи під-регулятора

Вигляд структурної схеми простої системи автоматичного регулювання (далі САР) зі зворотнім зв'язком зображена на рис 1.1.

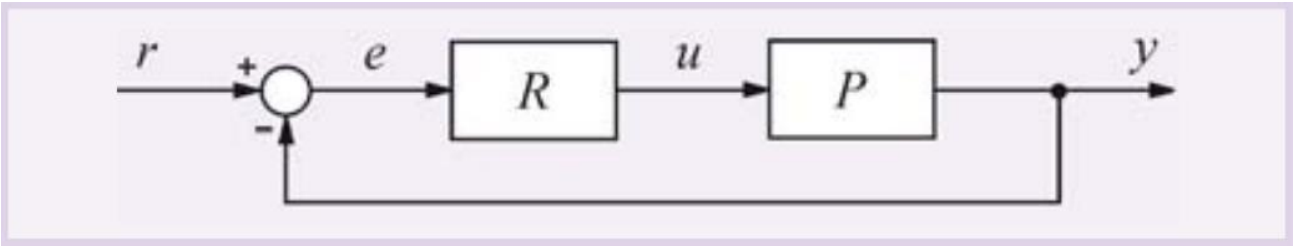

Рисунок 1.1 – Структурна схема САР зі зворотнім зв'язком [1]

R – регулятор;

P – керований процес / об'єкт регулювання;

r – сигнал завдання (потрібне значення регульованої величини);

e – сигнал «помилка керування»;

u – сигнал керування;

y – значення регульованої величини [1].

Визначення сигналу керування має такі форми:

$$u(t) = K \cdot e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d\frac{de(t)}{dt} \quad (1.1)$$

$$u(t) = K_0\left(e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d\frac{de(t)}{dt}\right) \quad (1.2)$$

$$u(t) = K_p e(t) + K_i\int_0^t e(t)dt + K_d\frac{de(t)}{dt} \quad (1.3)$$

Знайдена література зазвичай використовує усі три форми запису як в виразах 1.1, 1.2, та 1.3, і було б важливо продемонструвати їх усі та зазначити, що в цій роботі, в програмі, використовуватиметься саме остання форма за

номером 1.3 через її зручність реалізації в програмній формі. Різниці між вибором принципової немає. Налаштування проходить завжди через три коефіцієнта. Від обраної форми залежатимуть лише числові значення при одному і тому ж стані налаштувань регулятора. Тобто коли перехідна характеристика однакова, то обрана форма розрахунків впливатиме лише на числові значення коефіцієнтів[2].

Згідно до обраної форми фрагмент пов'язаний з блоком регулятора R попередньої структурної схеми можна деталізувати наступним чином як на рис 1.2.

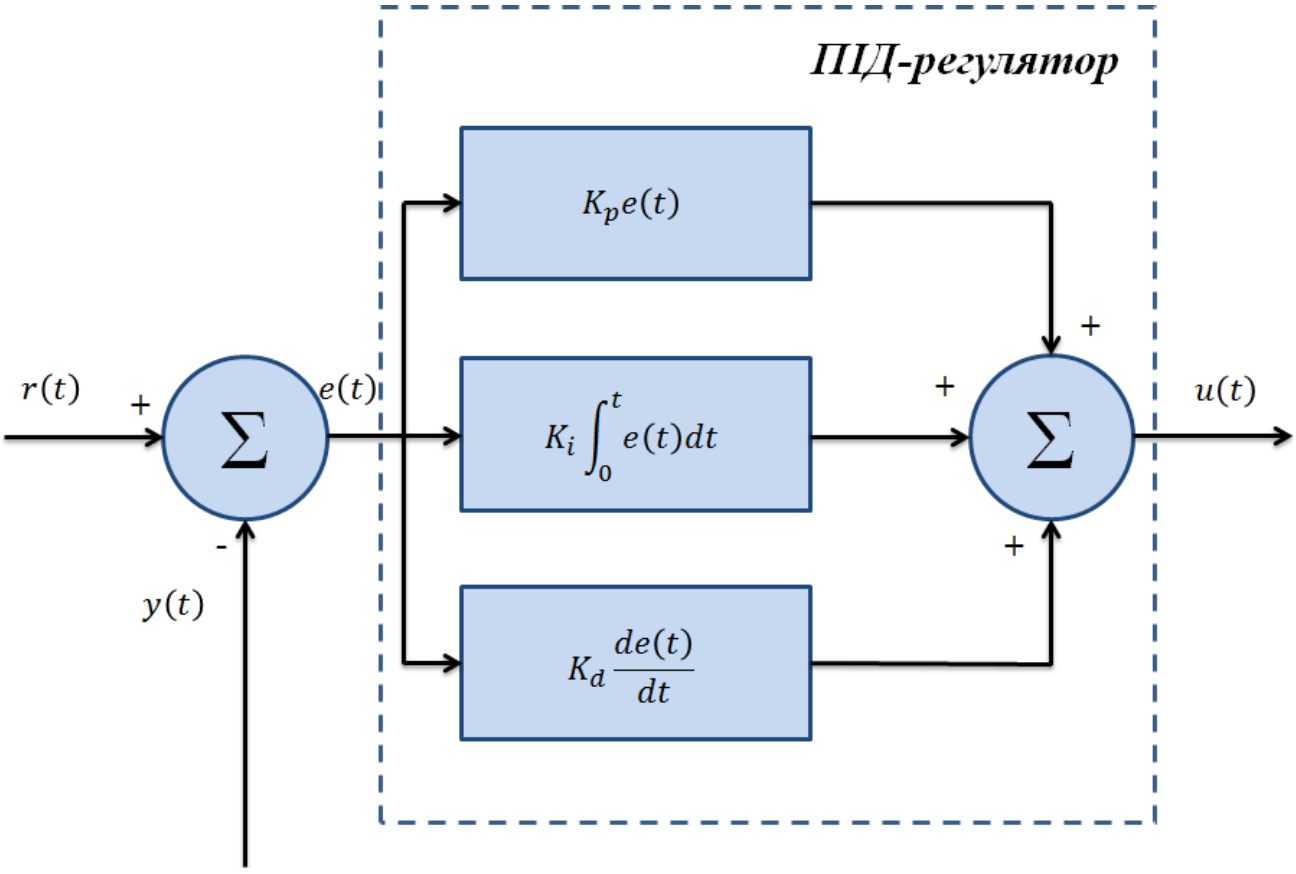

Рисунок 1.2 – Структурна схема ПІД-регулятора детально [3]

Об'єкт дослідження визначається поняттям «Налаштування регулятора», котре іменує процес підбору оптимальних коефіцієнтів K_p, K_i, K_d для задовільного характеру перехідних характеристик під час виконання підтримання значень регульованої величини на рівні зі значенням сигналу

завдання. Кожен випадок налаштування є індивідуальним, бо визначається динамічними характеристиками об’єкту керування (керованого процесу) або діапазоном їх змін, властивостей та особливостей виконуючих механізмів та власне і самим сенсором, характер фіксації значень керованої величини має значний вплив на процес регулювання, тому як саме від його точності, дискретності та подібним характеристик фіксацій залежить розрахована помилка регулювання і утворюється різною за значенням та іншими параметрами, що у свою чергу вимагає індивідуальних налаштувань. Отже, складність цього процесу визначається саме цими деталями і має бути знайденням рішення як його автоматизувати наскільки це можливо та наскільки якісно в цілому [9][10].

1.2 Проектування математичної моделі

Проводитись вони будуть під час комп’ютерного моделювання на базі програмної реалізації математичної моделі.

Тому як ПІД-регулятор є частиною САР, то будь які його програмні модифікації можуть бути проаналізовані лише за умови створення математичної моделі, що відтворюватиме усю систему цілком.

Система має складатися як мінімум з трьох основних елементів:

- джерело енергії;
- регулятор;
- керований об’єкт / регульований процес.

Задля більш наближеного до реального випадку системи та задля уникання занадто високого абстрагування було вирішено обрати певний тип подібного роду системи.

Отже, було обрано реалізовувати САР двигуном постійного струму незалежного збудження обмоток за значенням швидкості обертів або результуючим кутом (положенням) його валу.

Такий тип вимагає наявність в нашій математичній моделі сім елементів:

- джерело постійної напруги (як джерело енергії для функціювання системи);
- перетворювач електроенергії (як виконавчий механізм);
- двигун постійного струму певного типу (оберти якого є регульованим процесом), що є перетворювачем електричної форми енергії до механічної. (технічні характеристики якого завдаватимуть динамічні характеристики системи електричного та механічного походження);
- механічне навантаження на валу двигуна (тип якого завдаватиме динамічні характеристики системи механічного походження);
- вплив зовнішнього середовища на систему (завдає певний вплив на механічну складову динаміки системи);
- тахогенератор або енкодер (аналоговий або цифровий сенсор для фіксації значень регульованої величини відповідного процесу, що в конкретному випадку є швидкістю);
- ПІД-регулятор (як контролер або мікроконтролер у якості регулятора, що відтворює алгоритми ПІД-регулювання на базі отримуваної інформації за допомогою сенсорів та виконуючий регулювання за допомогою виконавчого механізму).

Задля практичності проектованої математичної моделі, центральним елементом для дослідження котрої є регулятор, можна виконати деякі спрощення через приведення деяких елементів до одного об’єкту, що відтворюватиме ті ж самі обчислення без нехтування впливу кожного з семи елементів але з основним акцентом на сам регулятор.

Приведення семи елементів обраного типу системи до їх меншої кількості можна виконати за наступним алгоритмом:

1) Визначити суміжні елементи до головного — регулятора.
2) Усі суміжні елементи до визначених вище наведеним пунктом, що напряму не впливають на регулятор, об’єднати в один об’єкт-

елемент САР під назвою відповідного суміжного до регулятора елемента.

Алгоритм приведення елементів був успішно застосований тому як регулятор в нашому конкретному випадку взаємодіє з кожною групою елементів через лише два елементи: сенсор та виконавчий механізм, які легко об’єднати

Тоді ми отримаємо наступне розподілення елементів по таким групам:

а) джерело енергії:
   1) джерело постійної напруги (як джерело енергії для функціювання системи)
   2) перетворювач електроенергії (як виконавчий механізм)

б) керований об’єкт / регульований процес:
   1) двигун постійного струму певного типу (оберти або результуючий кут положення його валу якого є регульованим вихідним значенням зазначеного процесу як регульованого), що є перетворювачем електричної форми енергії до механічної. (технічні характеристики якого завдаватимуть динамічні характеристики системи електричного та механічного походження)
   2) механічне навантаження на валу двигуна (тип якого завдаватиме динамічні характеристики системи механічного походження).
   3) вплив зовнішнього середовища на систему (завдає певний вплив на механічну складову динаміки системи)
   4) тахогенератор або енкодер (сенсор)

в) регулятор:
   1) ПІД-регулятор.

Обрана структура математичної моделі має таку відзначну перевагу як можливість враховувати вплив зовнішнього середовища за допомогою одного з приведених елементів системи[11].

1.2.1 Джерело енергії як елемент математичної моделі

Простіший для математичного опису елемент тому як єдина його функція — це виставити ліміти (мінімальні та максимальні значення) для вхідного сигналу на керований процес як продукт обробки (за допомогою цього елементу) вихідного сигналу регулятора, що не має обмежень для своїх можливих значень.

1.2.2 Регульований процес як елемент математичної моделі

Найскладніший елемент для математичного опису через громіздкі системи диференційних рівнянь та необхідність звести декілька елементів без нехтування до цього єдиного.

Двигун постійного струму незалежного збудження описується наступною системою диференційних рівнянь (див. вираз 1.4):

$$\begin{cases} U_{\text{я}} = kF \cdot \omega + I_{\text{я}} R_{\text{я}} + L_{\text{я}} \frac{dI_{\text{я}}}{dt} \\ M = kF \cdot I_{\text{я}} \\ M - M_{\text{С}} = J_{\Sigma} \frac{d\omega}{dt} \end{cases} \quad (1.4)$$

де $U_{\text{я}}$ — напруга на якорі двигуна;

kF — коефієнт відношення утворюваного моменту на валу двигуна до струму якоря;

ω — кутова швидкість обертів валу двигуна;

$I_{\text{я}}$ — струм якоря;

$R_{\text{я}}$ — активний опір якоря двигуна;

$L_{\text{я}}$ — індуктивність обмотки якоря двигуна;

M — утворюваний момент на валу двигуна;

M_C — момент навантаження прикладений до валу двигуна;

J_Σ — сумма енерції ротора та навантаження.

Два поєднаних в один елемент за відсутності різниці одного між іншим у випадку математичного опису можуть легко бути описані наступним рівнянням, що й було обрано для подальшого інтегрування в спільну систему рівнянь (див. вирази 1.5 та 1.6):

$$M_C(\omega) = \begin{cases} k_0 + k_1\omega + k_{lim}(1 - e^{-\omega \cdot k_{curv}}), \ \omega > 0 \\ k_0 + k_1\omega + k_{lim}(-1 + e^{\omega \cdot k_{curv}}), \ \omega \leq 0 \end{cases} \quad (1.5)$$

тоді:

$$\frac{dM_C}{dt}(\omega) = \begin{cases} k_1 + k_{curv} \cdot k_{lim} \cdot e^{-\omega \cdot k_{curv}}, \ \omega > 0 \\ k_1 + k_{curv} \cdot k_{lim} \cdot e^{\omega \cdot k_{curv}}, \ \omega \leq 0 \end{cases} \quad (1.6)$$

де k_0 — незалежна від швидкості, константна складова значення момента навантаження на валу двигуна.

$k_1\omega$ — прямопропорційна швидкості складова значення момента навантаження на валу двигуна згідно до вентиляторного типу навантаження.

k_{lim} та k_{curv} — коефіцієнти за допомогою яких симулюється сила тертя за використанням експонентної функції задля уникання розриву функції через особливості Рунге-Кута метода, що не працює за наявності розривів функції. Перший коефіцієнт завдає максимальне значення опору, а другий визначає швидкість зростання цієї складової до свого ліміта в залежності від швидкості обертів валу двигуна.

Тип механічної характеристики навантаження передбачається типу навантаження вентилятора, що використовується у випадку застосування електроприводу в системах вентиляційних, насосах, компресорах, турбінах.

Тахогенератор або енкодер (сенсор) може виконувати деякі перетворення вихідної величини, такі як дискретування (для імітації використовування енкодера) або спотворення сигналу (для імітації завад аналогового сигналу) для детального дослідження їх впливу на якість регулювання, але це не є однією з задач цієї роботи, а тому нехтуватимемо цими подробицями вважаючи, що в нас встановлено аналоговий прилад (тахогенератор) з мінімальними спотворенням сигналу, що наближаються до нескінченно малих значень, що дозволятиме нам їх вважати нульовими. А тому цей пункт не привноситиме нічого до спільної системи рівнянь[5].

Отже, в результаті маємо наступну систему диференційних рівнянь (див. вираз 1.7), що описують даний елемент із задовільною нашим потребам точністю:

$$\begin{cases} \begin{cases} U_{\text{я}} = kF \cdot \omega + I_{\text{я}} R_{\text{я}} + L_{\text{я}} \frac{dI_{\text{я}}}{dt} \\ M = kF \cdot I_{\text{я}} \\ M - M_{\text{C}} = J_{\Sigma} \frac{d\omega}{dt} \end{cases} \\ M_{\text{C}}(\omega) = \begin{cases} k_0 + k_1\omega + k_{lim}(1 - e^{-\omega \cdot k_{curv}}),\ \omega > 0 \\ k_0 + k_1\omega + k_{lim}(-1 + e^{\omega \cdot k_{curv}}),\ \omega \leq 0 \end{cases} \end{cases} \tag{1.7}$$

1.2.3 Регулятор як елемент математичної моделі

Як вже було вище зазначено сигнал регулювання (вихідний сигнал елементу) визначатиметься наступною формою виразу (див. вираз 1.8):

$$\begin{cases} u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \\ e(t) = r(t) - y(t) \end{cases} \tag{1.8}$$

, де $u(t)$ – вихідний сигнал регулювання;

$e(t)$ – сигнал помилки регулювання;

$r(t)$ – завдання регулятору (значення величини, якому має наближатися САР);

$y(t)$– сигнал реального значення регульованої величини з сенсора (вихідний сигнал процесу регульованого);

K_p – коефіцієнт пропорційної складової регулятора (налаштовуваний параметр);

K_i – коефіцієнт інтегральної складової регулятора (налаштовуваний параметр);

K_d – коефіцієнт диференційної складової регулятора (налаштовуваний параметр);

РОЗДІЛ 2 ПРОГРАМНА РЕАЛІЗАЦІЯ МАТЕМАТИЧНОЇ МОДЕЛІ

2.1 Адаптація математичної моделі до програмної реалізації

Перш за все потрібно визначити чисельні методи для розв'язування системи диференційних рівнянь не аналітичним методом.

Було вирішено передбачити імплементацію методу Ейлера (вирази 1.9-15) та метода Рунге-Кути четвертого порядку (вирази 1.16-47) для елемента регульованого процесу[4].

Метод Ейлера:

$$\frac{dM_{\text{С}}}{dt} = M_{\text{С}}(t_n) - M_{\text{С}}(t_{n-1}) \tag{1.9}$$

$$\frac{dI_{\text{я}}}{dt} = \frac{kF\cdot\omega + I_{\text{я}}R_{\text{я}} - U_{\text{я}}}{L_{\text{я}}} \tag{1.10}$$

$$\frac{d\omega}{dt} = \frac{kF\cdot I_{\text{я}} - M_{\text{С}}}{J_{\Sigma}} \tag{1.11}$$

$$M_{\text{С}}(t_n + 1) = M_{\text{С}}(t_n) + \frac{dM_{\text{С}}}{dt}\cdot \Delta t \tag{1.12}$$

$$I_{\text{я}}(t_n + 1) = I_{\text{я}}(t_n) + \frac{dI_{\text{я}}}{dt}\cdot \Delta t \tag{1.13}$$

$$\omega(t_n + 1) = \omega(t_n) + \frac{d\omega}{dt}\cdot \Delta t \tag{1.14}$$

$$\Theta(t_n + 1) = \Theta(t_n) + \omega(t_n)\cdot \Delta t \tag{1.15}$$

де Θ – результуючий кут обертання валу двигуна.

Метод Рунге-Кути четвертого порядку:

$$K1_{\frac{dM_{\text{С}}}{d\omega}} = \frac{dM_{\text{С}}}{d\omega}(\omega) = \begin{cases} k_1 + k_{curv}\cdot k_{lim}\cdot e^{-\omega\cdot k_{curv}}, & \omega > 0 \\ k_1 + k_{curv}\cdot k_{lim}\cdot e^{\omega\cdot k_{curv}}, & \omega \le 0 \end{cases} \tag{1.16}$$

$$K1_{\frac{dI_{\text{я}}}{dt}} = \frac{dI_{\text{я}}}{dt} = \frac{kF\cdot\omega + I_{\text{я}}R_{\text{я}} - U_{\text{я}}}{L_{\text{я}}} \tag{1.17}$$

$$K1_{\frac{d\omega}{dt}} = \frac{d\omega}{dt} = \frac{kF\cdot I_{\text{я}} - M_{\text{С}}}{J_{\Sigma}} \tag{1.18}$$

$$K1_{M_C} = \frac{dM_C}{dt} = K1_{\frac{dM_C}{d\omega}} \cdot K1_{\frac{d\omega}{dt}} \cdot \Delta t \quad (1.19)$$

$$K1_{I_я} = I_я(t_n) = I_я(t_{n-1}) + K1_{\frac{dI_я}{dt}} \cdot \Delta t \quad (1.20)$$

$$K1_{\omega} = \omega(t_n) = \omega(t_{n-1}) + K1_{\frac{d\omega}{dt}} \cdot \Delta t \quad (1.21)$$

$$K1_{\Theta} = \Theta(t_n) = \Theta(t_{n-1}) + K1_{\omega} \cdot \Delta t \quad (1.22)$$

$$K2_{\frac{dM_C}{d\omega}} = \frac{dM_C}{d\omega}(\omega) = \begin{cases} k_1 + k_{curv} \cdot k_{lim} \cdot e^{-(\omega + \frac{K1_\omega}{2}) \cdot k_{curv}}, & \omega > 0 \\ k_1 + k_{curv} \cdot k_{lim} \cdot e^{(\omega + \frac{K1_\omega}{2}) \cdot k_{curv}}, & \omega \le 0 \end{cases} \quad (1.23)$$

$$K2_{\frac{dI_я}{dt}} = \frac{dI_я}{dt} = \frac{kF \cdot (\omega + \frac{K1_\omega}{2}) + (I_я + \frac{K1_{I_я}}{2}) R_я - U_я}{L_я} \quad (1.24)$$

$$K2_{\frac{d\omega}{dt}} = \frac{d\omega}{dt} = \frac{kF \cdot (I_я + \frac{K1_{I_я}}{2}) - (M_C + \frac{K1_{M_C}}{2})}{J_\Sigma} \quad (1.25)$$

$$K2_{M_C} = \frac{dM_C}{dt} = K2_{\frac{dM_C}{d\omega}} \cdot K2_{\frac{d\omega}{dt}} \cdot \Delta t \quad (1.26)$$

$$K2_{I_я} = I_я(t_n) = I_я(t_{n-1}) + K2_{\frac{dI_я}{dt}} \cdot \Delta t \quad (1.27)$$

$$K2_{\omega} = \omega(t_n) = \omega(t_{n-1}) + K2_{\frac{d\omega}{dt}} \cdot \Delta t \quad (1.28)$$

$$K2_{\Theta} = \Theta(t_n) = \Theta(t_{n-1}) + K2_{\omega} \cdot \Delta t \quad (1.29)$$

$$K3_{\frac{dM_C}{d\omega}} = \frac{dM_C}{d\omega}(\omega) = \begin{cases} k_1 + k_{curv} \cdot k_{lim} \cdot e^{-(\omega + \frac{K2_\omega}{2}) \cdot k_{curv}}, & \omega > 0 \\ k_1 + k_{curv} \cdot k_{lim} \cdot e^{(\omega + \frac{K2_\omega}{2}) \cdot k_{curv}}, & \omega \le 0 \end{cases} \quad (1.30)$$

$$K3_{\frac{dI_я}{dt}} = \frac{dI_я}{dt} = \frac{kF \cdot (\omega + \frac{K2_\omega}{2}) + (I_я + \frac{K2_{I_я}}{2}) R_я - U_я}{L_я} \quad (1.31)$$

$$K3_{\frac{d\omega}{dt}} = \frac{d\omega}{dt} = \frac{kF \cdot (I_я + \frac{K2}{2}) - (M_C + \frac{K2_{M_C}}{2})}{J_\Sigma} \quad (1.32)$$

$$K3_{M_C} = \frac{dM_C}{dt} = K3_{\frac{dM_C}{d\omega}} \cdot K3_{\frac{d\omega}{dt}} \cdot \Delta t \quad (1.33)$$

$$K3_{I_я} = I_я(t_n) = I_я(t_{n-1}) + K3_{\frac{dI_я}{dt}} \cdot \Delta t \quad (1.34)$$

$$K3_{\omega} = \omega(t_n) = \omega(t_{n-1}) + K3_{\frac{d\omega}{dt}} \cdot \Delta t \quad (1.35)$$

$$K3_{\Theta} = \Theta(t_n) = \Theta(t_{n-1}) + K3_{\omega} \cdot \Delta t \quad (1.36)$$

$$K4_{\frac{dM_C}{d\omega}} = \frac{dM_C}{d\omega}(\omega) = \begin{cases} k_1 + k_{curv} \cdot k_{lim} \cdot e^{-(\omega+K3_\omega)\cdot k_{curv}}, & \omega > 0 \\ k_1 + k_{curv} \cdot k_{lim} \cdot e^{(\omega+K3_\omega)\cdot k_{curv}}, & \omega \leq 0 \end{cases} \quad (1.37)$$

$$K4_{\frac{dI_я}{dt}} = \frac{dI_я}{dt} = \frac{kF\cdot(\omega+K3_\omega)+(I_я+K3_{I_я})R_я - U_я}{L_я} \quad (1.38)$$

$$K4_{\frac{d\omega}{dt}} = \frac{d\omega}{dt} = \frac{kF\cdot(I_я+K3_{I_я})-(M_C+K3_{M_C})}{J_\Sigma} \quad (1.39)$$

$$K4_{M_C} = \frac{dM_C}{dt} = K4_{\frac{dM_C}{d\omega}} \cdot K4_{\frac{d\omega}{dt}} \cdot \Delta t \quad (1.40)$$

$$K4_{I_я} = I_я(t_n) = I_я(t_{n-1}) + K4_{\frac{dI_я}{dt}} \cdot \Delta t \quad (1.41)$$

$$K4_\omega = \omega(t_n) = \omega(t_{n-1}) + K4_{\frac{d\omega}{dt}} \cdot \Delta t \quad (1.42)$$

$$K4_\Theta = \Theta(t_n) = \Theta(t_{n-1}) + K4_\omega \cdot \Delta t \quad (1.43)$$

$$I_я(t) = (K1_{I_я} + 2 \cdot K2_{I_я} + 2 \cdot K3_{I_я} + K4_{I_я})/6 \quad (1.44)$$

$$\omega(t_n) = (K1_\omega + 2 \cdot K2_\omega + 2 \cdot K3_\omega + K4_\omega)/6 \quad (1.45)$$

$$\Theta(t_n) = (K1_\Theta + 2 \cdot K2_\Theta + 2 \cdot K3_\Theta + K4_\Theta)/6 \quad (1.46)$$

$$M_C(t_n) = (K1_{M_C} + 2 \cdot K2_{M_C} + 2 \cdot K3_{M_C} + K4_{M_C})/6 \quad (1.47)$$

По-друге, для програмної реалізації вигідно було б ввести елемент, що виконував би функцію завдання значення величини, котру має підтримувати регульований процес. Такий елемент є відтворенням оператора-людини, що виставляє необхідне завдання для величини певного технологічного процесу, яким він керує, та водночас і слугуватиме інструментом для програмних налаштовувачів регулятора, за допомогою якого налаштовувачі матимуть можливість виконувати маніпуляції над системою для оцінювання успіху виконаного налаштування або визначення фітнес-функції.

2.2 Обрані технології для програмної реалізації математичної моделі та проведення експериментів

Розробка програмного забезпечення проводиться на мові C++ в програмному середовищі QT6 за допомогою технологій Cmake та Git (+Github) з використанням фреймворку для написання юніт-тестів на базі бібліотеки BOOST (boost/test/unit_test.hpp).

В результаті збірки програми в результаті отримується два виконавчих файла:

- acs_model_for_experiments;
- unit_test.

Що використовують також утворювані динамічні бібліотеки (так звані “shared objects”) такі як:

- libControlled_process.so (бібліотека керованих процесів);
- libDC_source.so (бібліотека джерел енергії);
- libRegulator.so (бібліотека регуляторів);
- libRegulator_tunner.so (ця остання бібліотека не є частиною математичної моделі САР та буде згадана та розібрана пізніше).

Проектування програмного забезпечення здійснюється згідно парадигми програмування ООП зі застосуванням принципів SOLID.

Тому кожен клас, що розроблявся для імплементації трьох виділених елементів системи були успадковані від спільного базового абстрактного класу, роль якого визначити спільний інтерфейс для певних методів, спільних для кожного елемента системи.

Також для запуску комп’ютерного моделювання на певних умовах, для фіксації та запису результатів у різних режимах було утворено відповідно ще три додаткові ієрархії класів[14][15].

2.3 Структура розробленого програмного забезпечення

2.3.1 Структура програмної моделі САР

Клас Automated_control_system спроектований відповідно до патерну «фасад» (див. рис 2.1) являє собою абстракцію автоматизованої системи регулювання згідно до математичної моделі, що було розроблено у попередньому розділі, та інкапсулює виконання однієї ітерації моделювання за допомогою програмної реалізації математичної моделі. Виконує він це маючи вказівники на відповідні підкласи абстрактного класу Automated_control_system_element_interface, що абстрагують певні елементи моделі САР та інкапсулюють ітерацію розрахунків необхідних для моделювання в межах відповідного елементу, де обов'язок зумовити їх комунікації одне між одним покладені на сам клас-фасад.

Проектування інтерфейсу та ієрархії класів, що є абстракцією елементів САР, де класу Automated_control_system_element_interface — абстрактний клас, що визначає інтерфейс такими підкласами як:

Reference_signal_definder_static — задатчик сигналу керування (оператор або зовнішній сигнал з надсистеми) додатково введений новий елемент системи для зручності моделювання та використанні цієї підсистеми фасаду при тому фактично не є частиною САР, а лише програмна імплементація зовнішнього впливу на її роботу як вхідний сигнал, та де вихідним її сигналом виступає вихідний сигнал регульованого процесу або керованого об'єкту.

PID_regulator — регулятор.

DC_source — джерело постійної напруги (джерело електроенергії системи).

DC_engine — керований об'єкт, що абстрагує регульований процес, вплив на нього вхідних величин системи та включає також і вплив зовнішнього середовища, тобто і вплив ззовні системиі.

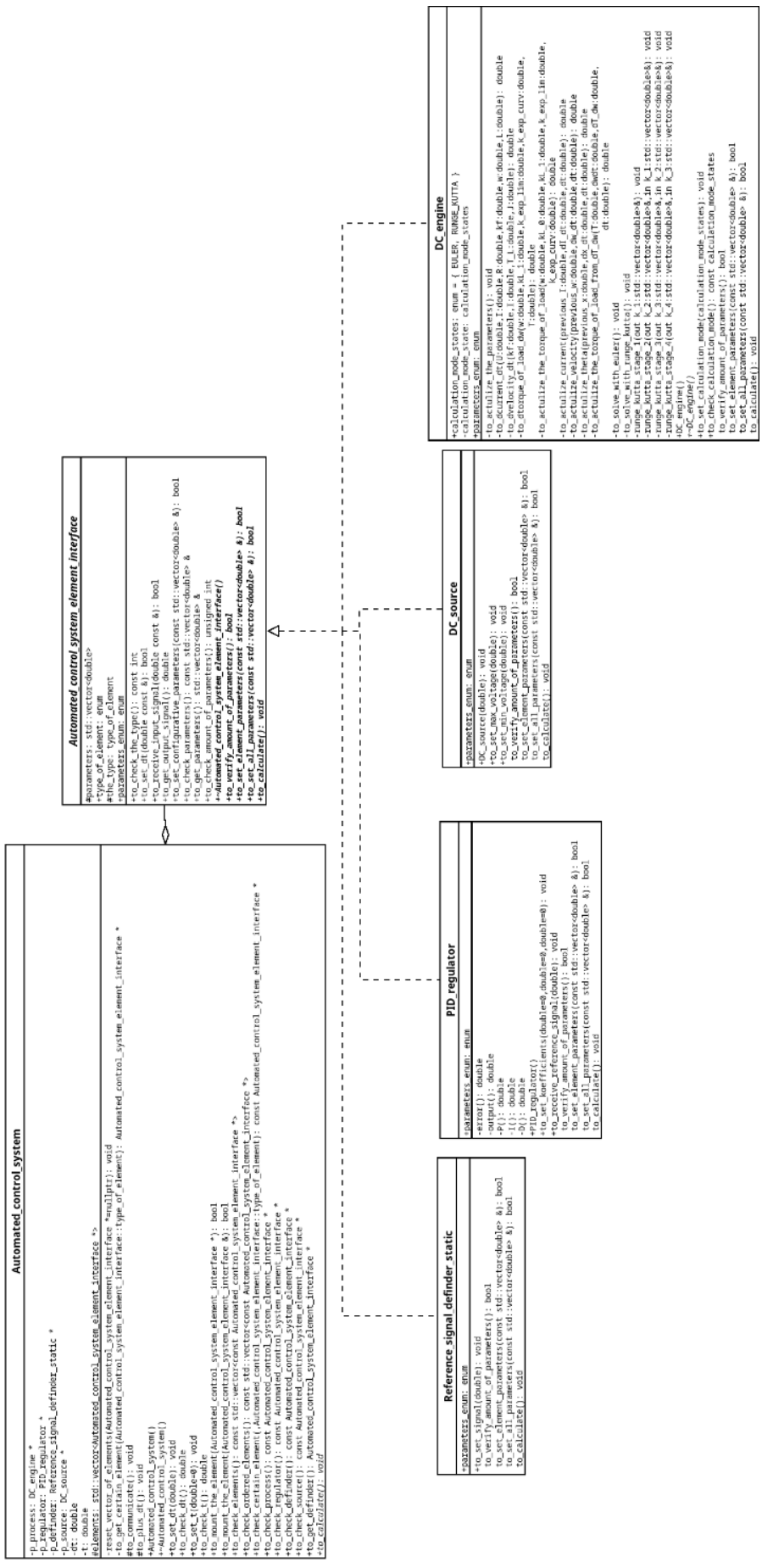

Рисунок 2.1 – UML-схема ієрархій класів програмної моделі САР

За допомогою розробленої програмної моделі САР у структурному вигляді такому як фасад та його підсистеми було зумовлено наступні можливості для клієнта:

- за спрощеним інтерфейсом клієнт може провести ітерацію моделювання за допомогою класу automated_control_system без прямої взаємодії із агрегованими ним класами;
- клієнту залишається доступ до усіх елементів підсистеми фасаду у випадку необхідності.

Отже, сама модель спроектована і тепер потребується демонстрація проектування класів відповідальних за проведення власне самого моделювання за встановленими умовами та фіксацією результатів моделювання.

2.3.2 Структура службових модулів програмного забезпечення для проведення експерименту

Кожен підклас абстрактного класу Experiment_executor, що було спроектовано відповідно до патерну «міст», успадковують роль свого батьківського класу та являють собою частину-абстракцію автоматизованої системи регулювання, котра використовує частину-реалізацію, де функції останньої частини покладені на вже розкритий раніше клас-фасад Automated_control_system.

Абстракція моста, підклас абстрактного класу Experiment_executor, використовує реалізацію ітерації розрахунків моделювання інкапсульовану в Automated_control_system таку кількість разів, що відповідатиме вказаній тривалості експерименту в Experiment_executor та кроку ітерації за часом.

Також на підкласи абстракції моста покладені додаткові необхідні функції в фіксації результатів моделювання з паралельним демонструванням його результатів за необхідності, та завжди відкриті для утворення нових підкласів для збільшення листу обов’язків при необхідності.

Тепер детальніше про розроблені підкласи Experiment_executor_interface та покладені на них обов’язки (див. рис 2.2):

Experiment_executor — запуск моделі та запис у файл усіх параметрів кожного елемента САР та виводяться у консоль.

Experiment_executor_short_report — те саме, проте лише обрані параметри у невеликій кількості.

Experiment_executor_for_fitness_function — запуск моделі та фіксація величини за якою відбувається регулювання у тип даних “std::vector”. (Створений для обслуговування потреб налаштовувача регулятора, що буде розглядатиметься пізніше).

Experiment_executor_for_fitness_function_with_varied_reference_signal — те саме проте за умови змінного сигналу завдання за обраними параметрами величини часу та повторень. (Також створений для обслуговування потреб налаштовувача регулятора, що розглядатиметься пізніше)

Підкласи Experiment_executor як клієнти використовують можливості класів наступної ієрархії, де Registrator — абстрактний клас, що визначає інтерфейс для похідних класів таких як:

Registrator_to_txt_file — обслуговує клас-клієнт Experiment_executor (виконує запис до txt-файлу).

Registrator_to_txt_file_short — те саме, але обслуговує Experiment_executor_short_report.

Registrator_to_std_vector — обслуговує клас Experiment_executor_for_fitness_function (виконує запис у тип даних “std::vector” значень регульованої величини).

Registrator_to_std_vector_difference — те саме, але обслуговує клас під назвою Experiment_executor_for_fitness_function_with_varied_reference_signal.

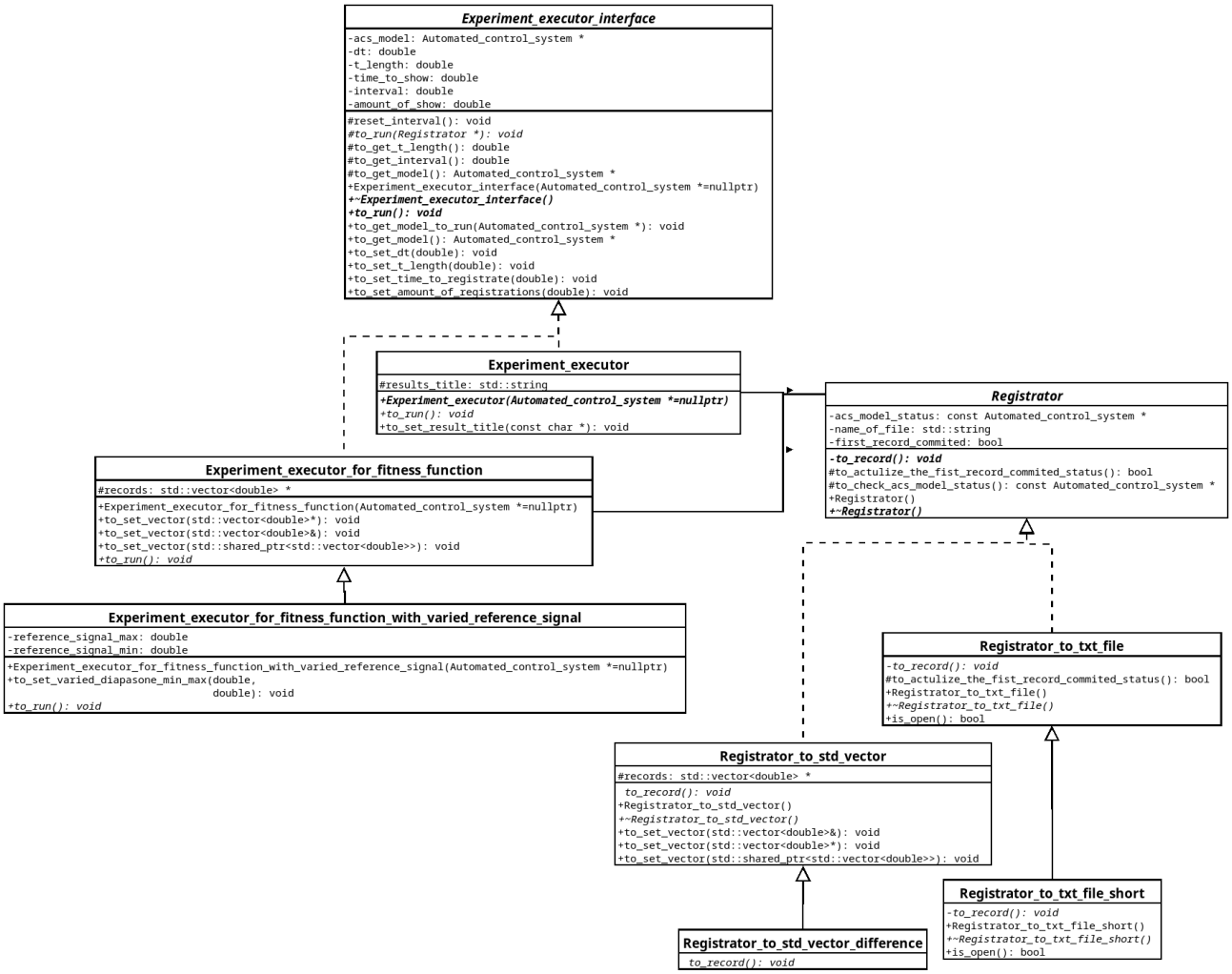

Рисунок 2.2 – UML-схема ієрархій класів службових модулей

Відтепер, коли усе готово для автоматизованого виконання багаторазових експериментів із запуском моделі САР та налагоджено отримання експериментальних даних, залишається лише розробити алгоритми для виконання оцінки експериментальних даних що можуть потребувати кожен з різновидів налаштовувачів. Але спершу було вирішено визначити єдиний інтерфейс незалежно від реалізації алгоритму оцінки та налаштовування і його мала визначити ієрархія класів, що абстрагуватиме налаштовувач.

2.3.3 Структура класів регулятора

Абстрактний клас Regulator_tuner_interface спроектований відповідно до патерну «адаптер» (див. рис 2.3) адаптує інтерфейси застосовуваних різноманітних алгоритмів оптимізації до використання клієнтом за одним стандартним інтерфейсом, що визначено цим класом та реалізації котрого імплементовані його підкласами.

Його підкласи для успішного початку налаштовування обов'язково мають отримати посилання на структуру parameters_for_optimizer, що визначає необхідні конфігурації для оптимізаторів використовуваних в алгоритмах налаштування, визначає умови проведення експериментів за допомогою котрих оптимізатори багаторазово визначатимуть значення фітнес-функцій протягом процесу налаштування, визначають параметри за котрими функціонуватимуть самі алгоритми налаштування та в решті решт передає необхідні вказівники на модель САР, регулятор котрої потребується налаштувати та яку треба використовувати для проведення експериментів необхідних протягом процесу налаштування.

Підкласами Regulator_tuner_interface є такі ж абстрактні класи Regulator_tuner_my_optimizer_interface та Regulator_tuner_my_generative_algorithm для відділення одне від одного алгоритмів, що використовують фітнес-функцію та тих, що використовують інші методи оцінки.

Regulator_tuner_my_optimizer_interface має два конкретних підкласи такі як Regulator_tuner_my_gradient_algorithm, Regulator_tuner_my_generative_algorithm, і власне відповідає паттерну "адаптер" тому, що у більшій своїй частині він виконує приведення інтерфейсу використовуваних ними оптимізаторів та в меншій частині алгоритм виконуючий експерименти та той, що вносить відповідні коефіцієнти до регулятора отримані у результаті вирішення проблеми оптимізації фітнес функції отримуваної експериментальним шляхом.

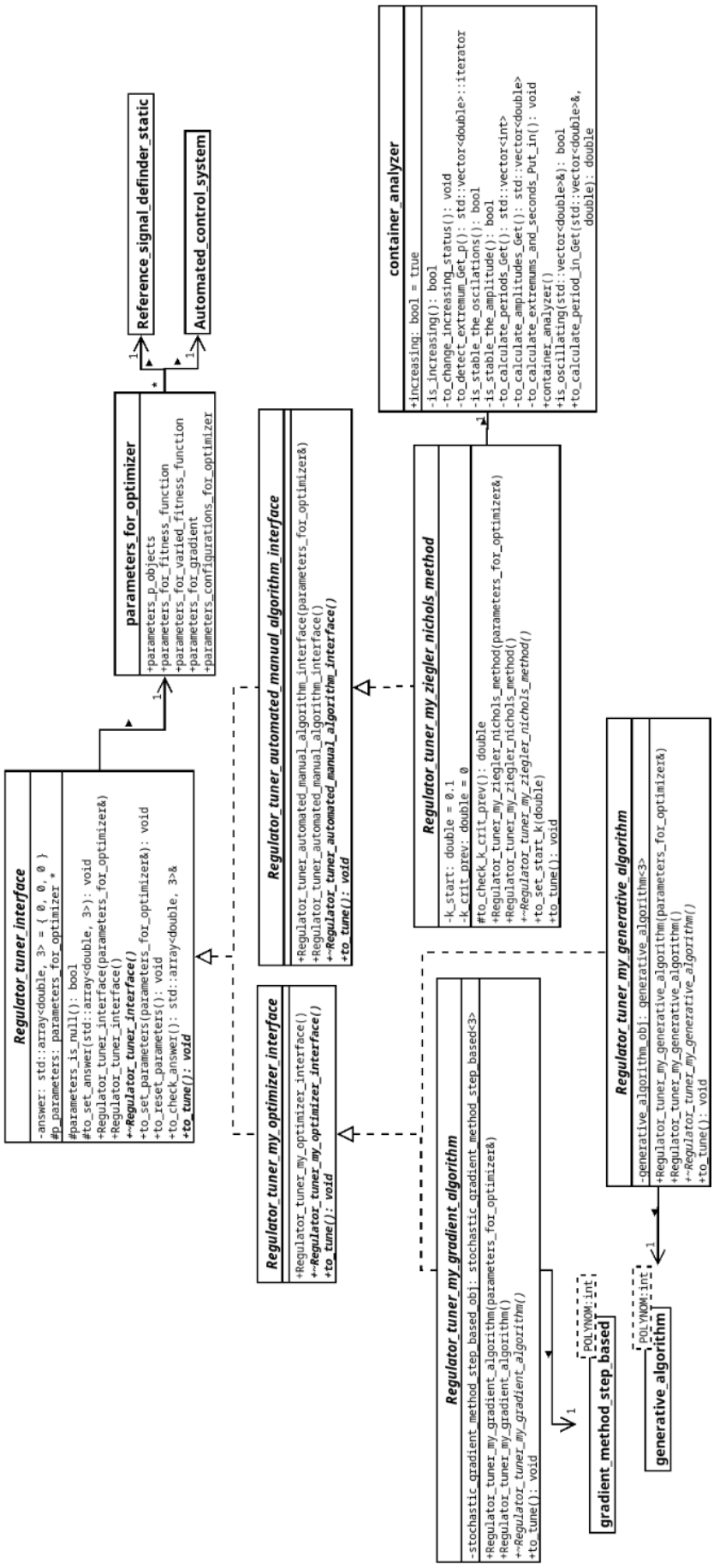

Рисунок 2.3 – UML-схема ієрархій класів тюнерів ПІД-регулятора

Regulator_tuner_my_gradient_algorithm використовує екземпляр шаблона класу Regulator_tuner_my_gradient_algorithm параметризованого кількістю коефіцієнтів потрібного для розрахунку, і Regulator_tuner_my_generative_algorithm з generative_algorithm відповідно також.

Regulator_tuner_my_gradient_algorithm використовує чисельний метод знаходження мінімуму фітнес-функції за допомогою руху вздовж градієнту під назвою “градієнтний спуск” (див. рис 2.4).

Regulator_tuner_my_generative_algorithm використовує евристичний та стохастичний метод пошуку мінімуму функції, що має назву “генетичний алгоритм” (див. рис 2.4).

Успадкований від Regulator_tuner_automated_manual_algorithm_interface підклас Regulator_tuner_my_ziegler_nichols_method реалізує алгоритм призначений для налаштування регулятора власноруч емпіричним шляхом одним з числа методів параметричної оптимізації та розроблений для налаштування саме ПІД-регулятора, що має назву “метод Циглера-Ніколса”.

Останній регулятор не використовує фітнес-функцій, проте має надавати оцінку коливанням з характеру стабільності коливань та визначення періоду коливань для розрахунку наближених до найкращих коефіцієнтів регулятора. Таку функціональність йому надає екземпляр класу під назвою container_analyzer.

Отже, регулятор був реалізований на базі різних алгоритмів налаштування, що відрізнялися не лише побудовою та порядком організації обчислень, а принципово різні в самих його видах. Реалізовані алгоритми належали таким принципово різним методам находження локального мінімума функції таким як чисельного, евристичного та стохастичного, а також і до методів параметричної оптимізації розроблених спеціально для мануального налаштування ПІД-регуляторів.

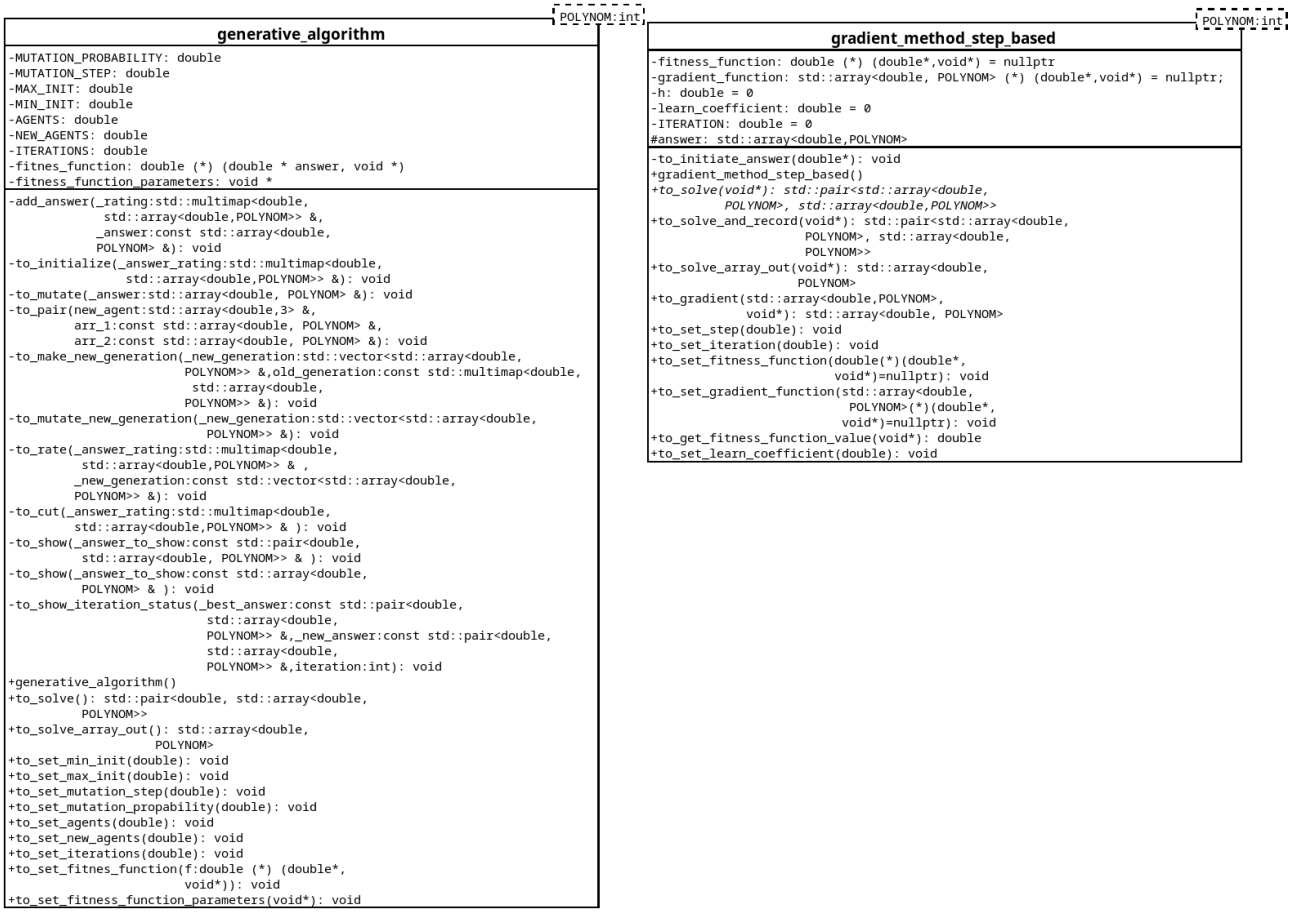

Рисунок 2.4 – UML-схема розроблених шаблонних класів використовуваних тюнерами регулятора, що інкапсулюють різні типи числельних алгоритмів пошуку мінімуму функції

Метод Циглера-Ніколса різко відрізняється від генетичного алгоритму та градієнтного методу. Останні використовують фітнес-функцію для оцінки поточної якості налаштування. Вона в цій роботі визначалася як сума квадратів значень, що дорівнюють різниці між вихідним значенням системи (результуючий кут обертання двигуна) та його поточним завданням на проміжку часу що підпало на момент фіксації протягом експерименту. Функція визначення градієнта використовувала цей самий алгоритм розрахунку фітнес-функції для визначення його зміни при частковому диференціюванні за кожним з трьох коефіцієнтів[7].

Генетичний алгоритм в цій імплементації під час утворення нащадку традиційним чином його складає з коефіцієнтів батьківських особин, обираючи довільним чином в якого з них отримати значення для ініціалізації

відповідного у нащадку, а процес мутації визначився випадковими змінами одного чи більше коефіцієнтів ініціалізованої щойно особини на випадкову величину своїм модулем не більше заданого значення[6][12].

Градієнтний спуск традиційним чином наближає нас у зворотній від градієнта напрямку на значення що відповідає добутку коефіцієнту навчання та градієнта.

Метод Циглера-Ніколса полягає у наступному порядку кроків:

а) обнулити усі коефіцієнти;

б) розпочати поступове збільшення пропорційного коефіцієнта доки не буде отримано стабільні коливання;

в) отримане значення коефіцієнта при перших проявах стабільних коливань фіксується разом із періодом коливань;

г) зафіксовані значення використовуються у розрахунках коефіцієнтів регулятора за одним набором формул з переліку, де кожен з них відповідає певним режимам його роботи.

Алгоритм оцінки коливань був реалізований оцінкою того чи виходить хоча б один період коливання за певні ліміти від середніх значень протягом усього проміжку вимірювання за такими величинами як амплітуда коливань та їх період.

Набір формул для розрахунку був обраний за принципом, що нам потрібен загальний випадок, що буде симетричним для режимів роботи для котрих знаходити відповідь мають інші алгоритми[8].

2.3.4 Узагальнена структура передбачених взаємодій між підсистемами

Варто прояснити відношення між такими класами як Automated_control_system, Experiment_executor_interface, Regulator_tuner_interface, структура взаємодії яких відображає розроблене програмне забезпечення в узагальненому вигляді (див. рис 2.5).

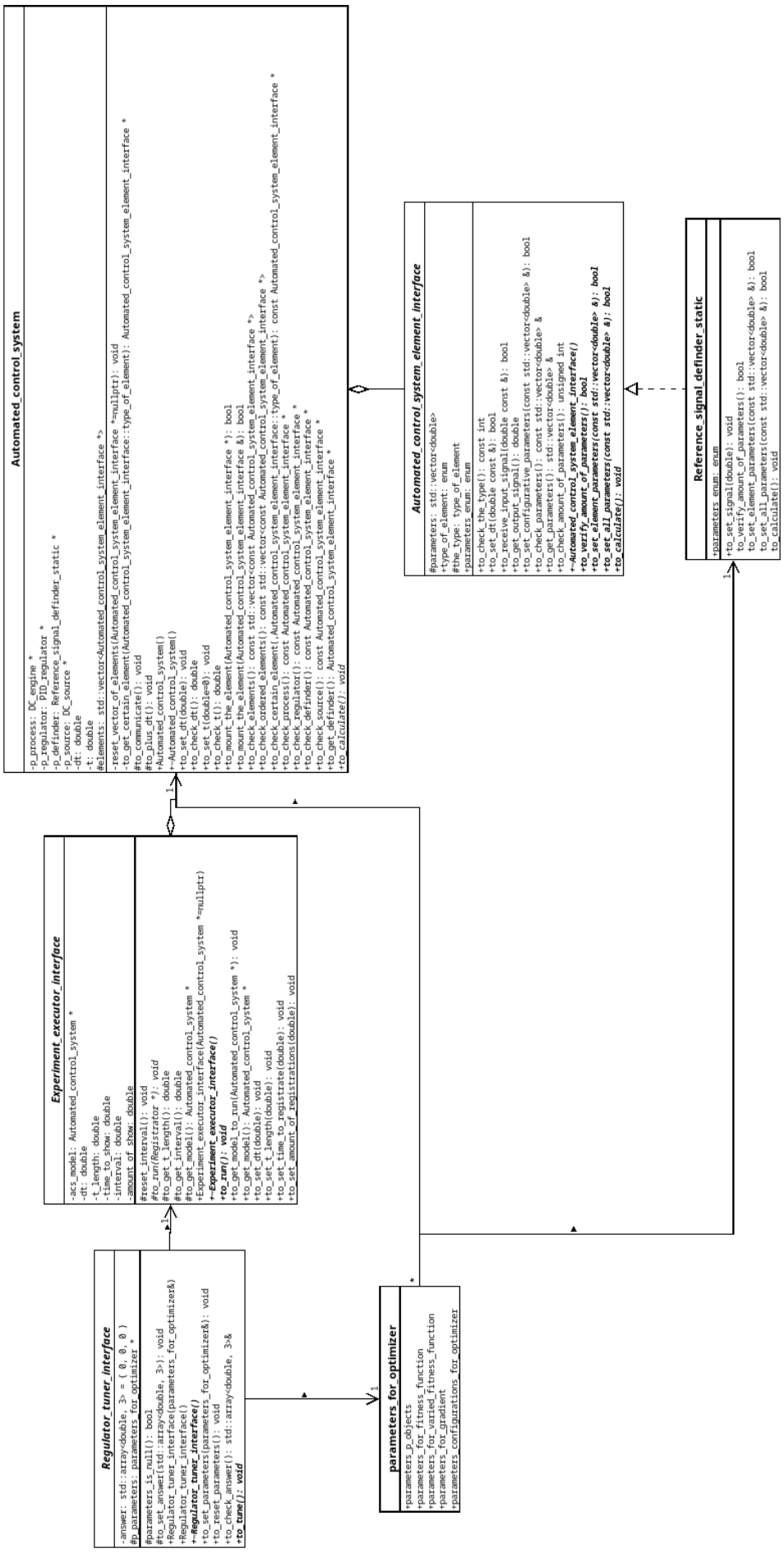

Рисунок 2.5 – UML-схема взаємодії інтерфейсів розроблених ієрархій класів та узагальнена структура передбачених взаємодій між модулями

Регулятори як абстракції утворювані підкласами визначених інтерфейсом абстрактного класу Regulator_tuner_interface мають асоціативний зв’язок з підкласами абстрактного класу Experiment_executor_interface, де як клієнт останньому надсилає запит на проведення експерименту та за допомогою наданих вказівників та певних конфігурацій структурою parameters_for_optimizer виконує оцінювання якості поточного стану налаштування регулятора системи та приймає рішення про наступний крок відповідно до використовуваного алгоритму та конфігурацій налаштовувача.

Experiment_executor_interface та його підкласи у свій час агрегують Automated_control_system, клас для виконання ітерації обчислень математичної моделі.

Regulator_tuner_interface після отримання остаточних результатів обчислень власноруч вводить їх до регулятора САР, таким чином закінчуючи його налаштування.

РОЗДІЛ 3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ ПРОГРАМНИХ МЕТОДІВ НАЛАШТУВАННЯ РЕГУЛЯТОРА

Як і було пригадано у вступі протягом дослідження використовувався метод досліджень у ході котрого поставлено було багато різноманітних експериментів під час комп'ютерного моделювання за допомогою програмних модулів, яким був наданий детальний опис у розділі другому.

Було проведено налаштування кожним з розроблених налаштовувачів за змінними конфігураціями їх алгоритмів налаштувань для дослідження впливу їх змін безпосередньо на якість налаштування, змінювалися лише конфігурації зміна котрих передбачає чи малий вплив на якість.

Також налаштування з різними конфігураціями передбачають пошук кращих випадків налаштування кожним типом тюнера для більш доречного порівняння результатів між кожного з типів та зниження впливу фактора випадку під час підбору конфігурацій на результати порівнянь.

Порівняння налаштовувачів буде відбуватися за певними метриками, подробиці котрих буде роз'яснено пізніше.

3.1 Експерименти з отримання налаштувань регулятора для подальшого аналізу алгоритмів

Всі експерименти в цій роботі ставилися з умовами ініціалізації об'єкта електродвигуна згідно до технічних характеристик реального зразка моделі 4ITO100L1 з потужністю 2500 Вт, номінальною напругою 220 В, номінальним струмом 14.2 А, номінальною швидкістю обертань 1300 об/хв, активним опором в 1.33 В та моментом інерції 0.05 кг·м^2 та номінальним електромагнітним моментом вираховуваним з наданих технічних характеристик.

Відбувалося регулювання електроприводом за значенням результуючого кута положення валу двигуна, а тобто за значенням сумарного кута на який було обернено двигун з початку пуску моделі.

Усі експерименти з налаштування ПІД-регулятора відбувалися за умови завдання значення 20 в радіанах як величини результуючого кута що має підтримувати система.

Оцінка результатів відбувалася за умови як статичного моменту механічного навантаження на двигун:

```
        arr[DC_engine::LOAD_K_0] = 1 * load;
```

Так і змінного:

```
        arr[DC_engine::LOAD_K_0] = 0.5 * load;
        arr[DC_engine::LOAD_K_1] = 0.2 * load;
```

Як із сигналом завдання для котрого відбувалося налаштування:

```
        definder->to_set_signal(20);
```

Так і для інших:

```
        definder->to_set_signal(50);

          . . .

        definder->to_set_signal(100);
```

І в тому числі і змінним у часі сигналом завдання між останніми двома згадуваними значеннями.

3.1.1 Постановка експерименту з параметричною оптимізацією методом Циглера-Ніколса

Алгоритм Циглера-Ніколса з різновиду параметричної оптимізації традиційно використовуваний в налаштуванні ПІД-регуляторів було автоматизовано за допомогою відповідного класу було протестовано за різних типах конфігурації, таких як:

- за різних умов роботи ним використовуваного класу аналізатора вихідних значень під час пусків моделі такими як зміна ступеня суворості в оцінюванні стабільності коливань;
- за різною швидкістю підвищення пропорційного коефіцієнту;
- за різною стартовою величиною пропорційно коефіцієнту після нульового значення.

Першим варіантом налаштування виступатиме дослідження впливу ступеня суворості в оцінюванні стабільності коливань:

а) найжорстокіший — налаштування ніколи не завершувалося під час експериментальних пусків, тому викреслено за неможливістю отримати результати як нежиттєспроможній метод оцінки стабільності коливань під час налаштування:

```
```
bool
container_analyzer::is_stable_the_oscilations(const
std::vector<int>& seconds)
{
    …
    for (auto i : amplitudes)
        ans *= (double(average_amplitude) * 0.9 < i
&& i < double(average_amplitude) * 1.1) ;
}
…
```
```

```
    bool
container_analyzer::is_stable_the_amplitude(const
std::vector<double>& extremums)
    {
        … // те саме
    }
    ```
```

б) середній ступінь — надавав стабільні результати налаштування $\{ 0.175, 0.0190217, 1.0626 \}$ :

```
    ```
    bool
container_analyzer::is_stable_the_oscilations(const
std::vector<int>& seconds)
    {
        …
        for (auto i : amplitudes)
            ans *= (double(average_amplitude) * 0.5 < i &&
i < double(average_amplitude) * 2) ;
    }
    …
    bool container_analyzer::is_stable_the_amplitude(const
std::vector<double>& extremums)
    {
        … // те саме
    }
    ```
```

в) дуже вільний ступінь — надавав стабільні результати налаштування { 0.175, 0.0190217, 1.0626 }, що ідентичні до попереднього ступеню:

```
    ```
    bool
container_analyzer::is_stable_the_oscilations(const
std::vector<int>& seconds)
    {
        …
        for (auto i : amplitudes)
            ans *= (double(average_amplitude) * 0.33 < i &&
i < double(average_amplitude) * 3) ;
    }
    …
    bool container_analyzer::is_stable_the_amplitude(const
std::vector<double>& extremums)
    {
        … // те саме
    }
```

```
```

Отже, в результаті трьох експериментів залишилось лише двоє варіантів конфігурацій за ступенем суворості оцінки стабільності коливань та й ті мають однакові результати а тому отримані налаштування в результаті різних експериментів матимуть одне найменування “Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass”, де частина “0.5 step k_crit” відповідає обраному одного виду конфігурацій з наступної низки їх варіацій, що відповідають наступному етапу експериментів.

Другим варіантом налаштування виступатиме дослідження впливу стартової величини та швидкості зміни пропорційного коефіцієнту для дослідження на предмет впливу на якість налаштування. Показник був змінюваним величиною зі значення 0,1 до 1 й у тому числі враховувалися попередні результати, де позначка “0.5 step k_crit” це і демонструвала. Нові експерименти проводилися із середнім ступенем оцінки коливань і результати були наступними:

Налаштування зі значенням 0.1 вимагало 33 ітерації з підвищення пропорційного коефіцієнту та оцінки коливань перш ніж визначити коефіцієнти та дало наступний результат { 0.595, 0.111215, 2.10095 } отримало найменування “Zeingler-Nichols method 0.1 step k_crit”:

```
```
    optimizer->to_set_start_k(0.1);
```
```

Налаштування зі значенням 1 вимагало 5 ітерацій та дало наступний результат { 0.8, 0.137339, 3.0756 } отримало найменування “Zeingler-Nichols method 1 step k_crit”:

```
```
    optimizer->to_set_start_k(1);
```
```

А результат з попереднього етапу зі значенням 0,5 вимагав лише 3 ітерації має вже відомий результат під найменуванням “Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass”:

```
    optimizer->to_set_start_k(0.5);
```

3.1.2 Постановка експерименту з евристичним та стохастичним генетичним алгоритмом

Генетичний алгоритм один з різновиду евристичного та стохастичного методу пошуку мінімуму функції гіпотетично не найкращий метод налаштування через свою властивість таку як стохастичність, тому що пошук оптимального рішення покладається на випадок збільшуючи кількість виклику фітнес-фукції, значення котрої визначається експериментальним шляхом, що в свою чергу вимагає більш довгого часу роботи обладнання у режимі налаштовування, що відповідає потенційним збиткам тому як обладнання САР не має можливості гарантовано виконувати роботу проте споживає енергію завдаючи фінансових збитків. Тобто цей метод налаштування може нести забагато накладних розходів та мати завелику вартість у порівнянні з іншими.

Цей алгоритм було протестовано багаторазово при двох різних типах конфігурацій алгоритму:

- понижена схильність до мутацій у агентів;
- підвищена схильність до мутації агентів.

Налаштування відповідно:

а) під найменуваннями оформлених у стилі “Generative method non mutationable a lot #i”, де “i” — порядковий номер проведеного експерименту з отриманим результатів:

```
void
Default_configuration_setter::to_set_configurations_in_parameter
s_for_optimizer(parameters_for_optimizer& _arg) const
```

```
    {
        …
_arg.parameters_configurations_for_optimizer_obj.mutation_step =
0.1 * scale;

_arg.parameters_configurations_for_optimizer_obj.mutation_propab
ility = 0.5;
        …
    }
    ```
```

б) під найменуваннями оформлених у стилі “Generative method mutationable #i”, де “i” — порядковий номер проведеного експерименту з отриманим результатів:

```
    ```
    void
Default_configuration_setter::to_set_configurations_in_parameter
s_for_optimizer(parameters_for_optimizer& _arg) const
    {
        …
_arg.parameters_configurations_for_optimizer_obj.mutation_step =
1.0 * scale;

_arg.parameters_configurations_for_optimizer_obj.mutation_propab
ility = 0.9;
        …
    }
    ```
```

У результаті багаторазових експериментів через свою принципову властивість недетермінованості у наслідок своєї стохастичної природи генетичний алгоритм завжди видавав різні результати навіть при незмінних конфігураціях на відміну від попереднього алгоритму налаштування. Їх усі результати згідно обумовленому стилю іменування було зафіксовано у кількості п’яти штук на кожен вид налаштування:

«Generative method non mutationable a lot #1» { 3.9, 3.5, 5.3 }

«Generative method non mutationable a lot #2» { 5.1, 2.2, 6.7 }

«Generative method non mutationable a lot #3» { 5.2, 1.2, 2.8 }

«Generative method non mutationable a lot #4» { 5.8, 6.4, 7.2 }
```

«Generative method non mutationable a lot #5» { 4, 4.3, 0.1 }

«Generative method mutationable #1» { 9.4, 7, 3.5 }
«Generative method mutationable #2» { 9.1, 5, 0 }
«Generative method mutationable #3» { 8.8, 2.5, 5.2 }
«Generative method mutationable #4» { 1.7, 0.8, 0.3 }
«Generative method mutationable #5» { 0.5, 6.7, 1.3 }

3.1.3 Постановка експерименту з евристичним та чисельним методом градієнтного спуску

Градієнтний спуск один з різновиду евристичних та чисельних методів знаходження мінімуму фітнес-функції. Гіпотетично має бути одним з найкращих методів налаштування регулятора завдяки пошуку чисельного рішення базуючись на залежностях, що мають ним отримувані експериментальні дані, що оцінюються через фітнес-функцію, тим самим знижуючи частоту її викликів і протилежно до випадку із генетичним алгоритмом відповідно матиме меншу вартість налаштування з економічної точки зору на процес налаштування з огляду витрат енергії до виконання корисної роботи обладнанням САР.

Алгоритм тестувався багаторазово за цілою низкою стартових умов, а саме ініціалізації початкової відповіді від котрої значно залежать результати градієнтного спуску, проте сам алгоритм має детерміновану поведінку через свої властивості і тому на кожну конфігурацію має лише одну й ту саму відповідь.

Їх найменування виконується за наступним стилем: “Gradient method ‘kp,ki,kd’ init”:

«Gradient method ‘1,0,0’ init» { 1.06758, 0.317347, 0.579046 }
«Gradient method ‘2,0,0’ init» { 8.38471, -29403.1, -341273 }
«Gradient method ‘5,0,0’ init» { 6.13157, 1.49139, 1.30597 }

«Gradient method ‘7,0,0’ init» { 7.69889, 0.876194, 0.881272 }

«Gradient method ‘10,0,0’ init» { 10.474, 0.568234, 0.571808 }

Де kp, ki, kd є коефіцієнтами початкових налаштувань ПІД-регулятоа відповідно, якими ініціалізується на початку кожного експерименту стартова відповідь, як стартова точка пошуків відповіді градієнтним алгоритмом.

3.1.4 Постановка експерименту з методом градієнтного спуску з ініціалізацією початкової відповіді методом Циглера-Ніколса

Було зроблено припущення, що по суті являє собою гіпотезу, що ініціалізація початкової відповіді методом Циглера-Ніколса може бути кращим кроком за випадкове значення як стартове для метода градієнтного спуску через можливість виконати початкове наближення до теоретично найкращого діапазону значень коефіцієнтів, а градієнтний метод лише знайде локальний мінімум тієї попередньо наближеної області, що унаслідок матиме гіпотетично найкращий з глобальних мінімумів, особливо у порівнянні з градієнтним методом без попереднього наближення.

Ця гіпотеза буде перевірена протягом постановки експериментів та аналізу даних.

А доки маємо результати експериментів під відповідним стилем найменувань:

«Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit « { 5.7759, 5.34597, 7.63554 }

«Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit « { 4.70099, 4.2221, 6.21183 }

«Gradient method with init by the Zeingler-Nichols method 1 step k_krit « { 2.49195, 1.82929, 4.76755 }

Де в “Gradient method with init by the Zeingler-Nichols method 1 step k_krit” позначає обрану швидкість підвищення коефіцієнту та його стартову величину метода Циглера-Ніколса відповідно до опису у параграфі 3.1.1.

3.2 Розробка методу чисельного оцінювання отриманих експериментальних даних та власне її результати у повному обсязі

Було розроблено наступні метрики для оцінки алгоритмів за якістю налаштування:

а) кількість викликів фітнес-функції як деякий покажчик витрат на налаштування;

б) значення фітнес-функції протягом трьох різних за тривалістю проміжків часу виконання експериментальних пусків за умовами:

1) незмінного сигналу завдання для котрого проводилися налаштування;

2) двох різних незмінних сигналів завдання не для котрих проводилося налаштування та середнє значення між ними;

3) змінний сигнал зі значеннями завдання не для котрих відбувалося налаштування;

в) ті самі умови пусків але вже зі змінним навантаженням на систему, а тобто все те ж саме але з імітацією зовнішнього впливу та більш наближеного до реального випадку типу навантаження;

г) стабільність налаштувань як різниця значень фітнес-функцій між найдовшим проміжком часу вимірювань та другим за тривалістю проміжком часу.

За отриманими метриками можна виконати оцінку кожного з методів

3.2.1 Рейтинг за економічністю в еквіваленті кількості викликів фітнес-функції за алгоритмами як еквівалент витраченого часу та фінансів

Економічність можна порівнювати за допомогою кількості викликів фітнес-функції за алгоритмами тому, що ця кількість є еквівалентом витраченого часу та фінансів на витрати енергії та збитків протягом роботи обладнання у режимі налаштування. До того ж збитки можуть носити як характер простою, коли промисловий процес не виробляє продукту, так і характер браку продукції, коли регулятор ще не встиг налаштуватися.

Генетичний і градієнтний метод виконали п’ять ітерацій протягом процесу налаштування, де генетичний метод викликав фітнес-функцій десять разів для своїх десяти агентів-нащадків, а градієнтний на кожну ітерацію по чотири виклику на ітерацію для визначення поточного значення та за допомогою нього визначити часткове диференціювання за кожною з трьох координат відповідно.

Отже, кількість викликів фітнес функції складала 50 для генетичного та 20 для градієнтного методу.

А для Циглера-Ніколса потребувалося від 33 викликів зі стартовим коефіцієнтом та швидкістю його наростання в значення 0,1 до 5 викликів — значення 1, та 3 викликів — значення 0,5. А комбінація його з градієнтним методом 53, 25 та 23 відповідно (див. табл. 3.1).

Таблиця 3.1 – Рейтингування за методів налаштування з кількістю викликів фітнес функції (за витратами на налаштування)

Рейтинг	Певний випадок налаштувань	Кількість викликів фітнес-функціїі
1	Метод Циглера-Ніколса k_start = 0,5	3
2	Метод Циглера-Ніколса k_start = 1	5
3	Градієнтний спуск	20
4	Метод Циглера-Ніколса k_start = 0,1	33
5	Генетичний алгоритм	50

3.2.2 Рейтинг за значенням фітнес-функції протягом трьох різних за тривалістю проміжків часу виконання експериментальних пусків

За умовами незмінного сигналу завдання для котрого проводилися налаштування:

– 360 секунд (див. табл. 3.2);

Таблиця 3.2 – Оцінка при *r(t)=const,* t_n=360 с, налаштовано для *r(t)*=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	7933.8
2	Generative method non mutationable a lot #3	12125.3
3	Generative method non mutationable a lot #2	12462.2
4	Gradient method ‘10,0,0’ init	14415.8
5	Gradient method ‘7,0,0’ init	17432.9
6	Zeingler-Nichols method 1 step k_crit	166038
7	Zeingler-Nichols method 0.1 step k_crit	395448
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	7.14289e+06
9	Gradient method ‘5,0,0’ init	5.9584e+09
10	Gradient method ‘1,0,0’ init	1.803e+10
11	Generative method non mutationable a lot #1	3.57176e+10
12	Generative method non mutationable a lot #4	3.86145e+10
13	Generative method mutationable #4	3.88705e+10
14	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	3.92763e+10

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
15	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	4.18367e+10
16	Generative method mutationable #2	4.50287e+10
17	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	4.98165e+10
18	Generative method mutationable #1	5.31742e+10
19	Generative method non mutationable a lot #5	5.38634e+10
20	Generative method mutationable #5	6.63403e+10
21	Gradient method ‘2,0,0’ init	6.17963e+12

– 720 секунд (див. табл. 3.3);

Таблиця 3.3 – Оцінка при $r(t)=const$, t_n=720 с, налаштовано для $r(t)$=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	7933.8
2	Generative method non mutationable a lot #3	12125.3
3	Generative method non mutationable a lot #2	12462.2
4	Gradient method ‘10,0,0’ init	14415.8
5	Gradient method ‘7,0,0’ init	17432.9
6	Zeingler-Nichols method 1 step k_crit	166038
7	Zeingler-Nichols method 0.1 step k_crit	395448
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	7.14284e+06
9	Gradient method ‘5,0,0’ init	5.63513e+11
10	Gradient method ‘1,0,0’ init	7.06777e+11

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
11	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	1.02779e+12
12	Generative method mutationable #1	1.07954e+12
13	Generative method mutationable #2	1.0857e+12
14	Generative method mutationable #4	1.09695e+12
15	Generative method non mutationable a lot #1	1.09814e+12
16	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	1.09879e+12
17	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	1.15468e+12
18	Generative method non mutationable a lot #5	1.1689e+12
19	Generative method non mutationable a lot #4	1.33388e+12
20	Generative method mutationable #5	1.5439e+12
21	Gradient method ‘2,0,0’ init	4.18962e+14

– 1140 секунд (див. табл. 3.4);

Таблиця 3.4 – Оцінка при $r(t)=const$, $t_n=1140$ с, налаштовано для $r(t)=20$

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	7933.8
2	Generative method non mutationable a lot #3	12125.3
3	Generative method non mutationable a lot #2	12462.2
4	Gradient method ‘10,0,0’ init	14415.8

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
5	Gradient method ‘7,0,0’ init	17432.9
6	Zeingler-Nichols method 1 step k_crit	166038
7	Zeingler-Nichols method 0.1 step k_crit	395448
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	7.14284e+06
9	Gradient method ‘5,0,0’ init	5.63513e+11
10	Gradient method ‘1,0,0’ init	7.06777e+11
11	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	1.02779e+12
12	Generative method mutationable #1	1.07954e+12
13	Generative method mutationable #2	1.0857e+12
14	Generative method mutationable #4	1.09695e+12
15	Generative method non mutationable a lot #1	1.09814e+12
16	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	1.09879e+12
17	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	1.15468e+12
18	Generative method non mutationable a lot #5	1.1689e+12
19	Generative method non mutationable a lot #4	1.33388e+12
20	Generative method mutationable #5	1.5439e+12
21	Gradient method ‘2,0,0’ init	4.18962e+14

Двох різних незмінних сигналів завдання не для котрих проводилося налаштування та середнє значення між ними.

– 360 секунд перше значення (див. табл. 3.5);

Таблиця 3.5 – Оцінка при *r(t)=const,* t_n=360 с, налаштовано для *0.4r(t)*=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	78484.9
2	Gradient method ‘10,0,0’ init	97479.7
3	Generative method non mutationable a lot #2	99333.1
4	Generative method non mutationable a lot #3	114749
5	Gradient method ‘7,0,0’ init	150177
6	Zeingler-Nichols method 1 step k_crit	236840
7	Zeingler-Nichols method 0.1 step k_crit	431264
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	6.6484e+06
9	Gradient method ‘5,0,0’ init	1.85535e+10
10	Gradient method ‘1,0,0’ init	2.46394e+10
11	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	3.36734e+10
12	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	3.97422e+10
13	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	4.15997e+10
14	Generative method mutationable #4	4.17452e+10
15	Generative method non mutationable a lot #1	4.18692e+10
16	Generative method mutationable #2	4.42951e+10
17	Generative method non mutationable a lot #5	4.68945e+10
18	Generative method mutationable #5	5.02991e+10
19	Generative method mutationable #1	5.63382e+10
20	Generative method non mutationable a lot #4	6.13424e+10
21	Gradient method ‘2,0,0’ init	6.18729e+12

– 360 секунд друге значення (див. табл. 3.6);

Таблиця 3.6 – Оцінка при *r(t)=const,* t_n=360 с, налаштовано для *0.2r(t)*=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Gradient method ‘10,0,0’ init	457724
2	Zeingler-Nichols method 1 step k_crit	666458
3	Gradient method ‘7,0,0’ init	889542
4	Generative method non mutationable a lot #3	1.00809e+06
5	Zeingler-Nichols method 0.1 step k_crit	1.03424e+06
6	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	6.65399e+06
7	Gradient method ‘5,0,0’ init	1.39424e+11
8	Generative method mutationable #3	1.4784e+11
9	Generative method non mutationable a lot #2	1.52826e+11
10	Gradient method ‘1,0,0’ init	1.65548e+11
11	Generative method mutationable #4	2.15131e+11
12	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	2.23251e+11
13	Generative method mutationable #1	2.2651e+11
14	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	2.30718e+11
15	Generative method non mutationable a lot #1	2.3131e+11
16	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	2.31708e+11
17	Generative method mutationable #2	2.57062e+11
18	Generative method non mutationable a lot #4	2.76725e+11
19	Generative method non mutationable a lot #5	2.7848e+11

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
20	Generative method mutationable #5	3.19554e+11
21	Gradient method ‘2,0,0’ init	5.14633e+13

– 720 секунд перше значення (див. табл. 3.7);

Таблиця 3.7 – Оцінка при *r(t)=const,* t_n=720 с, налаштовано для *0.4r(t)*=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	78484.9
2	Gradient method ‘10,0,0’ init	97479.7
3	Generative method non mutationable a lot #2	99333.1
4	Generative method non mutationable a lot #3	114749
5	Gradient method ‘7,0,0’ init	150177
6	Zeingler-Nichols method 1 step k_crit	236840
7	Zeingler-Nichols method 0.1 step k_crit	431264
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	6.64835e+06
9	Gradient method ‘5,0,0’ init	1.16601e+11
10	Gradient method ‘1,0,0’ init	1.53557e+11
11	Generative method mutationable #4	1.93005e+11
12	Generative method mutationable #2	2.00711e+11
13	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	2.20806e+11
14	Generative method non mutationable a lot #5	2.4108e+11
15	Generative method mutationable #1	2.41898e+11

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
16	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	2.62017e+11
17	Generative method mutationable #5	2.63236e+11
18	Generative method non mutationable a lot #1	2.69486e+11
19	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	2.69576e+11
20	Generative method non mutationable a lot #4	2.73378e+11
21	Gradient method ‘2,0,0’ init	5.14109e+13

– 720 секунд друге значення (див. табл. 3.8);

Таблиця 3.8. – Оцінка при $r(t)=const$, t_n=720 с, налаштовано для $0.2r(t)$=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Gradient method ‘10,0,0’ init	457724
2	Zeingler-Nichols method 1 step k_crit	666458
3	Gradient method ‘7,0,0’ init	889542
4	Generative method non mutationable a lot #3	1.00809e+06
5	Zeingler-Nichols method 0.1 step k_crit	1.03424e+06
6	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	6.65399e+06
7	Gradient method ‘5,0,0’ init	1.39424e+11
8	Generative method mutationable #3	1.4784e+11
9	Generative method non mutationable a lot #2	1.52826e+11
10	Gradient method ‘1,0,0’ init	1.65548e+11

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
11	Generative method mutationable #4	2.15131e+11
12	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	2.23251e+11
13	Generative method mutationable #1	2.2651e+11
14	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	2.30718e+11
15	Generative method non mutationable a lot #1	2.3131e+11
16	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	2.31708e+11
17	Generative method mutationable #2	2.57062e+11
18	Generative method non mutationable a lot #4	2.76725e+11
19	Generative method non mutationable a lot #5	2.7848e+11
20	Generative method mutationable #5	3.19554e+11
21	Gradient method ‘2,0,0’ init	5.14633e+13

– 1140 секунд перше значення (див. табл. 3.9);

Таблиця 3.9 – Оцінка при $r(t)=const$, t_n=1140 с, налаштовано для $0.4r(t)$=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	78484.9
2	Gradient method ‘10,0,0’ init	97479.7
3	Generative method non mutationable a lot #2	99333.1
4	Generative method non mutationable a lot #3	114749
5	Gradient method ‘7,0,0’ init	150177

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
6	Zeingler-Nichols method 1 step k_crit	236840
7	Zeingler-Nichols method 0.1 step k_crit	431264
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	6.64835e+06
9	Gradient method ‘5,0,0’ init	6.19321e+11
10	Gradient method ‘1,0,0’ init	6.98053e+11
11	Generative method mutationable #4	9.22533e+11
12	Generative method mutationable #1	1.10969e+12
13	Generative method mutationable #2	1.13962e+12
14	Generative method non mutationable a lot #5	1.15072e+12
15	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	1.22879e+12
16	Generative method non mutationable a lot #4	1.26875e+12
17	Generative method non mutationable a lot #1	1.29641e+12
18	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	1.30006e+12
19	Generative method mutationable #5	1.3005e+12
20	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	1.31868e+12
21	Gradient method ‘2,0,0’ init	4.19089e+14

– 1140 секунд перше друге значення (див. табл. 3.10);

Таблиця 3.10 – Оцінка при *r(t)=const,* t_n=1140 с, налаштовано для *0.2r(t)*=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Gradient method ‘10,0,0’ init	457724
2	Zeingler-Nichols method 1 step k_crit	666458
3	Gradient method ‘7,0,0’ init	889542
4	Generative method non mutationable a lot #3	1.00809e+06
5	Zeingler-Nichols method 0.1 step k_crit	1.03424e+06
6	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	6.65399e+06
7	Gradient method ‘5,0,0’ init	7.15194e+11
8	Gradient method ‘1,0,0’ init	7.2562e+11
9	Generative method mutationable #3	8.03985e+11
10	Generative method mutationable #4	9.59222e+11
11	Generative method non mutationable a lot #2	9.97941e+11
12	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	1.0588e+12
13	Generative method mutationable #1	1.09009e+12
14	Generative method non mutationable a lot #1	1.09998e+12
15	Generative method mutationable #2	1.1001e+12
16	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	1.10489e+12
17	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	1.13114e+12
18	Generative method non mutationable a lot #4	1.32819e+12
19	Generative method non mutationable a lot #5	1.368e+12
20	Generative method mutationable #5	1.47221e+12
21	Gradient method ‘2,0,0’ init	4.19301e+14

Змінний сигнал зі значеннями завдання не для котрих відбувалося налаштування

– 360 секунд (див. табл. 3.11);

Таблиця 3.11 – Оцінка при $r(t) \neq const$, t_n=360 с, налаштовано для значення 20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	390011
2	Generative method non mutationable a lot #2	464792
3	Gradient method ‘10,0,0’ init	493856
4	Generative method non mutationable a lot #3	578055
5	Gradient method ‘7,0,0’ init	765018
6	Zeingler-Nichols method 1 step k_crit	808379
7	Zeingler-Nichols method 0.1 step k_crit	1.2953e+06
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	9.6949e+06
9	Gradient method ‘5,0,0’ init	1.78783e+10
10	Gradient method ‘1,0,0’ init	2.35183e+10
11	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	3.37411e+10
12	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	3.99902e+10
13	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	4.11928e+10
14	Generative method non mutationable a lot #1	4.13851e+10
15	Generative method mutationable #4	4.171e+10
16	Generative method mutationable #2	4.38918e+10
17	Generative method non mutationable a lot #5	4.60766e+10

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
18	Generative method mutationable #5	5.11235e+10
19	Generative method mutationable #1	5.55545e+10
20	Generative method non mutationable a lot #4	6.07955e+10
21	Gradient method ‘2,0,0’ init	6.19856e+12

– 720 секунд (див. табл. 3.12);

Таблиця 3.12 – Оцінка при $r(t)\neq const$, t_n=720 с, налаштовано для значення 20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	389988
2	Generative method non mutationable a lot #2	464403
3	Gradient method ‘10,0,0’ init	498065
4	Generative method non mutationable a lot #3	582379
5	Gradient method ‘7,0,0’ init	804565
6	Zeingler-Nichols method 1 step k_crit	820910
7	Zeingler-Nichols method 0.1 step k_crit	1.45637e+06
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	7.92699e+06
9	Gradient method ‘5,0,0’ init	1.18716e+11
10	Gradient method ‘1,0,0’ init	1.53943e+11
11	Generative method mutationable #4	1.92876e+11
12	Generative method mutationable #2	2.0101e+11
13	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	2.21875e+11

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
14	Generative method non mutationable a lot #5	2.40556e+11
15	Generative method mutationable #1	2.4117e+11
16	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	2.63259e+11
17	Generative method mutationable #5	2.64126e+11
18	Generative method non mutationable a lot #1	2.70369e+11
19	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	2.70442e+11
20	Generative method non mutationable a lot #4	2.72099e+11
21	Gradient method ‘2,0,0’ init	5.14572e+13

– 1140 секунд (див. табл. 3.13);

Таблиця 3.13 – Оцінка при $r(t)\neq const$, t_n=1140 с, налаштовано для значення 20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	389988
2	Generative method non mutationable a lot #2	464403
3	Gradient method ‘10,0,0’ init	498073
4	Generative method non mutationable a lot #3	582366
5	Gradient method ‘7,0,0’ init	802003
6	Zeingler-Nichols method 1 step k_crit	820988
7	Zeingler-Nichols method 0.1 step k_crit	1.44995e+06

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	8.21382e+06
9	Gradient method ‘5,0,0’ init	6.17015e+11
10	Gradient method ‘1,0,0’ init	6.98871e+11
11	Generative method mutationable #4	9.19363e+11
12	Generative method mutationable #1	1.11513e+12
13	Generative method mutationable #2	1.13719e+12
14	Generative method non mutationable a lot #5	1.14763e+12
15	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	1.22927e+12
16	Generative method non mutationable a lot #4	1.27492e+12
17	Generative method non mutationable a lot #1	1.29049e+12
18	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	1.29464e+12
19	Generative method mutationable #5	1.30054e+12
20	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	1.31692e+12
21	Gradient method ‘2,0,0’ init	4.19276e+14

3.2.3 Аналогічні рейтинги вже зі змінним навантаженням на систему

За умовами незмінного сигналу завдання для котрого проводилися налаштування:

– 360 секунд (див. табл. 3.14);

Таблиця 3.14 – Оцінка при *r(t)=const, M(ω)≠const,* t_n=360 с, налаштовано для *r(t)*=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	7933.8
2	Generative method non mutationable a lot #3	12125.3
3	Generative method non mutationable a lot #2	12462.2
4	Gradient method ‘10,0,0’ init	14415.8
5	Gradient method ‘7,0,0’ init	17432.9
6	Zeingler-Nichols method 1 step k_crit	166038
7	Zeingler-Nichols method 0.1 step k_crit	395448
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	7.14289e+06
9	Gradient method ‘5,0,0’ init	5.9584e+09
10	Gradient method ‘1,0,0’ init	1.803e+10
11	Generative method non mutationable a lot #1	3.57176e+10
12	Generative method non mutationable a lot #4	3.86145e+10
13	Generative method mutationable #4	3.88705e+10
14	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	3.92763e+10
15	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	4.18367e+10
16	Generative method mutationable #2	4.50287e+10
17	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	4.98165e+10
18	Generative method mutationable #1	5.31742e+10
19	Generative method non mutationable a lot #5	5.38634e+10

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
20	Generative method mutationable #5	6.63403e+10
21	Gradient method ‘2,0,0’ init	6.17963e+12

– 720 секунд (див. табл. 3.15);

Таблиця 3.15 – Оцінка при *r(t)=const, M(ω)≠const,* t_n=720 с, налаштовано для *r(t)*=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Gradient method ‘10,0,0’ init	4745.66
2	Gradient method ‘7,0,0’ init	5726.89
3	Generative method mutationable #3	6380.64
4	Gradient method ‘5,0,0’ init	7411.42
5	Generative method non mutationable a lot #3	8551.93
6	Generative method mutationable #2	10548.9
7	Generative method non mutationable a lot #2	11346.3
8	Generative method mutationable #1	11699.1
9	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	18292.7
10	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	22057.6
11	Generative method non mutationable a lot #4	26549
12	Generative method non mutationable a lot #1	27236.2
13	Generative method mutationable #4	29190

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
14	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	30417.2
15	Gradient method ‘1,0,0’ init	39805.4
16	Zeingler-Nichols method 1 step k_crit	67629
17	Zeingler-Nichols method 0.1 step k_crit	98415.7
18	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	1.08994e+06
19	Generative method non mutationable a lot #5	9.08211e+07
20	Generative method mutationable #5	6.27809e+08
21	Gradient method ‘2,0,0’ init	6.2676e+11

Двох різних незмінних сигналів завдання не для котрих проводилося налаштування та середнє значення між ними.

– 360 секунд перше значення (див. табл. 3.16);

Таблиця 3.16 – Оцінка при $r(t)=const$, $M(\omega)\neq const$, t_n=360 с, налаштовано для $0.4r(t)$=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	78484.9
2	Gradient method ‘10,0,0’ init	97479.7
3	Generative method non mutationable a lot #2	99333.1
4	Generative method non mutationable a lot #3	114749
5	Gradient method ‘7,0,0’ init	150177
6	Zeingler-Nichols method 1 step k_crit	236840

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
7	Zeingler-Nichols method 0.1 step k_crit	431264
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	6.6484e+06
9	Gradient method ‘5,0,0’ init	1.85535e+10
10	Gradient method ‘1,0,0’ init	2.46394e+10
11	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	3.36734e+10
12	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	3.97422e+10
13	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	4.15997e+10
14	Generative method mutationable #4	4.17452e+10
15	Generative method non mutationable a lot #1	4.18692e+10
16	Generative method mutationable #2	4.42951e+10
17	Generative method non mutationable a lot #5	4.68945e+10
18	Generative method mutationable #5	5.02991e+10
19	Generative method mutationable #1	5.63382e+10
20	Generative method non mutationable a lot #4	6.13424e+10
21	Gradient method ‘2,0,0’ init	6.18729e+12

– 720 секунд перше значення (див. табл. 3.17);

Таблиця 3.17 – Оцінка при *r(t)=const, M(ω)≠const,* t_n=720 с, налаштовано для *0.4r(t)*=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Gradient method ‘10,0,0’ init	46803.5
2	Gradient method ‘7,0,0’ init	50753.1
3	Generative method non mutationable a lot #3	59759.2
4	Gradient method ‘5,0,0’ init	62885.8
5	Generative method mutationable #3	65995
6	Generative method non mutationable a lot #2	79743.6
7	Generative method mutationable #2	117546
8	Generative method mutationable #1	157813
9	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	175293
10	Generative method mutationable #4	182988
11	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	183355
12	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	195203
13	Generative method non mutationable a lot #1	204994
14	Gradient method ‘1,0,0’ init	209921
15	Zeingler-Nichols method 1 step k_crit	284819
16	Generative method non mutationable a lot #4	320986
17	Zeingler-Nichols method 0.1 step k_crit	374239
18	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	1.85982e+06
19	Generative method non mutationable a lot #5	1.0386e+08
20	Generative method mutationable #5	6.63794e+08
21	Gradient method ‘2,0,0’ init	6.30257e+11

– 360 секунд друге значення (див. табл. 3.18);

Таблиця 3.18 – Оцінка при *r(t)=const, M(ω)≠const,* t_n=360 с, налаштовано для *0.2r(t)*=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Gradient method ‘10,0,0’ init	457724
2	Zeingler-Nichols method 1 step k_crit	666458
3	Gradient method ‘7,0,0’ init	889542
4	Generative method non mutationable a lot #3	1.00809e+06
5	Zeingler-Nichols method 0.1 step k_crit	1.03424e+06
6	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	6.65404e+06
7	Generative method mutationable #3	2.04577e+10
8	Gradient method ‘5,0,0’ init	2.26438e+10
9	Gradient method ‘1,0,0’ init	2.59467e+10
10	Generative method non mutationable a lot #2	3.20344e+10
11	Generative method mutationable #2	3.84074e+10
12	Generative method non mutationable a lot #1	4.61027e+10
13	Generative method non mutationable a lot #5	4.62812e+10
14	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	4.67316e+10
15	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	4.90011e+10
16	Generative method mutationable #4	4.99198e+10
17	Generative method mutationable #1	5.03485e+10

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
18	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	5.15883e+10
19	Generative method non mutationable a lot #4	5.73611e+10
20	Generative method mutationable #5	6.92311e+10
21	Gradient method ‘2,0,0’ init	6.20007e+12

– 720 секунд друге значення (див. табл. 3.19);

Таблиця 3.19 – Оцінка при $r(t)=const$, $M(\omega)\neq const$, t_n=720 с, налаштовано для $0.2r(t)$=20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Gradient method ‘10,0,0’ init	316008
2	Gradient method ‘7,0,0’ init	362191
3	Generative method non mutationable a lot #3	460112
4	Gradient method ‘5,0,0’ init	467443
5	Generative method mutationable #3	482973
6	Generative method non mutationable a lot #2	615544
7	Gradient method ‘1,0,0’ init	836975
8	Generative method mutationable #4	881995
9	Generative method mutationable #2	933833
10	Zeingler-Nichols method 1 step k_crit	1.07774e+06
11	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	1.21753e+06
12	Zeingler-Nichols method 0.1 step k_crit	1.38498e+06

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
13	Generative method mutationable #1	1.57439e+06
14	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	1.94974e+06
15	Generative method non mutationable a lot #1	2.03124e+06
16	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	2.06661e+06
17	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	4.69737e+06
18	Generative method non mutationable a lot #4	3.55451e+07
19	Generative method non mutationable a lot #5	1.31872e+08
20	Generative method mutationable #5	6.81605e+08
21	Gradient method ‘2,0,0’ init	6.36114e+11

Змінний сигнал зі значеннями завдання не для котрих відбувалося налаштування.

– 360 секунд (див. табл. 3.20);

Таблиця 3.20 – Оцінка при $r(t)\neq const$, $M(\omega)\neq const$, t_n=360 с, налаштовано для значення 20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	390011
2	Generative method non mutationable a lot #2	464792
3	Gradient method ‘10,0,0’ init	493856

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
4	Generative method non mutationable a lot #3	578055
5	Gradient method ‘7,0,0’ init	765018
6	Zeingler-Nichols method 1 step k_crit	808379
7	Zeingler-Nichols method 0.1 step k_crit	1.2953e+06
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	9.6949e+06
9	Gradient method ‘5,0,0’ init	1.78783e+10
10	Gradient method ‘1,0,0’ init	2.35183e+10
11	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	3.37411e+10
12	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	3.99902e+10
13	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	4.11928e+10
14	Generative method non mutationable a lot #1	4.13851e+10
15	Generative method mutationable #4	4.171e+10
16	Generative method mutationable #2	4.38918e+10
17	Generative method non mutationable a lot #5	4.60766e+10
18	Generative method mutationable #5	5.11235e+10
19	Generative method mutationable #1	5.55545e+10
20	Generative method non mutationable a lot #4	6.07955e+10
21	Gradient method ‘2,0,0’ init	6.19856e+12

– 720 секунд (див. табл. 3.21).

Таблиця 3.21 – Оцінка при *r(t)≠const, M(ω)≠const,* t_n=720 с, налаштовано для значення 20

Рейтинг	Певний випадок налаштувань	Значення фітнес-функції
1	Generative method mutationable #3	389988
2	Generative method non mutationable a lot #2	464403
3	Gradient method ‘10,0,0’ init	498065
4	Generative method non mutationable a lot #3	582379
5	Gradient method ‘7,0,0’ init	804565
6	Zeingler-Nichols method 1 step k_crit	820910
7	Zeingler-Nichols method 0.1 step k_crit	1.45637e+06
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	7.92699e+06
9	Gradient method ‘5,0,0’ init	1.18716e+11
10	Gradient method ‘1,0,0’ init	1.53943e+11
11	Generative method mutationable #4	1.92876e+11
12	Generative method mutationable #2	2.0101e+11
13	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	2.21875e+11
14	Generative method non mutationable a lot #5	2.40556e+11
15	Generative method mutationable #1	2.4117e+11
16	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	2.63259e+11
17	Generative method mutationable #5	2.64126e+11
18	Generative method non mutationable a lot #1	2.70369e+11
19	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	2.70442e+11
20	Generative method non mutationable a lot #4	2.72099e+11
21	Gradient method ‘2,0,0’ init	5.14572e+13

3.2.4 Стабільність налаштувань як різниця значень фітнес-функцій між найдовшим проміжком часу вимірювань та другим за тривалістю проміжком часу

Стабільність оцінюватимемо за двома випадками експериментів такими як:

- за умов незмінного сигналу завдання для котрого проводилися налаштування;
- з постійним навантаженням на систему змінний сигнал зі значеннями завдання не для котрих відбувалося налаштування;
- зі змінним навантаженням на систему змінний сигнал зі значеннями завдання не для котрих відбувалося налаштування.

Стабільність регулювання розраховуватиметься наступним чином (див. вираз 3.1):

$$fi(t2) - fi(t1) = \sum_0^{t2}(r(t) - u(t))^2 - \sum_0^{t1}(r(t) - u(t))^2 \qquad (3.1)$$

, де$\sum_0^{t2}(r(t) - u(t))^2$ величина відхилення протягом усього часу;

$\sum_0^{t1}(r(t) - u(t))^2$ величина відхилення протягом половини від часу попереднього випадку, протягом часу коли регульована величина з нульового значення виходить на завданий рівень;

А їх різниця дорівнює величині відхилення коли вже перехідні процеси завершилися, після половини усього проміжку часу та враховує вона лише останню частину, де виконується підтримка певного заданого значення й чим краще це відбувається тим нижче буде це значення.

Незмінний сигнал завдання для котрого проводилися налаштування (див. табл. 3.22):

Таблиця 3.22 – Різниця значень фітнес функцій між пусками табл. 3.4 та 3.2

Рейтинг	Певний випадок налаштувань	Різниця значеніь фітнес-функції
1	Generative method mutationable #3	0,00
2	Generative method non mutationable a lot #3	0,00
3	Generative method non mutationable a lot #2	0,00
4	Gradient method ‘10,0,0’ init	0,00
5	Gradient method ‘7,0,0’ init	0,00
6	Zeingler-Nichols method 1 step k_crit	0,00
7	Zeingler-Nichols method 0.1 step k_crit	0,00
8	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	0,00
9	Gradient method ‘5,0,0’ init	4,73099E+11
10	Gradient method ‘1,0,0’ init	5,61922E+11
11	Generative method non mutationable a lot #1	8,30663E+11
12	Generative method mutationable #2	8,33753E+11
13	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	8,40693E+11
14	Generative method mutationable #1	8,48259E+11
15	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	8,92526E+11
16	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	9,09264E+11
17	Generative method mutationable #4	9,17823E+11
18	Generative method non mutationable a lot #5	9,25918E+11
19	Generative method non mutationable a lot #4	1,07521E+12
20	Generative method mutationable #5	1,22322E+12
21	Gradient method ‘2,0,0’ init	3,67583E+14

Змінний сигнал зі значеннями завдання не для котрих відбувалося налаштування з постійним навантаженням на систему (деякі різниці вийшли від'ємними, що пояснюється вірогідною похибкою розрахунків особливо за умови, що кожні показання це незалежні одне від одного пуски і замість від'ємних результатів будемо їх сприйняти прямуючими до нульових значень) (див. табл. 3.23).

Таблиця 3.23 – Різниця значень фітнес функцій між пусками табл. 3.13 та 3.11.

Рейтинг	Певний випадок налаштувань	Різниця значень фітнес-функції
1	Zeingler-Nichols method 0.1 step k_crit	-6420
2	Gradient method '7,0,0' init	-2562
3	Generative method non mutationable a lot #3	-13
4	Gradient method '10,0,0' init	8
5	Generative method mutationable #3	0
6	Generative method non mutationable a lot #2	0
7	Zeingler-Nichols method 1 step k_crit	78
8	Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and x3' at '0.5 step k_crit' to avrg pass	286830
9	Gradient method '5,0,0' init	4,98299E+11
10	Gradient method '1,0,0' init	5,44928E+11
11	Generative method mutationable #4	7,26487E+11
12	Generative method mutationable #1	8,7396E+11
13	Generative method non mutationable a lot #5	9,07074E+11
14	Generative method mutationable #2	9,3618E+11
15	Generative method non mutationable a lot #4	1,00282E+12
16	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	1,0074E+12

Рейтинг	Певний випадок налаштувань	Різниця значень фітнес-функції
17	Generative method non mutationable a lot #1	1,02012E+12
18	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	1,0242E+12
19	Generative method mutationable #5	1,03641E+12
20	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	1,05366E+12
21	Gradient method ‘2,0,0’ init	3,67819E+14

Змінний сигнал зі значеннями завдання не для котрих відбувалося налаштування зі змінним навантаженням на систему (див. табл. 3.24).

Таблиця 3.24 – Різниця значень фітнес функцій між пусками табл. 3.20 та 3.21

Рейтинг	Певний випадок налаштувань	Різниця значеніь фітнес-функції
1	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass	-1767910
2	Generative method non mutationable a lot #2	-389
3	Generative method mutationable #3	-23,00
4	Gradient method ‘10,0,0’ init	4209
5	Generative method non mutationable a lot #3	4324
6	Zeingler-Nichols method 1 step k_crit	12531
7	Gradient method ‘7,0,0’ init	39547
8	Zeingler-Nichols method 0.1 step k_crit	161070
9	Gradient method ‘5,0,0’ init	1,0084E+11
10	Gradient method ‘1,0,0’ init	1,3042E+11

Рейтинг	Певний випадок налаштувань	Різниця значеніь фітнес-функції
11	Generative method mutationable #4	1,5117E+11
12	Generative method mutationable #2	1,5712E+11
13	Generative method mutationable #1	1,8562E+11
14	Gradient method with init by the Zeingler-Nichols method 1 step k_krit	1,8813E+11
15	Generative method non mutationable a lot #5	1,9448E+11
16	Generative method non mutationable a lot #4	2,113E+11
17	Generative method mutationable #5	2,13E+11
18	Gradient method with init by the Zeingler-Nichols method 0.5 step k_krit	2,2327E+11
19	Generative method non mutationable a lot #1	2,2898E+11
20	Gradient method with init by the Zeingler-Nichols method 0.1 step k_krit	2,2925E+11
21	Gradient method ‘2,0,0’ init	4,5259E+13

3.3 Економічна ефективність впровадження кожного з досліджених регуляторів

3.3.1 Задовільні налаштування за швидкістю та стабільністю

Протягом застосування усіх можливих типів метрик, що застосовувались для чисельного оцінювання експериментально отриманих результатів налаштувань різними алгоритмами з різними конфігураціями можна із впевненістю заявити що ми мали певну низку найкращих результатів налаштувань, що від інших відрізнялись на істотний порядок, та в числовому вираженні метрик складало відношення більше ніж у тисячу в кожній метриці.

Навіть якщо ми складатимемо різні списки за різними показниками такими як:

- за швидкістю виходу на задану величину регулювання;
- за стабільністю підтримки заданої величину регулювання після виходу на неї.

То ми отримуємо завжди один й той же список налаштувань:

- швидкість виходу на величину (див. табл. 3.25):

Таблиця 3.25 – Підсумки за усіма метриками.

Кількість ідеальних застосувань згідно метрикам	Найменування експериментів в результаті котрих налаштування були отримані
9	Zeingler-Nichols method 0.1 step k_crit
8	Gradient method ‘10,0,0’ init
8	Gradient method ‘7,0,0’ init
8	Zeingler-Nichols method 1 step k_crit
8*	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass
7	Generative method non mutationable a lot #3
6	Generative method mutationable #3
6	Generative method non mutationable a lot #2

- стабільність підтримки величини (див. табл. 3.26):

Таблиця 3.26 – Підсумки за стабільністю налаштувань.

Кількість ідеальних застосувань згідно метрикам	Найменування експериментів в результаті котрих налаштування були отримані
3	Zeingler-Nichols method 0.1 step k_crit

Кількість ідеальних застосувань згідно метрикам	Найменування експериментів в результаті котрих налаштування були отримані
3	Gradient method ‘7,0,0’ init
3	Generative method non mutationable a lot #3
3	Gradient method ‘10,0,0’ init
3	Generative method mutationable #3
3	Generative method non mutationable a lot #2
3	Zeingler-Nichols method 1 step k_crit
3*	Zeingler-Nichols method ‘x0.5 and x2’+’x0.33 and x3’ at ‘0.5 step k_crit’ to avrg pass

Де «*» як позначка, що цей вид налаштувань не один раз займав пограничне положення.

3.3.2 Порівняння типів регулятора за ефективністю та аналіз впливу конфігурацій алгоритмів використовуваних ними

Тепер проаналізуємо який з різновидів регулятора виявився згідно проведеним дослідженням більш доцільним за інші, які використовувані алгоритми регуляторами робили їх налаштування більш якісними та яким чином на їх якість впливали зміни в конфігураціях алгоритмів.

Генетичний алгоритм має лише три з десяти задовільних для використання отриманих налаштувань регулятора та при цьому має найбільшу кількість викликів фітнес-функції.

Це підтверджує нульову гіпотезу про нульові перспективи генетичного алгоритму у використанні в налаштуванні ПІД-регуляторів САР.

Метод градієнтного спуску з п’яти різними налаштуваннями за різними стартовими точками пошуку відповіді задовільними виявилося лише два випадки зі стартовими точками { 10, 0, 0 } та { 7, 0, 0 }, тобто у випадках за

найбільшими пропорційними коефіцієнтами як стартові. І це за умов що він так само робив зовсім не малу кількість викликів фітнес-функції проте й менше ніж генетичний алгоритм більш ніж у два рази.

Метод Циглера-Ніколса виявився єдиним з усіх методів налаштувань, чиї результаті задовільні більше ніж в половині випадків налаштування з різноманітними конфігураціями алгоритму, а саме п’ять з шести налаштувань завершилося з успіхом та в результаті мало задовільними усі п’ять з шести успішно завершені налаштування, де двоє налаштувань привели до одних й тих же результатів. І при цьому має найменший еквівалент викликам фітнес-функцій у порівнянні з іншими методами.

А також варто зазначити, що доволі важливим є вибір швидкості наростання пропорційного коефіцієнту, що значно впливає на швидкість налаштування та на його якість, проте його некоректний вибір має куди менш фатальні наслідки за інше методи налаштування. Заради економічності процесу налаштування бажано обирати швидкість по-вище тому, як згідно проведеним дослідженням якість налаштувань зростає зовсім несуттєво, проте занижена швидкість викликає витрати (еквівалент кількості визову фітнес-функції) до десяти разів(!) більше за необхідні.

Комбінація не виправдала сподівань, за якою була сформульована нульова гіпотеза про можливу точність налаштувань, що наразі відхилена за результатами проведених досліджень, тому що з чотирьох різних випадків налаштувань ми не отримали ні одного задовільного результата.

Залишилося лише візуально та схематично відобразити у вигляді рейтингу регулятори та їх алгоритмів конфігурації в табл. 3.27.

Таблиця 3.27 – Коротке відображення інтерпретації результатів досліджень.

<table>
<tr><th>Викликів фітнес-функції (Витрати)</th><th>Конфігурації алгоритмів</th><th>Різновид налаштовувала регулятора за використовуваним алгоритмом</th><th>Успішність застосування (Якість)</th></tr>
<tr><td>3</td><td>Оптимальний крок</td><td rowspan="3">Метод Циглера-Ніколса</td><td rowspan="3">80%</td></tr>
<tr><td>5</td><td>Завеликий крок</td></tr>
<tr><td>33</td><td>Замалий крок</td></tr>
<tr><td>20</td><td colspan="2">Метод градієнтного спуску</td><td>40%</td></tr>
<tr><td>50</td><td colspan="2">Генетичний алгоритм</td><td>30%</td></tr>
<tr><td>23, 25, 55</td><td colspan="2">Метод градієнтного спуску на базі методу Циглера-Ніколса</td><td>0%</td></tr>
</table>

ВИСНОВКИ

Протягом досліджень не було виявлено можливостей ефективно застосувати методи вирішення проблем машинного навчання у вирішенні проблеми налаштування ПІД-регулятора, як мінімум через те, що складнощі налаштування ПІД-регулятора заміняються складнощами налаштування чисельних алгоритмів пошуку мінімуму функції найоптимальнішим шляхом, а як максимум поступаються традиційним методам налаштування в ефективності, що визначається відношенням якості налаштування до витрат (часу) налаштування навіть якщо знехтувати проблемами налаштування чисельних алгоритмів.

Протягом досліджень було знайдено підтвердження гіпотезі про можливість через автоматизування мануальних алгоритмів налаштування зменшити економічне навантаження на підприємство, що використовує ПІД-регулятор в колах САР. Тому у подальших роботах варто було б продовжувати розвиток саме за цим напрямом, а саме параметричної оптимізації. Саме цей напрям досліджень показав себе найперспективнішим, проте також має деякі мінуси такі як нездатність налаштувати регулятор у випадку занадто далекої від передбачуваної поведінки системи регулювання або процесу. Але в такі випадки завжди спеціалісти мають бути в зоні доступу, або ефективного реагування, попри навіть те, що їх втручання здатне бути зведено до мінімуму. Випадками виключеннями ще й можуть бути в час виникнення специфічних потреб від регулювання, але ці питання й близько не розглядались в цій роботі через те, що вони виходять за рамки її теми та вирішуваної проблеми. Проте як напрямок є досі доволі перспективним на погляд автора цієї роботи.

Отже, в результаті проведених досліджень можна з впевненістю стверджувати, що існує потенціал замінити роль спеціаліста з автоматизованих систем керування технологічними процесами, або спростити її як мінімум, за допомогою автоматизації мануальних методів налаштування призначених спеціально для нього, прикладом з яких являє собою одним з різновидів

параметричної оптимізації та першим методом налаштування ПІД-регуляторів з часів першої половини двадцятого століття, котрий в імплементації програмного алгоритму справляється суттєво краще за евристичні методи, чисельних або стохастичних методів як за показниками економічності так і за показниками якості.

ПЕРЕЛІК ПОСИЛАНЬ

1. «ПИД-регуляторы: принципы построения и модификации» / Виктор Денисенко // «В записную книжку инженера» – СТА 4/2006.

2. «PID without a PhD» / Tim Wescott FLIR // Systems. Режим доступу до джерела: http://surl.li/jwxoia.

3. «Розробка лабораторного стенду для дослідження роботи дискретного ПІД-регулятора на базі ARDUINO» / Ткач Павло // Криворізький національний університет – 2023.

4. Trench, William F., "Elementary Differential Equations with Boundary Value Problems" (2013). Faculty Authored and Edited Books & CDs. 9. https://digitalcommons.trinity.edu/mono/9

5. Моделювння електромеханічних систем: Підручник / Чорний О.П., Луговой А.В., Д.Й.Родькін, Сисюк Г.Ю., Садовой О.В.– Кременчук, 2001. – 410 с.

6. «Програмна система інтелектуального формування стартових популяцій» / Рибніков Владислав // Київський політехнічний інститут імені Ігоря Сікорського – 2021.

7. «Градієнтні методи розв'язання задач безумовної оптимізації» конспект лекцій (Частина 3) з курсу "Методи синтезу та оптимізації" для студентів базового напрямку "Комп'ютерні науки " / В.М.Теслюк // Національний університет "Львівська політехніка" – 2013.

8. «Chemical Process Dynamics and Controls» Open Textbook / Peter J. Woolf // University of Michigan – 2009.

9. Теорія автоматичного управління: Навчальний посібник [Електронний ресурс] : навч. посіб. для студ. спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології», освітньо-професійна програма «Автоматизація та комп'ютерно-інтегровані технології кібер-енергетичних систем»; уклад.: О. Й. Штіфзон, П. В.

Новіков, В.П. Бунь. – Електронні текстові дані (1 файл: 2,2 Мбайт). – Київ : КПІ ім.

10. Попович М. Г., Ковальчук О. В. П58 Теорія автоматичного керування: Підручник. — 2-ге вид., перероб. і доп. ¬ К.: Либідь, 2007. ¬ 656с.

11. Азарян А.А., Трачук А.А., Гриценко А.М., Швець Д.В. // Методичні вказівки з виконання лабораторних робіт з дисципліни "Математичне моделювання систем та процесів", для студентів спеціальності 121 – «Інженерія програмного забезпечення». – Кривий Ріг – 2023.

12. Матренин П.В. М 346 Методы стохастической оптимизации: учебное пособие / П.В. Матренин, М.Г. Гриф, В.Г. Секаев. – Новосибирск: Изд-во НГТУ, 2016. – 67 с.

13. Азарян А.А., Трачук А.А., Гриценко А.М., Швець Д.В. // Методичні вказівки з виконання лабораторних робіт з дисципліни «Основи наукових досліджень», для студентів спеціальності 121 – «Інженерія програмного забезпечення». – Кривий Ріг – 2023.

14. Якість програмного забезпечення та тестування: базовий курс. Навчальний посібник / За ред. Крепич С.Я., Співак І.Я. / для бакалаврів галузі знань 12 «Інформаційні технології» спеціальності 121 «Інженерія програмного забезпечення». – Тернопіль: ФОП Паляниця В.А., 2020. – 478с.

15. Гамма Э., Хелм Р., Джонсон Р ., Влиссидес Дж. П75 Паттерны объектно-ориентированного проектирования. — СПб.: Питер, 2020. — 448 с.: ил. — (Серия «Библиотека программиста»).

Додаток А

Зміст файлу Evaluation_report.txt після пуску оцінювання налаштувань:

```
Results #1:
1:          7933.8            Generative method mutationable #3        8.8
      2.5         5.2
2:          12125.3           Generative method non mutationable a lot #3
      5.2         1.2         2.8
3:          12462.2           Generative method non mutationable a lot #2
      5.1         2.2         6.7
4:          14415.8           Gradient method '10,0,0' init      10.474
      0.568234          0.571808
5:          17432.9           Gradient method '7,0,0' init       7.69889
      0.876194          0.881272
6:          166038            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          395448            Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
8:          7.14289e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
9:          5.9584e+09        Gradient method '5,0,0' init       6.13157
      1.49139           1.30597
10:         1.803e+10         Gradient method '1,0,0' init       1.06758
      0.317347          0.579046
11:         3.57176e+10       Generative method non mutationable a lot #1
      3.9         3.5         5.3
12:         3.86145e+10       Generative method non mutationable a lot #4
      5.8         6.4         7.2
13:         3.88705e+10       Generative method mutationable #4        1.7
      0.8         0.3
14:         3.92763e+10       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
15:         4.18367e+10       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
16:         4.50287e+10       Generative method mutationable #2        9.1
      5           0
17:         4.98165e+10       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
18:         5.31742e+10       Generative method mutationable #1        9.4
      7           3.5
19:         5.38634e+10       Generative method non mutationable a lot #5
      4           4.3         0.1
20:         6.63403e+10       Generative method mutationable #5        0.5
      6.7         1.3
21:         6.17963e+12       Gradient method '2,0,0' init       8.38471
      -29403.1          -341273
The end of rating list


Results #2:
1:          78484.9           Generative method mutationable #3        8.8
      2.5         5.2
2:          97479.7           Gradient method '10,0,0' init      10.474
      0.568234          0.571808
3:          99333.1           Generative method non mutationable a lot #2
      5.1         2.2         6.7
4:          114749            Generative method non mutationable a lot #3
      5.2         1.2         2.8
```

```
5:          150177            Gradient method '7,0,0' init       7.69889
      0.876194          0.881272
6:          236840            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          431264            Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
8:          6.6484e+06        Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
9:          1.85535e+10       Gradient method '5,0,0' init       6.13157
      1.49139           1.30597
10:         2.46394e+10       Gradient method '1,0,0' init       1.06758
      0.317347          0.579046
11:         3.36734e+10       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
12:         3.97422e+10       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
13:         4.15997e+10       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
14:         4.17452e+10       Generative method mutationable #4         1.7
      0.8         0.3
15:         4.18692e+10       Generative method non mutationable a lot #1
      3.9         3.5         5.3
16:         4.42951e+10       Generative method mutationable #2         9.1
      5           0
17:         4.68945e+10       Generative method non mutationable a lot #5
      4           4.3         0.1
18:         5.02991e+10       Generative method mutationable #5         0.5
      6.7         1.3
19:         5.63382e+10       Generative method mutationable #1         9.4
      7           3.5
20:         6.13424e+10       Generative method non mutationable a lot #4
      5.8         6.4         7.2
21:         6.18729e+12       Gradient method '2,0,0' init       8.38471
      -29403.1          -341273
The end of rating list


Results #3:
1:          457724            Gradient method '10,0,0' init      10.474
      0.568234          0.571808
2:          666458            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
3:          889542            Gradient method '7,0,0' init       7.69889
      0.876194          0.881272
4:          1.00809e+06       Generative method non mutationable a lot #3
      5.2         1.2         2.8
5:          1.03424e+06       Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
6:          6.65404e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
7:          2.04577e+10       Generative method mutationable #3         8.8
      2.5         5.2
8:          2.26438e+10       Gradient method '5,0,0' init       6.13157
      1.49139           1.30597
9:          2.59467e+10       Gradient method '1,0,0' init       1.06758
      0.317347          0.579046
10:         3.20344e+10       Generative method non mutationable a lot #2
      5.1         2.2         6.7
11:         3.84074e+10       Generative method mutationable #2         9.1
      5           0
12:         4.61027e+10       Generative method non mutationable a lot #1
      3.9         3.5         5.3
```

```
13:         4.62812e+10       Generative method non mutationable a lot #5
      4           4.3         0.1
14:         4.67316e+10       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
15:         4.90011e+10       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
16:         4.99198e+10       Generative method mutationable #4         1.7
      0.8         0.3
17:         5.03485e+10       Generative method mutationable #1         9.4
      7           3.5
18:         5.15883e+10       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
19:         5.73611e+10       Generative method non mutationable a lot #4
      5.8         6.4         7.2
20:         6.92311e+10       Generative method mutationable #5         0.5
      6.7         1.3
21:         6.20007e+12       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #4:
1:          390011            Generative method mutationable #3         8.8
      2.5         5.2
2:          464792            Generative method non mutationable a lot #2
      5.1         2.2         6.7
3:          493856            Gradient method '10,0,0' init       10.474
      0.568234          0.571808
4:          578055            Generative method non mutationable a lot #3
      5.2         1.2         2.8
5:          765018            Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
6:          808379            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          1.2953e+06        Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
8:          9.6949e+06        Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
9:          1.78783e+10       Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
10:         2.35183e+10       Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
11:         3.37411e+10       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
12:         3.99902e+10       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
13:         4.11928e+10       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
14:         4.13851e+10       Generative method non mutationable a lot #1
      3.9         3.5         5.3
15:         4.171e+10         Generative method mutationable #4         1.7
      0.8         0.3
16:         4.38918e+10       Generative method mutationable #2         9.1
      5           0
17:         4.60766e+10       Generative method non mutationable a lot #5
      4           4.3         0.1
18:         5.11235e+10       Generative method mutationable #5         0.5
      6.7         1.3
19:         5.55545e+10       Generative method mutationable #1         9.4
      7           3.5
20:         6.07955e+10       Generative method non mutationable a lot #4
      5.8         6.4         7.2
```

```
21:         6.19856e+12       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #5:
1:          7933.8            Generative method mutationable #3         8.8
      2.5         5.2
2:          12125.3           Generative method non mutationable a lot #3
      5.2         1.2         2.8
3:          12462.2           Generative method non mutationable a lot #2
      5.1         2.2         6.7
4:          14415.8           Gradient method '10,0,0' init       10.474
      0.568234          0.571808
5:          17432.9           Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
6:          166038            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          395448            Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
8:          7.14284e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass           0.175       0.0190217
      1.0626
9:          9.04144e+10       Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
10:         1.44855e+11       Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
11:         1.79127e+11       Generative method mutationable #4         1.7
      0.8         0.3
12:         1.87097e+11       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
13:         2.06264e+11       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
14:         2.31281e+11       Generative method mutationable #1         9.4
      7           3.5
15:         2.42982e+11       Generative method non mutationable a lot #5
      4           4.3         0.1
16:         2.45416e+11       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
17:         2.51947e+11       Generative method mutationable #2         9.1
      5           0
18:         2.58673e+11       Generative method non mutationable a lot #4
      5.8         6.4         7.2
19:         2.67477e+11       Generative method non mutationable a lot #1
      3.9         3.5         5.3
20:         3.20685e+11       Generative method mutationable #5         0.5
      6.7         1.3
21:         5.13795e+13       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #6:
1:          78484.9           Generative method mutationable #3         8.8
      2.5         5.2
2:          97479.7           Gradient method '10,0,0' init       10.474
      0.568234          0.571808
3:          99333.1           Generative method non mutationable a lot #2
      5.1         2.2         6.7
4:          114749            Generative method non mutationable a lot #3
      5.2         1.2         2.8
5:          150177            Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
```

```
6:          236840          Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          431264          Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
8:          6.64835e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
9:          1.16601e+11       Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
10:         1.53557e+11       Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
11:         1.93005e+11       Generative method mutationable #4         1.7
      0.8         0.3
12:         2.00711e+11       Generative method mutationable #2         9.1
      5           0
13:         2.20806e+11       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
14:         2.4108e+11        Generative method non mutationable a lot #5
      4           4.3         0.1
15:         2.41898e+11       Generative method mutationable #1         9.4
      7           3.5
16:         2.62017e+11       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
17:         2.63236e+11       Generative method mutationable #5         0.5
      6.7         1.3
18:         2.69486e+11       Generative method non mutationable a lot #1
      3.9         3.5         5.3
19:         2.69576e+11       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
20:         2.73378e+11       Generative method non mutationable a lot #4
      5.8         6.4         7.2
21:         5.14109e+13       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #7:
1:          457724            Gradient method '10,0,0' init       10.474
      0.568234          0.571808
2:          666458            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
3:          889542            Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
4:          1.00809e+06       Generative method non mutationable a lot #3
      5.2         1.2         2.8
5:          1.03424e+06       Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
6:          6.65399e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
7:          1.39424e+11       Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
8:          1.4784e+11        Generative method mutationable #3         8.8
      2.5         5.2
9:          1.52826e+11       Generative method non mutationable a lot #2
      5.1         2.2         6.7
10:         1.65548e+11       Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
11:         2.15131e+11       Generative method mutationable #4         1.7
      0.8         0.3
12:         2.23251e+11       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
13:         2.2651e+11        Generative method mutationable #1         9.4
      7           3.5
```

```
14:         2.30718e+11       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
15:         2.3131e+11        Generative method non mutationable a lot #1
      3.9         3.5         5.3
16:         2.31708e+11       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
17:         2.57062e+11       Generative method mutationable #2         9.1
      5           0
18:         2.76725e+11       Generative method non mutationable a lot #4
      5.8         6.4         7.2
19:         2.7848e+11        Generative method non mutationable a lot #5
      4           4.3         0.1
20:         3.19554e+11       Generative method mutationable #5         0.5
      6.7         1.3
21:         5.14633e+13       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #8:
1:          389988            Generative method mutationable #3         8.8
      2.5         5.2
2:          464403            Generative method non mutationable a lot #2
      5.1         2.2         6.7
3:          498065            Gradient method '10,0,0' init       10.474
      0.568234          0.571808
4:          582379            Generative method non mutationable a lot #3
      5.2         1.2         2.8
5:          804565            Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
6:          820910            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          1.45637e+06       Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
8:          7.92699e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
9:          1.18716e+11       Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
10:         1.53943e+11       Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
11:         1.92876e+11       Generative method mutationable #4         1.7
      0.8         0.3
12:         2.0101e+11        Generative method mutationable #2         9.1
      5           0
13:         2.21875e+11       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
14:         2.40556e+11       Generative method non mutationable a lot #5
      4           4.3         0.1
15:         2.4117e+11        Generative method mutationable #1         9.4
      7           3.5
16:         2.63259e+11       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
17:         2.64126e+11       Generative method mutationable #5         0.5
      6.7         1.3
18:         2.70369e+11       Generative method non mutationable a lot #1
      3.9         3.5         5.3
19:         2.70442e+11       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
20:         2.72099e+11       Generative method non mutationable a lot #4
      5.8         6.4         7.2
21:         5.14572e+13       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list
```

```
Results #9:
1:          7933.8           Generative method mutationable #3        8.8
      2.5         5.2
2:          12125.3          Generative method non mutationable a lot #3
      5.2         1.2         2.8
3:          12462.2          Generative method non mutationable a lot #2
      5.1         2.2         6.7
4:          14415.8          Gradient method '10,0,0' init       10.474
      0.568234          0.571808
5:          17432.9          Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
6:          166038           Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          395448           Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
8:          7.14284e+06      Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
9:          5.63513e+11      Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
10:         7.06777e+11      Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
11:         1.02779e+12      Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
12:         1.07954e+12      Generative method mutationable #1        9.4
      7           3.5
13:         1.0857e+12       Generative method mutationable #2        9.1
      5           0
14:         1.09695e+12      Generative method mutationable #4        1.7
      0.8         0.3
15:         1.09814e+12      Generative method non mutationable a lot #1
      3.9         3.5         5.3
16:         1.09879e+12      Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
17:         1.15468e+12      Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
18:         1.1689e+12       Generative method non mutationable a lot #5
      4           4.3         0.1
19:         1.33388e+12      Generative method non mutationable a lot #4
      5.8         6.4         7.2
20:         1.5439e+12       Generative method mutationable #5        0.5
      6.7         1.3
21:         4.18962e+14      Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #10:
1:          78484.9          Generative method mutationable #3        8.8
      2.5         5.2
2:          97479.7          Gradient method '10,0,0' init       10.474
      0.568234          0.571808
3:          99333.1          Generative method non mutationable a lot #2
      5.1         2.2         6.7
4:          114749           Generative method non mutationable a lot #3
      5.2         1.2         2.8
5:          150177           Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
6:          236840           Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          431264           Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
```

```
8:          6.64835e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
9:          6.19321e+11       Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
10:         6.98053e+11       Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
11:         9.22533e+11       Generative method mutationable #4         1.7
      0.8         0.3
12:         1.10969e+12       Generative method mutationable #1         9.4
      7           3.5
13:         1.13962e+12       Generative method mutationable #2         9.1
      5           0
14:         1.15072e+12       Generative method non mutationable a lot #5
      4           4.3         0.1
15:         1.22879e+12       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
16:         1.26875e+12       Generative method non mutationable a lot #4
      5.8         6.4         7.2
17:         1.29641e+12       Generative method non mutationable a lot #1
      3.9         3.5         5.3
18:         1.30006e+12       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
19:         1.3005e+12        Generative method mutationable #5         0.5
      6.7         1.3
20:         1.31868e+12       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
21:         4.19089e+14       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #11:
1:          457724            Gradient method '10,0,0' init       10.474
      0.568234          0.571808
2:          666458            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
3:          889542            Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
4:          1.00809e+06       Generative method non mutationable a lot #3
      5.2         1.2         2.8
5:          1.03424e+06       Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
6:          6.65399e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
7:          7.15194e+11       Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
8:          7.2562e+11        Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
9:          8.03985e+11       Generative method mutationable #3         8.8
      2.5         5.2
10:         9.59222e+11       Generative method mutationable #4         1.7
      0.8         0.3
11:         9.97941e+11       Generative method non mutationable a lot #2
      5.1         2.2         6.7
12:         1.0588e+12        Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
13:         1.09009e+12       Generative method mutationable #1         9.4
      7           3.5
14:         1.09998e+12       Generative method non mutationable a lot #1
      3.9         3.5         5.3
15:         1.1001e+12        Generative method mutationable #2         9.1
      5           0
```

```
16:         1.10489e+12       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
17:         1.13114e+12       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
18:         1.32819e+12       Generative method non mutationable a lot #4
      5.8         6.4         7.2
19:         1.368e+12         Generative method non mutationable a lot #5
      4           4.3         0.1
20:         1.47221e+12       Generative method mutationable #5         0.5
      6.7         1.3
21:         4.19301e+14       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #12:
1:          389988            Generative method mutationable #3         8.8
      2.5         5.2
2:          464403            Generative method non mutationable a lot #2
      5.1         2.2         6.7
3:          498073            Gradient method '10,0,0' init       10.474
      0.568234          0.571808
4:          582366            Generative method non mutationable a lot #3
      5.2         1.2         2.8
5:          802003            Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
6:          820988            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          1.44995e+06       Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
8:          8.21382e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
9:          6.17015e+11       Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
10:         6.98871e+11       Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
11:         9.19363e+11       Generative method mutationable #4         1.7
      0.8         0.3
12:         1.11513e+12       Generative method mutationable #1         9.4
      7           3.5
13:         1.13719e+12       Generative method mutationable #2         9.1
      5           0
14:         1.14763e+12       Generative method non mutationable a lot #5
      4           4.3         0.1
15:         1.22927e+12       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
16:         1.27492e+12       Generative method non mutationable a lot #4
      5.8         6.4         7.2
17:         1.29049e+12       Generative method non mutationable a lot #1
      3.9         3.5         5.3
18:         1.29464e+12       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
19:         1.30054e+12       Generative method mutationable #5         0.5
      6.7         1.3
20:         1.31692e+12       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
21:         4.19276e+14       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #13:
```

```
1:          4745.66          Gradient method '10,0,0' init      10.474
      0.568234          0.571808
2:          5726.89          Gradient method '7,0,0' init       7.69889
      0.876194          0.881272
3:          6380.64          Generative method mutationable #3        8.8
      2.5         5.2
4:          7411.42          Gradient method '5,0,0' init       6.13157
      1.49139           1.30597
5:          8551.93          Generative method non mutationable a lot #3
      5.2         1.2         2.8
6:          10548.9          Generative method mutationable #2        9.1
      5           0
7:          11346.3          Generative method non mutationable a lot #2
      5.1         2.2         6.7
8:          11699.1          Generative method mutationable #1        9.4
      7           3.5
9:          18292.7          Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
10:         22057.6          Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
11:         26549       Generative method non mutationable a lot #4
      5.8         6.4         7.2
12:         27236.2          Generative method non mutationable a lot #1
      3.9         3.5         5.3
13:         29190       Generative method mutationable #4        1.7
      0.8         0.3
14:         30417.2          Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
15:         39805.4          Gradient method '1,0,0' init       1.06758
      0.317347          0.579046
16:         67629       Zeingler-Nichols method 1 step k_crit          0.8
      0.137339          3.0756
17:         98415.7          Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
18:         1.08994e+06      Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
19:         9.08211e+07      Generative method non mutationable a lot #5
      4           4.3         0.1
20:         6.27809e+08      Generative method mutationable #5        0.5
      6.7         1.3
21:         6.2676e+11       Gradient method '2,0,0' init       8.38471
      -29403.1          -341273
The end of rating list


Results #14:
1:          46803.5          Gradient method '10,0,0' init      10.474
      0.568234          0.571808
2:          50753.1          Gradient method '7,0,0' init       7.69889
      0.876194          0.881272
3:          59759.2          Generative method non mutationable a lot #3
      5.2         1.2         2.8
4:          62885.8          Gradient method '5,0,0' init       6.13157
      1.49139           1.30597
5:          65995       Generative method mutationable #3        8.8
      2.5         5.2
6:          79743.6          Generative method non mutationable a lot #2
      5.1         2.2         6.7
7:          117546           Generative method mutationable #2        9.1
      5           0
8:          157813           Generative method mutationable #1        9.4
      7           3.5
```

```
9:          175293            Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
10:         182988            Generative method mutationable #4         1.7
      0.8         0.3
11:         183355            Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
12:         195203            Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
13:         204994            Generative method non mutationable a lot #1
      3.9         3.5         5.3
14:         209921            Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
15:         284819            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
16:         320986            Generative method non mutationable a lot #4
      5.8         6.4         7.2
17:         374239            Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
18:         1.85982e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
19:         1.0386e+08        Generative method non mutationable a lot #5
      4           4.3         0.1
20:         6.63794e+08       Generative method mutationable #5         0.5
      6.7         1.3
21:         6.30257e+11       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #15:
1:          316008            Gradient method '10,0,0' init       10.474
      0.568234          0.571808
2:          362191            Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
3:          460112            Generative method non mutationable a lot #3
      5.2         1.2         2.8
4:          467443            Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
5:          482973            Generative method mutationable #3         8.8
      2.5         5.2
6:          615544            Generative method non mutationable a lot #2
      5.1         2.2         6.7
7:          836975            Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
8:          881995            Generative method mutationable #4         1.7
      0.8         0.3
9:          933833            Generative method mutationable #2         9.1
      5           0
10:         1.07774e+06       Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
11:         1.21753e+06       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
12:         1.38498e+06       Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
13:         1.57439e+06       Generative method mutationable #1         9.4
      7           3.5
14:         1.94974e+06       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
15:         2.03124e+06       Generative method non mutationable a lot #1
      3.9         3.5         5.3
16:         2.06661e+06       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
```

```
17:         4.69737e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
18:         3.55451e+07       Generative method non mutationable a lot #4
      5.8         6.4         7.2
19:         1.31872e+08       Generative method non mutationable a lot #5
      4           4.3         0.1
20:         6.81605e+08       Generative method mutationable #5         0.5
      6.7         1.3
21:         6.36114e+11       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #16:
1:          188301            Gradient method '10,0,0' init       10.474
      0.568234          0.571808
2:          207039            Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
3:          248625            Generative method non mutationable a lot #3
      5.2         1.2         2.8
4:          258416            Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
5:          264193            Generative method mutationable #3         8.8
      2.5         5.2
6:          328708            Generative method non mutationable a lot #2
      5.1         2.2         6.7
7:          485718            Generative method mutationable #2         9.1
      5           0
8:          652422            Generative method mutationable #1         9.4
      7           3.5
9:          719040            Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
10:         751088            Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
11:         764367            Generative method mutationable #4         1.7
      0.8         0.3
12:         800822            Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
13:         840969            Generative method non mutationable a lot #1
      3.9         3.5         5.3
14:         852854            Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
15:         1.08727e+06       Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
16:         1.33279e+06       Generative method non mutationable a lot #4
      5.8         6.4         7.2
17:         1.41009e+06       Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
18:         4.65041e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
19:         8.08954e+07       Generative method non mutationable a lot #5
      4           4.3         0.1
20:         7.58259e+08       Generative method mutationable #5         0.5
      6.7         1.3
21:         6.34078e+11       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #17:
1:          7933.8            Generative method mutationable #3         8.8
      2.5         5.2
```

```
2:          12125.3           Generative method non mutationable a lot #3
      5.2         1.2         2.8
3:          12462.2           Generative method non mutationable a lot #2
      5.1         2.2         6.7
4:          14415.8           Gradient method '10,0,0' init       10.474
      0.568234          0.571808
5:          17432.9           Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
6:          166038            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          395448            Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
8:          7.14289e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
9:          5.9584e+09        Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
10:         1.803e+10         Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
11:         3.57176e+10       Generative method non mutationable a lot #1
      3.9         3.5         5.3
12:         3.86145e+10       Generative method non mutationable a lot #4
      5.8         6.4         7.2
13:         3.88705e+10       Generative method mutationable #4         1.7
      0.8         0.3
14:         3.92763e+10       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
15:         4.18367e+10       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
16:         4.50287e+10       Generative method mutationable #2         9.1
      5           0
17:         4.98165e+10       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
18:         5.31742e+10       Generative method mutationable #1         9.4
      7           3.5
19:         5.38634e+10       Generative method non mutationable a lot #5
      4           4.3         0.1
20:         6.63403e+10       Generative method mutationable #5         0.5
      6.7         1.3
21:         6.17963e+12       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #18:
1:          78484.9           Generative method mutationable #3         8.8
      2.5         5.2
2:          97479.7           Gradient method '10,0,0' init       10.474
      0.568234          0.571808
3:          99333.1           Generative method non mutationable a lot #2
      5.1         2.2         6.7
4:          114749            Generative method non mutationable a lot #3
      5.2         1.2         2.8
5:          150177            Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
6:          236840            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          431264            Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
8:          6.6484e+06        Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
9:          1.85535e+10       Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
```

```
10:         2.46394e+10       Gradient method '1,0,0' init       1.06758
      0.317347          0.579046
11:         3.36734e+10       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
12:         3.97422e+10       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
13:         4.15997e+10       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
14:         4.17452e+10       Generative method mutationable #4        1.7
      0.8         0.3
15:         4.18692e+10       Generative method non mutationable a lot #1
      3.9         3.5         5.3
16:         4.42951e+10       Generative method mutationable #2        9.1
      5           0
17:         4.68945e+10       Generative method non mutationable a lot #5
      4           4.3         0.1
18:         5.02991e+10       Generative method mutationable #5        0.5
      6.7         1.3
19:         5.63382e+10       Generative method mutationable #1        9.4
      7           3.5
20:         6.13424e+10       Generative method non mutationable a lot #4
      5.8         6.4         7.2
21:         6.18729e+12       Gradient method '2,0,0' init       8.38471
      -29403.1          -341273
The end of rating list


Results #19:
1:          457724            Gradient method '10,0,0' init      10.474
      0.568234          0.571808
2:          666458            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
3:          889542            Gradient method '7,0,0' init       7.69889
      0.876194          0.881272
4:          1.00809e+06       Generative method non mutationable a lot #3
      5.2         1.2         2.8
5:          1.03424e+06       Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
6:          6.65404e+06       Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
7:          2.04577e+10       Generative method mutationable #3        8.8
      2.5         5.2
8:          2.26438e+10       Gradient method '5,0,0' init       6.13157
      1.49139           1.30597
9:          2.59467e+10       Gradient method '1,0,0' init       1.06758
      0.317347          0.579046
10:         3.20344e+10       Generative method non mutationable a lot #2
      5.1         2.2         6.7
11:         3.84074e+10       Generative method mutationable #2        9.1
      5           0
12:         4.61027e+10       Generative method non mutationable a lot #1
      3.9         3.5         5.3
13:         4.62812e+10       Generative method non mutationable a lot #5
      4           4.3         0.1
14:         4.67316e+10       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
15:         4.90011e+10       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
16:         4.99198e+10       Generative method mutationable #4        1.7
      0.8         0.3
17:         5.03485e+10       Generative method mutationable #1        9.4
      7           3.5
```

```
18:         5.15883e+10       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
19:         5.73611e+10       Generative method non mutationable a lot #4
      5.8         6.4         7.2
20:         6.92311e+10       Generative method mutationable #5         0.5
      6.7         1.3
21:         6.20007e+12       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list


Results #20:
1:          390011            Generative method mutationable #3         8.8
      2.5         5.2
2:          464792            Generative method non mutationable a lot #2
      5.1         2.2         6.7
3:          493856            Gradient method '10,0,0' init       10.474
      0.568234          0.571808
4:          578055            Generative method non mutationable a lot #3
      5.2         1.2         2.8
5:          765018            Gradient method '7,0,0' init        7.69889
      0.876194          0.881272
6:          808379            Zeingler-Nichols method 1 step k_crit
      0.8         0.137339          3.0756
7:          1.2953e+06        Zeingler-Nichols method 0.1 step k_crit
      0.595       0.111215          2.10095
8:          9.6949e+06        Zeingler-Nichols method 'x0.5 and x2'+'x0.33 and
x3' at '0.5 step k_crit' to avrg pass          0.175       0.0190217
      1.0626
9:          1.78783e+10       Gradient method '5,0,0' init        6.13157
      1.49139           1.30597
10:         2.35183e+10       Gradient method '1,0,0' init        1.06758
      0.317347          0.579046
11:         3.37411e+10       Gradient method with init by the Zeingler-Nichols
method 1 step k_krit          2.49195           1.82929           4.76755
12:         3.99902e+10       Gradient method with init by the Zeingler-Nichols
method 0.5 step k_krit        5.7759            5.34597           7.63554
13:         4.11928e+10       Gradient method with init by the Zeingler-Nichols
method 0.1 step k_krit        4.70099           4.2221            6.21183
14:         4.13851e+10       Generative method non mutationable a lot #1
      3.9         3.5         5.3
15:         4.171e+10         Generative method mutationable #4         1.7
      0.8         0.3
16:         4.38918e+10       Generative method mutationable #2         9.1
      5           0
17:         4.60766e+10       Generative method non mutationable a lot #5
      4           4.3         0.1
18:         5.11235e+10       Generative method mutationable #5         0.5
      6.7         1.3
19:         5.55545e+10       Generative method mutationable #1         9.4
      7           3.5
20:         6.07955e+10       Generative method non mutationable a lot #4
      5.8         6.4         7.2
21:         6.19856e+12       Gradient method '2,0,0' init        8.38471
      -29403.1          -341273
The end of rating list
```

Додаток Б

Приклади звітів у консоль під час налаштування регулятора різними регуляторами за різними алгоритмами (центральна частина відсутня через відсутність змістового навантаження та однотипність звіту, що був реалізованим лише як засіб спостереження за станом процесу налаштування чи моделювання пуску).

Варто зазначити, що спочатку модель розроблялася для регулювання за швидкістю, а потім для дослідження роботи регулятора та його налаштовувачів було вирішено перейти на регулювання за результуючим кутом виконаних обертів, проте модулі, що формували звіти відредаговано не було і тому варто пам’ятати, що в колонці «Velocity» насправді виводяться значення «Theta», тобто не швидкості, а результуючого кута.

Приклад налаштування за градієнтним методом та моделюванням пуску обладнання САР з отриманими налаштуваннями регулятора:

```
20:00:25: Starting
/home/tkachpavlo/Документы/HomeWork/Codding/diploma/model/build-
acs_model_for_experiments-Desktop-
Debug/acs_model_for_experiments...
1    0    0

NEXT_STEP:

cnf: 1          0          0
fi():     3.46493e+06
grad:     -6757.59        -31734.7        -57904.6
ans: 1.06758        0.317347        0.579046

NEXT_STEP:

cnf: 1.06758        0.317347        0.579046
fi():     3.46493e+06
grad:     5.57499e+08         5.64618e+09         1.34761e+10

ans: -5573.92       -56461.5        -134761

NEXT_STEP:

cnf: 0          0          0
fi():     1.3566e+10
```

```
grad:     -1.28504e+10        -1.30973e+10        -1.32383e+10

ans: 122930         74512          -2377.37

NEXT_STEP:

cnf: 122930         74512          0
fi():     2.12643e+08
grad:     6.74068e+12         2.01406e+12         1.49661e+11

ans: -6.72839e+07        -2.00661e+07        -1.49899e+06

NEXT_STEP:

cnf: 0         0         0
fi():     1.20048e+12
grad:     3.96562e+12         1.08463e+13         2.06419e+13

ans: -1.0694e+08         -1.28529e+08        -2.07918e+08
t    r    PID  Source    Velocity  EngVoltage     Torque
0    20   0    0    0    0    0
0.10001   20   21.9098   21.9097   0.0101022 21.9097   9.05446
0.2  20   22.5289   22.5288   0.0216546 22.5288   9.30979
0.3  20   23.1471   23.1471   0.0336771 23.147    9.56431
0.4  20   23.7644   23.7643   0.0461813 23.7643   9.8183
0.5  20   24.3806   24.3806   0.0591804 24.3805   10.0717
0.60001   20   24.9959   24.9958   0.0726898 24.9958   10.3246
0.70001   20   25.61     25.6099   0.0867214 25.6099   10.5769
0.80001   20   26.223    26.2229   0.101292  26.2229   10.8285
0.90001   20   26.8348   26.8347   0.116419  26.8347   11.0795
1.00001   20   27.4454   27.4454   0.13212   27.4453   11.3297
1.10001   20   28.0548   28.0547   0.148412  28.0547   11.5793
```

(Пропуск)

```
719.2     20   -8.13395  -8.13395  20.0373   -8.13395  -3.37728
719.3     20   -8.13147  -8.13147  20.034    -8.13147  -3.37625
719.4     20   -8.12889  -8.12889  20.0306   -8.12889  -3.37518
719.5     20   -8.1262   -8.1262   20.0272   -8.1262   -3.37407
719.6     20   -8.12341  -8.12341  20.0238   -8.12341  -3.37291
719.7     20   -8.1205   -8.1205   20.0204   -8.1205   -3.37171
719.8     20   -8.1175   -8.1175   20.017    -8.1175   -3.37047
719.9     20   -8.11439  -8.11439  20.0137   -8.11439  -3.36918
720  20   -8.11117  -8.11117  20.0103   -8.11117  -3.36785
1.06758   0.317347  0.579046
20:09:14:
/home/tkachpavlo/Документы/HomeWork/Codding/diploma/model/build-
acs_model_for_experiments-Desktop-
Debug/acs_model_for_experiments exited with code 0
```

Приклад налаштування за генетичним алгоритмом та моделюванням пуску обладнання САР з отриманими налаштуваннями регулятора:

```
00:10:20:                                                    Starting
/home/tkachpavlo/Документы/HomeWork/Codding/diploma/model/build-
acs_model_for_experiments-Desktop-
Debug/acs_model_for_experiments...
............
0:
globaly best:       1.65567e+09         3.9       3.5       5.30
locally best::      1.0078e+11          4.7       3.5       5.010

..........
1:
globaly best:       1.65567e+09         3.9       3.5       5.31
locally best::      1.64579e+11         5.46      4.67      0.921

..........
2:
globaly best:       1.65567e+09         3.9       3.5       5.32
locally best::      1.95489e+11         5.69      5.15      -
0.692

..........
3:
globaly best:       1.65567e+09         3.9       3.5       5.33
locally best::      2.13838e+11         5.25      5.15      0.273

..........
4:
globaly best:       1.65567e+09         3.9       3.5       5.34
locally best::      3.73767e+11         5.25      4.31      -
1.384

t     r     PID   Source     Velocity   EngVoltage      Torque
0     20    0     0     0     0     0
0.10001    20    73.0638    73.0641    0.11769    73.0644    18.364
0.2   20    70.3142    70.3144    0.417126   70.3146    18.364
                              (Пропуск)
719.9      20    0.0714089 0.0714091 20.0067    0.0714093 0.029723
720   20    0.0689982 0.0689985 20.0067    0.0689987 0.0287216
3.9   3.5   5.3
00:29:21:
/home/tkachpavlo/Документы/HomeWork/Codding/diploma/model/build-
acs_model_for_experiments-Desktop-
Debug/acs_model_for_experiments exited with code 0
```

Приклад налаштування за методом Циглера-Ніколса та моделюванням пуску обладнання САР з отриманими налаштуваннями регулятора:

```
21:12:30:                                                    Starting
/home/tkachpavlo/Документы/HomeWork/Codding/diploma/model/build-
acs_model_for_experiments-Desktop-
Debug/acs_model_for_experiments...
```

```
firstStep_start
vel: 0    -0.0116638     -0.0463454     -0.103776 ...
sec: 111  110
ext: 0    -43.181   -19.4299
amp: 21.5905   11.8756
firstStep_cycle     1
vel: 50.4388   50.5696   50.7073   50.8517   51.0026   ...
sec: 128
ext: 71.0098   60.2166
amp: 5.39657
vel: 62.8847   62.9479   63.0044   63.0542   ...
sec: 119  118
ext: 63.2203   50.2032   57.0623
amp: 6.50855   3.42953
aproximateStep_cycle     0.875
t    r    PID  Source    Velocity  EngVoltage     Torque
0    20   0    0    0    0    0
0.10001   20   43.894    43.8941   0.0386403 43.8943   17.5047
0.2  20   43.218    43.2181   0.0919481 43.2181   17.6619
                          (Пропуск)
719.8     20   -0.760903 -0.760903 20.0564   -0.760903 -0.316119
719.9     20   -0.760261 -0.760262 20.0561   -0.760262 -0.315853
720  20   -0.75962  -0.75962  20.0558   -0.75962  -0.315586
0.175     0.0190217 1.0626
21:15:34:
/home/tkachpavlo/Документы/HomeWork/Codding/diploma/model/build-
acs_model_for_experiments-Desktop-
Debug/acs_model_for_experiments exited with code 0
```

Приклад налаштування за градієнтним методом та методом Циглера-Ніколса в поєднанні одне з одним та моделюванням пуску обладнання САР з отриманими налаштуваннями регулятора немає сенсу тому як для них вже приклади наведені і налаштування виглядало як одне за одним в ряд налаштування виконуються та наприкінці традиційно запуск обладнання, тобто нічого нового.

Додаток В

Лістинги вихідного коду проекту.

Лістинг automated_control_system.h:

```
#ifndef AUTOMATED_CONTROL_SYSTEM_H
#define AUTOMATED_CONTROL_SYSTEM_H

#include"controlled_process.h"
#include"regulator.h"
#include"reference_signal_definder_static.h"
#include"dc_source.h"
#include<vector>
#include<array>

class Automated_control_system
{
private:
    DC_engine * p_process;
    PID_regulator * p_regulator;
    Reference_signal_definder_static * p_definder;
    DC_source * p_source;
    double dt;
    double t;

    void
reset_vector_of_elements(Automated_control_system_element_interf
ace * = nullptr);
    Automated_control_system_element_interface                  *
to_get_certain_element(Automated_control_system_element_interfac
e::type_of_element);
protected:
    std::vector<Automated_control_system_element_interface     *>
elements;
    void to_communicate();
    void to_plus_dt();
public:
    Automated_control_system();
    ~Automated_control_system();
    void to_set_dt(double);
    double to_check_dt() const; //not tested
    void to_set_t(double = 0); //not tested
    double to_check_t() const; //not tested
    bool
to_mount_the_element(Automated_control_system_element_interface
*);
    bool
to_mount_the_element(Automated_control_system_element_interface
&);
```

```
    const                                          std::vector<const
Automated_control_system_element_interface *> to_check_elements()
const;
    const                                          std::vector<const
Automated_control_system_element_interface                        *>
to_check_ordered_elements() const;
    const      Automated_control_system_element_interface      *
to_check_certain_element(Automated_control_system_element_interf
ace::type_of_element) const;
    const      Automated_control_system_element_interface      *
to_check_process() const;
    const      Automated_control_system_element_interface      *
to_check_regulator() const;
    const      Automated_control_system_element_interface      *
to_check_definder() const;
    const      Automated_control_system_element_interface      *
to_check_source() const;
    Automated_control_system_element_interface                 *
to_get_definder();
    virtual void to_calculate();
};

class      Automated_control_system_paralleled      :      public
Automated_control_system
{
    virtual void to_calculate() override;
};

#endif // AUTOMATED_CONTROL_SYSTEM_H
```

Лістинг container_analyzer.h:

```
#ifndef CONTAINER_ALANYZER_H
#define CONTAINER_ALANYZER_H

#include <algorithm>
#include <functional>
#include <numeric>

class container_analyzer
{
private:
    bool increasing = true;
protected:
    bool is_increasing() const { return increasing; }
    void      to_change_increasing_status()      {      increasing
= !increasing; }
    std::vector<double>::iterator  to_detect_extremum_Get_p(const
std::vector<double>::iterator&,                              const
std::vector<double>::iterator&);
    bool is_stable_the_oscilations(const std::vector<int>&);
```

```
    bool is_stable_the_amplitude(const std::vector<double>&);
    std::vector<int>               to_calculate_periods_Get(const
std::vector<int>&);
    std::vector<double>         to_calculate_amplitudes_Get(const
std::vector<double>&);
    void
to_calculate_extremums_and_seconds_Put_in(std::vector<double>&
r_vector_obj,   std::vector<double>&   extremums,std::vector<int>&
seconds);
public:
    container_analyzer() {}
    bool is_oscillating(std::vector<double>&);
    double
to_calculate_period_in_Get(std::vector<double>&,double);
};


#endif // CONTAINER_ALANYZER_H
```

Лістинг container_analyzer.cpp:

```
#include"container_analyzer.h"

std::vector<double>::iterator
container_analyzer::to_detect_extremum_Get_p(const
std::vector<double>::iterator&          it_begin,          const
std::vector<double>::iterator& it_end)
{
    //data
    auto ans = it_begin;
    //alg
    if (is_increasing())
    {
        ans      =      std::adjacent_find(it_begin,      it_end,
std::greater());
    }
    else
    {
        ans = std::adjacent_find(it_begin, it_end, std::less());
    }

    if (ans != it_end) to_change_increasing_status();

    return ans;
}

#include<iostream>
bool      container_analyzer::is_oscillating(std::vector<double>&
r_vector_obj)
{
    //data
    std::vector<double> extremums;
    std::vector<int> seconds;
```

```
    bool isStable_last_check = false;
    //alg
    to_calculate_extremums_and_seconds_Put_in(r_vector_obj,
extremums, seconds);
    std::cout << "vel:\t"; for (auto i : r_vector_obj) std::cout
<< i << "\t"; std::cout << std::endl << std::flush;
    std::cout << "sec:\t"; for (auto i : seconds) std::cout << i
<< "\t"; std::cout << std::endl << std::flush;
    std::cout << "ext:\t"; for (auto i : extremums) std::cout <<
i << "\t"; std::cout << std::endl << std::flush;
    isStable_last_check = is_stable_the_oscilations(seconds);
    isStable_last_check *= is_stable_the_amplitude(extremums);

    return isStable_last_check;
}

void
container_analyzer::to_calculate_extremums_and_seconds_Put_in(st
d::vector<double>&       r_vector_obj,       std::vector<double>&
extremums,std::vector<int>& seconds)
{
    //data
    std::vector<double>::iterator p_extremum;
    std::vector<double>::iterator p_extremum_prev;
    //alg
    p_extremum  =  to_detect_extremum_Get_p(r_vector_obj.begin(),
r_vector_obj.end());
    p_extremum_prev = p_extremum;
    while( p_extremum != r_vector_obj.end() )
    {
        extremums.push_back(*p_extremum);
        if     (     p_extremum     !=     p_extremum_prev     )
seconds.push_back(p_extremum - p_extremum_prev);
        if  ( p_extremum  !=  p_extremum_prev  )  p_extremum_prev  =
p_extremum;

        p_extremum     =     to_detect_extremum_Get_p(p_extremum,
r_vector_obj.end());
    }
}

bool          container_analyzer::is_stable_the_oscilations(const
std::vector<int>& seconds)
{
    //data
    std::vector<int> amplitudes;
    double average_amplitude = 0;
    bool ans = true;
    //alg
    amplitudes = to_calculate_periods_Get(seconds);
    average_amplitude                                          =
double(   std::accumulate(amplitudes.begin(),   amplitudes.end(),
0) ) / double(amplitudes.size());
```

```
    for (auto i : amplitudes)
        ans *= (double(average_amplitude) * 0.5 < i && i <
double(average_amplitude) * 2) ;
    return ans;
}

std::vector<int>
container_analyzer::to_calculate_periods_Get(const
std::vector<int>& seconds)
{
    //data
    std::vector<int> periods;
    int period = 0;
    //alg
    auto i_prev = seconds.begin();
    for(auto i = seconds.begin(); i < seconds.end(); ++i)
    {
        period += *i;
        if (i - i_prev != 0)
        {
            periods.push_back(period);
            i_prev = i + 1;
            period = 0;
        }
    }
    return periods;
}

bool            container_analyzer::is_stable_the_amplitude(const
std::vector<double>& extremums)
{
    //data
    std::vector<double> amplitudes;
    double average_amplitude;
    bool ans = true;
    //alg
    amplitudes = to_calculate_amplitudes_Get(extremums);
    average_amplitude    =    std::accumulate(amplitudes.begin(),
amplitudes.end(), 0.0) / double(amplitudes.size());
    for (auto i : amplitudes)
        ans   *=   (average_amplitude   *   0.3   <   i   &&   i   <
average_amplitude * 3) ;
    if (amplitudes.size() < 2) ans = false;

    std::cout << "amp:\t"; for (auto i : amplitudes) std::cout <<
i << "\t"; std::cout << std::endl << std::flush;

    return ans;
}

std::vector<double>
container_analyzer::to_calculate_amplitudes_Get(const
std::vector<double>& extremums)
```

```
{
    //data
    std::vector<double> amplitudes;
    //alg
    auto i_prev = extremums.begin();
    for(auto i = extremums.begin() + 1; i < extremums.end(); ++i)
    {
        amplitudes.push_back( std::abs(*i - *i_prev) / 2 );
        i_prev = i;
    }
    return amplitudes;
}

double
container_analyzer::to_calculate_period_in_Get(std::vector<doubl
e>& r_vector_obj, double dt)
{
    //data
    std::vector<double> extremums;
    std::vector<int> seconds;
    std::vector<int> periods;
    double average_period;
    //alg
    to_calculate_extremums_and_seconds_Put_in(r_vector_obj,
extremums, seconds);
    periods = to_calculate_periods_Get(seconds);
    average_period  =  double(  std::accumulate(periods.begin(),
periods.end(), 0.0) ) / double(periods.size());
    return average_period * dt;
}
```

Лістинг controlled_process.h:

```
#ifndef CONTROLLED_PROCESS_H
#define CONTROLLED_PROCESS_H

#include "automated_control_system_element_interface.h"

class DC_engine_tester;

class               DC_engine               :               public
Automated_control_system_element_interface
{
private:
    friend class DC_engine_tester;
    void to_actulize_the_parameters();
    double to_dcurrent_dt(double U, double I, double R, double kf,
double w, double L);
    double to_dvelocity_dt(double kf, double I, double T_L, double
J);
    double  to_dtorque_of_load_dw(double  w,  double  kL_1,  double
k_exp_lim, double k_exp_curv, double T);
```

```
    double to_actulize_the_torque_of_load(double w, double kL_0,
double kL_1, double k_exp_lim, double k_exp_curv);
    double to_actulize_current(double previous_I, double dI_dt,
double dt);
    double to_actulize_velocity(double previous_w, double dw_dt,
double dt);
    double to_actulize_theta(double previous_x, double dx_dt,
double dt);
    double       to_actulize_the_torque_of_load_from_dT_dw(double
T,double dwdt, double dT_dw, double dt);
    void to_solve_with_euler();
    void to_solve_with_runge_kutta();
    void runge_kutta_stage_1(std::vector<double>&);
    void
runge_kutta_stage_2(std::vector<double>&,std::vector<double>&);
    void
runge_kutta_stage_3(std::vector<double>&,std::vector<double>&);
    void
runge_kutta_stage_4(std::vector<double>&,std::vector<double>&);
public:
    DC_engine();
    virtual ~DC_engine() override;
    enum calculation_mode_states { EULER, RUNGE_KUTTA };
private:
    calculation_mode_states calculation_mode_state;
public:
    void to_set_calculation_mode(calculation_mode_states);
    const calculation_mode_states to_check_calculation_mode();


    // BEGIN OF THE STATIC PARAMETERS
    // Torque of the load STATIC koefficients
    enum
    {
        BEGIN_STATIC = END_INTERFACE,        /// BE CRARE FOR THE
TRUE OF THE EQUATION !!
        BEGIN_LOAD_K = END_INTERFACE,        /// BE CRARE FOR THE
TRUE OF THE EQUATION !!

        // K_of_T_load
        LOAD_K_0 = BEGIN_LOAD_K + 0,
        LOAD_K_1 = BEGIN_LOAD_K + 1,
        LOAD_K_EXP_LIMIT = BEGIN_LOAD_K + 2,
        LOAD_K_EXP_CURVATURE = BEGIN_LOAD_K + 3,


        LAST_LOAD_K = LOAD_K_EXP_CURVATURE,                /// BE
CRARE FOR THE TRUE OF THE EQUATION !!
        END_LOAD_K = LAST_LOAD_K + 1,        /// BE CRARE FOR THE
TRUE OF THE EQUATION !!

    };
    // DC_engine STATIC parameters
```

```
    enum
    {
        BEGIN_DC_ENGINE = END_LOAD_K,        /// BE CRARE FOR THE
TRUE OF THE EQUATION !!

        RESISTANCE = BEGIN_DC_ENGINE + 0,
        INDUCTIVITY = BEGIN_DC_ENGINE + 1,
        KF = BEGIN_DC_ENGINE + 2,

        // J
        MOMENT_OF_INERTIA = BEGIN_DC_ENGINE + 3,
        MOMENT_OF_INERTIA_OF_ENGINE = BEGIN_DC_ENGINE + 4,
        MOMENT_OF_INERTIA_OF_MECHANICAL_LOAD = BEGIN_DC_ENGINE +
5,


        LAST_SATIC    =    MOMENT_OF_INERTIA_OF_MECHANICAL_LOAD,
/// BE CRARE FOR THE TRUE OF THE EQUATION !!
        END_STATIC = LAST_SATIC + 1,         /// BE CRARE FOR THE
TRUE OF THE EQUATION !!
    };
    // END OF THE STATIC PARAMETERS

    // VARIED PARAMETERS "NONSTATIC"
    enum
    {
        BEGIN_NONSTATIC = END_STATIC,        /// BE CRARE FOR THE
TRUE OF THE EQUATION !!

        // X V A
        THETA = BEGIN_NONSTATIC + 0,

        DTHETA_DT = BEGIN_NONSTATIC + 1,
        VELOCITY = DTHETA_DT,

        DDTHETA_DTT = BEGIN_NONSTATIC + 2,
        DVELOCITY_DT = DDTHETA_DTT,
        ACCELERATION = DVELOCITY_DT,

        // q I dI/dt
        COULOMBLS = BEGIN_NONSTATIC + 3,

        DCOULOMBS_DT = BEGIN_NONSTATIC + 4,
        CURRENT = DCOULOMBS_DT,

        DDCOULUMBS_DTT = BEGIN_NONSTATIC + 5,
        DCURRENT_DT = DDCOULUMBS_DTT,

        // V
        VOLTAGE = BEGIN_NONSTATIC + 6,

        // T_engine
        TORQUE = BEGIN_NONSTATIC + 7,
```

```
        TORQUE_OF_LOAD = BEGIN_NONSTATIC + 8,
        DTORQUE_OF_LOAD_DVELOCITY = BEGIN_NONSTATIC + 9,



        LAST_ENGINE         =         DTORQUE_OF_LOAD_DVELOCITY,
/// BE CRARE FOR THE TRUE OF THE EQUATION !!
        END_ENGINE = LAST_ENGINE + 1,                          ///
BE CRARE FOR THE TRUE OF THE EQUATION !!
    };

    // THE END OF ALL
    enum
    {
        LAST_NONSTATIC = LAST_ENGINE,                          ///
BE CRARE FOR THE TRUE OF THE EQUATION !!
        END_NONSTATIC = LAST_NONSTATIC + 1,                    ///
BE CRARE FOR THE TRUE OF THE EQUATION !!

        LAST_DC_ENGINE = LAST_ENGINE,                          ///
BE CRARE FOR THE TRUE OF THE EQUATION !!
        END_DC_ENGINE = LAST_DC_ENGINE + 1,                    ///
BE CRARE FOR THE TRUE OF THE EQUATION !!

        SIZE = END_DC_ENGINE                                   ///
BE CRARE FOR THE TRUE OF THE EQUATION !!
    };

    bool to_verify_amount_of_parameters() const override;
    bool  to_set_element_parameters(const  std::vector<double>  &)
override;
    bool   to_set_all_parameters(const   std::vector<double>   &)
override;
    void to_calculate() override;
};

#endif // CONTROLLED_PROCESS_H
```

Лістинг dc_source.h:

```
#ifndef DC_SOURCE_H
#define DC_SOURCE_H

#include "automated_control_system_element_interface.h"

class               DC_source               :               public
Automated_control_system_element_interface
{
public:
    DC_source();
    enum
    {
```

```
        BEGIN_SOURCE = END_INTERFACE,            /// BE CRARE FOR
THE TRUE OF THE EQUATION !!

        MAX_VOLTAGE = BEGIN_SOURCE + 0,

        MIN_VOLTAGE = BEGIN_SOURCE + 1,

        LAST_SOURCE = MIN_VOLTAGE,               /// BE CRARE FOR
THE TRUE OF THE EQUATION !!
        END_SOURCE = LAST_SOURCE + 1,
        SIZE = END_SOURCE
    };
    void to_set_max_voltage(double);
    void to_set_min_voltage(double);
    //interface:
    virtual bool to_verify_amount_of_parameters() const override;
    virtual          bool          to_set_element_parameters(const
std::vector<double> &) override;
    virtual bool to_set_all_parameters(const std::vector<double>
&) override;
    virtual void to_calculate() override;
};

class DC_source_inerted : public DC_source
{
private:
    double change_step();
public:
    virtual void to_calculate() override;
};

#endif // DC_SOURCE_H
```

Лістинг experiment_executor.h:

```
#ifndef EXPERIMENT_EXECUTOR_H
#define EXPERIMENT_EXECUTOR_H

#include"automated_control_system.h"
#include"registrator.h"

#include<string>
#include<vector>
#include<memory>

class Experiment_executor_interface
{
private:
    Automated_control_system * acs_model;
    double dt;
    double t_length;
    double time_to_show;
    double interval;
```

```
    double amount_of_show;
protected:
    void reset_interval();
    virtual void to_run(Registrator *) const;
    double to_get_t_length() const;
    double to_get_interval() const;
    Automated_control_system * to_get_model() const;
public:
    Experiment_executor_interface(Automated_control_system  *  =
nullptr);
    virtual ~Experiment_executor_interface() = 0;
    virtual void to_run() = 0;
    void to_get_model_to_run(Automated_control_system *);
    Automated_control_system * to_get_model();
    void to_set_dt(double);
    void to_set_t_length(double);
    void to_set_time_to_registrate(double);
    void to_set_amount_of_registrations(double);
};

class      Experiment_executor      :      virtual      public
Experiment_executor_interface // for txt-file report
{
protected:
    std::string results_title;
public:
    Experiment_executor(Automated_control_system * = nullptr);
    void to_set_result_title(const char *);
    virtual void to_run() override;
};

class      Experiment_executor_short_report      :      public
Experiment_executor // for short txt-file report
{
public:
    Experiment_executor_short_report(Automated_control_system * =
nullptr);
    virtual void to_run() override;
};

class Experiment_executor_for_fitness_function : virtual public
Experiment_executor_interface
{
protected:
    std::vector<double> * records = nullptr;
public:

Experiment_executor_for_fitness_function(Automated_control_syste
m * = nullptr);
    void to_set_vector(std::vector<double>*);
    void to_set_vector(std::vector<double>&);
    void to_set_vector(std::shared_ptr<std::vector<double>>);
    virtual void to_run() override;
```

```
};

class      Experiment_executor_velocity_record      :      public
Experiment_executor_for_fitness_function
{
public:
    void to_run() override;
};

class
Experiment_executor_for_fitness_function_with_varied_reference_s
ignal : public Experiment_executor_for_fitness_function
{
    double reference_signal_max;
    double reference_signal_min;
public:

Experiment_executor_for_fitness_function_with_varied_reference_s
ignal(Automated_control_system * = nullptr);

    void to_set_varied_diapasone_min_max(double,double);

    virtual void to_run() override;
};


#endif // EXPERIMENT_EXECUTOR_H
```

Лістинг fitness_function.cpp:

```
#include "regulator_tuner.h"
#include "experiment_executor.h"

#include "tkachpavlo2001lib/myExeption.hpp"

#include <numeric>

class  exception_for_fitness_function_varied_reference_signal  :
public myException
{
public:
    exception_for_fitness_function_varied_reference_signal()    :
myException()                    {std::cerr                    <<
"::fitness_function_varied_reference_signal"; }
};

class exception_for_gradient_by_step : public myException
{
public:
    exception_for_gradient_by_step() : myException() { std::cerr
<< "::gradient_by_step"; }
};
```

```
double
fitness_function_varied_reference_signal(double*ans,void*param)
{
    int status = -1;
    parameters_for_optimizer * p_parameters_for_optimizer;

std::shared_ptr<Experiment_executor_for_fitness_function_with_va
ried_reference_signal> p_experiment;
    try
    {
        if  (param  !=  nullptr)    p_parameters_for_optimizer  =
static_cast<parameters_for_optimizer *>(param);
        else
        {
            status = 1;
            throw
exception_for_fitness_function_varied_reference_signal();
        }
        if (p_parameters_for_optimizer == nullptr)
        {
            status = 2;
            throw
exception_for_fitness_function_varied_reference_signal();
        }
        else                    p_experiment                    =
std::make_shared<Experiment_executor_for_fitness_function_with_v
aried_reference_signal>
                (

p_parameters_for_optimizer->parameters_p_objects_parameters_obj.
p_acs_model
                );


p_experiment->to_set_t_length(p_parameters_for_optimizer->parame
ters_for_fitness_function_obj.length);

p_experiment->to_set_dt(p_parameters_for_optimizer->parameters_f
or_fitness_function_obj.dt);
        p_experiment->to_set_varied_diapasone_min_max
                (

p_parameters_for_optimizer->parameters_for_varied_fitness_functi
on_obj.min,

p_parameters_for_optimizer->parameters_for_varied_fitness_functi
on_obj.max
                );

p_experiment->to_set_time_to_registrate(p_parameters_for_optimiz
er->parameters_for_fitness_function_obj.t_registrate);

        std::vector<double> records;
```

```
        p_experiment->to_set_vector(records);


p_experiment->to_get_model_to_run(p_parameters_for_optimizer->pa
rameters_p_objects_parameters_obj.p_acs_model);


p_parameters_for_optimizer->parameters_p_objects_parameters_obj.
p_regulator->to_get_parameters()[PID_regulator::K_P] = ans[0];

p_parameters_for_optimizer->parameters_p_objects_parameters_obj.
p_regulator->to_get_parameters()[PID_regulator::K_I] = ans[1];

p_parameters_for_optimizer->parameters_p_objects_parameters_obj.
p_regulator->to_get_parameters()[PID_regulator::K_D] = ans[2];

        for       (int       i       =       0;       i       <
p_parameters_for_optimizer->parameters_for_varied_fitness_functi
on_obj.times; ++i) p_experiment->to_run();

        status = 0;
        return std::accumulate(records.begin(), records.end(), 0,
[&](double acc, double num)->double {return acc + num * num;} );
    }
    catch
(exception_for_fitness_function_varied_reference_signal&)
    {
        std::cerr << ": status = " << status << std::endl <<
std::endl << std::endl;
        std::abort();
    }

}

std::array<double, 3> gradient_by_step(double*ans,void*param)
{
    int status = -1;
    parameters_for_optimizer * p_parameters_for_optimizer;
    std::array<double, 3> gradient {0};
    try
    {
        if (param == nullptr)
        {
            status = 1;
            throw exception_for_gradient_by_step();
        }
        p_parameters_for_optimizer                                =
static_cast<parameters_for_optimizer*>(param);
        if (p_parameters_for_optimizer == nullptr)
        {
            status = 2;
            throw exception_for_gradient_by_step();
        }
```

```
        PID_regulator          *          p_regulator          =
p_parameters_for_optimizer->parameters_p_objects_parameters_obj.
p_regulator;
        double temp;
        double stepped_value;

        double                    fitness_value                 =
fitness_function_varied_reference_signal(ans,param);

        temp                                                    =
p_regulator->to_get_parameters()[PID_regulator::K_P];
        ans[0]              =              ans[0]               +
p_parameters_for_optimizer->parameters_for_gradient_obj.dx;
        gradient[0]                                             =
fitness_function_varied_reference_signal(ans,      param)      -
fitness_value;
        p_regulator->to_get_parameters()[PID_regulator::K_P]    =
ans[0] = temp;

        temp                                                    =
p_regulator->to_get_parameters()[PID_regulator::K_I];
        ans[1]              =              ans[1]               +
p_parameters_for_optimizer->parameters_for_gradient_obj.dx;
        gradient[1]                                             =
fitness_function_varied_reference_signal(ans,      param)      -
fitness_value;
        p_regulator->to_get_parameters()[PID_regulator::K_I]    =
ans[1] = temp;

        temp                                                    =
p_regulator->to_get_parameters()[PID_regulator::K_D];
        ans[2]              =              ans[2]               +
p_parameters_for_optimizer->parameters_for_gradient_obj.dx;
        gradient[2]                                             =
fitness_function_varied_reference_signal(ans,      param)      -
fitness_value;
        p_regulator->to_get_parameters()[PID_regulator::K_D]    =
ans[2] = temp;

        status = 0;
        return gradient;
    }
    catch (exception_for_gradient_by_step&)
    {
        std::cerr << ": status = " << status << std::endl <<
std::endl << std::endl;
        std::abort();
    }
}
```

Лістинг generative_algorithm.h:

```
#ifndef GENERATIVE_ALGORITHM_H
#define GENERATIVE_ALGORITHM_H

#include<iostream>
#include<vector>
#include<ctime>
#include<random>
#include<array>
#include<map>
#include<array>

#include"tkachpavlo2001lib/tkachpavlo2001lib.hpp"

template <int POLYNOM>
class generative_algorithm
{
private:
    void                          add_answer(std::multimap<double,
std::array<double,POLYNOM>>        &        _rating,        const
std::array<double,POLYNOM> & _answer);
    void                        to_initialize(std::multimap<double,
std::array<double,POLYNOM>> & _answer_rating);
    void to_mutate(std::array<double, POLYNOM> & _answer);
    void   to_pair(std::array<double,3>   &   new_agent,   const
std::array<double,  POLYNOM>  &  arr_1,  const  std::array<double,
POLYNOM> & arr_2);
    void
to_make_new_generation(std::vector<std::array<double,POLYNOM>>  &
_new_generation,           const           std::multimap<double,
std::array<double,POLYNOM>> & old_generation);
    void  to_mutate_new_generation(std::vector<std::array<double,
POLYNOM>> & _new_generation);
    void                             to_rate(std::multimap<double,
std::array<double,POLYNOM>>      &      _answer_rating,      const
std::vector<std::array<double,POLYNOM>> & _new_generation);
    void to_cut(std::multimap<double, std::array<double,POLYNOM>>
& _answer_rating);
    void   to_show(const   std::pair<double,   std::array<double,
POLYNOM>> & _answer_to_show);
    void    to_show(const    std::array<double,    POLYNOM>    &
_answer_to_show);
    void     to_show_iteration_status(const     std::pair<double,
std::array<double,     POLYNOM>>     &     _best_answer,     const
std::pair<double, std::array<double, POLYNOM>> & _new_answer, int
iteration);
    double MUTATION_PROBABILITY = 0.1;
    double MUTATION_STEP = 0.1;
    double MAX_INIT = 1;
    double MIN_INIT = 0;
    double AGENTS = 1000;
    double NEW_AGENTS = 3000;
    double ITERATIONS = 100;
```

```
    double (*fitnes_function) (double * answer, void *) = nullptr;
    void * fitness_function_parameters = nullptr;

public:
    generative_algorithm();
    void to_solve_record_mode();
    std::pair<double, std::array<double,POLYNOM>> to_solve();
    std::array<double,POLYNOM> to_solve_array_out();
    void to_set_min_init(double = 0);
    void to_set_max_init(double = 1);
    void to_set_mutation_step(double = 0.1);
    void to_set_mutation_propability(double = 0.1);
    void to_set_agents(double = 1000);
    void to_set_new_agents(double = 3000);
    void to_set_iterations(double = 100);
    void to_set_fitnes_function( double (*f) (double*,void*));
    void to_set_fitness_function_parameters (void*);
};

template <int POLYNOM>
generative_algorithm<POLYNOM>::generative_algorithm()
{
    to_set_min_init();
    to_set_max_init();
    to_set_mutation_step();
    to_set_mutation_propability();
}

template <int POLYNOM>
void
generative_algorithm<POLYNOM>::to_initialize(std::multimap<doubl
e, std::array<double,POLYNOM>> & _answer_rating)
{
    for (unsigned int i = 0; i < AGENTS; i++)
    {
        std::array<double, POLYNOM> _answer;
        for (int i = 0; i < POLYNOM; i++)
        {
            _answer[i] = myrand() * (MAX_INIT - MIN_INIT) +
MIN_INIT;
        }
        add_answer(_answer_rating, _answer);
    }
}

template <int POLYNOM>
void
generative_algorithm<POLYNOM>::add_answer(std::multimap<double,
std::array<double,POLYNOM>>        &        _rating,        const
std::array<double,POLYNOM> & _answer)
{
    double array[POLYNOM];
    for (int i = 0; i < POLYNOM; ++i)
```

```
        array[i] = _answer[i];
    double          fi          =          fitnes_function(array,
fitness_function_parameters);
    _rating.insert(std::pair(fi, _answer));
}

template <int POLYNOM>
void  generative_algorithm<POLYNOM>::to_mutate(std::array<double,
POLYNOM> & _answer)
{
    for  (auto  i  =  std::begin(_answer);  i  !=  std::end(_answer);
++i)
    {
        if ( myrand() >= MUTATION_PROBABILITY ) *i += (rand() % 2
- 1) * myrand() * MUTATION_STEP;
    }
}

template <int POLYNOM>
void  generative_algorithm<POLYNOM>::to_pair(std::array<double,3>
& new_agent, const std::array<double, POLYNOM> & arr_1, const
std::array<double, POLYNOM> & arr_2)
{
    for (long unsigned int i = 0; i < new_agent.size(); ++i)
        if (myrand() > 0.5)
            new_agent[i] = arr_1[i];
        else
            new_agent[i] = arr_2[i];
}

template <int POLYNOM>
void
generative_algorithm<POLYNOM>::to_make_new_generation(std::vecto
r<std::array<double,POLYNOM>>     &     _new_generation,     const
std::multimap<double,        std::array<double,POLYNOM>>        &
old_generation)
{
    _new_generation.clear();
    int i = 0;
    if     (!old_generation.empty())     for     (auto     j     =
std::begin(old_generation); i < NEW_AGENTS; ++i, std::advance(j,
1))
    {
        _new_generation.emplace_back();
        auto j_2 = j;
        std::advance(j_2, 1);
        if (j_2 == std::end(old_generation) )
        {
            j = std::begin(old_generation);
            j_2 = j;
            std::advance(j_2, 1);
        }
```

```
        if    (j_2    !=    old_generation.end()    &&    j    !=
old_generation.end())  to_pair(_new_generation.at(i),  j->second,
j_2->second);
    }
}

template <int POLYNOM>
void
generative_algorithm<POLYNOM>::to_mutate_new_generation(std::vec
tor<std::array<double, POLYNOM>> & _new_generation)
{
    for   (auto   i   =   std::begin(_new_generation);   i   !=
std::end(_new_generation); ++i)
        to_mutate(*i);
}

template <int POLYNOM>
void generative_algorithm<POLYNOM>::to_rate(std::multimap<double,
std::array<double,POLYNOM>>     &     _answer_rating,     const
std::vector<std::array<double,POLYNOM>> & _new_generation)
{
    _answer_rating.clear();
    if (!_new_generation.empty()) for (auto i : _new_generation)
        add_answer(_answer_rating, i);
}

template <int POLYNOM>
void  generative_algorithm<POLYNOM>::to_cut(std::multimap<double,
std::array<double,POLYNOM>> & _answer_rating)
{
    while (_answer_rating.size() > AGENTS)
    {
        _answer_rating.erase(--
(std::rbegin(_answer_rating).base()));
    }
}

template <int POLYNOM>
void                generative_algorithm<POLYNOM>::to_show(const
std::pair<double, std::array<double, POLYNOM>> & _answer_to_show)
{
    std::cout << _answer_to_show.first;
    for (auto i : _answer_to_show.second) std::cout << '\t' << i;
}

template <int POLYNOM>
void                generative_algorithm<POLYNOM>::to_show(const
std::array<double, POLYNOM> & _answer_to_show)
{
    std::cout << "NODATA";
    for (auto i : _answer_to_show) std::cout << '\t' << i;
}
```

```
template <int POLYNOM>
void
generative_algorithm<POLYNOM>::to_show_iteration_status(const
std::pair<double, std::array<double, POLYNOM>> & _best_answer,
const    std::pair<double,    std::array<double,    POLYNOM>>    &
_new_answer, int iteration)
{
    std::cout << iteration << ":\t" << _new_answer.first << '\t'
<< _best_answer.first;
    for (auto i : _best_answer.second) std::cout << '\t' << i;
}

template <int POLYNOM>
void generative_algorithm<POLYNOM>::to_solve_record_mode()
{
    if    (fitnes_function    ==    nullptr)    {    std::cerr    <<
"fitness_function is NULL\n"; return; }
    if (fitness_function_parameters == nullptr) { std::cerr <<
"fitness_function_parameters is NULL\n"; return; }
    std::vector<std::array<double,POLYNOM>> answer_array;
    std::multimap<double,             std::array<double,POLYNOM>>
answer_rating;
    std::pair<double, std::array<double,POLYNOM>> answer;
    std::pair<double, std::array<double,POLYNOM>> best_answer;
    srand(time(0));
    to_initialize(answer_rating);
    answer = *std::begin(answer_rating);
    std::cout << "Initializing:\n";
    to_show(answer);
    std::cout << '\n';
    best_answer = answer;
    for ( int i = 0 ; i < ITERATIONS; ++i)
    {
        to_make_new_generation(answer_array,answer_rating);
        to_mutate_new_generation(answer_array);
        to_rate(answer_rating, answer_array);
        to_cut(answer_rating);
        answer = *(std::begin(answer_rating));
        if  (best_answer.first  >  answer.first)  best_answer  =
answer;
        to_show_iteration_status(best_answer, answer, i);
        std::cout << std::endl;
    }
    std::cout << "\n\nFinal answer: ";
    to_show(best_answer);
    std::cout << std::endl;
}

template <int POLYNOM>
std::pair<double,                    std::array<double,POLYNOM>>
generative_algorithm<POLYNOM>::to_solve()
{
```

```
    if (fitnes_function == nullptr) { std::cerr << "fitnes_function is NULL\n"; std::abort(); }
    if (fitness_function_parameters == nullptr) { std::cerr << "fitness_function_parameters is NULL\n"; std::abort(); }
    std::vector<std::array<double,POLYNOM>> answer_array;
    std::multimap<double, std::array<double,POLYNOM>> answer_rating;
    std::pair<double, std::array<double,POLYNOM>> answer;
    std::pair<double, std::array<double,POLYNOM>> best_answer;
    srand(time(0));
    to_initialize(answer_rating);
    answer = *std::begin(answer_rating);
    best_answer = answer;
    for ( int i = 0 ; i < ITERATIONS; ++i)
    {
        to_make_new_generation(answer_array,answer_rating);
        to_mutate_new_generation(answer_array);
        to_rate(answer_rating, answer_array);
        to_cut(answer_rating);
        answer = *(std::begin(answer_rating));
        if (best_answer.first > answer.first) best_answer = answer;
    }
    return best_answer;
}

template <int POLYNOM>
std::array<double,POLYNOM>
generative_algorithm<POLYNOM>::to_solve_array_out()
{
    auto answer = this->to_solve();
    return answer.second;
}

template <int POLYNOM>
void generative_algorithm<POLYNOM>::to_set_min_init(double num)
{
    MIN_INIT = num;
}

template <int POLYNOM>
void generative_algorithm<POLYNOM>::to_set_max_init(double num)
{
    MAX_INIT = num;
}

template <int POLYNOM>
void generative_algorithm<POLYNOM>::to_set_mutation_step(double num)
{
    MUTATION_STEP = num;
}
```

```
template <int POLYNOM>
void
generative_algorithm<POLYNOM>::to_set_mutation_propability(doubl
e num)
{
    MUTATION_PROBABILITY = num;
}

template <int POLYNOM>
void generative_algorithm<POLYNOM>::to_set_agents(double num)
{
    AGENTS = num;
}

template <int POLYNOM>
void generative_algorithm<POLYNOM>::to_set_new_agents(double num)
{
    NEW_AGENTS = num;
}

template <int POLYNOM>
void generative_algorithm<POLYNOM>::to_set_iterations(double num)
{
    ITERATIONS = num;
}

template <int POLYNOM>
void
generative_algorithm<POLYNOM>::to_set_fitnes_function(     double
(*f) (double*,void*))
{
    fitnes_function = f;
}

template <int POLYNOM>
void
generative_algorithm<POLYNOM>::to_set_fitness_function_parameter
s(void* _parameters)
{
    fitness_function_parameters = _parameters;
}

#endif // GENERATIVE_ALGORITHM_H
```

Лістинг gradient_method.h:

```
#ifndef GRADIENT_METHOD_H
#define GRADIENT_METHOD_H

#include<array>
#include<iostream>
#include<map>
```

```
#include<random>
#include<time.h>


#include"tkachpavlo2001lib/myRand.h"



template <int POLYNOM>
class gradient_method_step_based
{
private:
    double (*fitness_function) (double*,void*) = nullptr;
    std::array<double,        POLYNOM>        (*gradient_function)
(double*,void*) = nullptr;
    double h = 0;
    double learn_coefficient = 0;
    double ITERATION = 0;
    void to_initiate_answer(double*);
protected:
    std::array<double,POLYNOM> answer;
public:
    gradient_method_step_based();
    virtual                 std::pair<std::array<double,POLYNOM>,
std::array<double,POLYNOM> > to_solve(void*);
    std::pair<std::array<double,POLYNOM>,
std::array<double,POLYNOM> > to_solve_and_record(void * param);
    std::array<double,POLYNOM> to_solve_array_out(void * param);
    std::array<double,                                      POLYNOM>
to_gradient(std::array<double,POLYNOM>, void*);
    void to_set_step(double);
    void to_set_iteration(double);
    void
to_set_fitness_function(double(*)(double*,void*)=nullptr);
    void              to_set_gradient_function(std::array<double,
POLYNOM>(*)(double*,void*)=nullptr);
    //void
to_set_gradient_function(double(*)(double*,void*)=nullptr);
    double to_get_fitness_function_value(void*);
    void          to_set_learn_coefficient(double          _arg)
{ learn_coefficient = _arg; }
};

template <int POLYNOM>
class     stochastic_gradient_method_step_based     :     public
gradient_method_step_based<POLYNOM>
{
private:
    unsigned int tries_amount = 0;
    double init_min = 0;
    double init_max = 0;
    std::array<double,   POLYNOM>   (*initiation_function)   ()   =
nullptr;
```

```
    std::array<double, POLYNOM> init_funct_def();
    std::array<double, POLYNOM> call_init_funct();
public:
    stochastic_gradient_method_step_based();
    void to_set_tries_amount(unsigned int);
    void to_set_initiation_minimum(double);
    void to_set_initiation_maximum(double);
    void  to_set_initiation_function(std::array<double,  POLYNOM>
(*) ());
    std::pair<std::array<double,POLYNOM>,
std::array<double,POLYNOM> > to_solve(void*) override;
};

template <int POLYNOM>
gradient_method_step_based<POLYNOM>::gradient_method_step_based(
)
{
    for (auto & i : answer) i = 0;
}

template <int POLYNOM>
std::array<double,                                          POLYNOM>
gradient_method_step_based<POLYNOM>::to_gradient(std::array<doub
le,POLYNOM> _ans, void * param)
{
    std::array<double,         POLYNOM>         answer         =
gradient_function(_ans.begin(), param);
    return answer;
}

template <int POLYNOM>
std::pair<std::array<double,POLYNOM>,
std::array<double,POLYNOM>                                       >
gradient_method_step_based<POLYNOM>::to_solve(void * param)
{
    to_initiate_answer(answer.begin());
    for (auto i : answer) std::cout << i << "\t";          //
REDUNDANCE
    std::cout << std::endl << std::flush;                 //
REDUNDANCE
    if   (fitness_function   ==   nullptr)   {   std::cerr   <<
"fitnes_function is NULL\n"; abort(); }
    std::array<double, POLYNOM> gradient;
    for (int i = 0; i < ITERATION; ++i)
    {
        gradient = this->to_gradient(answer, param);
        for(int j = 0; j < POLYNOM; ++j)
            answer[j] -= gradient[j] * learn_coefficient;
        for (auto i : gradient) std::cout << i << "\t";      //
REDUNDANCE
        for (auto i : answer) std::cout << i << "\t";        //
REDUNDANCE
```

```
        std::cout << std::endl << std::flush;                 //
REDUNDANCE
    }
    return                  std::pair<std::array<double,POLYNOM>,
std::array<double,POLYNOM>> (gradient,answer);;
}

template <int POLYNOM>
std::pair<std::array<double,POLYNOM>,
std::array<double,POLYNOM>                                     >
gradient_method_step_based<POLYNOM>::to_solve_and_record(void   *
param)
{
    to_initiate_answer(answer.begin());
    if   (fitness_function   ==   nullptr)   {   std::cerr   <<
"fitnes_function is NULL\n"; abort(); }
    std::array<double, POLYNOM> gradient;
    for (int i = 0; i < ITERATION; ++i)
    {
        std::cout << i << ":\t";
        gradient = to_gradient(answer);
        for(int j = 0; j < POLYNOM; ++j)
            std::cout << gradient[j] << "\t\t";
        std::cout << "||";
        for(int j = 0; j < POLYNOM; ++j)
            std::cout << (answer[j] -= gradient[j] * h) << "\t\t";
        std::cout << std::endl;
    }
    return                  std::pair<std::array<double,POLYNOM>,
std::array<double,POLYNOM>> (gradient,answer);
}

template <int POLYNOM>
std::array<double,POLYNOM>
gradient_method_step_based<POLYNOM>::to_solve_array_out(void    *
param)
{
    std::pair<std::array<double,POLYNOM>,
std::array<double,POLYNOM> > ans = this->to_solve(param);
    return ans.second;
}

template <int POLYNOM>
void
gradient_method_step_based<POLYNOM>::to_initiate_answer(double*
_init)
{
    for(int j = 0; j < POLYNOM; ++j) answer[j] = _init[j];
}

template <int POLYNOM>
void gradient_method_step_based<POLYNOM>::to_set_step(double _h)
{
```

```
    h = _h;
}

template <int POLYNOM>
void gradient_method_step_based<POLYNOM>::to_set_iteration(double
_iteration)
{
    ITERATION = _iteration;
}

template <int POLYNOM>
void
gradient_method_step_based<POLYNOM>::to_set_fitness_function(dou
ble(*_f)(double*,void*))
{
    fitness_function = _f;
}


template <int POLYNOM>
void
gradient_method_step_based<POLYNOM>::to_set_gradient_function(st
d::array<double, POLYNOM>(*_f)(double*,void*))
{
    gradient_function = _f;
}


template <int POLYNOM>
double
gradient_method_step_based<POLYNOM>::to_get_fitness_function_val
ue(void* param)
{
    return fitness_function(answer.begin(), param);
}

template <int POLYNOM>
stochastic_gradient_method_step_based<POLYNOM>::stochastic_gradi
ent_method_step_based()
{
    srand(time(0));
    call_init_funct();
}

template <int POLYNOM>
std::array<double,                                          POLYNOM>
stochastic_gradient_method_step_based<POLYNOM>::init_funct_def()
{
    for (int i = 0; i < POLYNOM; ++i)
        this->answer[i] = myrand() * (init_max - init_min) +
init_min;
    return this->answer;
}
```

```
template <int POLYNOM>
std::array<double,                                          POLYNOM>
stochastic_gradient_method_step_based<POLYNOM>::call_init_funct(
)
{
    if (initiation_function == nullptr) return init_funct_def();
    return initiation_function();
}

template <int POLYNOM>
void
stochastic_gradient_method_step_based<POLYNOM>::to_set_tries_amo
unt(unsigned int _n)
{
    tries_amount = _n;
}

template <int POLYNOM>
void
stochastic_gradient_method_step_based<POLYNOM>::to_set_initiatio
n_minimum(double _num)
{
    init_min = _num;
}

template <int POLYNOM>
void
stochastic_gradient_method_step_based<POLYNOM>::to_set_initiatio
n_maximum(double _num)
{
    init_max = _num;
}

template <int POLYNOM>
void
stochastic_gradient_method_step_based<POLYNOM>::to_set_initiatio
n_function(std::array<double, POLYNOM> (*_f) ())
{
    initiation_function = _f;
}

template <int POLYNOM>
std::pair<std::array<double,POLYNOM>,
std::array<double,POLYNOM>                                  >
stochastic_gradient_method_step_based<POLYNOM>::to_solve(void*pa
ram)
{
    std::multimap<double, std::pair<std::array<double, POLYNOM>,
std::array<double, POLYNOM> > > rating;
    std::pair<std::array<double,   POLYNOM>,   std::array<double,
POLYNOM> > result;
    for (int i = 0; i < tries_amount; ++i)
```

```
    {
        call_init_funct();
        result                                                    =
gradient_method_step_based<POLYNOM>::to_solve(param);
        rating.insert(std::pair<double,
std::pair<std::array<double,     POLYNOM>,     std::array<double,
POLYNOM>    >    >    (this->to_get_fitness_function_value(param),
result) );
    }
    this->answer = rating.begin()->second.second;
    return rating.begin()->second;
}
#endif // GRADIENT_METHOD_H
```

Лістинг reference_signal_definder_static.h:

```
#ifndef REFERENCE_SIGNAL_DEFINDER_STATIC_H
#define REFERENCE_SIGNAL_DEFINDER_STATIC_H

#include "automated_control_system_element_interface.h"

class       Reference_signal_definder_static       :       public
Automated_control_system_element_interface
{
public:
    Reference_signal_definder_static(double=0);

    void to_set_signal(double);

    enum { SIZE = END_INTERFACE };

    //interface:
    virtual bool to_verify_amount_of_parameters() const override;
    virtual         bool         to_set_element_parameters(const
std::vector<double> &) override;
    virtual bool to_set_all_parameters(const std::vector<double>
&) override;
    virtual void to_calculate() override;
};

#endif // REFERENCE_SIGNAL_DEFINDER_STATIC_H
```

Лістинг registrator.h:

```
#ifndef REGISTRATOR_H
#define REGISTRATOR_H

#include"automated_control_system.h"
#include<string>
#include <fstream>
#include <vector>
```

```
class Registrator
{
private:
    const Automated_control_system * acs_model_status;
    std::string name_of_file;
    bool first_record_commited;
    virtual void to_record() = 0;
protected:
    bool to_actulize_the_fist_record_commited_status();
    const Automated_control_system * to_check_acs_model_status()
const;
public:
    Registrator();
    virtual ~Registrator() = 0;

    const char * to_check_name_of_file() const;
    void to_set_name_of_file(std::string);
    Registrator & operator<<(const Automated_control_system&);
};

class Registrator_to_txt_file : public Registrator
{
private:
    virtual void to_record() override;
protected:
    std::ofstream fout;
    bool to_actulize_the_fist_record_commited_status();
public:
    Registrator_to_txt_file();
    virtual ~Registrator_to_txt_file() override;
    bool is_open();
};

class        Registrator_to_txt_file_short        :        public
Registrator_to_txt_file
{
private:
    virtual void to_record() override;
public:
    Registrator_to_txt_file_short();
    virtual ~Registrator_to_txt_file_short() override;
};

class Registrator_to_std_vector : public Registrator
{
public:
    void to_record() override;
protected:
    std::vector<double> * records = nullptr;
public:
    Registrator_to_std_vector();
    virtual ~Registrator_to_std_vector();
    void to_set_vector(std::vector<double>&);
```

```
    void to_set_vector(std::vector<double>*);
    void to_set_vector(std::shared_ptr<std::vector<double>>);
};

class     Registrator_to_std_vector_difference     :     public
Registrator_to_std_vector
{
    void to_record() override;
};

#endif // REGISTRATOR_H
```

Лістинг regulator.h:

```
#ifndef REGULATOR_H
#define REGULATOR_H

#include"automated_control_system_element_interface.h"
#include<memory>

class             PID_regulator             :             public
Automated_control_system_element_interface
{
private:
    double error();
    double output();
    double P();
    double I();
    double D();
public:
    PID_regulator();

    void to_set_koefficients(double=0, double=0, double=0);
    void to_receive_reference_signal(double);

    enum {
        BEGIN_REGULATOR = END_INTERFACE,                    ///
BE CRARE FOR THE TRUE OF THE EQUATION !!

        K_P = BEGIN_REGULATOR + 0,

        K_D = BEGIN_REGULATOR + 1,

        K_I = BEGIN_REGULATOR + 2,

        PROPORTIONAL_SIGNAL = BEGIN_REGULATOR + 3,

        INTEGRAL_SIGNAL = BEGIN_REGULATOR + 4,

        DERIVATIVE_SIGNAL = BEGIN_REGULATOR + 5,

        PREVIOUS_ERROR = BEGIN_REGULATOR + 6,
```

```
        REFERENCE_SIGNAL = BEGIN_REGULATOR + 7,

        ERROR_SIGNAL = BEGIN_REGULATOR + 8,

        LAST_REGULATOR = ERROR_SIGNAL,                          ///
BE CRARE FOR THE TRUE OF THE EQUATION !!
        END_REGULATOR = LAST_REGULATOR + 1,
        SIZE = END_REGULATOR
    };

    //interface:
    virtual bool to_verify_amount_of_parameters() const override;
    virtual         bool         to_set_element_parameters(const
std::vector<double> &) override;
    virtual bool to_set_all_parameters(const std::vector<double>
&) override;
    virtual void to_calculate() override;
};


#endif // REGULATOR_H
```

Лістинг regulator.cpp:

```
#include "regulator.h"

double PID_regulator::error()
{
    return             parameters[REFERENCE_SIGNAL]            -
parameters[INPUT_SIGNAL];
}

double PID_regulator::output()
{
    return P() + I() + D();
}

double PID_regulator::P()
{
    return  parameters[PROPORTIONAL_SIGNAL]  =  parameters[K_P]  *
parameters[ERROR_SIGNAL];
}

double PID_regulator::I()
{
    return  parameters[INTEGRAL_SIGNAL]  +=  parameters[K_I]  *
parameters[ERROR_SIGNAL] * parameters[DT];
}

double PID_regulator::D() // can be paralleled
{
```

```
    parameters[DERIVATIVE_SIGNAL]    =       parameters[K_D]    *
(  parameters[ERROR_SIGNAL]  -  parameters[PREVIOUS_ERROR]  )  /
parameters[DT];
    parameters[PREVIOUS_ERROR] = parameters[ERROR_SIGNAL];
    return parameters[DERIVATIVE_SIGNAL];
}

PID_regulator::PID_regulator()                                  :
Automated_control_system_element_interface()
{
    the_type = PID_regulator::REGULATOR;
    while (parameters.size() < SIZE) parameters.push_back(0);
}

void  PID_regulator::to_set_koefficients(double  kp,  double  ki,
double kd)
{
    parameters[K_P] = kp > 0 ? kp : 0;
    parameters[K_I] = ki > 0 ? ki : 0;
    parameters[K_D] = kd > 0 ? kd : 0;
}

void PID_regulator::to_receive_reference_signal(double r)
{
    parameters[REFERENCE_SIGNAL] = r;
}

bool PID_regulator::to_verify_amount_of_parameters() const
{
    if (parameters.size() != SIZE) return false;
    return true;
}

bool               PID_regulator::to_set_element_parameters(const
std::vector<double> & _r_parameters)
{
    {
        auto i = std::begin(parameters);
        std::advance(i, END_INTERFACE);
        auto j = std::begin(_r_parameters);
        std::advance(j, END_INTERFACE);
        int k = 0;
        for (
                ;
                i    !=    std::end(parameters)    &&    j    !=
std::end(_r_parameters) && k < END_REGULATOR;
                ++i, ++j, ++k
             )
            *i = *j;
        if    (    !(i    ==    std::end(parameters)    &&    j    ==
std::end(_r_parameters)) ) return false;
    }
    return true;
```

```
}

bool                    PID_regulator::to_set_all_parameters(const
std::vector<double> & _r_parameters)
{
    {
        auto i = std::begin(parameters);
        auto j = std::begin(_r_parameters);
        int k = 0;
        for (
                ;
                i    !=    std::end(parameters)    &&    j    !=
std::end(_r_parameters) && k < END_REGULATOR;
                ++i, ++j, ++k
             )
            *i = *j;
        if    (    !(i    ==    std::end(parameters)    &&    j    ==
std::end(_r_parameters)) ) return false;
    }
    return true;
}

void PID_regulator::to_calculate()
{
    parameters[ERROR_SIGNAL] = error();
    parameters[OUTPUT_SIGNAL] = output();
}
```

Лістинг regulator_tuner.h:

```
#ifndef REGULATOR_TUNER_H
#define REGULATOR_TUNER_H

#include"automated_control_system.h"

#include "generative_algorithm.h"
#include "gradient_method.h"

struct parameters_for_fitness_function
{
    double dt = 0;
    double length = 0;
    double t_registrate = 0;
};

struct parameters_for_varied_fitness_function
{
    double times = 0;
    double min = 0;
    double max = 0;
};

struct parameters_for_gradient
```

```
{
    double last_value_f = 0;
    double dx = 0;
    double learn_step = 0;

    int tries = 1;

    double learn_step_distortion = 1;
    double learn_step_distortion_velocity = 0;
};

struct parameters_p_objects
{
    Automated_control_system * p_acs_model = nullptr;
    PID_regulator * p_regulator = nullptr;
};

inline  void  to_copy_parameters_p_objects(parameters_p_objects&
_receiver, parameters_p_objects& _sender)
{
    _receiver.p_acs_model = _sender.p_acs_model;
    _receiver.p_regulator = _sender.p_regulator;
}

struct parameters_configurations_for_optimizer
{
    unsigned int iterations = 0;
    unsigned int agents = 0;
    unsigned int new_agents = 0;
    double mutation_step = 0;
    double mutation_propability = 0;
    double min_init = 0;
    double max_init = 0;
};

struct parameters_for_optimizer
{
    parameters_p_objects parameters_p_objects_parameters_obj;

    parameters_for_fitness_function
parameters_for_fitness_function_obj;
    parameters_for_varied_fitness_function
parameters_for_varied_fitness_function_obj;
    parameters_for_gradient parameters_for_gradient_obj;

    parameters_configurations_for_optimizer
parameters_configurations_for_optimizer_obj;
};

inline                                                  void
to_copy_parameters_for_optimizer(parameters_for_optimizer&
_receiver, parameters_for_optimizer& _sender)
{
```

```
to_copy_parameters_p_objects(_receiver.parameters_p_objects_para
meters_obj, _sender.parameters_p_objects_parameters_obj);
    _receiver.parameters_for_fitness_function_obj               =
_sender.parameters_for_fitness_function_obj;
    _receiver.parameters_for_varied_fitness_function_obj        =
_sender.parameters_for_varied_fitness_function_obj;
    _receiver.parameters_for_gradient_obj                       =
_sender.parameters_for_gradient_obj;
    _receiver.parameters_configurations_for_optimizer_obj       =
_sender.parameters_configurations_for_optimizer_obj;
}

double fitness_function_varied_reference_signal(double*,void*);

std::array<double, 3> gradient_by_step(double*,void*);

class Regulator_tuner_interface
{
private:
    std::array<double, 3> answer { 0, 0, 0 };
protected:
    parameters_for_optimizer * p_parameters = nullptr;

    bool parameters_is_null() { return p_parameters == nullptr; }
    void to_set_answer(std::array<double, 3> _arg) { answer =
_arg; }
public:
    Regulator_tuner_interface (parameters_for_optimizer&);
    Regulator_tuner_interface ();
    virtual ~Regulator_tuner_interface() = 0;

    void to_set_parameters(parameters_for_optimizer&);
    void to_reset_parameters();
    std::array<double, 3> & to_check_answer();

    virtual void to_tune() = 0;
};

class   Regulator_tuner_automated_manual_algorithm_interface   :
virtual public Regulator_tuner_interface
{
public:
    Regulator_tuner_automated_manual_algorithm_interface() {}

Regulator_tuner_automated_manual_algorithm_interface(parameters_
for_optimizer&);
    virtual
~Regulator_tuner_automated_manual_algorithm_interface() = 0;
    virtual void to_tune() = 0;
};
```

```
class Regulator_tuner_my_optimizer_interface : virtual public
Regulator_tuner_interface
{
public:
    Regulator_tuner_my_optimizer_interface() {}
    virtual ~Regulator_tuner_my_optimizer_interface() = 0;
    virtual void to_tune() = 0;
};

class Regulator_tuner_side_optimizer_interface : virtual public
Regulator_tuner_interface
{};

class    Regulator_tuner_my_generative_algorithm    :    public
Regulator_tuner_my_optimizer_interface
{
private:
    generative_algorithm<3> generative_algorithm_obj;
public:

Regulator_tuner_my_generative_algorithm(parameters_for_optimizer
&                       _parameters_to_set)                      :
Regulator_tuner_interface(_parameters_to_set) {}
    Regulator_tuner_my_generative_algorithm() {}
    virtual ~Regulator_tuner_my_generative_algorithm() {}

    void to_tune() override;
};

class     Regulator_tuner_my_gradient_algorithm     :     public
Regulator_tuner_my_optimizer_interface
{
private:
    stochastic_gradient_method_step_based<3>
stochastic_gradient_method_step_based_obj;
    double learn_step_change = 1;
    double learn_step_change_velocity = 1;
public:

Regulator_tuner_my_gradient_algorithm(parameters_for_optimizer&
_parameters_to_set)                                              :
Regulator_tuner_interface(_parameters_to_set) {}
    Regulator_tuner_my_gradient_algorithm() {}
    virtual ~Regulator_tuner_my_gradient_algorithm() {}

    void to_tune() override;
};

class    Regulator_tuner_my_ziegler_nichols_method    :    public
Regulator_tuner_automated_manual_algorithm_interface
{
private:
    double k_start = 0.1;
```

```
    double k_crit_prev = 0;
protected:
    double to_check_k_crit_prev() { return k_crit_prev; }
public:

//Regulator_tuner_my_ziegler_nichols_method(parameters_for_optim
izer&);
    Regulator_tuner_my_ziegler_nichols_method() {}

Regulator_tuner_my_ziegler_nichols_method(parameters_for_optimiz
er&                      _parameters_to_set)                    :
Regulator_tuner_automated_manual_algorithm_interface(_parameters
_to_set) {}
    virtual ~Regulator_tuner_my_ziegler_nichols_method() {}
    void to_set_start_k(double _k = 0.1) { k_start = _k; }

    void to_tune() override;

    virtual void to_do_first_step();
    virtual bool to_do_aproximating_step_Is_aproximated();
    virtual void to_do_second_step();
    virtual void to_do_third_step();
    virtual void to_do_final_step();
};

class Regulator_tuner
{};
#endif // REGULATOR_TUNER_H
```

Лістинг regulator_tuner.cpp:

```
#include "regulator_tuner.h"

Regulator_tuner_interface::Regulator_tuner_interface(parameters_
for_optimizer& _parameters_to_set)
{
    p_parameters = new parameters_for_optimizer;
    to_copy_parameters_for_optimizer(*p_parameters,
_parameters_to_set);
}

Regulator_tuner_interface::Regulator_tuner_interface()          :
p_parameters(nullptr)
{
    ;
}

Regulator_tuner_interface::~Regulator_tuner_interface()
{
    if (!parameters_is_null()) delete p_parameters;
}
```

```
void
Regulator_tuner_interface::to_set_parameters(parameters_for_opti
mizer& _parameters_to_set)
{
    if(parameters_is_null())       p_parameters       =       new
parameters_for_optimizer;
    to_copy_parameters_for_optimizer(*p_parameters,
_parameters_to_set);
}


void Regulator_tuner_interface::to_reset_parameters()
{
    if(!parameters_is_null()) delete p_parameters;
}


std::array<double,                          3>                          &
Regulator_tuner_interface::to_check_answer()
{
    return answer;
}

Regulator_tuner_my_optimizer_interface::~Regulator_tuner_my_opti
mizer_interface() {}

void Regulator_tuner_my_generative_algorithm::to_tune()
{

generative_algorithm_obj.to_set_min_init(p_parameters->parameter
s_configurations_for_optimizer_obj.min_init);

generative_algorithm_obj.to_set_max_init(p_parameters->parameter
s_configurations_for_optimizer_obj.max_init);

generative_algorithm_obj.to_set_mutation_step(p_parameters->para
meters_configurations_for_optimizer_obj.mutation_step);

generative_algorithm_obj.to_set_mutation_propability(p_parameter
s->parameters_configurations_for_optimizer_obj.mutation_propabil
ity);

generative_algorithm_obj.to_set_agents(p_parameters->parameters_
configurations_for_optimizer_obj.agents);

generative_algorithm_obj.to_set_new_agents(p_parameters->paramet
ers_configurations_for_optimizer_obj.new_agents);

generative_algorithm_obj.to_set_iterations(p_parameters->paramet
ers_configurations_for_optimizer_obj.iterations);
```

```
generative_algorithm_obj.to_set_fitnes_function(fitness_function
_varied_reference_signal);

generative_algorithm_obj.to_set_fitness_function_parameters(p_pa
rameters);


to_set_answer( generative_algorithm_obj.to_solve_array_out() );

    auto k = to_check_answer();

p_parameters->parameters_p_objects_parameters_obj.p_regulator->t
o_set_koefficients(k[0], k[1], k[2]);
}

void Regulator_tuner_my_gradient_algorithm::to_tune()
{



stochastic_gradient_method_step_based_obj.to_set_tries_amount(p_
parameters->parameters_for_gradient_obj.tries);

stochastic_gradient_method_step_based_obj.to_set_initiation_mini
mum(p_parameters->parameters_configurations_for_optimizer_obj.mi
n_init);

stochastic_gradient_method_step_based_obj.to_set_initiation_maxi
mum(p_parameters->parameters_configurations_for_optimizer_obj.ma
x_init);

stochastic_gradient_method_step_based_obj.to_set_initiation_func
tion(nullptr);


stochastic_gradient_method_step_based_obj.to_set_step(p_paramete
rs->parameters_for_gradient_obj.dx);

stochastic_gradient_method_step_based_obj.to_set_learn_coefficie
nt(p_parameters->parameters_for_gradient_obj.learn_step);


stochastic_gradient_method_step_based_obj.to_set_iteration(p_par
ameters->parameters_configurations_for_optimizer_obj.iterations)
;

stochastic_gradient_method_step_based_obj.to_set_fitness_functio
n(fitness_function_varied_reference_signal);

stochastic_gradient_method_step_based_obj.to_set_gradient_functi
on(gradient_by_step);
```

```
//stochastic_gradient_method_step_based_obj.to_get_fitness_funct
ion_value();


to_set_answer(  stochastic_gradient_method_step_based_obj.to_solv
e_array_out(p_parameters) );

    auto k = to_check_answer();

    //for ( auto & i : k ) if (1) i = 0;


p_parameters->parameters_p_objects_parameters_obj.p_regulator->t
o_set_koefficients(k[0], k[1], k[2]);
}

Regulator_tuner_automated_manual_algorithm_interface::Regulator_
tuner_automated_manual_algorithm_interface(parameters_for_optimi
zer& _parameters_to_set)
{
    p_parameters = new parameters_for_optimizer;
    to_copy_parameters_for_optimizer(*p_parameters,
_parameters_to_set);
}

void Regulator_tuner_my_ziegler_nichols_method::to_tune()
{
    PID_regulator            *            p_regulator            =
p_parameters->parameters_p_objects_parameters_obj.p_regulator;
    Automated_control_system        *        p_acs_model        =
p_parameters->parameters_p_objects_parameters_obj.p_acs_model;

    to_do_first_step();
    while (!to_do_aproximating_step_Is_aproximated()) continue;
    to_do_second_step();
    to_do_third_step();
    to_do_final_step();
}

#include "experiment_executor.h"
#include "default_configuration_setter.h"
#include "container_analyzer.h"

Regulator_tuner_automated_manual_algorithm_interface::~Regulator
_tuner_automated_manual_algorithm_interface() {}

void
Regulator_tuner_my_ziegler_nichols_method::to_do_first_step()
{
    // DELETE
    std::cout << "firstStep_start\n" << std::flush;
    // END_DELETE
```

```
    //data
    PID_regulator           *           p_regulator           =
p_parameters->parameters_p_objects_parameters_obj.p_regulator;
    Automated_control_system        *        p_acs_model        =
p_parameters->parameters_p_objects_parameters_obj.p_acs_model;
    double                      t_registrare                   =
p_parameters->parameters_for_fitness_function_obj.t_registrate;

    //data setting //Experiment_executor_velocity_record
std::make_shared<Experiment_executor_for_fitness_function>();
    std::shared_ptr<Experiment_executor_velocity_record>
p_experiment                                                   =
std::make_shared<Experiment_executor_velocity_record>();
    Default_configuration_setter setter;
    setter.to_set_experiment_parameters(p_experiment.get());

p_experiment->to_set_t_length(p_parameters->parameters_for_fitne
ss_function_obj.length);
    p_experiment->to_get_model_to_run(p_acs_model);

    //alg
    bool cycle_must_proceed = true;
    std::vector<double> output;
    p_experiment->to_set_vector(output);
    container_analyzer analyzer;
    std::array<double, 3> & ans = to_check_answer();
    ans[0] += k_start;
    ans[1] = 0;
    ans[2] = 0;
    while(cycle_must_proceed)
    {
        k_crit_prev = ans[0];
        ans[0] += k_start;
        p_regulator->to_set_koefficients(ans[0]);
        p_experiment->to_run();
        if ( analyzer.is_oscillating(output) ) cycle_must_proceed
= false;

        // DELETE
        std::cout  <<  "firstStep_cycle\t"  <<  ans[0]  <<  "\n"<<
std::flush;
        // END_DELETE

        output.clear();
    }
}

bool
Regulator_tuner_my_ziegler_nichols_method::to_do_aproximating_st
ep_Is_aproximated()
{
    //data
```

```
    PID_regulator            *            p_regulator            =
p_parameters->parameters_p_objects_parameters_obj.p_regulator;
    Automated_control_system        *        p_acs_model        =
p_parameters->parameters_p_objects_parameters_obj.p_acs_model;
    std::array<double, 3> & ans = to_check_answer();
    std::shared_ptr<Experiment_executor_for_fitness_function>
p_experiment                                                    =
std::make_shared<Experiment_executor_for_fitness_function>();
    container_analyzer analyzer;
    std::vector<double> output;
    double                      t_registrare                    =
p_parameters->parameters_for_fitness_function_obj.t_registrate;

    //data setting
    Default_configuration_setter setter;
    setter.to_set_experiment_parameters(p_experiment.get());

p_experiment->to_set_t_length(p_parameters->parameters_for_fitne
ss_function_obj.length);
    p_experiment->to_get_model_to_run(p_acs_model);
    p_experiment->to_set_vector(output);

    //alg
    ans[0] *= 0.75;
    p_regulator->to_set_koefficients(ans[0]);
    p_experiment->to_run();

    if ( analyzer.is_oscillating(output) ) return false;

    else
    {
        output.clear();
        k_crit_prev = ans[0];

        ans[0] /= 0.75;
        ans[0] *= 0.875;
        p_regulator->to_set_koefficients(ans[0]);
        p_experiment->to_run();


        if ( !analyzer.is_oscillating(output) ) ans[0] /= 0.875;
    }
    // DELETE
    std::cout << "aproximateStep_cycle\t" << ans[0] << "\n" <<
std::flush;
    // END_DELETE

    return true;
}

void
Regulator_tuner_my_ziegler_nichols_method::to_do_second_step()
{
```

```
    //data
    PID_regulator           *           p_regulator           =
p_parameters->parameters_p_objects_parameters_obj.p_regulator;
    Automated_control_system        *       p_acs_model        =
p_parameters->parameters_p_objects_parameters_obj.p_acs_model;
    double                      t_registrare                   =
p_parameters->parameters_for_fitness_function_obj.t_registrate;

    //data setting
    std::shared_ptr<Experiment_executor_for_fitness_function>
p_experiment                                                   =
std::make_shared<Experiment_executor_for_fitness_function>();
    Default_configuration_setter setter;
    setter.to_set_experiment_parameters(p_experiment.get());

p_experiment->to_set_t_length(p_parameters->parameters_for_fitne
ss_function_obj.length);
    p_experiment->to_get_model_to_run(p_acs_model);

    //alg
    std::array<double, 3> & ans = to_check_answer();
    std::vector<double> output;
    p_experiment->to_set_vector(output);
    p_experiment->to_run();

    container_analyzer analyzer;
    ans[1]     =     analyzer.to_calculate_period_in_Get(output,
t_registrare);
}

void
Regulator_tuner_my_ziegler_nichols_method::to_do_third_step()
{
    std::array<double, 3> results = to_check_answer();
    to_set_answer( std::array<double, 3> { 0.6 * results[0] , 1.2
* results[0] / results[1], 0.125 * results[0] * results[1]} );
}

void
Regulator_tuner_my_ziegler_nichols_method::to_do_final_step()
{
    PID_regulator           *           p_regulator           =
p_parameters->parameters_p_objects_parameters_obj.p_regulator;

    auto ans = to_check_answer();
    p_regulator->to_set_koefficients(ans[0], ans[1], ans[2]);
}
```