МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра фундаментальної та прикладної математики

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«АВТОМАТИЗАЦІЯ ПРОЦЕСУ МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ, АНАЛІЗУ ТА КЕРУВАННЯ СКЛАДНОЮ ДЕТЕРМІНОВАНОЮ ПОЗИТИВНОЮ ДИНАМІЧНОЮ СИСТЕМОЮ»**

Виконала: студент 2 курсу, групи 8.1132
спеціальності 113 Прикладна математика
(шифр і назва спеціальності)
освітньої програми Прикладна математика
(назва освітньої програми)
С. С. Полос
(ініціали та прізвище)

Керівник доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Леонтьєва В. В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент декан математичного факультету, професор, д.т.н. Гоменюк С. І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра фундаментальної та прикладної математики

Рівень вищої освіти магістр

Спеціальність 113 Прикладна математика

(шифр і назва)

Освітня програма Прикладна математика

ЗАТВЕРДЖУЮ

Завідувач кафедри фундаментальної та прикладної математики, д.т.н., професор

Гребенюк С.М.

(підпис)

« » 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Полосу Станіславу Сергійовичу

(прізвище, ім'я та по-батькові)

1. Тема роботи (проєкту) Автоматизація процесу математичного моделювання, аналізу та керування складною детермінованою позитивною динамічною системою

керівник роботи (проєкту) Леонтьєва Вікторія Володимирівна, к.ф.-м.н., доцент

(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 28.11.2023

3. Вихідні дані до роботи 1) Постановка задачі
2) Перелік літератури

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Збір даних
2) Реалізація
3) Перевірка працездатності програмного продукту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1			
2			
3			

7. Дата видачі завдання 01.05.2023

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	10.05.2023	
2.	Збір вихідних даних.	25.05.2023	
3.	Обробка методичних та теоретичних	03.09.2023	
	джерел.		
4.	Розробка першого розділу.	15.10.2023	
5.	Розробка другого розділу.	24.10.2023	
6.	Розробка третього розділу.	10.11.2023	
7.	Оформлення та нормоконтроль	28.11.2023	
	кваліфікаційної роботи.		
8.	Захист кваліфікаційної роботи.	12.12.2023	

Студент _____ С. С. Полос
(підпис) (ініціали та прізвище)

Керівник роботи _____ В. В. Леонтьєва
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О. Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Автоматизація процесу математичного моделювання, аналізу та керування складною детермінованою позитивною динамічною системою»: 124 с., 17 рис., 2 табл., 29 джерел, 2 додатки.

АВТОМАТИЗАЦІЯ, АЛГОРИТМІЗАЦІЯ, АНАЛІЗ, ДИСКРЕТНА Й НЕПЕРЕРВНА МОДЕЛІ, КЕРУВАННЯ, ПОЗИТИВНА СИСТЕМА, СПОСТЕРЕЖЕННЯ.

Об’єкт дослідження – складна детермінована позитивна динамічна система дискретного або неперервного часу.

Мета роботи – розробка методики дослідження та реалізуючої її програми для автоматизації процесів моделювання й розв’язання задач аналізу й керування за складною детермінованою позитивною динамічною системою із забезпеченням можливості отримання аналітичних результатів за довільний проміжок часу.

Метод дослідження – аналітичний.

Робота присвячена проведенню дослідження з моделювання поведінки, розв’язання задач аналізу й керування за складною позитивною динамічною системою із визначенням алгоритму послідовних дій в визначенні та розв’язанні зазначених задач та розробкою реалізуючого його програмного забезпечення.

Структурно робота складається з 3 розділів. В 1 розділі роботи розглянуто основні поняття теорії позитивних систем; розкрито особливості об’єкта дослідження, наведено основні задачі та підходи їх розв’язання з метою аналізу та здійснення різних видів керування. У 2 та 3 розділах наведено основні етапи розробленого алгоритму здійснюваних в роботі досліджень, за яким отримано програмну реалізацію змін основних характеристик досліджуваного об’єкта, наведено приклади роботи програми для систем дискретного та неперервного часу.

SUMMARY

Master's Qualifying Thesis "Automation of the process of mathematical modeling, analysis and control of complex deterministic positive dynamical system": 124 p., 17 figs., 2 tables, 29 sources, 2 applications.

AUTOMATION, ALGORITHMIZATION, ANALYSIS, DISCRETE AND CONTINUOUS MODELS, CONTROL, POSITIVE SYSTEM, OBSERVATION.

The object of research is the complex deterministic positive dynamical system of discrete or continuous time.

The purpose of the work is to develop a research methodology and its implementation program for automating the processes of modeling and solving problems of analysis and control of a complex deterministic positive dynamical system with the ensuring the possibility of obtaining analytical results in an arbitrary period of time.

The method of research is analytical.

The work is devoted to conducting research on modeling behavior, solving problems of analysis and control of a complex positive dynamical system with defining an algorithm for sequential actions in determining and solving these problems and developing software that implements it.

Structurally, the work consists of 3 chapters. In the 1 chapter of the work the basic concepts of the theory of positive systems and control theory are considered; features of the research object are revealed; the main problems and methods for solving them are given for the purpose of analyzing and implementing various types of control. Chapters 2 and 3 present the main stages of the developed algorithm for the carrying out work' research, according to which the software implementation of main characteristics' changes of the studied object is obtained, examples of the developed program for discrete and continuous time systems are given.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Позитивна динамічна система: загальне поняття, математичні моделі динаміки руху, властивості, основні розв’язувані задачі 12
1.1 Загальне поняття про позитивну динамічну систему 12
1.2 Огляд основних видів математичних моделей, які застосовуються для опису динаміки й статики детермінованої позитивної системи 15
1.2.1 Динамічна дискретна математична модель детермінованої позитивної динамічної системи 16
1.2.2 Динамічна неперервна математична модель детермінованої позитивної динамічної системи 19
1.2.3 Статична математична модель детермінованої позитивної системи 21
1.3 Основні властивості позитивної динамічної системи як об’єкту аналізу та керування 23
1.4 Основні задачі керування, розв’язувані для керованої позитивної динамічної системи: огляд задач та обґрунтування вибору тої чи іншої задачі керування 26
2 Алгоритмізація процесу розв’язання задач аналізу й керування для складної детермінованої позитивної динамічної системи та аналізу отримуваних результатів 31
2.1 Вхідні дані для проведення дослідження 31
2.2 Основні етапи дослідження та розв’язання поставлених задач аналізу й керування 33
2.3 Вихідні дані за здійснюваним дослідженням 38

3 Автоматизація процесу розв’язання задач аналізу й керування за складною детермінованою позитивною динамічною системою 40
3.1 Вибір технологій. Загальна характеристика розробленого програмного продукту 40
3.2 Інтерфейс користувача. Приклади з проведення обчислень 42
3.3 Структура організації програмного коду 57
Висновки 60
Перелік посилань 62
Додаток А Лістинг вихідного коду програмного продукту 65
Додаток Б Вміст файлу README.md для розробленого програмного продукту 119

ВСТУП

При здійсненні наукових досліджень досить часто виникає проблема обчислювальної складності, розв’язувана завдяки використанню ресурсів обчислювальної техніки із застосуванням відповідного програмного забезпечення [6]. Дана робота присвячена розв’язанню зазначеної проблеми відносно певного підкласу складних динамічних систем, що характеризується властивістю позитивності вхідних змінних, траєкторій руху систем та вихідних характеристик. Основною особливістю окреслених динамічних систем при цьому виступає використання таких підходів та методів розв’язання задач, які здатні зберегти володіння системою властивості позитивності на протязі всього досліджуваного періоду часу [10-13, 28]. При цьому такі складні динамічні системи, з урахуванням можливостей їх початкового відхилення від затребуваних (за своїм фізичним змістом) властивостей, а також виникнення в процесі дослідження необхідності їх стабілізації й покращення певних динамічних властивостей, потребують детального дослідження та розв’язання додаткових задач аналізу й керування для досягнення як їх повноцінного функціонування, так і ефективного керування в межах поставлених цілей. Проведення таких комплексних досліджень зазвичай є досить трудомістким та ресурсовитратним процесом, особливо при моделюванні динамічних систем із великою кількістю взаємопов’язаних підсистем [2, 14, 17, 20]. А оскільки використовувані при проведенні такого дослідження математичні методи та підходи для збереження властивості позитивності досліджуваних систем в своїй більшості були попередньо модифіковані, то задача автоматизації моделювання, аналізу й керування за позитивними динамічними системами стає особливо актуальною.

В якості мети кваліфікаційної роботи обрано розробку методики дослідження та реалізуючої її програми для автоматизації процесів

моделювання й розв’язання задач аналізу й керування за складною детермінованою позитивною динамічною системою із забезпеченням можливості отримання аналітичних результатів за довільний проміжок часу. Розв’язання поставленої задачі дозволить забезпечити встановлений рівень точності та ефективності досліджень, знизити трудомісткість здійснюваних обчислень, скоротити затребувані витрати часу та ресурсів для обробки інформації, а також дозволить покращити процеси моделювання, аналізу й керування досліджуваними складними системами та візуалізувати отримувані результати дослідження.

Об'єктом дослідження в роботі виступає складна детермінована позитивна динамічна система дискретного або неперервного часу. Поведінка такої системи, з урахуванням сталості структури свого опису з плином або дискретного, або неперервного часу, описується дискретною або неперервною математичними моделями, які представляються відповідно лінійними неоднорідними векторно-матричними різницевими або диференціальними рівняннями із матрицями сталих коефіцієнтів. Такі системи мають місце в таких галузях знань, де досліджуються процеси, що оперують невід’ємними протягом всього аналізовуваного періоду часу показниками, наприклад, в галузях економіки, екології, біології і т. ін. [26, 28, 29].

Предметом дослідження виступають основні характеристики, особливості, властивості складних детермінованих позитивних динамічних систем та підходи до їх моделювання, аналізу й керування, а також можливості розробки та подальшого використання розробленого спеціального програмного забезпечення, здатного автоматизувати окреслені процеси дослідження з відносно великим ступенем участі людини у процесі дослідження для прийняття рішення стосовно вибору необхідної (з позиції поставленої мети) розв’язуваної задачі та з абсолютно низьким ступенем її участі у процесах отримання, перетворення, передавання і використання оперовуваних матеріалів та інформації, а також спрямованого на зменшення

трудомісткості виконуваних при проведенні дослідження операцій й скорочення обсягів витрачуваного при цьому часу, а, отже, орієнтованого на збільшення точності отримуваних результатів та спрощення процесу їх отримання у цілому у найменш можливий термін.

Завданнями роботи виступають аналіз предметної області, який окреслює постановку проблеми та характеризує сучасний стан її розв'язаності у науковій площині сьогодення; дослідження основних характеристик й особливостей складної детермінованої позитивної динамічної системи з дотриманням володіння нею її основних властивостей як системи керування й регулювання; дослідження математичних моделей використовуваних для опису поведінки аналізовуваної системи з урахуванням сталості структури свого опису з плином або дискретного, або неперервного часу; аналіз основних властивостей досліджуваної системи як системи керування; аналіз підходів до розв'язання задач керування в умовах повної інформації про стани досліджуваної системи; розробка загальної методики з проведення комплексного дослідження визначення змін основних характеристик об'єкта та її практична реалізація у вигляді програмного продукту.

Структурно робота складається з 3 розділів.

У першому розділі роботи наводяться основні теоретичні відомості, проводиться розкриття понять та основних визначень теорії позитивних систем, що використовуються при описі складної детермінованої позитивної динамічної системи, представляються результати аналізу основних властивостей, притаманних досліджуваній системі як системі керування, надається огляд дискретної та неперервної динамічних та статичної моделей, що описують поведінку досліджуваної позитивної системи, наводиться огляд основних задач керування, що можуть ставитись для досягнення додаткових цілей дослідження, на які спирається пропонований у другому розділі роботи алгоритм проведення аналізу математичних моделей системи, розв'язання основних задач керування складною керованою позитивною динамічною

системою й отримуваних за зазначеними моделями результатів.

У другому розділі представляються основні етапи пропонованого в роботі алгоритму проведення аналізу математичної моделі, основних властивостей складної позитивної динамічної системи, розв’язання задач керування й отримуваних за ними результатів, на які спирається реалізація розробленого в роботі програмного забезпечення для автоматизації процесу здійснення відповідних досліджень.

У третьому розділі висвітлюються особливості програмної реалізації розробленого програмного забезпечення за представленим алгоритмом здійснення дослідження для спрощення розв’язання задач моделювання, аналізу й керування позитивними динамічними системами, що допомагає автоматизувати деякі етапи розв’язання задач виділеного класу складних детермінованих динамічних систем; видідяються основні переваги та особливості використання обраної мови програмування, надаються детальні описи інтерфейсу програми та інструкції з використання створеного програмного забезпечення для кінцевого користувача із результатами проведених обчислювальних експериментів на дискретній та неперервній математичних моделях досліджуваної складної детермінованої позитивної динамічної системи із використанням розробленого програмного продукту.

Результати, отримані в кваліфікаційній роботі, є важливими, науково й практично обґрунтованими. Розроблений програмний продукт може бути використано при дослідженні будь-яких складних динамічних систем (процесів, явищ), що вимагають за своїм змістовим навантаженням від основних характеристик модельованого об’єкта володіння властивістю позитивності на протязі всього інтервалу досліджуваного часу, а також затребують стабілізацію системи та/або покращення її динамічних властивостей в умовах повної інформації про стани досліджуваної системи.

1 ПОЗИТИВНА ДИНАМІЧНА СИСТЕМА: ЗАГАЛЬНЕ ПОНЯТТЯ, МАТЕМАТИЧНІ МОДЕЛІ ДИНАМІКИ РУХУ, ВЛАСТИВОСТІ, ОСНОВНІ РОЗВ'ЯЗУВАНІ ЗАДАЧІ

В даному розділі надається детальна характеристика об'єкта дослідження, в якості якого обрано складну детерміновану позитивну динамічну систему, наводяться результати з аналізу існуючих дискретної й неперервної динамічних, а також статичної математичних моделей з виокремленням основних основні вимог та обмежень, що забезпечують виконання властивості позитивності досліджуваної системи та асимптотичної стійкості отримуваних за моделями розв'язків. Також в даному розділі досліджувана система подається як система керування, для якої наводяться основні властивості складної керованої позитивної динамічної системи із наданням алгебраїчних критеріїв для їх перевірки, виокреслюються основні задачі керування, що можуть бути розв'язані при певних умовах.

1.1 Загальне поняття про позитивну динамічну систему

У сучасному світі, де складність технологічних, економічних, соціальних та природничих систем невпинно зростає, важливим аспектом стає розуміння та керування цими системами. Одним із ключових підходів у цьому контексті є теорія позитивних систем, яка займає особливе місце в сфері математичного моделювання та аналізу динамічних систем. Теорія позитивних систем досліджує клас систем, для яких можливо гарантувати позитивність певних характеристик (таких як стани, виходи тощо), що є критично важливим у багатьох практичних застосуваннях [2, 10, 11, 26, 29].

При цьому динамічна система називається позитивною, якщо її стан і вихід є невід'ємними для будь-якого невід'ємного початкового стану та

невід’ємного входу [10-13, 28, 29]. Цю властивість можна побачити в економічних системах, екологічних й біологічних процесах, мережевих комунікаціях та ймовірнісних системах [10-12, 26, 28, 29]. Так, позитивні системи знаходять застосування в сфері економіки через дослідження, пов’язані з позитивними економічними моделями, в екології (моделі популяційних динамік), біотехнології (моделі хімічних реакцій), інженерії (системи управління та автоматизація) та багатьох інших. Отже, визначення та аналіз позитивних динамічних систем має фундаментальне значення у різноманітних галузях – від екології та біології до економіки та інженерії.

Для лінійних систем позитивність призводить до певних видів вхідних матриць системи. Зокрема, система з дискретним або неперервним часом описується невід’ємними матрицями, які здавна були предметом вивчення в математичних науках [2, 10-12, 29]. Основні результати стосуються характеристики розташування власних значень і належать Перрону, Фробеніусу і Карпелевичу. Теоретико-системний підхід до позитивних лінійних систем було започатковано Луенбергером у його фундаментальній роботі [28] протягом 1980-х років. З того часу з'явилася вражаюча кількість теоретичних внесків у цю область. Питання дослідження позитивних систем підіймались у багатьох роботах вітчизняних та закордонних науковців [2, 10-13, 26, 28, 29]. Використання теорії позитивних систем дозволяє вченим та інженерам більш точно моделювати та передбачати поведінку складних систем, а також розробляти ефективні стратегії керування.

В основі теорії позитивних систем лежать основні постулати з різних галузей науки, наприклад, системного аналізу [7, 18, 19], теорії динамічних систем [5, 8, 9, 27, 28] та системології, кожна з яких висуває окрему низку понять та спеціальний математичний інструментарій, здатні з різних позицій описати досліджуваний об’єкт з дотриманням основних закономірностей та характеристик, притаманних певному класу досліджуваних систем, та використати його при здійсненні відповідних науково-обгрунтованих теоретичних й практичних досліджень.

У контексті системної теорії, позитивна система визначається як система, яка в будь-який момент часу має позитивні або невід'ємні стани та виходи [10, 12]. Це означає, що всі змінні стану та виходу системи обмежені знизу нулем, що є особливо важливим у багатьох застосуваннях, де негативні значення не мають фізичного чи логічного сенсу.

У контексті позитивних систем, динамічна система розглядається як математична модель, що описує, як система розвивається у часі у відповідь на вхідні сигнали або впливи [27, 28]. Ці системи можуть бути неперервними (описувані диференціальними рівняннями) або дискретними (описувані різницевими рівняннями).

Позитивні системи при цьому можуть бути як лінійними, так і нелінійними. Лінійні системи мають ту властивість, що відповідь системи на лінійну комбінацію входів є лінійною комбінацією відповідей на ці входи. Нелінійні системи не дотримуються цього правила, що робить їх аналіз значно складнішим. При цьому у лінійних позитивних системах використовуються невід'ємні матриці для опису переходів між станами. Аналізу властивостей цих матриць, особливо їхніх домінуючих власних значень та векторів, які визначають динаміку системи, проводиться за використанням теореми Фробеніусу-Перрону [11, 28], яка займає особливе місце при дослідженні позитивних динамічних систем. У нелінійних позитивних системах використовуються конвексні конуси для опису областей допустимих станів. Це дозволяє формалізувати поняття позитивності в більш широкому контексті.

Крім того, потрібно зауважити, що ключовою характеристикою позитивних систем виступає властивість стійкості розв’язків за математичними моделями системи [28]. Зазначена властивість визначає, чи повернеться досліджувана система до свого сталого стану після збурення. У контексті позитивних систем особливо важливою є властивість асимптотичної стійкості розв’язків за математичними моделями системи, виконання якої визначає здатність системи повертатися до стану рівноваги з

плином часу. Така властивість для розглядуваної в роботі детермінованої позитивної динамічної системи, з урахуванням певних обмежень, встановлюється виконуваною на всіх рівнях дослідження [10-13].

Наостанок потрібно зазначити, що наведені в даному розділі основні концепції та ключові положення, що лежать в основі теорії позитивних систем, надають можливість проводити аналіз досліджуваних систем та отримувати математичний опис поведінки зазначених систем з урахуванням характерних для встановлених внутрішніх особливостей процесів, що протікають в позитивній системі. В наступних підрозділах даного розділу на основі отриманих концепцій будуть надані результати огляду основних математичних моделей руху позитивної динамічної системи, результати аналізу основних властивостей та характеристик позитивних систем, а також визначено основні підходи до здійснюваного до досліджуваних систем керування.

1.2 Огляд основних видів математичних моделей, які застосовуються для опису динаміки й статики детермінованої позитивної системи

В даному підрозділі проводиться огляд основних видів математичних моделей, які застосовуються для опису динаміки й статики детермінованої позитивної системи і є об’єктом автоматизації у кваліфікаційній роботі, наводяться основні відомості з дискретної й неперервної динамічних та статичної моделей детермінованої позитивної системи з наданням основних властивостей моделей. При цьому математичний опис детермінованої системи передбачає, що зв'язок між входами та виходами досліджуваної позитивної системи описується за допомогою певних відомих детермінованих функцій, а вхідні значення системи представляються точно визначеними. В такому випадку поведінка детермінованої моделі,

використовуваної для математичного опису детермінованої позитивної динамічної системи, передбачається однозначним чином.

1.2.1 Динамічна дискретна математична модель детермінованої позитивної динамічної системи

Як відомо з класичної літератури, наприклад робіт [4, 8, 9, 21, 23], динамічне моделювання систем застосовується для дослідження поведінки системи у часі. У випадку, якщо опис досліджуваної системи передбачає врахування дискретних змінювань її характеристик у відповідні моменти часу (в таких системах всі процеси формуються лише в окремі моменти часу), то для опису її руху будується дискретна динамічна математична модель, яка використовується для аналізу процесів у дискретній системі та відображає процеси змінювання стану та характеристик системи в окремі дискретні моменти часу.

В даній роботі, при побудові математичної моделі позитивної системи, перш за все, враховується умова детермінованості системи, тобто припускається, що всі майбутні стани досліджуваної системи однозначно визначаються її поточним станом та не залежать від дії випадкових факторів.

За умови припущення дискретності змінювання стану та характеристик системи початковий математичний опис досліджуваної в роботі детермінованої позитивної динамічної системи представляється сукупністю лінійних різницевих рівнянь з деякими нелінійними правими частинами, за допомогою яких описуються фізичні процеси в певних функціональних елементах системи. В роботі, задля отримання аналітичних розв’язків, зазначений опис подається в векторно-матричній формі. В такому формулюванні математична модель детермінованої позитивної динамічної системи представляється векторно-матричним різницевим рівнянням виду [11]

$$X_{t+1} - (I - B)^{-1}(A - B)X_t = (I - B)^{-1}C_t,\ X(0) = X_0 \qquad (1)$$

або

$$X_{t+1} - \tilde{A}X_t = \tilde{B}C_t,\ X(0) = X_0, \qquad (2)$$

де $X_{t+1}, X_t -$ вектор-функції розмірності $n \times 1$ характеристик досліджуваної системи відповідно на момент $t+1$ та t її функціонування: $X_{t+1} = \left(X_{t+1}^1, X_{t+1}^2, \ldots, X_{t+1}^n\right)^T$, $X_t = \left(X_t^1, X_t^2, \ldots, X_t^n\right)^T$;

$A,\ B,\ \tilde{A} = (I - B)^{-1}(A - B),\ \tilde{B} = (I - B)^{-1} -$ матриці сталих коефіцієнтів розмірностей $n \times n$ із певними властивостями й обмеженностями, виконання яких забезпечують позитивність досліджуваної системи;

$C_t = \left(C_t^1, \ldots, C_t^n\right)^T -$ нелінійна в загальному вигляді вхідна вектор-функція, визначувана при дослідженні експериментальних або спостережуваних даних із застосуванням методології регресійного аналізу; $X(0) = X_0 = \left(X_0^1, X_0^2, \ldots, X_0^n\right)^T -$ вектор початкових станів системи.

Для представленої рівняннями (1) або (2) дискретної математичної моделі з метою забезпечення виконання властивості позитивності системи та отримання асимптотично стійких (за Ляпуновим) розв'язків за визначеним рівнянням моделі на складові (матриці коефіцієнтів та вхідні функції) моделі накладаються певні обмеження [10, 11]:

а) коефіцієнти сталих матриць $A = (a_{ij})_{n \times n}$, $B = (b_{ij})_{n \times n}$ мають задовольняти наступним умовам: $0 \le a_{ij} < 1,\ 0 \le b_{ij} < 1,\ a_{ij} \ge b_{ij},\ i, j = \overline{1, n}$;

б) матриці $A,\ B,\ (A - B),\ \tilde{A} = (I - B)^{-1}(A - B)$ сталих коефіцієнтів мають бути невід'ємними та продуктивними. *Перевірка невід'ємності* зазначених *матриць* моделей здійснюється шляхом перевірки невід'ємності

всіх їх елементів, що, в формулюванні для довільної матриці $P=\left(p_{ij}\right)$ полягає в наступному: матриця $P=\left(p_{ij}\right)$ вважається невід'ємною, якщо її елементи $p_{ij}\geq 0$. *Перевірка продуктивності матриць* здійснюється шляхом використання необхідних і достатніх умов продуктивності, сформульованих В. В. Леонтьєвим [10]. Причому, для запрограмовування перевірки умов продуктивності, з практичної точки зору, найбільш зручною є умова, яка в формулюванні для довільної матриці $P=\left(p_{ij}\right)$ полягає в наступному: квадратна матриця $P\geq 0$ вважається продуктивною, якщо існує зворотна матриця $\left(I-P\right)^{-1}\geq 0$, де $I-$ одинична матриця розмірності $n\times n$;

в) вектор-функція початкового стану $X\left(0\right)=X_0=\left(X_0^1,X_0^2,\ldots,X_0^n\right)^T$ має бути невід'ємною;

г) вектор-функція керування $C_t=\left(C_t^1,\ldots,C_t^n\right)^T$ має бути невід'ємною.

При цьому функції керування $C_t=\left(C_t^1,\ldots,C_t^n\right)^T$ розглядаються в наступних виглядах:

а) в лінійному вигляді: $C_t=C^0+Ct$;

б) в квадратичному вигляді: $C_t=C^0+Ct^2$;

в) в степеневому вигляді: $C_t=C^0+w^tC$;

Якщо в наявності повна інформація про стан системи, а також виконуються всі наведені обмеження, розв'язок за дискретною моделлю позитивної системи подається у вигляді [11]

$$X_t=\left(\left(I-B\right)^{-1}\left(A-B\right)\right)^t X_0+\sum_{i=1}^{t}\left(\left(I-B\right)^{-1}\left(A-B\right)\right)^{t-i}\left(I-B\right)^{-1}C_{i-1}$$

або

$$X_t = \tilde{A}^{t-1}\tilde{B}\left((A-B)X_0 + (I-A)X^*\right) + \tilde{A}^{t-1}\sum_{i=1}^{t-1}\tilde{A}^{-i}\tilde{B}C_i\,,$$

де $X^* = (I-A)^{-1}C^0$ – стаціонарний розв'язок (рівноважний стан системи);

$C^0 = \left(C_0^1, C_0^2, \ldots, C_0^n\right)^T$, $C_0^i \geq 0$.

1.2.2 Динамічна неперервна математична модель детермінованої позитивної динамічної системи

Як відомо [1, 4, 7-9, 22, 23], у класі неперервних систем всі процеси протікають у часі неперервно, тобто, щоб динамічна система відносилась до неперервних, всі характеристики в її елементах мають бути визначеними в кожний момент часу.

За умови припущення неперервності змінювання стану та характеристик системи початковий математичний опис досліджуваної в роботі детермінованої позитивної динамічної системи представляється сукупністю лінійних диференціальних рівнянь з нелінійними правими частинами, що описують фізичні процеси в окремих функціональних елементах системи. Як і для дискретної моделі, задля отримання аналітичних розв'язків, зазначений математичний опис функціонування системи неперервного часу подається в векторно-матричній формі. В такому формулюванні неперервна математична модель детермінованої позитивної динамічної системи представляється векторно-матричним диференціальним рівнянням виду [12]

$$\dot{X}(t) - (I-B)^{-1}(A-I)X(t) = (I-B)^{-1}C(t),\ X(0) = X_0 \qquad (3)$$

або

$$\dot{X}(t) - \tilde{\tilde{A}}X(t) = \tilde{B}C(t),\ X(0) = X_0\,, \qquad (4)$$

де $X(t)=(X_1(t), X_2(t),\ldots, X_n(t))^T$ – вектор-функція розмірності $n\times 1$ характеристик системи на даний момент її функціонування;

A, B, $\tilde{\tilde{A}}=(I-B)^{-1}(A-I)$, $\tilde{B}=(I-B)^{-1}$ – матриці сталих коефіцієнтів розмірностей $n\times n$ із певними властивостями й обмеженностями, виконання яких забезпечують позитивність досліджуваної системи;

$C(t)=(C_1(t), C_2(t),\ldots, C_n(t))^T$ – нелінійна в загальному вигляді вхідна вектор-функція;

$X(0)=X_0=\left(X_0^1, X_0^2,\ldots, X_0^n\right)^T$ – вектор початкових станів системи.

Для представленої рівняннями (3) або (4) неперервної математичної моделі з метою забезпечення виконання властивості позитивності системи та отримання асимптотично стійких (за Ляпуновим) розв'язків накладаються певні обмеження на основні матриці та компоненти моделі системи [10, 12]:

а) матриці A, B сталих коефіцієнтів мають бути невід'ємними та продуктивними, при цьому $A=(a_{ij})_{n\times n}$, $B=(b_{ij})_{n\times n}$, $0\le a_{ij}<1$, $0\le b_{ij}<1$, $a_{ij}\ge b_{ij}$, $i, j=\overline{1,n}$;

б) в матриці $\tilde{\tilde{A}}=(I-B)^{-1}(A-I)$ елементи, розташовані на головній діагоналі мають бути недодатними (тобто $\tilde{\tilde{a}}_{ii}\le 0$, $i=\overline{1,n}$), а недіагональні елементи – невід'ємними (тобто $\tilde{\tilde{a}}_{ij}\ge 0$ при $i\ne j$, $i, j=\overline{1,n}$);

в) вектор-функції $X(0)=X_0=\left(X_0^1, X_0^2,\ldots, X_0^n\right)^T$ та $C(t)=(C_1(t), C_2(t),\ldots, C_n(t))^T$ мають бути невід'ємними.

При цьому функції керування $C(t)=(C_1(t), C_2(t),\ldots, C_n(t))^T$ розглядаються в наступних виглядах:

а) в лінійному вигляді: $C(t)=C^0+Ct$;

б) в квадратичному вигляді: $C(t)=C^0+Ct^2$;

в) в гармонійному вигляді: $C(t)=C^0+C\sin\omega t$;

Якщо в наявності повна інформація про стан системи, а також виконуються всі наведені обмеження, розв'язок за неперервною моделлю позитивної системи має вигляд [12]

$$X(t) = e^{\tilde{\tilde{A}}t} X_0 + e^{\tilde{\tilde{A}}t} \int_0^t e^{-\tilde{\tilde{A}}\tau} d\tau \cdot \tilde{B} C^0$$

або

$$X(t) = e^{\tilde{A}t} (X_0 - X^*) + X^*,$$

де $X^* = (I - A)^{-1} C^0$ – рівноважний стан системи;

$C^0 = \left(C_0^1, C_0^2, \ldots, C_0^n\right)^T$, $C_0^i \geq 0$;

$\tau -$ параметр інтегрування, що має розмірність часу;

$e^{\tilde{A}t}, e^{-\tilde{A}\tau} -$ матричні експоненти виду $e^{\tilde{A}t} = I + \tilde{A}t + \frac{\tilde{A}^2 t^2}{2!} + \frac{\tilde{A}^3 t^3}{3!} + \ldots$ та

$e^{-\tilde{A}\tau} = I - \tilde{A}\tau + \frac{\tilde{A}^2 \tau^2}{2!} - \frac{\tilde{A}^3 \tau^3}{3!} + \ldots$

1.2.3 Статична математична модель детермінованої позитивної системи

В даному пункті роботи основна увага зосереджена на поняття та особливостях статичного моделювання об'єкта дослідження, яке застосовується для опису його стану у деякий фіксований момент часу. Результатом статичного моделювання виступає статична математична модель досліджуваної системи, що описує функціонування системи у деякий фіксований момент часу або у випадку, коли змінюванням показників у часі можна знехтувати. Така модель зазвичай виступає в якості ідеалізованого

випадку функціонування системи, але при цьому має важливе значення при визначенні максимально можливих здібностей досліджуваної системи, та, крім того, при поданні стійких рівноважних (стаціонарних) станів систем [8, 9].

Для позитивної системи в якості статичної математичної моделі виступає модель, векторно-матричне рівняння якої, згідно до роботи [11], отримується з представлених у попередніх пунктах векторно-матричних рівнянь (2), (4), що описують дискретну та неперервну моделі, за умови фіксації часу та при умові відсутності змінювань основних характеристик моделей. Так, статична модель позитивної системи описується рівнянням виду [11, 12]

$$(A-I)X+C^0=0,$$

де $C^0=(C_0^1,C_0^2,\ldots,C_0^n)^T-$ сталі коефіцієнти, що входять до складу вхідної вектор-функції C_t (для системи дискретного часу) або $C(t)$ (для системи неперервного часу), причому $C_0^i\geq 0$, $i=\overline{1,n}$.

Наведене векторно-матричне алгебраїчне рівняння є статичним, оскільки у ньому відсутніми є різниці між станами системи, динаміка змінювання стану з часом, похідні за часом. Таке рівняння описує функціонування різних підсистем позитивної системи в деякому фіксованому режимі. Розв'язком рівняння є невід'ємні значення змінних X_i^*, $i=\overline{1,n}$, які у векторно-матричній формі визначаються у вигляді

$$X^*=(I-A)^{-1}C^0$$

та показують значення характеристик позитивної системи, що відповідають певному відносно стабільному стану системи. Отже, статична модель позитивної системи демонструє статичний позитивний математичний опис

функціонування досліджуваної системи та характеризує конкретний стан об'єкта у заданий момент часу, та, крім того, виступає рівноважним станом системи, оскільки всі розв’язки, що отримуються при розв’язанні векторно-матричних рівнянь, що описують дискретну та неперервну математичні моделі позитивної системи, асимптотично наближуються до стаціонарного розв’язку X^* [12].

1.3 Основні властивості позитивної динамічної системи як об’єкту аналізу та керування

Розглянемо досліджувану динамічну систему з дискретним (або неперервним) часом як систему керування. Для такої системи вектор-функції X_{t+1}, X_t (для системи дискретного часу) та $X(t)$ (для системи неперервного часу) будемо вважати фазовими змінними системи, а вхідні вектор-функції C_t (для системи дискретного часу) та $C(t)$ (для системи неперервного часу) будемо вважати керуючими функціями (керуваннями).

Для складання дискретної та неперервної математичних моделей керованої позитивної динамічної системи повернемося до наведених у підрозділі 1.2 векторно-матричних рівнянь, що описують дискретну та неперервну математичні моделі динаміки руху позитивної динамічної системи відповідно дискретного та неперервного часу. Подані в (1), (2) (для системи дискретного часу) та в (3), (4) (для системи неперервного часу) векторно-матричні рівняння являють собою рівняння у змінних стану позитивної системи, розглядуваної в якості системи керування. Такі рівняння дозволяють отримати уявлення про внутрішні процеси, що відбуваються у досліджуваній системі керування. При цьому для такої динамічної системи може бути виміряно деякий вихід системи, який подається для системи дискретного часу у векторно-матричній формі у вигляді рівняння

$$Y_t = DX_t,$$

а для системи неперервного часу – у вигляді рівняння

$$Y(t) = DX(t),$$

де матриця $D = \{d_{ij}\}_{r\times n}$ – матриця виходу системи;

r – кількість векторів виходу системи, причому:

– кількість рядків r матриці D (кількість виходів $Y_t = (Y_t^1,\ldots,Y_t^r)^T$ (для дискретної моделі) або $Y(t) = (Y_1(t),\ldots,Y_r(t))^T$ (для неперервної моделі) системи) визначається умовою $r \geq 1$;

– кількість стовпців n матриці D має співпадати із кількістю n підсистем досліджуваної системи;

– всі елементи матриці $D = (d_{ij})_{r\times n}$ мають бути невід'ємними: $d_{ij} \geq 0$, причому хоча б один з цих елементів має бути строго додатним;

– всі елементи матриці $D = (d_{ij})_{r\times n}$ одночасно не можуть бути нульовими.

З урахуванням наданих міркувань, математична модель, що описує поведінку дискретної системи керування, у змінних стану системи подається у вигляді сукупності векторно-матричних різницевих рівнянь стану й виходу системи відповідно вигляду [12, 13]

$$\begin{cases} X_{t+1} = \tilde{A}X_t + \tilde{B}C_t, \\ Y_t = DX_t. \end{cases} \qquad (5)$$

Аналогічно, математична модель, що описує поведінку неперервної системи керування, у змінних стану системи подається у вигляді сукупності

векторно-матричних диференціальних рівнянь стану й виходу системи відповідно вигляду

$$\begin{cases} \dot{X}(t) = \tilde{\tilde{A}}X(t) + \tilde{B}C(t), \\ Y(t) = DX(t). \end{cases} \tag{6}$$

Для системи керування зазвичай ставиться задача аналізу для встановлення володіння нею певних властивостей, що відкривають можливості застосовувати до досліджуваної динамічної системи певні види керування при наявності повної інформації про стан системи або за умови її неповноти або відсутності.

Так, для досліджуваної позитивної динамічної системи основними такими властивостями є наступні:

– асимптотична стійкість розв’язків за математичними моделями системи – визначає здатність системи повертатися до рівноваги з часом та, з урахуванням певних обмежень, встановлюється виконуваною на всіх рівнях дослідження [10, 11];

– керованість системи за станом – визначає принципову можливість застосування до системи керування обраного типу керування за станом досліджуваної системи за умови повної інформації про стан та/або вихід системи; з урахуванням виконання всіх обмежень на відповідні матриці математичних моделей позитивної системи ця властивість, згідно до досліджень, проведених у роботі [13] є повністю виконуваною, а досліджувана система є повністю керованою за станом, а, отже, доступною для керування й регулювання.

Зазначені властивості є основними в теорії керування й регулювання, а їх виконання визначає можливості розв’язання тої чи іншої задачі керування й регулювання у досліджуваній системі.

1.4 Основні задачі керування, розв’язувані для керованої позитивної динамічної системи: огляд задач та обґрунтування вибору тої чи іншої задачі керування

В даному підрозділі наводиться огляд основних задач керування, що можуть ставитись для досліджуваної позитивної динамічної системи дискретного або неперервного часу. Головним при цьому є володіння досліджуваною системою відповідною властивістю системи керування, описаних у попередньому підрозділі 1.3 роботи. Крім того, для розв’язання задач даного підрозділу припустимо, що про всі стани досліджуваної системи керування в наявності є повна інформація.

В такому випадку можливими до розв’язання є наступні задачі керування [12]:

а) задача визначення програмних керувань позитивною системою, які виводять систему на бажану траєкторію руху;

б) задача технологічного керування, за розв’язанням якої є можливим проведення покращення (зменшення або збільшення) вихідних характеристик системи;

в) задача інноваційного керування, за розв’язанням якої є можливим здійснення прискорення збіжності вихідних характеристик системи до її рівноважного стану та проведення покращення їх значень;

г) задача модального керування, що передбачає перехід до замкнених математичних моделей системи із застосуванням зворотного зв’язку, за розв’язанням якої є можливим зведення непродуктивної матриці A до продуктивної та придатної до моделювання або одержання «покращених» значень вихідних характеристик системи при неможливості змінювання матриці A.

Розглянемо стисло кожну із зазначених задач керування стосовно досліджуваної в роботі детермінованої позитивної динамічної системи.

Задача програмного керування полягає у визначенні таких керуючих вектор-функцій C_t (для дискретної моделі) або $C(t)$ (для неперервної моделі), які виводять систему на задану бажану траєкторію руху $X_t^* = \varphi_t \geq 0$, $X_{t+1}^* = \varphi_{t+1} \geq 0$ (для дискретної моделі) або $X^*(t) = \varphi(t) \geq 0$ (для неперервної моделі). При цьому шуканий вектор керуючих функцій знаходиться наступним чином:

– для дискретної моделі у вигляді $C_t = (I - B)\Delta\varphi_t - (A - I)\varphi_t \geq 0$ або $C_t = (I - B)\varphi_{t+1} - (A - B)\varphi_t \geq 0$, де $\Delta\varphi_t = \varphi_{t+1} - \varphi_t$; $\varphi_{t+1} \geq 0$, $\varphi_t \geq 0$;

– для неперервної моделі у вигляді $C(t) = (I - B)\left(\dot{\varphi}(t) - \tilde{\tilde{A}}\varphi(t)\right) \geq 0$ або $C(t) = (I - B)\dot{\varphi}(t) - (A - I)\varphi(t) \geq 0$.

Знайдені таким чином функції керування задають закон руху досліджуваної системи у вигляді $X_t^* = \varphi_t$ (для дискретної моделі) або у вигляді $X^*(t) = \varphi(t)$ (для неперервної моделі).

Задача технологічного керування, за розв'язанням якої є можливим проведення покращення (зменшення або збільшення) вихідних характеристик системи, полягає в здійснюванні керуючого впливу через коефіцієнти вхідної матриці *A*. При цьому дана задача може бути розв'язаною тільки у випадку, коли вхідна невід'ємна матриця *A* володіє властивістю продуктивності та дозволяє себе змінювати. В такій ситуації для збільшення значень кожного з елементів вектора X_t до граничних значень $C_0^i \ \left(i = \overline{1,n}\right)$ вектора $C^0 = \left(C_0^1, C_0^2, \ldots, C_0^n\right)^T$ елементи заданої матриці A мають наближуватися до одиниці (якщо $C_0^i \neq 0$), для зменшення значень кожного з елементів вектора X_t до граничних значень $C_0^i \ \left(i = \overline{1,n}\right)$ елементи матриці A мають наближуватися до нуля. У ситуації, коли матриця *A* визначається непродуктивною або продуктивною, але її неможливо змінювати, дана задача не може бути розв'язаною.

Задача інноваційного керування, за розв'язанням якої є можливим здійснення прискорення збіжності вихідних характеристик системи до її рівноважного стану та проведення покращення їх значень, полягає в здійснюванні керуючого впливу через коефіцієнти вхідної матриці *B*. При цьому дана задача може бути розв'язаною за умови продуктивності матриць *A*, *B* системи та при $C_i^0 \neq 0 \ \ \left(i = \overline{1,n}\right)$. При розв'язанні зазначеної задачі потрібно виходити з наступного правила: щоб досягти збільшення (зменшення) значень кожного з елементів вектора X_t або $X(t)$ до значень елементів вектора рівноважного стану X^* системи, елементи матриці B мають наближатися до відповідних елементів матриці A (до нуля).

Задача модального керування, за розв'язанням якої є можливим зведення непродуктивної матриці *A* до продуктивної або одержання «покращених» значень вихідних характеристик системи при неможливості змінювання матриці *A*, полягає у побудові лінійного регулятора та переході до замкнених дискретної та неперервної моделей системи відповідно вигляду

$$X_{t+1} = (I - B)^{-1}(A - k - B)X_t + (I - B)^{-1}C_t^0,$$
$$\dot{X}(t) = (I - B)^{-1}(A - k - I)X(t) + (I - B)^{-1}C_0(t),$$

де $k = (k_{ij})_{n\times n}$ – матриця сталих коефіцієнтів k_{ij} $\left(i, j = \overline{1,n}\right)$ посилення зворотного зв'язку, елементи якої обираються з наступних нерівностей:

– для збільшення значень елементів вектора X_t або $X(t)$:

$$\sum_{j=1}^{n} a_{ij} - 1 < \sum_{j=1}^{n} k_{ij} < \sum_{j=1}^{n} a_{ij},$$

причому при продуктивній матриці A коефіцієнти k_{ij} можуть приймати довільні значення; при непродуктивній матриці A коефіцієнти k_{ij} мають обиратись додатними;

– для зменшення значень елементів вектора X_t або $X(t)$:

$$\sum_{j=1}^{n} k_{ij} > \sum_{j=1}^{n} a_{ij},$$

причому при продуктивній матриці A значення $k_{ij} > 0$; при непродуктивності матриці A значення $k_{ij} \geq 1$;

– для забезпечення рівності значень елементів вектора X_t або $X(t)$ значенням елементів вектора C^0:

$$\sum_{j=1}^{n} k_{ij} = \sum_{j=1}^{n} a_{ij}$$

З урахуванням визначених з наведених нерівностей значень коефіцієнтів k_{ij} посилення зворотного зв'язку отримується модифікована вхідна матриця виду $(A-k)$. З урахуванням цього, розв'язки за дискретною та неперервною моделями позитивної системи мають відповідно вигляд [12]

$$X_t = \tilde{A}^{t-1}\tilde{B}\left((A-k-B)X_0 + (I-A-k)X^*\right) + \tilde{A}^{t-1}\sum_{i=1}^{t-1}\tilde{A}^{-i}\tilde{B}C_i^0$$

або

$$X(t) = e^{\tilde{\tilde{A}}t}X_0 + e^{\tilde{\tilde{A}}t}\int_0^t e^{-\tilde{\tilde{A}}\tau}\tilde{B}C_0(\tau)d\tau,$$

де $\tilde{\tilde{A}} = \tilde{\tilde{B}}(A - k - B);$

$\tilde{\tilde{\tilde{A}}} = \tilde{\tilde{B}}(A - k - I);$

$\tilde{\tilde{B}} = (I - B)^{-1};$

$X^* = (I - (A - k))^{-1} C^0;$

$X_0 = X(t_0);$

$\tau -$ параметр інтегрування, що має розмірність часу.

2 АЛГОРИТМІЗАЦІЯ ПРОЦЕСУ РОЗВ'ЯЗАННЯ ЗАДАЧ АНАЛІЗУ Й КЕРУВАННЯ ДЛЯ СКЛАДНОЇ ДЕТЕРМІНОВАНОЇ ПОЗИТИВНОЇ ДИНАМІЧНОЇ СИСТЕМИ ТА АНАЛІЗУ ОТРИМУВАНИХ РЕЗУЛЬТАТІВ

В даному розділі наводиться опис алгоритму послідовних дій (методики) з розв'язання задач аналізу й керування для складної детермінованої позитивної динамічної системи та аналізу отримуваних результатів з встановленням основної вхідної інформації для проведення відповідних досліджень та визначенням вихідних даних за здійсненим дослідженням. Описана в даному розділі інформація виступає в якості основи для розробки автоматизованого програмного продукту для розв'язання зазначених задач.

2.1 Вхідні дані для проведення дослідження

При проведенні основних досліджень за детермінованою позитивною динамічною системою встановлюються наступні вхідні дані, завдання яких характеризує, за своєю сутністю, попередній (0-ий) етап алгоритму послідовних дій з розв'язання задач аналізу й керування для модельованого об'єкта дослідження та аналізу отримуваних результатів:

а) введення розмірності позитивної системи (завдання кількості n її підсистем), причому $n \geq 1$;

б) встановлення виду розглядуваного в моделях часу (дискретного або неперервного);

в) введення коефіцієнтів вхідних матриць моделей: $A = (a_{ij})_{n \times n}$, $B = (b_{ij})_{n \times n}$, при цьому $0 \leq a_{ij} < 1$, $0 \leq b_{ij} < 1$, $a_{ij} \geq b_{ij}$, $i, j = \overline{1,n}$;

г) введення коефіцієнтів матриці $D=(d_{ij})_{r\times n}$ виходу системи, де r – кількість векторів виходу системи, причому:

– кількість рядків r матриці D (кількість виходів $Y_t=(Y_t^1,\ldots,Y_t^r)^T$ (для дискретної моделі) або $Y(t)=\big(Y_1(t),\ldots,Y_r(t)\big)^T$ (для неперервної моделі) системи) визначається умовою $r\geq 1$;

– кількість стовпців n матриці D має співпадати із кількістю n підсистем досліджуваної системи (у випадку, якщо кількість стовпців матриці D не співпадає із кількістю n підсистем системи, видається помилка (оскільки не всі виходи системи є доступними до вимірювання) та здійснюється перехід до початку етапу 0, підпункту г), і далі здійснюється коригування введених коефіцієнтів матриці $D=(d_{ij})_{r\times n}$ виходу системи;

– всі елементи матриці $D=(d_{ij})_{r\times n}$ мають бути невід'ємними: $d_{ij}\geq 0$, причому хоча б один з цих елементів має бути строго додатним;

– всі елементи матриці $D=(d_{ij})_{r\times n}$ одночасно не можуть бути нульовими.

Якщо хоча б одна з умов не виконується, видається помилка та здійснюється перехід до початку етапу 0, підпункту г), і далі знов здійснюється введення коефіцієнтів матриці $D=(d_{ij})_{r\times n}$ виходу системи.

Також потрібно зауважити, яким чином у дослідженні задається відповідний дискретний або неперервний час. Так, при обчисленнях за дискретною моделлю час обираємо дискретним у проміжку від $t_0=0$ до t_k зі сталим однаковим кроком, що дорівнює одиниці: $(t+1)-t=1$. При обчисленнях за неперервною моделлю час обираємо нормованим: у проміжку від $t_0=0$ до $t_k=1$ зі сталим однаковим кроком; довжина кроку дискретизації s визначається шляхом поділу $t_k=1$ на кількість розглядуваних моментів k часу. Наприклад, якщо потрібно подивитись, як працює та чи інша модель на протязі $k=12$ років, то шаг дискретизації s

визначатиметься відношенням $s = \frac{t_k}{k} = \frac{1}{12} \approx 0{,}08(3)$, в результаті чого отримуємо наступні дискретизовані моменти часу на інтервалі $[0;1]$:

$$t_0 = 0;$$

$$t_1 = t_0 + s = 0 + \frac{1}{12} = \frac{1}{12} \approx 0{,}08(3);$$

$$t_2 = t_1 + s = \frac{1}{12} + \frac{1}{12} = \frac{2}{12} \approx 0{,}1(6);$$

$$t_3 = t_2 + s = \frac{2}{12} + \frac{1}{12} = \frac{3}{12} = 0{,}25;$$

$$\ldots\ldots\ldots$$

$$t_{11} = t_{10} + s = \frac{10}{12} + \frac{1}{12} = \frac{11}{12} \approx 0{,}91(6);$$

$$t_{12} = t_{11} + s = \frac{11}{12} + \frac{1}{12} = \frac{12}{12} = 1.$$

2.2 Основні етапи дослідження та розв'язання поставлених задач аналізу й керування

Алгоритм послідовних дій з розв'язання задач аналізу й керування для модельованого об'єкта дослідження та аналізу отримуваних результатів включає наступні етапи дослідження та розв'язання поставлених задач аналізу й керування:

0 етап: Введення вхідних даних моделей.

І етап: Аналіз компонентів дискретної (або неперервної) математичної моделі об'єкта дослідження. Даний етап включає:

а) дослідження на невід'ємність та продуктивність вхідних параметрів математичних моделей об'єкта:

– для дискретної моделі на невід'ємність та продуктивність досліджуються матриці A, B, $(A-B)$, $\tilde{A}=(I-B)^{-1}(A-B)$;

– для неперервної моделі на невід'ємність та продуктивність досліджуються матриці A та B; в матриці $\tilde{\tilde{A}}=(I-B)^{-1}(A-I)$ елементи, розташовані на головній діагоналі мають бути недодатними (тобто $\tilde{\tilde{a}}_{ii}\le 0$, $i=\overline{1,n}$), а недіагональні елементи – невід'ємними (тобто $\tilde{\tilde{a}}_{ij}\ge 0$ при $i\ne j$, $i,\ j=\overline{1,n}$);

Якщо матриця A є непродуктивною, то здійснюється перехід до етапу V, підпункту б) пункту 1);

б) перевірка повноти інформації про вхідні характеристики моделей, а саме про вектор-функцію початкового стану системи $X(0)=X_0=\left(X_0^1,X_0^2,\ldots,X_0^n\right)^T$ та вектор-функцію керування $C_t=\left(C_t^1,\ldots,C_t^n\right)^T$ (для дискретної моделі) та $C(t)=\left(C_1(t),C_2(t),\ldots,C_n(t)\right)^T$ (для неперервної моделі).

Якщо мається повна інформація про вектор початкового стану системи $X(0)=X_0=\left(X_0^1,X_0^2,\ldots,X_0^n\right)^T$, то проводиться введення початкових умов $X(0)=X_0=\left(X_0^1,X_0^2,\ldots,X_0^n\right)^T$ та здійснюється перехід до етапу I, підпункту в). Якщо ж про вектор початкового стану інформації немає (не можемо задати значення вектору $X(0)=X_0=\left(X_0^1,X_0^2,\ldots,X_0^n\right)^T$), то програма сповіщає про неможливість визначення вектору $X(0)=X_0=\left(X_0^1,X_0^2,\ldots,X_0^n\right)^T$ без розв'язання задачі спостереження та вимагає від дослідника визначення повної інформації про стан системи.

Якщо мається повна інформація про вектор керування $C_t=\left(C_t^1,\ldots,C_t^n\right)^T$ (для дискретної моделі) та $C(t)=\left(C_1(t),C_2(t),\ldots,C_n(t)\right)^T$ (для неперервної моделі), то виконується введення початкових умов для цих функцій:

– для дискретної моделі: $C^0 = (C_0^1, C_0^2, \ldots, C_0^n)^T$ та $C = (C^1, C^2, \ldots, C^n)^T$, причому $C << C^0$; $C_0^i \geq 0$, $C^i \geq 0$; $w = \left(w_{ij}\right)_{n \times n}$, $0 < w_{ij} < 1$;

– для неперервної моделі: $C^0 = (C_0^1, C_0^2, \ldots, C_0^n)^T$ та $C = (C^1, C^2, \ldots, C^n)^T$, причому $C << C^0$; $C_0^i \geq 0$, $C^i \geq 0$; $\omega \geq 0$.

Якщо ж інформація про вектор керування $C_t = \left(C_t^1, \ldots, C_t^n\right)^T$ (для дискретної моделі) та $C(t) = \left(C_1(t), C_2(t), \ldots, C_n(t)\right)^T$ (для неперервної моделі) відсутня, то ставиться питання про наявність інформації про бажаний вектор стану системи та здійснюється перехід до етапу III;

в) дослідження на невід'ємність вхідних характеристик моделей, а саме вектор-функцій початкового стану $X(0) = X_0 = \left(X_0^1, X_0^2, \ldots, X_0^n\right)^T$ та керування $C_t = \left(C_t^1, \ldots, C_t^n\right)^T$ (для дискретної моделі) та $C(t) = \left(C_1(t), C_2(t), \ldots, C_n(t)\right)^T$ (для неперервної моделі).

При цьому:

– якщо умови невід'ємності для всіх функцій виконуються, то здійснюється перехід до етапу II;

– якщо хоча б один з елементів векторів $X(0) = X_0 = \left(X_0^1, X_0^2, \ldots, X_0^n\right)^T$ є від'ємним, то видається інформація про помилку та здійснюється перехід до етапу I, підпункту б), і далі знов здійснюється перевірка введення даних;

– якщо хоча б один з елементів векторів $C_t = \left(C_t^1, \ldots, C_t^n\right)^T$ (для дискретної моделі) або $C(t) = \left(C_1(t), C_2(t), \ldots, C_n(t)\right)^T$ (для неперервної моделі) є від'ємним, то видається інформація про помилку та здійснюється перехід до етапу I, підпункту б), знов здійснюється перевірка введення даних;

II етап: Аналіз основних властивостей позитивної динамічної системи, розглядуваної в якості системи керування. Даний етап включає дослідження вконання властивості керованості за станом складної позитивної системи.

На даному кроці оцінка керованості проводиться на основі перевірки рангової умови

$$rank\, M_{кер} = n,$$

де $M_{кер} = \left[\tilde{B} \vdots (\tilde{A}\tilde{B}) \vdots (\tilde{A}^2\tilde{B}) \vdots \ldots \vdots (\tilde{A}^{n-1}\tilde{B}) \right]_{n \times nn}$ (для системи з дискретним часом) та $M_{кер} = \left[\tilde{B} \vdots (\tilde{\tilde{A}}\tilde{B}) \vdots (\tilde{\tilde{A}}^2\tilde{B}) \vdots \ldots \vdots (\tilde{\tilde{A}}^{n-1}\tilde{B}) \right]_{n \times nn}$ (для системи з неперервним часом) - матриці керованості за станом.

Якщо $rank\, M_{кер} = n$, то система є повністю керованою за станом, якщо $0 < rank\, M_{кер} < n$, то система є не повністю керованою за станом, якщо $rank\, M_{кер} = 0$, система є не керованою за станом, а, отже, не доступною до відповідного керування.

III етап: Програмне керування позитивної системи.

На даному етапі здійснюється перевірка завдання бажаного вектору станів об'єкта дослідження. Якщо такий вектор станів не заданий, то здійснюється перехід на етап IV. Якщо ж вектори бажаних станів $X_t^* = \varphi_t \geq 0$, $X_{t+1}^* = \varphi_{t+1} \geq 0$ (для дискретної моделі) або $X^*(t) = \varphi(t) \geq 0$ (для неперервної моделі) є відомими, то розв'язується задача визначення програмних керувань C_t (для дискретної моделі) або $C(t)$ (для неперервної моделі), які виводять систему на бажану траєкторію руху, після чого здійснюється перехід до VI етапу (виведення результатів дослідження).

При цьому шуканий вектор керувань знаходиться наступним чином:

– для дискретної моделі у вигляді $C_t = (I - B)\varphi_{t+1} - (A - B)\varphi_t \geq 0$, де $\Delta\varphi_t = \varphi_{t+1} - \varphi_t$; $\varphi_{t+1} \geq 0$, $\varphi_t \geq 0$;

– для неперервної моделі у вигляді $C(t) = (I - B)\dot{\varphi}(t) - (A - I)\varphi(t) \geq 0$.

IV етап: Визначення рівноважного стану системи X^* та вихідних характеристик X_t (для дискретної моделі) або $X(t)$ (для неперервної моделі) позитивної системи при наявних вхідних змінних C_t (або відповідно $C(t)$) і початкових умовах $X(0) = X_0$ системи за розімкненими дискретною та неперервною моделями позитивної системи.

Визначення стаціонарного розв'язку (рівноважного стану системи) X^* для дискретної та неперервної моделей проводиться за співвідношенням $X^* = (I - A)^{-1} C^0$, де $C^0 = \left(C_0^1, C_0^2, \ldots, C_0^n\right)^T$, $C_0^i \geq 0$.

Для визначення вихідних характеристик системи використовуються наступні співвідношення:

– для дискретної моделі:

$$X_t = \tilde{A}^{t-1}\tilde{B}\left((A-B)X_0 + (I-A)X^*\right) + \tilde{A}^{t-1}\sum_{i=1}^{t-1}\tilde{A}^{-i}\tilde{B}C_i,$$

– для неперервної моделі:

$$X(t) = e^{\tilde{A}t}\left(X_0 - X^*\right) + X^*,$$

де $\tilde{A} = \tilde{B}(A-B)$;

$\tilde{\tilde{A}} = \tilde{B}(A-I)$, $\tilde{B} = (I-B)^{-1}$;

$$e^{\tilde{A}t} = I + \tilde{A}t + \frac{\tilde{A}^2 t^2}{2!} + \frac{\tilde{A}^3 t^3}{3!} + \ldots$$

V етап: Дослідження результатів, одержаних на етапі IV. Якщо одержані вихідні характеристики системи не задовольняють дослідника (замовника), тоді з множини можливих керувань обирається такий вплив на систему, який забезпечує досягнення поставленої мети.

Даний етап включає розв'язання наступних задач керування, описаних у підрозділі 1.4 розділу 1 роботи:

а) задачу технологічного керування;

б) задачу інноваційного керування;

в) задачу модального керування;

VI етап: Отримання вихідних даних за проведеним дослідженням.

2.3 Вихідні дані за здійснюваним дослідженням

В якості вихідної інформації виступає отримання в табличній формі та графічне представлення одержаних результатів для стаціонарного розв'язку $X^* = \left(X_1^*, X_1^*, \ldots, X_n^*\right)^T$, вектору $X_t = \left(X_t^1, X_t^2, \ldots, X_t^n\right)^T$ (для дискретної моделі) або $X(t) = \left(X_1(t), X_2(t), \ldots, X_n(t)\right)^T$ (для неперервної моделі).

Таблиця результатів при цьому подається у вигляді табл. 2.1 (для дискретної моделі) та табл. 2.2 (для неперервної моделі).

Таблиця 2.1 – Вихідні дані за дискретною моделлю системи

Час t		Стаціонарні розв'язки				Вихідні характеристики системи			
		X_1^*	X_2^*	…	X_n^*	X_t^1	X_t^2	…	X_t^n
t_0	0	#значення	#значення	…	#значення	#значення	#значення	…	#значення
t_1	$t_0 + s$	#значення	#значення	…	#значення	#значення	#значення	…	#значення
…	…	…	…	…	…	…	…	…	…
t_k	k	#значення	#значення	…	#значення	#значення	#значення	…	#значення

Таблиця 2.2 – Вихідні дані за неперервною моделлю системи

Час t		Стаціонарні розв'язки				Вихідні характеристики системи			
		X_1^*	X_2^*	…	X_n^*	$X_1(t)$	$X_2(t)$	…	$X_n(t)$
t_0	0	#значення	#значення	…	#значення	#значення	#значення	…	#значення
t_1	$t_0 + s$	#значення	#значення	…	#значення	#значення	#значення	…	#значення
…	…	…	…	…	…	…	…	…	…
t_k	k	#значення	#значення	…	#значення	#значення	#значення	…	#значення

Графічне зображення результатів дослідження здійснюємо шляхом одночасного зображення на одному графіці всіх стаціонарних розв'язків $X^* = \left(X_1^*, X_1^*, \ldots, X_n^*\right)^T$ та вихідних характеристик $X_t = \left(X_t^1, X_t^2, \ldots, X_t^n\right)^T$ (для дискретної моделі) або $X(t) = \left(X_1(t), X_2(t), \ldots, X_n(t)\right)^T$ (для неперервної моделі) для всіх видів вектор-функції керування. При цьому, для дискретної моделі графічні зображення мають подаватися у точковому вигляді (точки відповідного графіка функції не з'єднуються між собою ламаними), для неперервної моделі графічні зображення мають подаватися у вигляді неперервних кривих.

Представлений в даному розділі алгоритм послідовних дій з розв'язання задач аналізу й керування для складної детермінованої позитивної динамічної системи та аналізу отримуваних результатів виступає в якості основи для розробки автоматизованого програмного продукту для розв'язання зазначених задач.

3 АВТОМАТИЗАЦІЯ ПРОЦЕСУ РОЗВ'ЯЗАННЯ ЗАДАЧ АНАЛІЗУ Й КЕРУВАННЯ ЗА СКЛАДНОЮ ДЕТЕРМІНОВАНОЮ ПОЗИТИВНОЮ ДИНАМІЧНОЮ СИСТЕМОЮ

В даному розділі кваліфікаційної роботи наводиться характеристика програмного забезпечення, розробленого на основі представленого в попередньому розділі роботи алгоритму послідовних дій з розв'язання задач аналізу й керування для модельованого об'єкта дослідження, аналізу отримуваних результатів та здатного автоматизувати процеси розв'язання задач аналізу й керування за складною детермінованою позитивною динамічною системою, а також візуалізувати отримувану на виході результативну інформацію.

3.1 Вибір технологій. Загальна характеристика розробленого програмного продукту

На основі проведених досліджень розглядаються два типи динамічних систем: дискретні та неперервні. Розроблений в попередньому розділі роботи алгоритм спрямований на визначення та вивчення вихідних характеристик складної керованої додатної динамічної системи в обох випадках. Результатом цієї роботи є програмний продукт, реалізований мовою програмування C++ із використанням бібліотеки Qt. Цей продукт автоматизує процес отримання рішень для задач визначення та дослідження вихідних характеристик як для дискретних, так і для неперервних позитивних динамічних систем.

Вибір мови програмування C++ та версії бібліотеки Qt 4.7.3 для розробки програми зумовлено кількома чинниками.

а) підтримка C++11. Стандарт C++11 вводить багато функцій та поліпшень, які полегшують розробку, роблять код більш зрозумілим та безпечним. Використання можливостей C++11 може значним чином спростити та покращити розробку програми, роблячи код сучаснішим та ефективнішим;

б) бібліотека Qt. Qt – потужна бібліотека для розробки графічних інтерфейсів та кросплатформених застосунків. Версія 4.7.3, хоча і не є останньою, але поряд з цим є стабільною і може бути обрана через надійність та широкі можливості. Якщо є конкретні функції чи можливості в Qt 4.7.3, які є необхідними для проекту, і немає потреби чи можливості переходити на новіші версії, це може бути вирішальним чинником при виборі конкретної версії бібліотеки.

Також при проведенні аналізу існуючих мов програмування для розробки програмного забезпечення для роботи із досліджуваною складною летерінованою позитивною динамічною системою розглядалися альтернативи, такі як Python та PyQt 4 або 5 версій, але рішення нахилилося на користь розробки мовою, на якій був написаний сам фреймворк. Хоча Python та PyQt були прийнятними варіантами, також, на мові Python, реалізована потужна бібліотека для роботи з матрицями numpy, але, вважалося більш вигідним використовувати мову, на якій спочатку був реалізований основний фреймворк. Це рішення забезпечує кращу інтеграцію з основними функціями фреймворку, можливо, використовуючи наявні функції та оптимізації, специфічні для обраної мови програмування, у цьому випадку – C++.

Отже, вибір C++11 та Qt 4.7.3 для написання програми є практично обґрунтованим та найкращим в даному випадку вибором для реалізації програмного продукту.

3.2 Інтерфейс користувача. Приклади з проведення обчислень

Розглянемо інтерфейс користувача у розробленому програмному продукті відносно представлених в розробленому алгоритмі основних етапів дослідження та розв’язання поставлених задач аналізу й керування:

0 етап: Введення вхідних даних моделей.

а) введення розмірності позитивної системи, причому $n \geq 1$ (рис. 3.1);

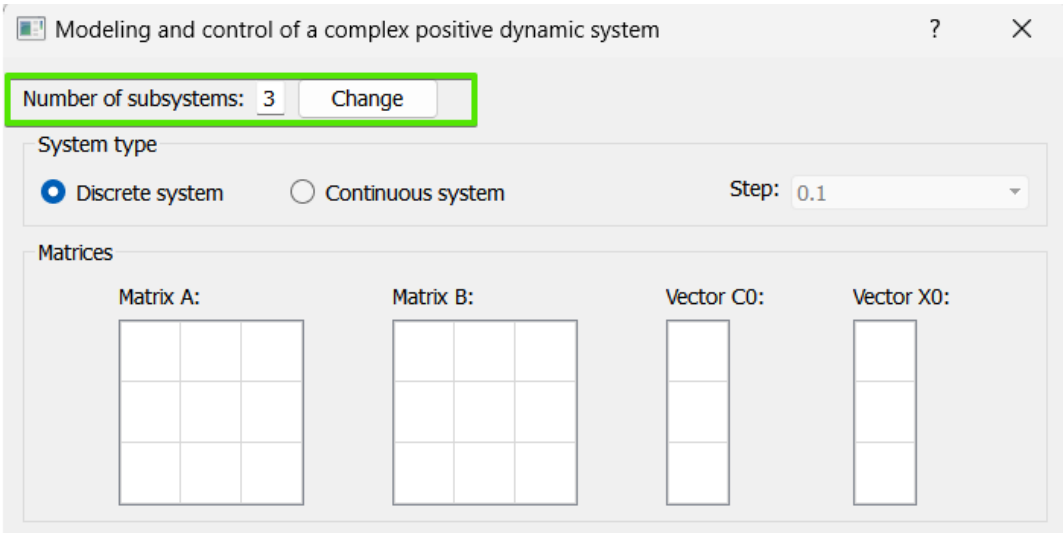


Рисунок 3.1 – Вікно введення даних програми із зазначенням розмірності досліджуваної системи

Для конфігурування кількості підсистем (розмірність системи), змініть кількість підсистем (розмірність системи) із значення 3 (значення за замовчування) на інше, та натисніть кнопку *Change*.

б) встановлення виду розглядуваного в моделях часу (дискретного або неперервного) (рис. 3.2);

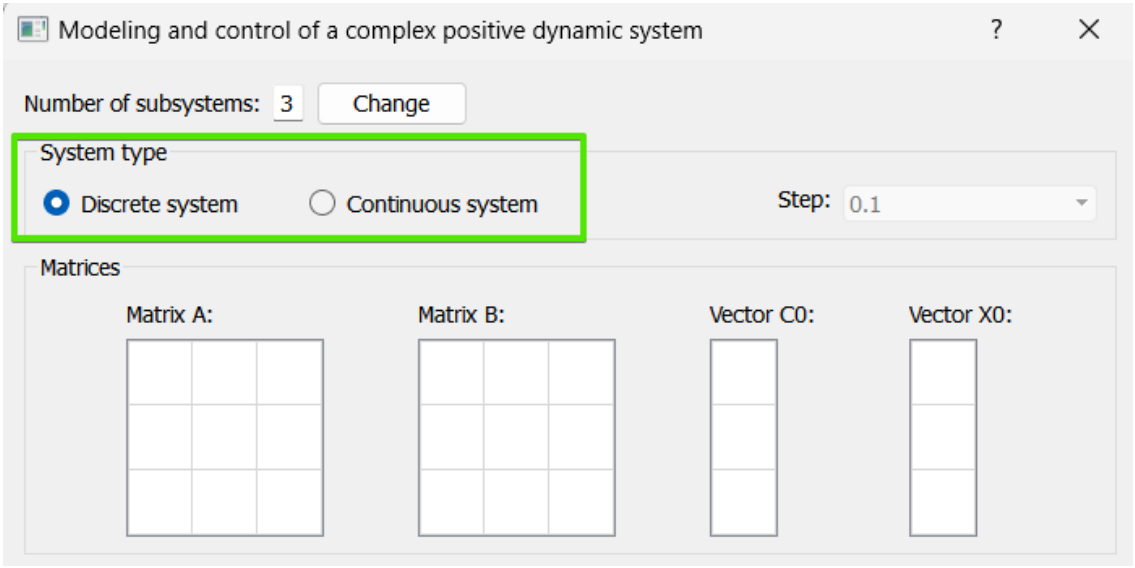


Рисунок 3.2 – Вікно введення даних програми із зазначенням виду розглядуваного в моделях часу

в) введення коефіцієнтів вхідних матриць моделей: $A=(a_{ij})_{n\times n}$, $B=(b_{ij})_{n\times n}$, при цьому $0\le a_{ij}<1$, $0\le b_{ij}<1$, $a_{ij}\ge b_{ij}$, $i,j=\overline{1,n}$ (рис. 3.3);

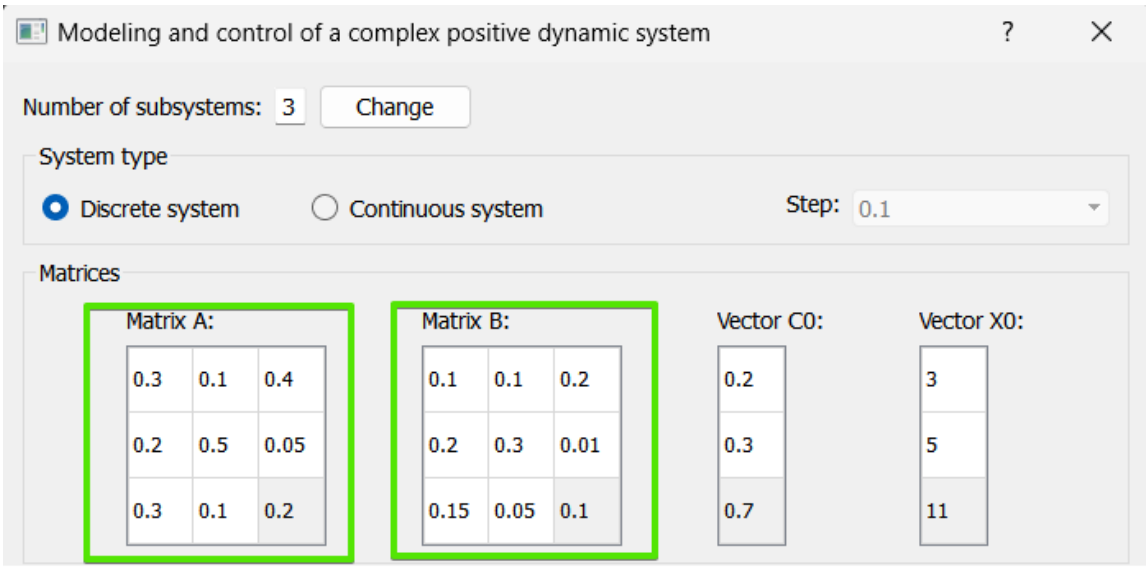


Рисунок 3.3 – Вікно введення даних програми із зазначенням коефіцієнтів вхідних матриць моделей

І етап: Аналіз компонентів дискретної (або неперервної) математичної моделі об'єкта дослідження:

а) дослідження на невід'ємність та продуктивність вхідних параметрів дискретної та неперервної математичних моделей об'єкта, а саме матриць коефіцієнтів моделей.

Якщо, на даному етапі виконуються всі умови невід'ємності та продуктивності для відповідних матриць моделей, програма дозволяє переходити до введення наступних вхідних даних. Якщо, на даному етапі не буде виконуватися будь-яка з окреслених умов, програма видає повідомлення про помилку (рис. 3.4).

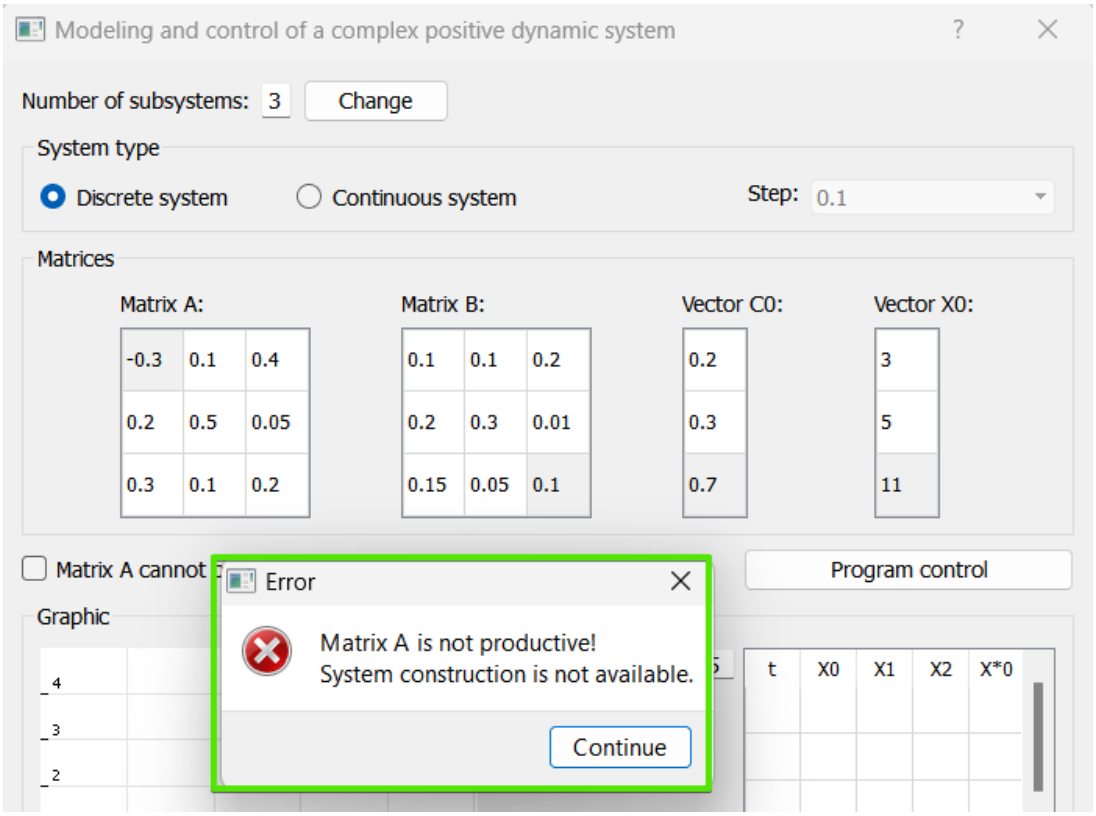


Рисунок 3.4 – Повідомлення про помилку при умові введення некоректних вхідних даних

б) перевірка повноти інформації про вхідні характеристики моделей, а саме про вектор-функцію початкового стану системи $X(0) = X_0 = \left(X_0^1, X_0^2, \ldots, X_0^n\right)^T$ та вектор-функцію керування $C_t = \left(C_t^1, \ldots, C_t^n\right)^T$

(для дискретної моделі) та $C(t)=(C_1(t),C_2(t),\ldots,C_n(t))^T$ (для неперервної моделі). Якщо мається повна інформація про вектор початкового стану системи $X(0)=X_0=\left(X_0^1,X_0^2,\ldots,X_0^n\right)^T$, то проводиться введення цих початкових умов (рис. 3.5) та здійснюється перехід до перевірки їх невід'ємності. Якщо ж про вектор початкового стану інформації немає, то в такому випадку програма видасть повідомлення про помилку при введенні початкових умов системи.

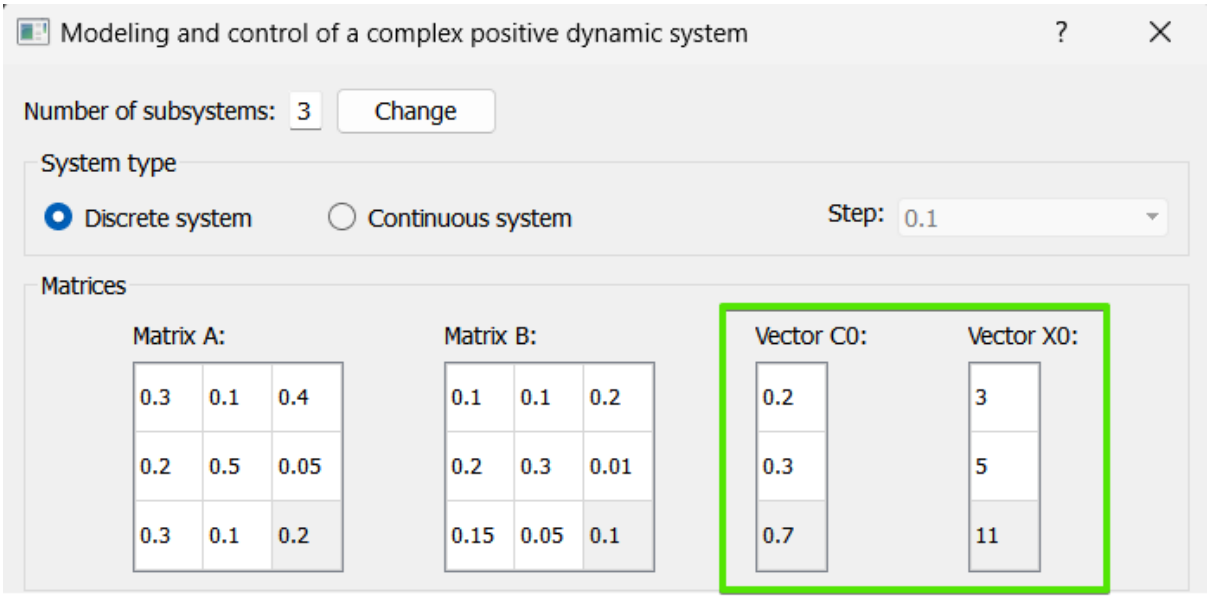


Рисунок 3.5 – Вікно введення даних програми із зазначенням коефіцієнтів вектору початкових умов $X(0)=X_0=\left(X_0^1,X_0^2,\ldots,X_0^n\right)^T$ та коефіцієнтів $C^0=(C_0^1,C_0^2,\ldots,C_0^n)^T$ вхідної вектор-функції керування

Якщо мається повна інформація про вектор керування $C_t=\left(C_t^1,\ldots,C_t^n\right)^T$ (для дискретної моделі) та $C(t)=(C_1(t),C_2(t),\ldots,C_n(t))^T$ (для неперервної моделі), то виконується введення початкових умов для цих функцій (рис. 3.5). В роботі функції керування розглядаються в визначених у розділі 2 виглядах (в залежності від обраної моделі).

в) дослідження на невід'ємність вхідних характеристик моделей, а саме вектор-функцій початкового стану $X(0) = X_0 = \left(X_0^1, X_0^2, \ldots, X_0^n\right)^T$ та керування $C_t = \left(C_t^1, \ldots, C_t^n\right)^T$ (для дискретної моделі) та $C(t) = \left(C_1(t), C_2(t), \ldots, C_n(t)\right)^T$ (для неперервної моделі). У випадку, якщо умови невід'ємності виконуються, програма надає можливість переходити до аналізу основних властивостей позитивної динамічної системи. В противному випадку програма видасть повідомлення про помилку та поверне користувача до введення відповідних даних.

II етап: Аналіз основних властивостей позитивної динамічної системи, розглядуваної в якості системи керування. Даний етап включає:

а) дослідження асимптотичної стійкості розв'язків за дискретною та неперервною математичними моделями об'єкта дослідження;

б) дослідження властивості керованості за станом позитивної системи.

Всі дослідження даного етапу в програмі реалізуються за допомогою побудови відповідних матриць керованості та перевірки відповідної рангової умови, визначеної в 1 розділі роботи.

Дані перевірки виконуються в програмному коді, представленому у Додатку А, L1006:L1309, або за посиланням

https://github.com/stas-polos/dynamic-system/blob/master/src/system/dynamic_system.cpp#L1006:L1309.

III етап: Програмне керування позитивної системи.

На даному етапі здійснюється перевірка завдання бажаного вектору станів об'єкта дослідження. Якщо такий вектор станів не заданий, то здійснюється перехід на етап IV. Якщо ж вектори бажаних станів $X_t^* = \varphi_t \geq 0$, $X_{t+1}^* = \varphi_{t+1} \geq 0$ (для дискретної моделі) або $X^*(t) = \varphi(t) \geq 0$ (для неперервної моделі) є відомими, то відкривається нове діалогове вікно, представлене на рис. 3.6 та розв'язується задача визначення програмних

керувань C_t (для дискретної моделі) або $C(t)$ (для неперервної моделі), які виводять систему на бажану траєкторію руху, після чого здійснюється перехід до VI етапу.

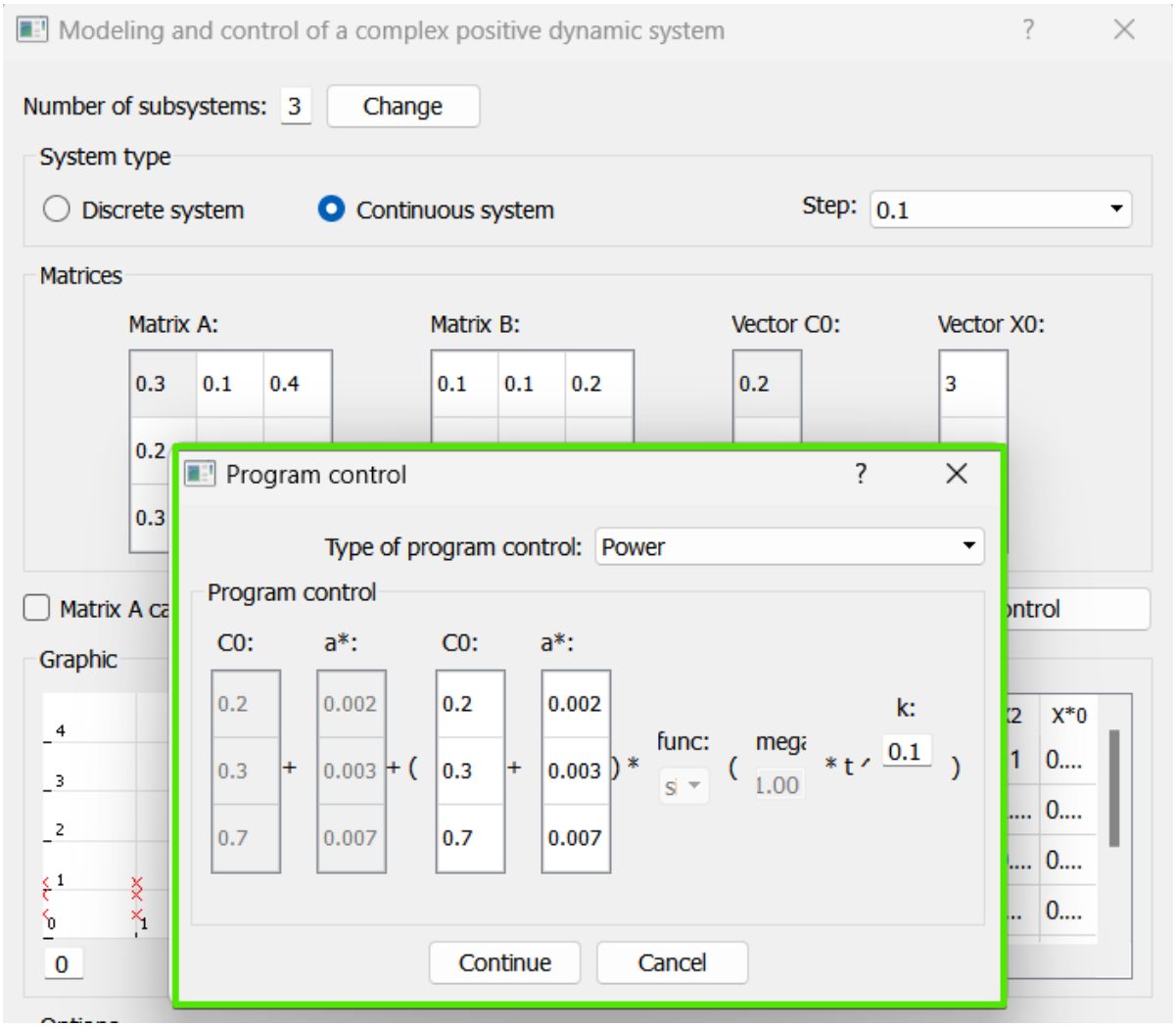


Рисунок 3.6 – Вікно здійснювання програмного керування системою

IV eman: Визначення рівноважного стану системи $X^* = (I - A)^{-1} C^0$ та вихідних характеристик X_t (для дискретної моделі) або $X(t)$ (для неперервної моделі) позитивної системи при наявних вхідних змінних C_t (або відповідно $C(t)$) і початкових умовах $X(0) = X_0$ системи за розімкненими дискретною та неперервною моделями позитивної системи.

Дані розрахунки виконуються в наступному коді:

```
void DynamicSystem::getX_st(int t, int index) {
  matrix newM(systemSize, 1);
  matrix E;
  E.setSingle(systemSize);
  matrix tmp1(A);
  matrix tmp2(C0);
  tmp1 = E - tmp1;
  tmp1.getInverse();
  newM = tmp1*tmp2;
  for(int i=0; i<systemSize; i++) Xst.setElement(i, 0, tmp1.getElement(i, 0));
}
```

V етап: Дослідження результатів, одержаних на етапі IV із використання різних видів керування. Даний етап проводиться у випадку, коли одержані вихідні характеристики системи не задовольняють дослідника, тоді з множини можливих керувань обирається такий вплив на систему, який забезпечує досягнення поставленої мети. Даний етап включає проведення технологічного, інноваційного й модального керувань за методикою, представленою у попередньому розділі роботи.

VI етап: Одержання в табличній формі та графічне подання отриманих результатів для стаціонарного розв'язку $X^* = \left(X_1^*, X_1^*, \ldots, X_n^*\right)^T$, вектору $X_t = \left(X_t^1, X_t^2, \ldots, X_t^n\right)^T$ (для дискретної моделі) або $X(t) = \left(X_1(t), X_2(t), \ldots, X_n(t)\right)^T$ (для неперервної моделі).

Результат виконання програми, виводиться як в табличному (рис. 3.7), так і в графічному (рис. 3.8) представленнях.

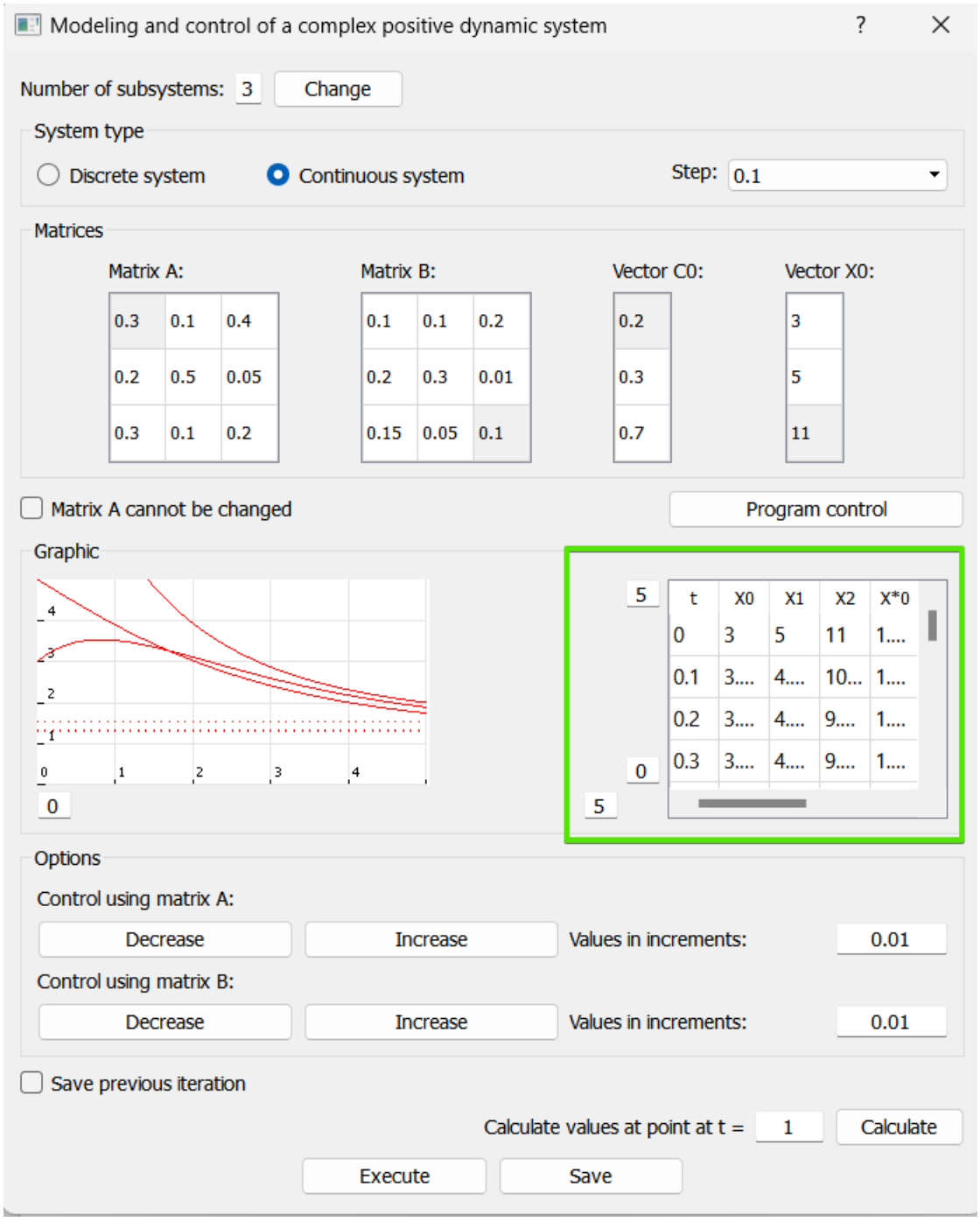


Рисунок 3.7 – Вікно роботи програми із виділенням табличних значень змінюваних у часі всіх станів досліджуваної системи

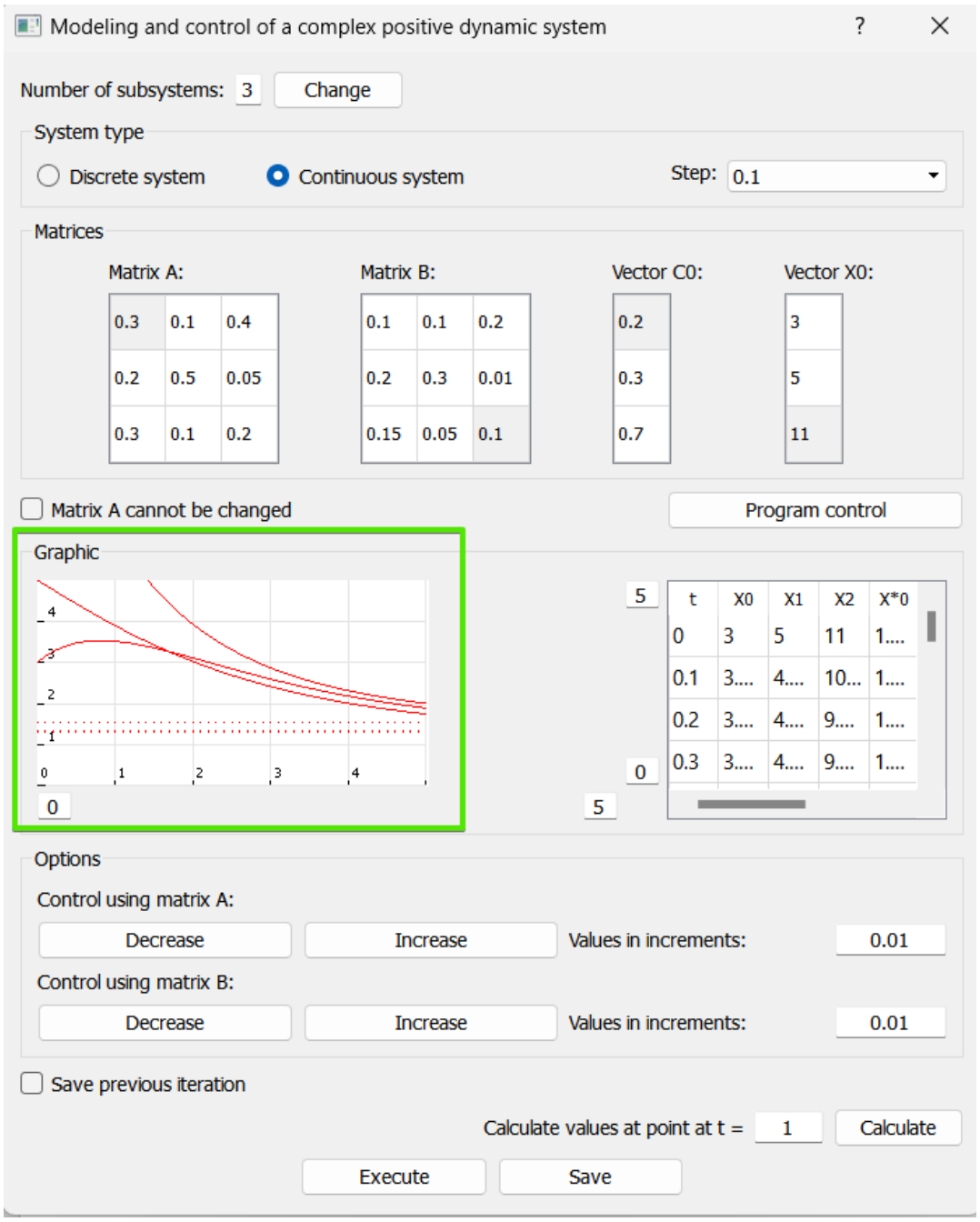


Рисунок 3.8 – Вікно роботи програми із виділенням графічного зображення змінюваних у часі всіх станів досліджуваної системи

При цьому на одному графіці зображуються всі стаціонарні розв’язки X^* та вихідні характеристики X_t (для дискретної моделі) або $X(t)$ (для неперервної моделі) для всіх видів вектор-функції керування. При цьому, для дискретної моделі графічні зображення подаються у точковому вигляді (точки відповідного графіка функції не з’єднуються між собою ламаними), для неперервної моделі – у вигляді неперервних кривих.

Також було реалізовано допоміжний функціонал для здійснення керування у системі через вхідні матриці моделей системи з метою затребуваного користувачем зменшення або збільшення всіх елементів матриці на фіксований крок (рис. 3.9).

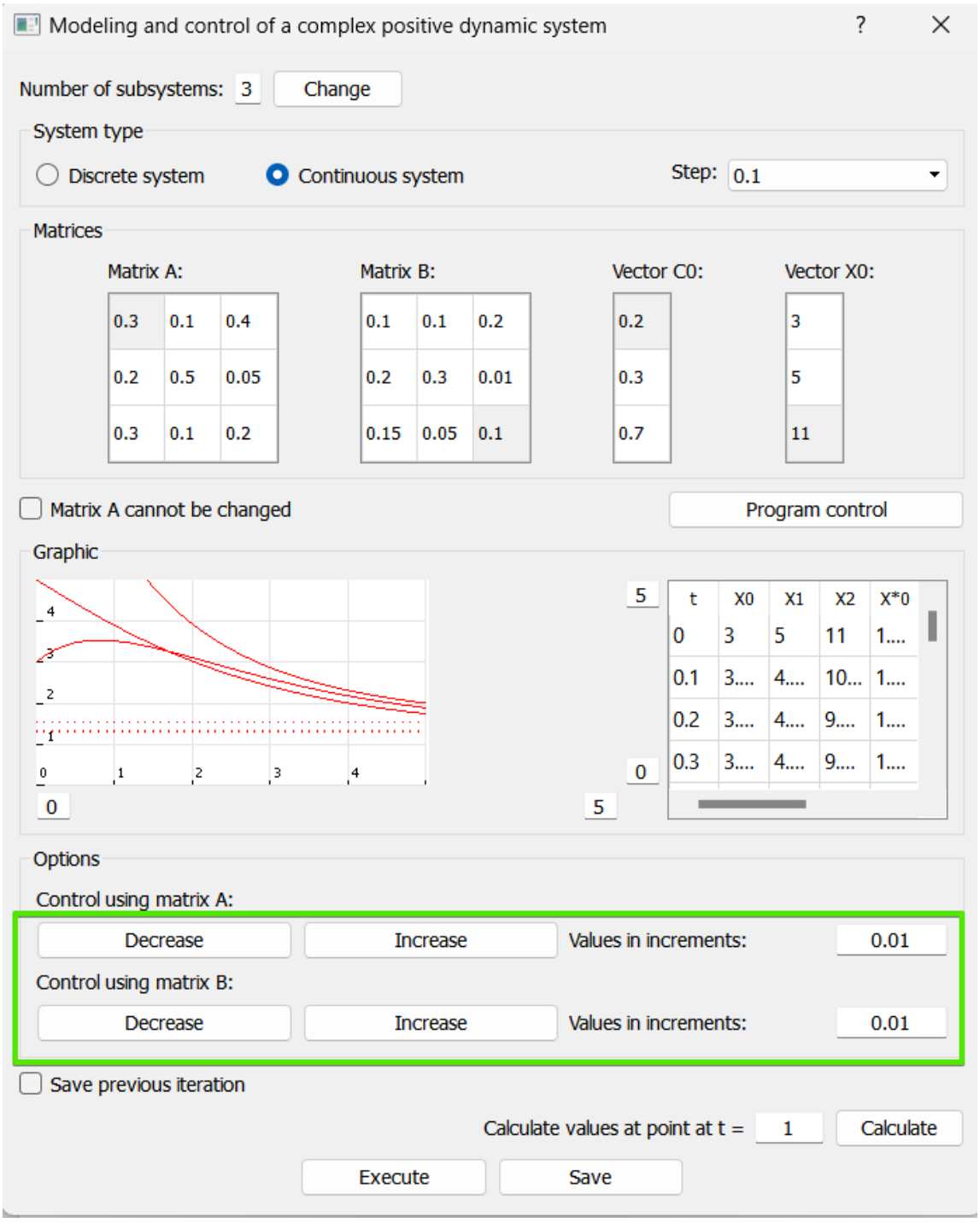


Рисунок 3.9 – Вікно роботи програми при застосуванні керуючих впливів на систему шляхом змінювання елементів вхідних матриць моделей системи

При цьому основні результати в програмі, окрім виведення на екран в повному обсязі, можливо виводити і для конкретно заданого моменту часу. Так, розрахунок значень за стаціонарним розв'язком моделі X^* та за вихідними станами системи X_t або $X(t)$ при заданому значенні t представлений на рис. 3.10.

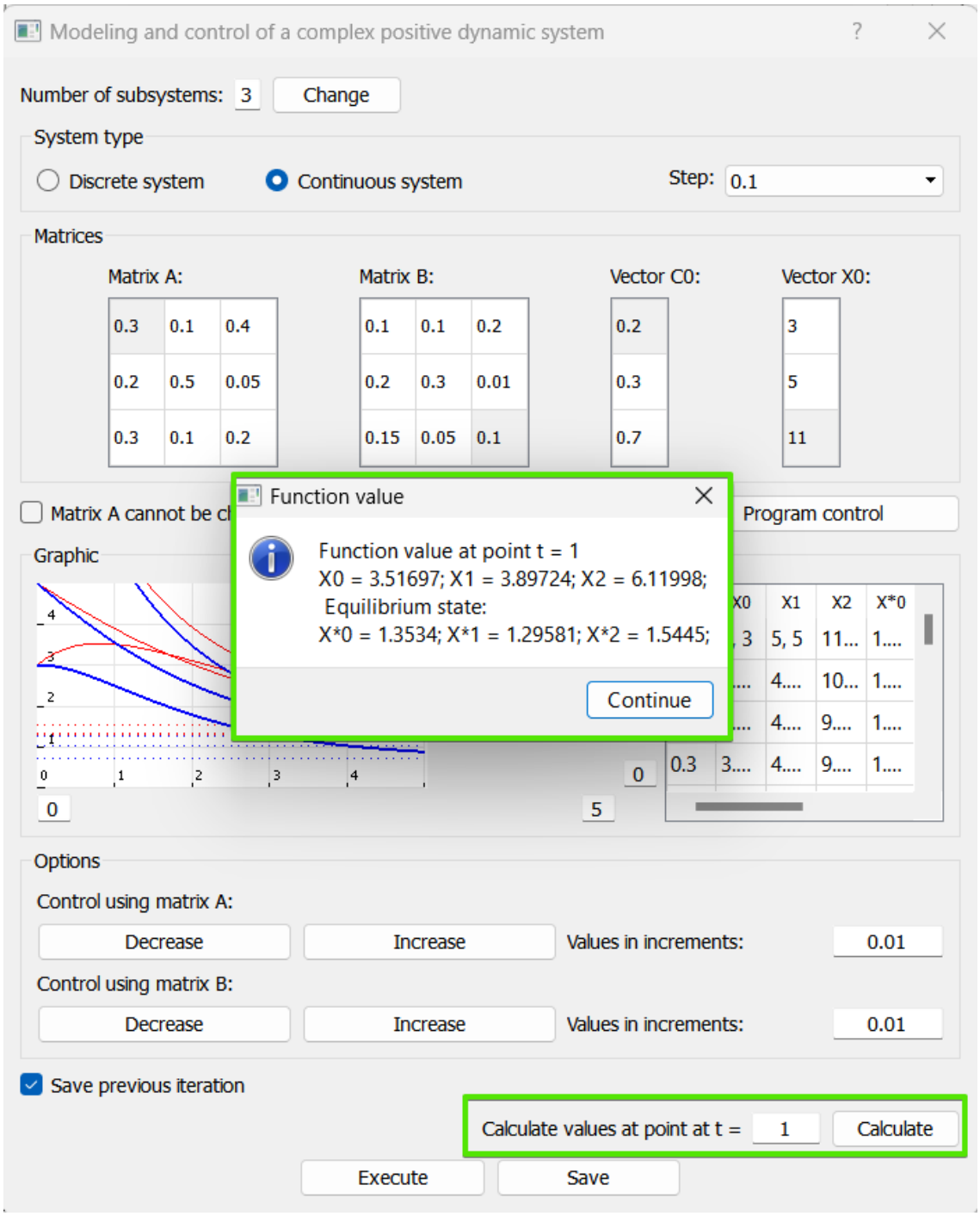


Рисунок 3.10 – Вікно роботи програми при визначенні основних характеристик системи для конкретно заданого моменту часу

Крім того, за результатами проведеного дослідження є можливим ще й збереження графічного зображення результатів моделювання. Для цього потрібно натиснути розташовану внизу вікна роботи програми кнопку «Save» (рис. 3.11, 3.12).

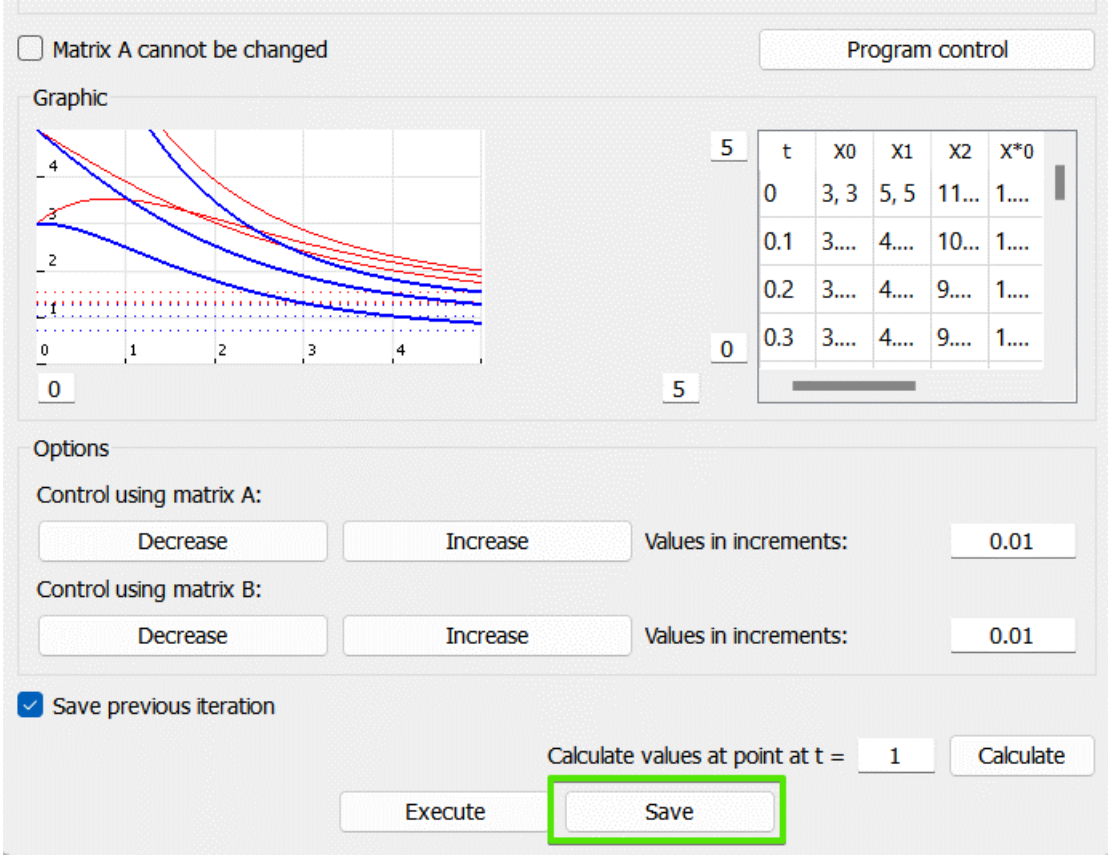


Рисунок 3.11 – Вікно роботи програми при необхідності збереження графічного зображення основних характеристик системи

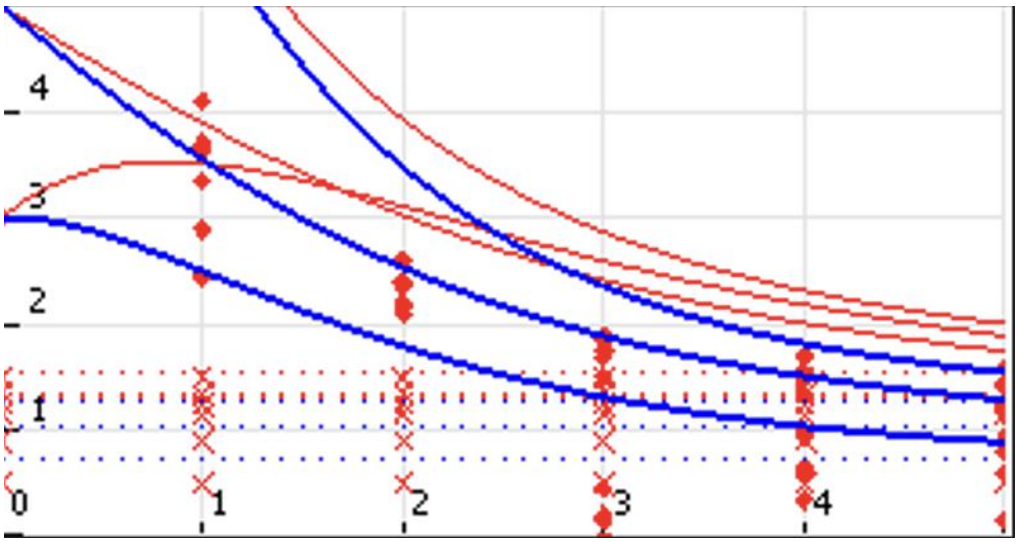


Рисунок 3.12 – Скріншот збережуваного графічного зображення результатів моделювання досліджуваної системи

Крім того, потрібно зауважити, що в процесі проведення дослідження за використанням розробленої програми, можливим є збереження проміжних результатів до текстових файлів (з розширенням txt) із відповідною назвою файлу, яке збігається з позначенням досліджуваної матриці (рис. 3.13-3.16).

Для перевірки проміжних результатів результати операцій над матрицями зберігається в файл наступним кодом:

```
void matrix::saveToFile(const std::string &filename) {
    std::fstream file;
    file.open(filename, std::fstream::out);

    if (file.is_open()) {
        file << sizeRow << " " << sizeCol << "\n"; // Write matrix dimensions

        for (int i = 0; i < sizeRow; i++) {
            for (int j = 0; j < sizeCol; j++) {
                file << array[i][j] << " ";
            }
```

```
            file << "\n"; // Move to the next row
        }

        file.close();
        std::cout << "Matrix saved to file: " << filename << std::endl;
    } else {
        std::cerr << "Unable to open file: " << filename << std::endl;
    }
}
```

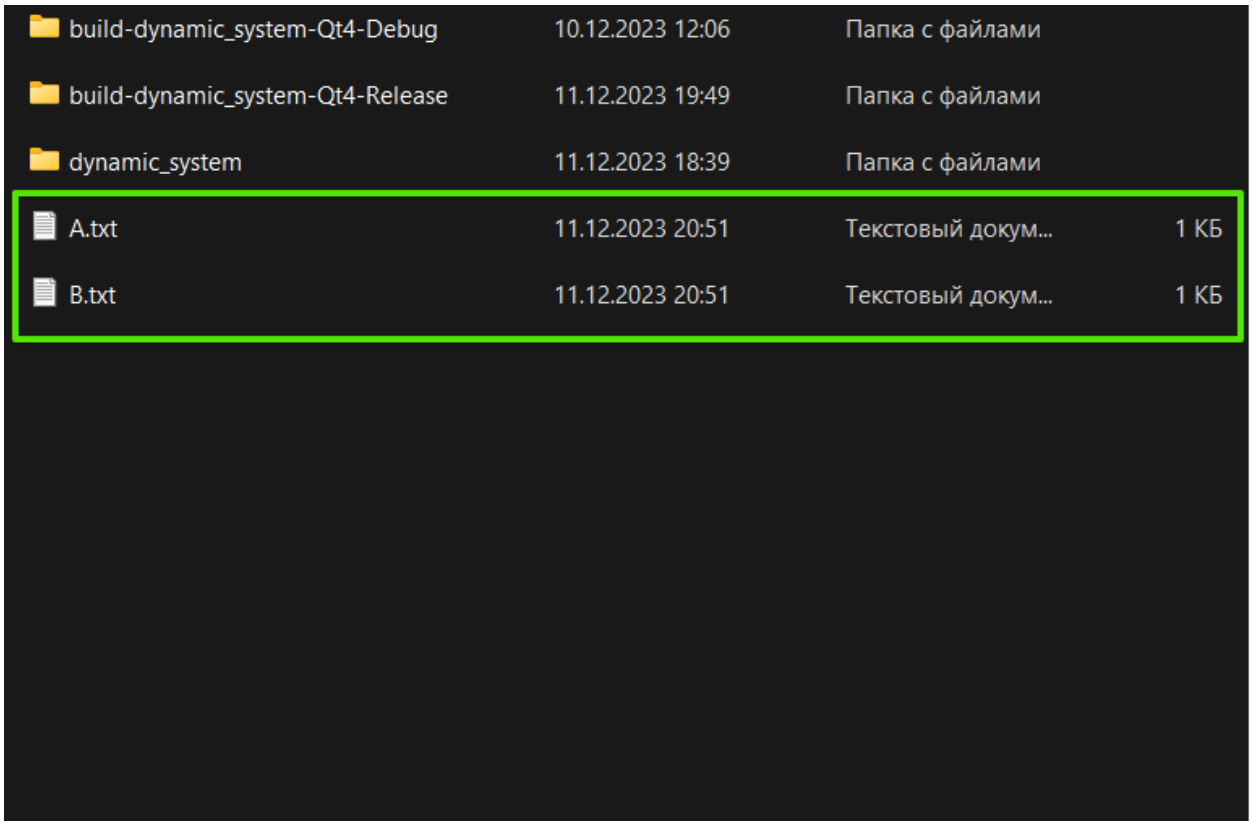


Рисунок 3.13 – Приклад збережених файлів при невідповідності умови продуктивності матриці *A*

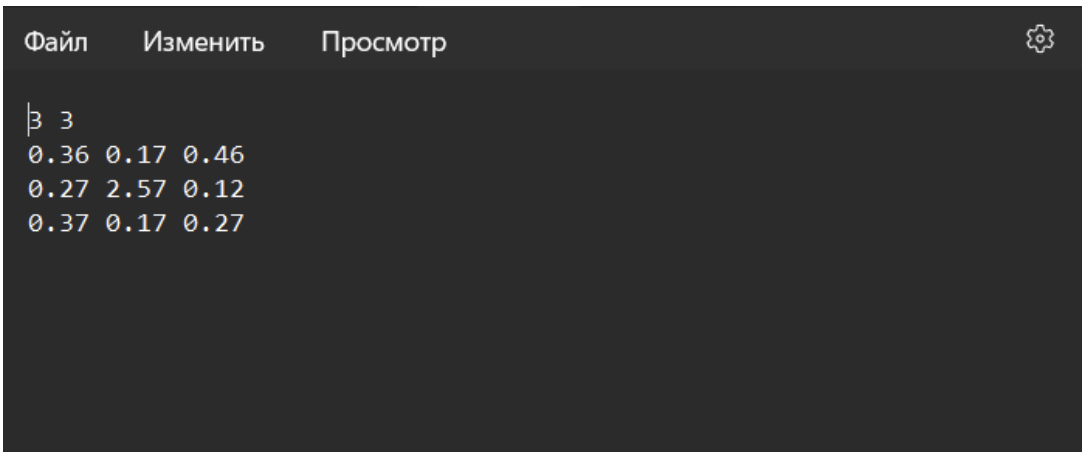


Рисунок 3.14 – Приклад вмісту збереженого файлу для матриці *A*

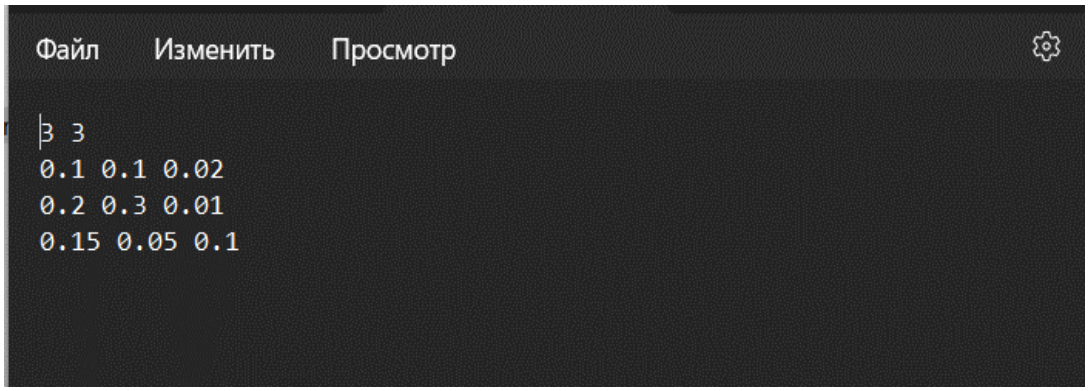


Рисунок 3.15 – Приклад вмісту збереженого файлу для матриці *B*

```
    E.setSingle(systemSize);
    tmpAB = tmpA - tmpB;
    tmpAB.saveToFile("C:\\programs\\A-B.txt");
    if (tmpAB.isProductive() == true) {

      tmp1 = E - B;
      tmp1.getInverse();
      tmp2 = tmpAB;
      tmp2 = tmp1 * tmpAB;
      tmp2.saveToFile("C:\\programs\\(E-B)^(-1)*(A-B).txt");
      if (!(tmp2.isProductive() == true)) {
          QMessageBox::critical(this, "Error", "The matrix (E-B)^(-1)*(A-B) is not productive!\nSystem construction is not
      }

    } else {
        QMessageBox::critical(this, "Error", "The A-B matrix is not productive!\nSystem building is not available.", "Cont
    }
  } else {
      QMessageBox::critical(this, "Error", "Matrix B is not productive!\nSystem construction is not available.", "Continue
  }
} else {
    QMessageBox::critical(this, "Error", "Matrix A is not productive!\nSystem construction is not available.", "Continue");
```

Рисунок 3.16 – Приклад перевірки продуктивності матриці

$$\tilde{A} = (I - B)^{-1}(A - B)$$

3.3 Структура організації програмного коду

Для контролю версій коду використовувався Git [25] та GitHub [16] для хостингу. Таким чином, репозиторій із кодом має наступну структуру, подану на рис. 3.17.

```
.
└── dynamic-system/
    ├── src/
    │   ├── matrix/                    1
    │   │   ├── matrix.h
    │   │   └── matrix.cpp
    │   ├── system/                    2
    │   │   ├── dynamic_system.cpp
    │   │   └── dynamic_system.h
    │   ├── widget/                    3
    │   │   ├── gwidget.cpp
    │   │   └── gwidget.h
    │   └── main.cpp                   4
    ├── images/
    │   ├── addition-functional.png
    │   ├── continuos-system.png
    │   ├── discrete-system.png
    │   ├── image_ui_1.png
    │   ├── programs-control.png
    │   └── select-type-systems.png
    ├── .gitignore
    ├── README.md
    ├── dynamic_system.pro
    └── dynamic_system.pro.user
```

Рисунок 3.17 – Деревовидна структура проекту

При цьому на рис. 3.17 введено наступні позначення:

а) модуль з реалізацією функціонала роботи із матрицями (додавання, віднімання, множення і т.д.);

б) модуль з реалізацією функціонала роботи з динамічною системою, дискретною та неперервною;

в) модуль з реалізацією функціонала побудови графіка розв’язку дискретної та неперервної систем;

г) директорія для зберігання зображень, які використовуються в README.md (Додаток Б);

д) файл main.cpp є початковою точкою для запуску програми;

е) файл dynamic_system.pro є конфігураційним файлом, який включає встановленні модулі, та декларування заголовних (.h) та вихідних (.cpp) файлів.

Репозиторій організований за модульною структурою, оскільки така структура має численні переваги, які сприяють ефективності розробки, обслуговуванню та розширенню програмного забезпечення. Ось деякі з найважливіших переваг:

– легкість розробки: модульна структура дозволяє розробникам концентруватися на роботі над окремими модулями, що полегшує вивчення, розуміння та виправлення помилок в коді; кожен модуль може розглядатися як самодостатній компонент, що спрощує взаємодію та комунікацію між членами команди;

– відновлення та розширення: модульна архітектура робить проєкт більш гнучким та легким для змін; заміна або розширення одного модулю не впливає на інші, якщо інтерфейси залишаються незмінними; це полегшує внесення змін у функціональність проєкту без необхідності повного перекомпілювання або переробки інших частин програми;

– повторне використання коду: модульність сприяє повторному використанню коду; якщо модуль правильно спроектований та ізольований, його можна використовувати в інших проектах або навіть у межах того ж самого проєкту, що економить час та зусилля розробника;

– тестування та сумісність: кожен модуль може бути протестований окремо, що полегшує виявлення та виправлення помилок; також це сприяє забезпеченню сумісності між модулями, оскільки їхні інтерфейси чітко визначені;

– паралельна розробка: розробка може проводитися паралельно для різних модулів, що сприяє швидшому виходу на ринок та зменшенню часу відведеного на проект;

– зменшення залежностей: модульна архітектура дозволяє мінімізувати залежності між різними частинами програми; це допомагає уникнути ситуацій, коли зміни в одному місці можуть призвести до неочікуваних проблем в інших частинах системи;

– зручність управління: модульна структура полегшує управління розробкою та технічними змінами, оскільки окремі модулі можуть бути відділеними та оновлюваними незалежно один від одного.

Загалом, модульна структура дозволяє створювати більш гнучкі, швидкі та легко змінювані програмні продукти.

ВИСНОВКИ

Робота присвячена розробці методики дослідження та реалізуючої її програми для автоматизації процесів моделювання й розв’язання задач аналізу й керування за складною детермінованою позитивною динамічною системою із забезпеченням можливості отримання аналітичних результатів за довільний проміжок часу.

В результаті виконання роботи мета дослідження була досягнута в повному обсязі.

Розв’язання поставленої задачі дозволило:

– забезпечити встановлений рівень точності та ефективності досліджень;

– знизити трудомісткість здійснюваних обчислень;

– скоротити затребувані витрати часу та ресурсів для обробки інформації;

– спростити процеси моделювання, аналізу й керування досліджуваними складними системами;

– візуалізувати отримувані результати дослідження.

Структурно кваліфікаційна робота складається з 3 розділів.

В 1 розділі роботи розглянуто основні поняття теорії позитивних систем; розкрито особливості об’єкта дослідження, наведено основні задачі та підходи їх розв’язання з метою аналізу та здійснення різних видів керування.

У 2 розділі роботи наведено основні етапи розробленого алгоритму здійснюваних в роботі досліджень, за яким отримано програмну реалізацію змін основних характеристик досліджуваного об’єкта, опис, структурні компоненти, особливості використання, інструкцію для кінцевого користувача та приклади роботи програми для систем дискретного та неперервного часу за якою представлено в розділі 3.

Розроблене в роботі програмне забезпечення дозволило автоматизувати окреслені процеси дослідження з відносно великим ступенем участі людини у процесі дослідження для прийняття рішення стосовно вибору необхідної (з позиції поставленої мети) розв’язуваної задачі та з абсолютно низьким ступенем її участі у процесах отримання, перетворення, передавання і використання оперовуваних матеріалів та інформації.

ПЕРЕЛІК ПОСИЛАНЬ

1. Абраменко І. Г., Абраменко Д. І. Теорія автоматичного керування. Харків : ХНАМГ, 2008. 178 с.

2. Алілуйко А. М., Мазко О. Г. Інваріантні конуси та стійкість лінійних динамічних систем. *Український математичний журн*ал, 2006. Т. 58, № 11. С. 1446–1461.

3. Блохін Л. М., Бурiченко М. Ю. Статистична динаміка систем управління. Київ : НАУ, 2003. 208 с.

4. Гуров А. П., Ольшевський С. І., Черно О. О., Бугрім Л. І. Теорія автоматичного керування : навч. посіб. у 2 ч. Ч. 1. Миколаїв : НУК ім. адмірала Макарова, 2018. 111 с.

5. Жуковська О.А., Новицький В. В. Неокласичні динамічні керовані системи. Лінійні моделі. *Вопросы аналитической механики и ее применений*: *Праці Інституту математики НАН України*. Київ : Ін-т математики НАН України, 1999. Т. 26. С. 84–93.

6. Кондратьева Н. А., Леонтьева В. В., Чопоров С. В. К вопросу построения автоматизированной системы исследования сложных систем. *Матеріали міжнародної конференції «Dynamical System Modeling and Stability Investigation» – DSMSI-2007* (22-25 травня 2007 р., м. Київ). Київ : КНУ ім. Т. Шевченко, 2007. С. 377.

7. Ладанюк А. П. Основи системного аналізу. Вінниця: Нова книга, 2004. 176 с.

8. Лазарєв Ю. Ф. Математичні моделі та методи теоретичного дослідження стаціонарних лінійних динамічних систем. Конспект лекцій. Київ : КПІ, 1991. 156 с.

9. Лазарєв Ю. Ф. Моделювання динамічних систем у Matlab. Електронний навчальний посібник. Київ : НТУУ «КПІ», 2011. 421 с.

10. Леонтьева В. В. Математическая модель динамики функционирования позитивных систем балансового типа. *Зб. наук. праць. Вісник ЗНУ*. Запоріжжя : ЗНУ, 2008. №1 С. 118–124.

11. Леонтьева В. В., Кондратьева Н. А. Разомкнутая дискретная математическая модель позитивных динамических систем балансового типа и ее анализ. *Зб. наук. праць. Вісник ЗНУ*. Запоріжжя : ЗНУ, 2009. №1. С. 132–137.

12. Леонтьева В. В., Кондратьева Н. А. Управление в непрерывной математической модели позитивной динамической системы балансового типа. *Вестник Херсонского национального технического университета: Сб. научных статей*. Херсон : ХНТУ, 2009. Вып. 2 (35). С. 273–278.

13. Леонтьева В.В., Кондратьева Н.А. Управляемость позитивной динамической системы балансового типа. *Зб. наук. праць. Вісник ЗНУ*. Запоріжжя : ЗНУ. 2011. № 1. С. 58–66.

14. Новицький В. В. Декомпозиція та керування в лінійних системах. Київ : Ін-т математики НАН України, 1995. 150 с.

15. Олешко М. І., Соломко Н. О. Теорія автоматичного керування : навч. посіб. Ніжин: НДУ ім. М. Гоголя, 2019. 284 с

16. Полос С. Проект «динамічна система» // Github.com – Free Git Terminal. URL: https://github.com/stas-polos/dynamic-system/.

17. Попович М. Г., Ковальчук О. В. Теорія автоматичного керування : підручник. Київ : Либідь, 1997. 544 с.

18. Системний аналіз – Wikipedia. URL: https://inlnk.ru/zaYR5.

19. Системний аналіз – Довідково-інформаційний сайт. URL: https://systems-analysis.ru/systems_analysis.html.

20. Сорока К. О. Теорія автоматичного керування : навч. посіб. Харків : ХНАМГ, 2006. 187 с.

21. Сорока К. О. Теорія автоматичного керування і комп’ютерне моделювання (неперервні лінійні системи). Ч. 1. Основи теорії систем автоматичного керування : навч. посіб. Харків : ФОП Тімченко, 2010. 218 с.

22. Сорока К. О. Теорія автоматичного керування і комп’ютерне моделювання (неперервні лінійні системи). Ч. 2. Аналіз систем автоматичного керування засобами комп’ютерного моделювання : навч. посіб. Харків : ФОП Тімченко, 2010. 156 с.

23. Стенцель Й. І. Математичне моделювання технологічних об'єктів керування : навч. посіб. Київ: ІСДО, 1993. 328 с.

24. Штіфзон О. Й., Новіков П. В., Бунь В. П. Теорія автоматичного управління : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2020. 144 с.

25. GIT – Розподільна система керування версіями. URL: https://git-scm.com/.

26. Kaczorek T. Some recent developments in positive systems. *Proc. 7th Conf. on Dynamical Systems: Theory and Applications*. Łydź, 2003. P. 25–35.

27. Kvakernaak H., Siwan R. Linear Optimal Control Systems. New York: Wiley-lnterscience, 1972. 575 pp.

28. Luenberger D. G. Introduction to Dynamic Systems: Theory, Models and Applications. New York: John Wiley & Sons, 1979. 446 pp.

29. Schuppen J. H. Control and System Theory of Positive Systems. Amsterdam : The Vrije Universitei, 2007. 245 p.

Додаток А

Лістинг вихідного коду програмного продукту

Файл dynamic_system.pro:

```
#-------------------------------------------------
#
# Project created by QtCreator 2023-09-08T14:43:58
#
#-------------------------------------------------

QMAKE_CXXFLAGS += -std=c++11

QT       += core gui opengl

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = dynamic_system
TEMPLATE = app


SOURCES += src/main.cpp\
           src/system/dynamic_system.cpp \
           src/matrix/matrix.cpp \
           src/widget/gwidget.cpp

HEADERS  += src/system/dynamic_system.h \
            src/matrix/matrix.h \
            src/widget/gwidget.h
```

Файл src/system/dynamic_system.h

```
#ifndef DSYS_H
#define DSYS_H

#include "src/matrix/matrix.h"
#include "src/widget/gwidget.h"
#include <QWidget>
#include <QDialog>

class QPushButton;
class QCheckBox;
class QComboBox;
class QGroupBox;
class QLabel;
class QLineEdit;
class QRadioButton;
class QTableWidget;
class QTableWidgetItem;
class QTableView;
class QFont;
class GLWidget;
```

```
class DynamicSystem: public QDialog {
  Q_OBJECT

  public:
    DynamicSystem(QWidget * parent = 0);
  ~DynamicSystem();
  matrix A;
  matrix B;
  matrix C0;
  matrix X0;
  matrix X;
  matrix Xst;
  matrix k;

  matrix A_old;
  matrix B_old;
  matrix C0_old;
  matrix X0_old;
  matrix X_old;
  matrix Xst_old;
  matrix k_old;

  matrix tempM;
  matrix tempM_old;
  int systemSize;
  int x0;
  int x1;
  int y0;
  int y1;
  int tableRowCount;
  private slots:
    void toggleEditA();
  void toggleSave();
  void toggleStepOff();
  void toggleStepOn();
  void progControlOpen();
  void calculateInPoint();
  void resizeX();
  void resizeY();
  void incMatrixA();
  void decMatrixA();
  void incMatrixB();
  void decMatrixB();
  void resizeData();
  void takeNewData();
  void saveData();
  void progProceed();
  void progCancel();
  void progSelect(int option);

  private:
    QLineEdit * editSize;
  QPushButton * resize;
  QRadioButton * radioDiscrete;
  QRadioButton * radioContinuity;
  QCheckBox * checkChangeA;
  QCheckBox * checkPrevIter;
  QComboBox * stepList;
```

```
QTableWidget * matrixA;
QTableWidget * matrixB;
QTableWidget * matrixC0;
QTableWidget * matrixX0;
GLWidget * graphArea;
QTableWidget * tableArea;
QLineEdit * editX0;
QLineEdit * editX1;
QLineEdit * editY0;
QLineEdit * editY1;
QPushButton * decA;
QPushButton * incA;
QPushButton * decB;
QPushButton * incB;
QLineEdit * editA;
QLineEdit * editB;
QLineEdit * editCalc;
QPushButton * buttonCalc;
QPushButton * buttonProgControl;
QPushButton * launch;
QPushButton * save;
bool systemType;
bool isFirstStart;
bool isFirstStartSave;
bool canSave;
int stepMultipler;
float stepIndex;
bool isSave;
bool isChangable;
bool isUsedFeedback;
int pharseMatrix(int n, int m, QTableWidget * matrix);
bool isAllProductive();
void addNewLine(matrix tmp, matrix tmp_old, int t);
void getX_st(float t);
void getX(float t);
void getX_st_old(float t);
void getX_old(float t);
void getC(float t);
void takeIncK();
void takeDecK();
void takeEqvK();
int takeAttr;
void saveOldData();
void resizeSystem(bool clear);
void calculateDiscreteX(float t);
void calculateContinuosX(float t);

QDialog * progDialog;
QPushButton * progDialogProceed;
QPushButton * progDialogCancel;
QComboBox * progDialogSelect;
QTableWidget * progMatrixC0;
QTableWidget * progMatrixa0;
QTableWidget * progMatrixd0;
QTableWidget * progMatrixa0_1;
QLineEdit * progOmega;
QLineEdit * progK;
QComboBox * progSelectFunc;
```

```
  QLabel * progGroupd0;
  matrix C0_prog;
  matrix a0;
  matrix a0_1;
  matrix d0;
  matrix Ct;
  float omega;
  float powK;
  bool isProgControl;
  void progDialogInit();
  void progDialogResize();
  bool progFiNegative();
  int progCurrentDialogSelect;
};

#endif
```

Файл src/system/dynamic_system.cpp

```
#include "dynamic_system.h"
#include "src/widget/gwidget.h"
#include "src/matrix/matrix.h"
#include <QtGui>
#include <QDialog>
#include <QDateTime>
#include <QPixmap>
#include <QGLWidget>
#include <math.h>
#include <stdio.h>

DynamicSystem::DynamicSystem(QWidget * parent): QDialog(parent) {
  systemType = true;
  tableRowCount = 0;
  isChangable = true;
  isSave = false;
  isUsedFeedback = false;
  isProgControl = false;
  isFirstStart = true;
  isFirstStartSave = true;
  canSave = false;
  stepMultipler = 1;
  stepIndex = 1;
  systemSize = 3;
  tempM.setSize(1, (2 * systemSize) + 1);
  tempM_old.setSize(1, (2 * systemSize) + 1);
  graphArea = new GLWidget;
  graphArea -> setMaximumSize(278, 147);
  graphArea -> setMinimumSize(278, 147);
  // Size of graphics
  graphArea -> systemSize = 3;
  graphArea -> stepIndex = stepIndex;
  tableArea = new QTableWidget(this);
  tableArea -> verticalHeader() -> hide();
  QFont headerF("Tahoma", 7);
  tableRowCount = 0;
  // UI
  QVBoxLayout * mainInfo = new QVBoxLayout(this);
  QHBoxLayout * topSize = new QHBoxLayout;
```

```
  QGridLayout * topSystem = new QGridLayout;
  QGridLayout * topInfoM = new QGridLayout;
  QGridLayout * topInfoC = new QGridLayout;
  QGridLayout * bottomInfoS = new QGridLayout;
  QHBoxLayout * bottomCalc = new QHBoxLayout;
  QHBoxLayout * bottomInfoL = new QHBoxLayout;
  QHBoxLayout * bottomInfoA = new QHBoxLayout;
  QHBoxLayout * bottomInfoAi = new QHBoxLayout;
  QHBoxLayout * bottomInfoB = new QHBoxLayout;
  QHBoxLayout * bottomInfoBi = new QHBoxLayout;
  QVBoxLayout * bottomAll = new QVBoxLayout;
  QGridLayout * graphAll = new QGridLayout;
  QGroupBox * matrixInfo = new QGroupBox("Matrices", this);
  QGroupBox * graphInfo = new QGroupBox("Graphic", this);
  QGroupBox * changeInfo = new QGroupBox("Options", this);
  QGroupBox * radioSystem = new QGroupBox("System type", this);
  QLabel * labelA = new QLabel("Matrix A:", this);
  QLabel * labelB = new QLabel("Matrix B:", this);
  QLabel * labelC0 = new QLabel("Vector C0:", this);
  QLabel * labelX0 = new QLabel("Vector X0:", this);
  QLabel * labelStep = new QLabel("Step:", this);
  QLabel * labelCalc = new QLabel("Calculate values at point at t =",
this);
  radioDiscrete = new QRadioButton("Discrete system");
  radioContinuity = new QRadioButton("Continuous system");
  radioDiscrete -> setChecked(true);

  matrixA = new QTableWidget(systemSize, systemSize, this);
  matrixB = new QTableWidget(systemSize, systemSize, this);
  matrixC0 = new QTableWidget(systemSize, 1, this);
  matrixX0 = new QTableWidget(systemSize, 1, this);

  matrixA -> setMinimumSize(122, 122);
  matrixB -> setMinimumSize(122, 122);
  matrixC0 -> setMinimumSize(42, 122);
  matrixX0 -> setMinimumSize(42, 122);
  matrixA -> setMaximumSize(122, 122);
  matrixB -> setMaximumSize(122, 122);
  matrixC0 -> setMaximumSize(42, 122);
  matrixX0 -> setMaximumSize(42, 122);

  for (int i = 0; i < systemSize; i++) {
    matrixA -> horizontalHeader() -> resizeSection(i, 120 /
systemSize);
    matrixA -> verticalHeader() -> resizeSection(i, 120 / systemSize);

    matrixB -> horizontalHeader() -> resizeSection(i, 120 /
systemSize);
    matrixB -> verticalHeader() -> resizeSection(i, 120 / systemSize);

    matrixC0 -> horizontalHeader() -> resizeSection(i, 40);
    matrixC0 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);

    matrixX0 -> horizontalHeader() -> resizeSection(i, 40);
    matrixX0 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
  }
```

```
// initialization of program control
progDialogInit();
resizeSystem(true);

matrixA -> verticalHeader() -> hide();
matrixA -> horizontalHeader() -> hide();

matrixB -> verticalHeader() -> hide();
matrixB -> horizontalHeader() -> hide();

matrixC0 -> verticalHeader() -> hide();
matrixC0 -> horizontalHeader() -> hide();

matrixX0 -> verticalHeader() -> hide();
matrixX0 -> horizontalHeader() -> hide();

checkChangeA = new QCheckBox("Matrix A cannot be changed", this);
checkPrevIter = new QCheckBox("Save previous iteration", this);
buttonProgControl = new QPushButton("Program control", this);

stepList = new QComboBox;
stepList -> setEnabled(false);
stepList -> addItem("0.1");
stepList -> addItem("0.01");
stepList -> addItem("0.001");

labelStep -> setAlignment(Qt::AlignRight);
topSystem -> addWidget(radioDiscrete, 0, 0);
topSystem -> addWidget(radioContinuity, 0, 1);
topSystem -> addWidget(labelStep, 0, 2);
topSystem -> addWidget(stepList, 0, 3);
radioSystem -> setLayout(topSystem);

topInfoM -> addWidget(labelA, 0, 0);
topInfoM -> addWidget(labelB, 0, 1);
topInfoM -> addWidget(labelC0, 0, 2);
topInfoM -> addWidget(labelX0, 0, 3);
topInfoM -> addWidget(matrixA, 1, 0);
topInfoM -> addWidget(matrixB, 1, 1);
topInfoM -> addWidget(matrixC0, 1, 2);
topInfoM -> addWidget(matrixX0, 1, 3);

topInfoC -> addWidget(checkChangeA, 0, 0);
topInfoC -> setHorizontalSpacing(250);
topInfoC -> addWidget(buttonProgControl, 0, 1);
bottomInfoS -> addWidget(checkPrevIter, 0, 0);
matrixInfo -> setLayout(topInfoM);

buttonCalc = new QPushButton("Calculate", this);
editCalc = new QLineEdit(this);
editCalc -> setText("1");
editCalc -> setAlignment(Qt::AlignCenter);
editCalc -> setFixedWidth(50);
editCalc -> setMaxLength(10);

bottomCalc -> addSpacing(400);
bottomCalc -> addWidget(labelCalc);
```

```
  bottomCalc -> addWidget(editCalc);
  bottomCalc -> addWidget(buttonCalc);

  tableArea -> setColumnCount((2 * systemSize) + 1);
  tableArea -> setRowCount(tableRowCount);

  for (int tableInc = 0; tableInc < ((2 * systemSize) + 1);
tableInc++) {
    tableArea -> horizontalHeader() -> resizeSection(tableInc, 36);
    tableArea -> horizontalHeader() -> setFont(headerF);
  }

  QStringList tableLabels;
  tableLabels += "t";
  tableLabels += "X0";
  tableLabels += "X1";
  tableLabels += "X2";
  tableLabels += "X*0";
  tableLabels += "X*1";
  tableLabels += "X*2";
  tableArea -> setHorizontalHeaderLabels(tableLabels);
  editX0 = new QLineEdit(this);
  editX1 = new QLineEdit(this);
  editY0 = new QLineEdit(this);
  editY1 = new QLineEdit(this);

  graphAll -> setHorizontalSpacing(5);
  graphAll -> setVerticalSpacing(5);

  graphAll -> addWidget(graphArea, 0, 0, 5, 20);
  graphAll -> addWidget(editX0, 5, 0, 1, 1);
  graphAll -> addWidget(editX1, 5, 19, 1, 1);
  graphAll -> addWidget(editY1, 0, 20, 1, 1);
  graphAll -> addWidget(editY0, 4, 20, 1, 1);
  graphAll -> addWidget(tableArea, 0, 21, 6, 2);

  editX0 -> setMaximumSize(25, 20);
  editX1 -> setMaximumSize(25, 20);
  editY0 -> setMaximumSize(25, 20);
  editY1 -> setMaximumSize(25, 20);

  graphArea -> setMaximumWidth(280);
  tableArea -> setMaximumWidth(200);

  editX0 -> setMaxLength(4);
  editX0 -> setAlignment(Qt::AlignCenter);
  editX1 -> setMaxLength(4);
  editX1 -> setAlignment(Qt::AlignCenter);
  editY0 -> setMaxLength(4);
  editY0 -> setAlignment(Qt::AlignCenter);
  editY1 -> setMaxLength(4);
  editY1 -> setAlignment(Qt::AlignCenter);

  editX0 -> setText("0");
  editX1 -> setText("5");
  editY0 -> setText("0");
  editY1 -> setText("5");
```

```
graphInfo -> setLayout(graphAll);
resize = new QPushButton("Change", this);

QLabel * labelSize = new QLabel("Number of subsystems:", this);
editSize = new QLineEdit(this);
editSize -> setText("3");
editSize -> setAlignment(Qt::AlignCenter);
editSize -> setFixedWidth(20);
editSize -> setMaxLength(2);

topSize -> addWidget(labelSize);
topSize -> addWidget(editSize);
topSize -> addWidget(resize);
topSize -> addSpacing(400);

QLabel * labelChangeA = new QLabel("Control using matrix A:", this);
decA = new QPushButton("Decrease", this);
incA = new QPushButton("Increase", this);
QLabel * labelStepA = new QLabel("Values in increments:", this);
editA = new QLineEdit(this);

QLabel * labelChangeB = new QLabel("Control using matrix B:", this);
decB = new QPushButton("Decrease", this);
incB = new QPushButton("Increase", this);
QLabel * labelStepB = new QLabel("Values in increments:", this);
editB = new QLineEdit(this);

launch = new QPushButton("Execute", this);
save = new QPushButton("Save", this);

editA -> setText("0.01");
editA -> setMaxLength(6);
editA -> setAlignment(Qt::AlignCenter);
editA -> setFixedWidth(80);

editB -> setText("0.01");
editB -> setMaxLength(6);
editB -> setAlignment(Qt::AlignCenter);
editB -> setFixedWidth(80);

bottomInfoA -> addWidget(labelChangeA);
bottomInfoAi -> addWidget(decA);
bottomInfoAi -> addWidget(incA);
bottomInfoAi -> addWidget(labelStepA);
bottomInfoAi -> addWidget(editA);

bottomInfoB -> addWidget(labelChangeB);
bottomInfoBi -> addWidget(decB);
bottomInfoBi -> addWidget(incB);
bottomInfoBi -> addWidget(labelStepB);
bottomInfoBi -> addWidget(editB);

bottomAll -> addLayout(bottomInfoA);
bottomAll -> addLayout(bottomInfoAi);
bottomAll -> addLayout(bottomInfoB);
bottomAll -> addLayout(bottomInfoBi);
changeInfo -> setLayout(bottomAll);
```

```
  bottomInfoL -> addSpacing(200);
  bottomInfoL -> addWidget(launch);
  bottomInfoL -> addWidget(save);
  bottomInfoL -> addSpacing(200);

  mainInfo -> addLayout(topSize);
  mainInfo -> addWidget(radioSystem);
  mainInfo -> addWidget(matrixInfo);
  mainInfo -> addLayout(topInfoC);
  mainInfo -> addWidget(graphInfo);
  mainInfo -> addWidget(changeInfo);
  mainInfo -> addLayout(bottomInfoS);
  mainInfo -> addLayout(bottomCalc);
  mainInfo -> addLayout(bottomInfoL);
  setLayout(mainInfo);

  // signals
  connect(resize, SIGNAL(clicked()), SLOT(resizeData()));
  connect(radioDiscrete, SIGNAL(clicked()), SLOT(toggleStepOff()));
  connect(radioContinuity, SIGNAL(clicked()), SLOT(toggleStepOn()));
  connect(checkChangeA, SIGNAL(clicked()), SLOT(toggleEditA()));
  connect(buttonProgControl, SIGNAL(clicked()),
SLOT(progControlOpen()));
  connect(editX0, SIGNAL(editingFinished()), SLOT(resizeX()));
  connect(editX1, SIGNAL(editingFinished()), SLOT(resizeX()));
  connect(editY0, SIGNAL(editingFinished()), SLOT(resizeY()));
  connect(editY1, SIGNAL(editingFinished()), SLOT(resizeY()));
  connect(incA, SIGNAL(clicked()), SLOT(incMatrixA()));
  connect(incB, SIGNAL(clicked()), SLOT(incMatrixB()));
  connect(decA, SIGNAL(clicked()), SLOT(decMatrixA()));
  connect(decB, SIGNAL(clicked()), SLOT(decMatrixB()));
  connect(checkPrevIter, SIGNAL(clicked()), SLOT(toggleSave()));
  connect(buttonCalc, SIGNAL(clicked()), SLOT(calculateInPoint()));
  connect(launch, SIGNAL(clicked()), SLOT(takeNewData()));
  connect(save, SIGNAL(clicked()), SLOT(saveData()));
  connect(progDialogProceed, SIGNAL(clicked()), SLOT(progProceed()));
  connect(progDialogCancel, SIGNAL(clicked()), SLOT(progCancel()));
  connect(progDialogSelect, SIGNAL(currentIndexChanged(int)),
SLOT(progSelect(int)));

}

DynamicSystem::~DynamicSystem() {}

int DynamicSystem::pharseMatrix(int n, int m, QTableWidget * matrix) {
  QString sNum;
  for (int exI = 0; exI < n; exI++) {
    for (int exJ = 0; exJ < m; exJ++) {
      if (matrix -> item(exI, exJ) == NULL) {
        QMessageBox::warning(this, "Error", "Not all matrix values
have been entered. Fill in all cells.", "Continue");
        return 1;
      }

      sNum = matrix -> item(exI, exJ) -> text();
      for (int strI = 0; strI < sNum.length(); strI++) {
        bool validNumber = sNum[strI] <= QChar('9') && sNum[strI] >=
QChar('0');
```

```
        bool validChar = sNum[strI] <= QChar('.') && sNum[strI] >=
QChar(',');
        if (!validNumber && !validChar) {
          QMessageBox::critical(this, "Error", "Invalid characters
were found in matrices.", "Continue");
          return 1;
        }
      }
    }
  }
  return 0;
}

void DynamicSystem::addNewLine(matrix tmp, matrix tmp_old, int t) {
  QString sTmp;
  QString sTmp_old;
  for (int i = 0; i < ((2 * systemSize) + 1); i++) {
    sTmp.setNum(tmp.getElement(0, i));
    if (!isFirstStartSave) {
      sTmp_old.setNum(tmp_old.getElement(0, i));
    }

    if (isFirstStartSave) {
      tableArea -> setItem(t, i, new QTableWidgetItem(sTmp));
    } else {
      if (i == 0) {
        tableArea -> setItem(t, i, new QTableWidgetItem(sTmp));
      } else {
        tableArea -> setItem(t, i, new QTableWidgetItem(sTmp + ", " +
sTmp_old));
      }
    }
  }
}

// function of X*
void DynamicSystem::getX_st(float t) {
  matrix newM(systemSize, 1);
  matrix E;
  E.setSingle(systemSize);
  matrix tmp1(A);
  matrix tmp2(C0);

  tmp1 = E - tmp1;
  tmp1.getInverse();
  newM = tmp1 * tmp2;
  for (int i = 0; i < systemSize; i++) {
    Xst.setElement(i, 0, tmp1.getElement(i, 0));
  }
}

void DynamicSystem::getX_st_old(float t) {
  matrix newM(systemSize, 1);
  matrix E;
  E.setSingle(systemSize);
  matrix tmp1(A_old);
  matrix tmp2(C0_old);
```

```
  tmp1 = E - tmp1;
  tmp1.getInverse();
  newM = tmp1 * tmp2;
  for (int i = 0; i < systemSize; i++) {
    Xst_old.setElement(i, 0, tmp1.getElement(i, 0));
  }
}

void DynamicSystem::getX(float t) {
  if (systemType == true) {
    calculateDiscreteX(t);
    return;
  }
  calculateContinuosX(t);
}

void DynamicSystem::calculateContinuosX(float t) {
  bool isExpBug = false;
  matrix expBug(systemSize, systemSize);
  expBug.init(0);
  matrix expBugInc(systemSize, systemSize);
  expBugInc.init(0);
  matrix tmp1(systemSize, systemSize);
  matrix tmp3(systemSize, systemSize);
  tmp1.setSingle(systemSize);
  tmp3.setSingle(systemSize);
  matrix tmp2(A);
  tmp1 = tmp1 - B;
  tmp1.getInverse();
  tmp2 = tmp2 - tmp3;
  tmp1 = tmp1 * tmp2; // (I-B)^-1 * (A-I)
  matrix tmp4(tmp1);
  tmp4.getExp(t); // first summator
  matrix tmp5(X0);
  matrix integral(systemSize, 1); // systemSize, 1
  integral.init(0);
  matrix tmpI1(tmp1);
  matrix tmpI3(B);
  matrix tmpI2(systemSize, systemSize);
  tmpI2.setSingle(systemSize);
  tmpI2 = tmpI2 - tmpI3;
  tmpI2.getInverse(); // Bt

  float a = 0;
  float b = t;
  float n = 200;
  float h = (b - a) / n;
  for (int k = 1; k <= n; k++) {
    getC(a + h * (k - 0.5));
    tmpI1.initMatrix(tmp1, systemSize, systemSize);
    tmpI1 = tmpI1 * (-1);
    tmpI1.getExp(a + h * (k - 0.5));
    tmpI1 = tmpI1 * tmpI2;
    tmpI1 = tmpI1 * C0;
    for (int p = 0; p < systemSize; p++) {
      integral.setElement(p, 0, integral.getElement(p, 0) +
tmpI1.getElement(p, 0));
    }
```

```
  }
  integral = integral * h;

  tmp5 = tmp5 + integral;
  tmp4 = tmp4 * tmp5;
  for (int j = 0; j < systemSize; j++) {
    X.setElement(j, 0, tmp4.getElement(j, 0));
  }
}

void DynamicSystem::calculateDiscreteX(float t) {
  matrix tmp1(systemSize, systemSize);
  tmp1.setSingle(systemSize);
  matrix tmp2(A);
  tmp1 = tmp1 - B;
  tmp1.getInverse();
  tmp2 = tmp2 - B;
  tmp1 = tmp1 * tmp2; // (I-B)^-1 * (A-B)
  matrix tmp3(tmp1);
  matrix tmp4(tmp1);
  tmp3.getPow(t);
  tmp3 = tmp3 * X0;
  matrix tmp5(systemSize, systemSize);
  tmp5.setSingle(systemSize);
  tmp5 = tmp5 - B;
  tmp5.getInverse();
  matrix sumTmp(systemSize, systemSize);
  sumTmp.init(0);
  matrix tmpSum(tmp4);
  for (int step = 1; step <= t; step++) {
    getC(step - 1);
    tmp4.initMatrix(tmpSum, systemSize, systemSize);
    tmp4.getPow(t - step);
    tmp4 = tmp4 * tmp5;
    tmp4 = tmp4 * C0;
    sumTmp = sumTmp + tmp4;
  }

  for (int i = 0; i < systemSize; i++) {
    X.setElement(i, 0, tmp3.getElement(i, 0) + sumTmp.getElement(i,
0));
  }
}

void DynamicSystem::getC(float t) {
  // discreate system
  if (systemType == true) {
    if (isProgControl == true) {
      matrix firstSum(systemSize, systemSize);
      firstSum.setSingle(systemSize);
      matrix secondSum(A);
      firstSum = firstSum - B;
      secondSum = secondSum - B;

      matrix fi(systemSize, 1); // fi(t)
      matrix fi1(systemSize, 1); // fi(t+1)
      matrix tmpLin1(Ct);
      matrix tmpLin2(Ct);
```

```
    matrix tmpStep(Ct);
    matrix tmpStep1(d0);
    matrix tmpStep2(Ct);
    matrix tmpStep3(d0);
    matrix tmpHarm(Ct);
    matrix tmpHarm1(d0);
    matrix tmpHarm2(Ct);
    matrix tmpHarm3(d0);

    switch (progDialogSelect -> currentIndex()) {
    case 0:
      fi = tmpLin1 * pow(t, powK);
      tmpLin2 = tmpLin2 + a0;
      fi1 = tmpLin2 * pow(t + 1, powK);
      break;
    case 1:
      tmpStep1 = tmpStep1 * pow(t, powK);
      tmpStep = tmpStep + tmpStep1;
      fi = tmpStep;
      tmpStep3 = tmpStep3 * pow(t + 1, powK);
      tmpStep2 = tmpStep2 + a0;
      tmpStep2 = tmpStep2 + tmpStep3;
      fi1 = tmpStep2;
      break;
    case 2:
      switch (progSelectFunc -> currentIndex()) {
      case 0:
        tmpHarm1 = tmpHarm1 * sin(omega * t);
        tmpHarm3 = tmpHarm3 * sin(omega * (t + 1));
        break;
      case 1:
        tmpHarm1 = tmpHarm1 * cos(omega * t);
        tmpHarm3 = tmpHarm3 * cos(omega * (t + 1));
        break;
      case 2:
        tmpHarm1 = tmpHarm1 * tan(omega * t);
        tmpHarm3 = tmpHarm3 * tan(omega * (t + 1));
        break;
      }
      tmpHarm = tmpHarm + tmpHarm1;
      fi = tmpHarm;
      tmpHarm2 = tmpHarm2 + a0;
      tmpHarm2 = tmpHarm2 + tmpHarm3;
      fi1 = tmpHarm2;
      break;
    }

    firstSum = firstSum * fi1;
    secondSum = secondSum * fi;

    firstSum = firstSum - secondSum;

    for (int i = 0; i < systemSize; i++) {
      C0.setElement(i, 0, firstSum.getElement(i, 0));
    }
  }
}
// continuos system
```

```
  else {
    if (isProgControl == true) {}
  }
}

void DynamicSystem::getX_old(float t) {
  // Вычисление X для дискретного случая
  if (systemType == true) {
    matrix pM1(systemSize, systemSize);
    matrix pM2(systemSize, systemSize);

    matrix tmp1;
    matrix tmp2(A_old);
    matrix tmp3;
    matrix tmp4(A_old);
    matrix tmp5(systemSize, systemSize);
    matrix tmp6;

    tmp1.setSingle(systemSize);
    tmp3.setSingle(systemSize);
    tmp6.setSingle(systemSize);

    tmp1 = tmp1 - B_old;
    tmp1.getInverse();
    tmp2 = tmp2 - B_old;
    tmp1 = tmp1 * tmp2;
    tmp5 = tmp1;
    tmp1.getPow(t);
    pM1 = tmp1 * X0_old;
    tmp3 = tmp3 - A_old;
    tmp3.getInverse();
    tmp4 = tmp4 - B_old;
    tmp3 = tmp3 * tmp4;
    tmp5.getPow(t - 1);
    tmp6 = tmp6 - B_old;
    tmp6.getInverse();
    tmp6 = tmp6 * C0_old;
    pM2 = tmp3 * tmp5;
    pM2 = pM2 * tmp6;

    for (int i = 0; i < systemSize; i++) {
      X_old.setElement(i, 0, pM1.getElement(i, 0) +
Xst_old.getElement(i, 0) - pM2.getElement(i, 0));
    }

  }
  else {
    matrix X1(systemSize, systemSize);
    matrix X2(X0_old);
    matrix X3(systemSize, systemSize);

    matrix tmp1;
    matrix tmp2(A_old);
    matrix tmp3;
    tmp1.setSingle(systemSize);
    tmp3.setSingle(systemSize);

    tmp1 = tmp1 - B_old;
```

```
    tmp1.getInverse();
    tmp2 = tmp2 - tmp3;
    tmp1 = tmp1 * tmp2;
    tmp1.getExp(t);

    X1 = tmp1;
    X2 = X2 - Xst_old;
    X3 = X1 * X2;

    for (int j = 0; j < systemSize; j++) {
      X_old.setElement(j, 0, X3.getElement(j, 0) +
Xst_old.getElement(j, 0));
    }
  }
}

// check productive matrices
bool DynamicSystem::isAllProductive() {
  matrix tmpA(A);
  matrix tmpB(B);
  matrix tmpAB(systemSize, systemSize);
  matrix E;
  E.setSingle(systemSize);

  matrix tmp1(systemSize, systemSize);
  matrix tmp2(systemSize, systemSize);
  matrix tmp3(systemSize, systemSize);

  if (tmpA.isProductive() == true) {
    if (tmpB.isProductive() == true) {
      tmpAB = tmpA - tmpB;
      if (tmpAB.isProductive() == true) {
        tmp1 = E - B;
        tmp1.getInverse(); /*\*/
        tmp2 = tmpAB;
        tmp2 = tmp1 * tmpAB;
        if (tmp2.isProductive() == true) return true;
        else return false;
      } else return false;
    } else return false;
  } else return false;

}

void DynamicSystem::takeIncK() {
  float tmpSum = 0.0;
  k.init(-0.01);
  for (int i = 0; i < systemSize; i++) {
    for (int j = 0; j < systemSize; j++) tmpSum += A.getElement(i, j);
    if ((tmpSum + 0.03) > 1.0) {
      for (int p = 0; p < systemSize; p++) k.setElement(i, p, 0.0);
    }
    tmpSum = 0.0;
  }
  A = A - k;
  QString sTmp;
  for (int x = 0; x < systemSize; x++)
    for (int y = 0; y < systemSize; y++) {
```

```
      sTmp.setNum(A.getElement(x, y));
      matrixA -> setItem(x, y, new QTableWidgetItem(sTmp));
    }
  isUsedFeedback = true;
}

void DynamicSystem::takeDecK() {
  k.init(0.0);
  for (int i = 0; i < systemSize; i++) {
      for (int j = 0; j < systemSize; j++) {
          if ((A.getElement(i, j) - B.getElement(i, j)) > 0.01)
k.setElement(i, j, 0.01);
      }
  }

  A = A - k;
  QString sTmp;
  for (int x = 0; x < systemSize; x++) {
    for (int y = 0; y < systemSize; y++) {
      sTmp.setNum(A.getElement(x, y));
      matrixA -> setItem(x, y, new QTableWidgetItem(sTmp));
    }
  }

  isUsedFeedback = true;
}

void DynamicSystem::takeEqvK() {
  matrix tmpA(A);
  matrix tmpB(B);
  matrix tmpK(systemSize, systemSize);
  tmpK.init(0.01);
  k = tmpA - tmpB - tmpK;
  A = A - k;
  QString sTmp;
  for (int x = 0; x < systemSize; x++) {
    for (int y = 0; y < systemSize; y++) {
      sTmp.setNum(A.getElement(x, y));
      matrixA -> setItem(x, y, new QTableWidgetItem(sTmp));
    }
  }
  isUsedFeedback = true;
}

void DynamicSystem::saveOldData() {
  for (int i = 0; i < systemSize; i++) {
    for (int j = 0; j < systemSize; j++) {
      A_old.setElement(i, j, A.getElement(i, j) + k.getElement(i, j));
// Probable BUG
      B_old.setElement(i, j, B.getElement(i, j));
      k_old.setElement(i, j, k.getElement(i, j));
    }
    C0_old.setElement(i, 0, C0.getElement(i, 0));
    X0_old.setElement(i, 0, X0.getElement(i, 0));
  }
  canSave = true;
}
```

```
void DynamicSystem::resizeSystem(bool clear) {
  if (clear == true) {
    A.setSize(systemSize, systemSize);
    B.setSize(systemSize, systemSize);
    k.setSize(systemSize, systemSize);
    k.init(0);
    C0.setSize(systemSize, 1);
    X0.setSize(systemSize, 1);
    X.setSize(systemSize, 1);
    Xst.setSize(systemSize, 1);

    A_old.setSize(systemSize, systemSize);
    B_old.setSize(systemSize, systemSize);
    k_old.setSize(systemSize, systemSize);
    k_old.init(0);
    C0_old.setSize(systemSize, 1);
    X0_old.setSize(systemSize, 1);
    X_old.setSize(systemSize, 1);
    Xst_old.setSize(systemSize, 1);

    tempM.setSize(1, (2 * systemSize) + 1);
    tempM_old.setSize(1, (2 * systemSize) + 1);

    matrixA -> setColumnCount(systemSize);
    matrixA -> setRowCount(systemSize);
    matrixB -> setColumnCount(systemSize);
    matrixB -> setRowCount(systemSize);
    matrixC0 -> setRowCount(systemSize);
    matrixX0 -> setRowCount(systemSize);

    int fSize = 8;
    if (systemSize > 4) {
      fSize = 7;
      if (systemSize > 6) {
        fSize = 5;
        if (systemSize > 8)
          fSize = 4;
      }
    } else {
        fSize = 8;
    }

    QFont matrixFont("Tahoma", fSize);
    matrixA -> setFont(matrixFont);
    matrixB -> setFont(matrixFont);
    matrixC0 -> setFont(matrixFont);
    matrixX0 -> setFont(matrixFont);
    for (int i = 0; i < systemSize; i++) {
      matrixA -> horizontalHeader() -> resizeSection(i, 120 /
systemSize);
      matrixA -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
      matrixB -> horizontalHeader() -> resizeSection(i, 120 /
systemSize);
      matrixB -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
      matrixC0 -> horizontalHeader() -> resizeSection(i, 40);
```

```
      matrixC0 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
      matrixX0 -> horizontalHeader() -> resizeSection(i, 40);
      matrixX0 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
    }

    matrixA -> clear();
    matrixB -> clear();
    matrixC0 -> clear();
    matrixX0 -> clear();

    progDialogResize();

    graphArea -> isGraphBuild = true;
    graphArea -> updateGL();

    tableRowCount = 0;
    tableArea -> setRowCount(tableRowCount);
    tableArea -> setColumnCount((2 * systemSize) + 1);

    for (int j = 0; j < ((2 * systemSize) + 1); j++) {
        tableArea -> horizontalHeader() -> resizeSection(j, 36);
    }

    QStringList tableLabels;
    QString sTmp;
    for (int k = 0; k < 3; k++) {
      if (k == 0) {
          tableLabels += "t";
      }

      if (k == 1) {
          for (int xi = 0; xi < systemSize; xi++) {
            sTmp.setNum(xi);
            tableLabels += "X" + sTmp;
          }
      }

      if (k == 2) {
          for (int xj = 0; xj < systemSize; xj++) {
            sTmp.setNum(xj);
            tableLabels += "X*" + sTmp;
          }
      }
    }
    tableArea -> setHorizontalHeaderLabels(tableLabels);
  }
  isChangable = true;
  isSave = false;
  isUsedFeedback = false;
  isFirstStart = true;
  isFirstStartSave = true;
  canSave = false;
}

void DynamicSystem::toggleStepOff() {
  stepList -> setEnabled(false);
```

```
  systemType = true;
  stepMultipler = 1;
  stepIndex = 1;
  graphArea -> stepIndex = 1;
  resizeSystem(false);
}

void DynamicSystem::toggleStepOn() {
  stepList -> setEnabled(true);
  systemType = false;
}

void DynamicSystem::progControlOpen() {
  if ((pharseMatrix(systemSize, 1, matrixC0) == 0)) {
    progDialog -> show();

    progOmega -> setText("1.00");
    progK -> setText("0.1");
    // Copy and initialize matrixes with numbers
    QString sTmp;
    double dTmp;

    for (int i = 0; i < systemSize; i++) {
      dTmp = matrixC0 -> item(i, 0) -> text().toDouble();
      sTmp.setNum(dTmp);
      progMatrixC0 -> setItem(i, 0, new QTableWidgetItem(sTmp));
      dTmp /= 100;
      sTmp.setNum(dTmp);
      progMatrixa0 -> setItem(i, 0, new QTableWidgetItem(sTmp));
      progMatrixa0_1 -> setItem(i, 0, new QTableWidgetItem(sTmp));
      dTmp = matrixC0 -> item(i, 0) -> text().toDouble();
      sTmp.setNum(dTmp);
      progMatrixd0 -> setItem(i, 0, new QTableWidgetItem(sTmp));
    }
  } else {
      QMessageBox::critical(this, "Error", "Matrix C0 is specified
incorrectly\nIt is impossible to go to program controls", "Continue");
  }
}

void DynamicSystem::toggleEditA() {
  if (incA -> isEnabled() == true) {
    incA -> setEnabled(false);
    decA -> setEnabled(false);
    editA -> setEnabled(false);
    isChangable = false;
  } else {
    incA -> setEnabled(true);
    decA -> setEnabled(true);
    editA -> setEnabled(true);
    isChangable = true;
  }
}

void DynamicSystem::toggleSave() {
  if (isSave == true) {
    isSave = false;
    graphArea -> isSavedOld = false;
```

```
    isFirstStartSave = true;
  } else {
    isSave = true;
    graphArea -> isSavedOld = false;
    isFirstStartSave = true;
  }
}

void DynamicSystem::incMatrixA() {
  double dTmp;
  QString sTmp;
  if (pharseMatrix(systemSize, systemSize, matrixA) == 0) {
    for (int i = 0; i < systemSize; i++) {
      for (int j = 0; j < systemSize; j++) {
        dTmp = matrixA -> item(i, j) -> text().toDouble() + editA ->
text().toDouble();
        sTmp.setNum(dTmp);
        matrixA -> setItem(i, j, new QTableWidgetItem(sTmp));
      }
    }
  }
}

void DynamicSystem::decMatrixA() {
  double dTmp;
  QString sTmp;
  if (pharseMatrix(systemSize, systemSize, matrixA) == 0) {
    for (int i = 0; i < systemSize; i++) {
      for (int j = 0; j < systemSize; j++) {
        dTmp = matrixA -> item(i, j) -> text().toDouble() - editA ->
text().toDouble();
        sTmp.setNum(dTmp);
        matrixA -> setItem(i, j, new QTableWidgetItem(sTmp));
      }
    }
  }
}

void DynamicSystem::incMatrixB() {
  double dTmp;
  QString sTmp;
  if (pharseMatrix(systemSize, systemSize, matrixB) == 0) {
    for (int i = 0; i < systemSize; i++) {
      for (int j = 0; j < systemSize; j++) {
        dTmp = matrixB -> item(i, j) -> text().toDouble() + editB ->
text().toDouble();
        sTmp.setNum(dTmp);
        matrixB -> setItem(i, j, new QTableWidgetItem(sTmp));
      }
    }
  }
}

void DynamicSystem::decMatrixB() {
  double dTmp;
  QString sTmp;
  if (pharseMatrix(systemSize, systemSize, matrixB) == 0) {
    for (int i = 0; i < systemSize; i++) {
```

```
      for (int j = 0; j < systemSize; j++) {
        dTmp = matrixB -> item(i, j) -> text().toDouble() - editB ->
text().toDouble();
        sTmp.setNum(dTmp);
        matrixB -> setItem(i, j, new QTableWidgetItem(sTmp));
      }
    }
  }
}

void DynamicSystem::resizeX() {
  QString tmp;
  int X0, X1;
  bool ok = TRUE;
  X0 = editX0 -> text().toInt( & ok, 10);
  if (ok == FALSE) {
    QMessageBox::warning(this, "Error", "The starting coordinate is
not an integer", "Continue");
    return;
  }
  X1 = editX1 -> text().toInt( & ok, 10);
  if (ok == FALSE) {
    QMessageBox::warning(this, "Error", "End coordinate is not an
integer", "Continue");
    return;
  }
  if ((X0 < 0) || (X1 < 0)) {
    QMessageBox::warning(this, "Error", "Due to the non-negativity of
solutions, it is impossible to specify a negative coordinate\n(must be
greater than 0)", "Continue");
    return;
  }
  if (X0 == X1) {
    QMessageBox::critical(this, "Error", "The start and end
coordinates along the X axis are the same. Unable to plot",
"Continue");
    return;
  }
  if (X0 > X1) {
    QMessageBox::warning(this, "Warning", "The coordinates will be
inverted (Start coordinate is greater than end coordinate)",
"Continue");
    tmp = editX0 -> text();
    editX0 -> setText(editX1 -> text());
    editX1 -> setText(tmp);
  }
}

void DynamicSystem::resizeY() {
  QString tmp;
  int Y0, Y1;
  bool ok = TRUE;
  Y0 = editY0 -> text().toInt( & ok, 10);
  if (ok == FALSE) {
    QMessageBox::warning(this, "Error", "The starting coordinate is
not an integer", "Continue");
    return;
  }
```

```
  Y1 = editY1 -> text().toInt( & ok, 10);
  if (ok == FALSE) {
    QMessageBox::warning(this, "Error", "End coordinate is not an
integer", "Continue");
    return;
  }

  if (Y0 == Y1) {
    QMessageBox::critical(this, "Error", "The start and end
coordinates along the Y axis are the same. Unable to plot",
"Continue");
    return;
  }

  if (Y0 > Y1) {
    QMessageBox::warning(this, "Warning", "The coordinates will be
inverted (Start coordinate is greater than end coordinate)",
"Continue");
    tmp = editY0 -> text();
    editY0 -> setText(editY1 -> text());
    editY1 -> setText(tmp);
  }
}

void DynamicSystem::takeNewData() {
  bool ok0 = TRUE;
  bool ok1 = TRUE;
  bool ok2 = TRUE;
  bool ok3 = TRUE;

  editX0 -> text().toInt( & ok0, 10);
  editX1 -> text().toInt( & ok1, 10);
  editY0 -> text().toInt( & ok2, 10);
  editY1 -> text().toInt( & ok3, 10);

  tableRowCount = 0;
  tableArea -> clear();
  QStringList tableLabels;
  QString sTmp;
  for (int sC = 0; sC < 3; sC++) {
    if (sC == 0) {
      tableLabels += "t";
    }

    if (sC == 1) {
      for (int sCi = 0; sCi < systemSize; sCi++) {
        sTmp.setNum(sCi);
        tableLabels += "X" + sTmp;
      }
    }

    if (sC == 2) {
      for (int sCj = 0; sCj < systemSize; sCj++) {
        sTmp.setNum(sCj);
        tableLabels += "X*" + sTmp;
      }
    }
```

```
  }

  tableArea -> setHorizontalHeaderLabels(tableLabels);
  auto isValidMatrixA = pharseMatrix(systemSize, systemSize, matrixA)
== 0;
  auto isValidMatrixB = pharseMatrix(systemSize, systemSize, matrixB)
== 0;
  auto isValidMatrixC0 = pharseMatrix(systemSize, 1, matrixC0) == 0;
  auto isValidMatrixX0 = pharseMatrix(systemSize, 1, matrixX0) == 0;

  if (isValidMatrixA && isValidMatrixB && isValidMatrixC0 &&
isValidMatrixX0) {
    if (!((ok0 == FALSE) || (ok1 == FALSE) || (ok2 == FALSE) || (ok3
== FALSE))) {
      if (progFiNegative()) {
        for (int i = 0; i < systemSize; i++) {
          C0.setElement(i, 0, matrixC0 -> item(i, 0) ->
text().toDouble());
          X0.setElement(i, 0, matrixX0 -> item(i, 0) ->
text().toDouble());

          for (int j = 0; j < systemSize; j++) {
            A.setElement(i, j, matrixA -> item(i, j) ->
text().toDouble());
            B.setElement(i, j, matrixB -> item(i, j) ->
text().toDouble());
          }
        }

        bool ok = TRUE;
        x0 = editX0 -> text().toInt( & ok, 10);
        x1 = editX1 -> text().toInt( & ok, 10);
        if ((x0 < 0) || (x1 < 0)) {
          QMessageBox::critical(this, "Error", "Due to the non-
negativity of solutions, it is impossible to specify a negative
coordinate\n(must be greater than 0)", "Continue");
          return;
        }

        y0 = editY0 -> text().toInt( & ok, 10);
        y1 = editY1 -> text().toInt( & ok, 10);
        if ((x0 == x1) || (y0 == y1)) {
          QMessageBox::critical(this, "Error", "The start and end
coordinates are the same. Unable to plot", "Continue");
          return;
        }

        if (systemType == false) {
          switch (stepList -> currentIndex()) {
          case 0:
            stepMultipler = 10;
            stepIndex = 0.1;
            graphArea -> stepIndex = 0.1;
            break;
          case 1:
            stepMultipler = 100;
            stepIndex = 0.01;
            graphArea -> stepIndex = 0.01;
```

```
            break;
          case 2:
            stepMultipler = 1000;
            stepIndex = 0.001;
            graphArea -> stepIndex = 0.001;
            break;
          }
        }
        feedbackLabel:
          tableRowCount = stepMultipler * (x1 - x0) + 1;

        tableArea -> setRowCount(tableRowCount);
        graphArea -> pArray = new float * [tableRowCount];
        for (int count = 0; count < tableRowCount; count++) {
             graphArea -> pArray[count] = new float[2 * systemSize];
        }

        if (isFirstStartSave == false) {
          graphArea -> pArray_old = new float * [tableRowCount];
          for (int count_old = 0; count_old < tableRowCount;
count_old++) {
               graphArea -> pArray_old[count_old] = new float[2 *
systemSize];
          }
        }

        graphArea -> isGraphBuild = true;
        graphArea -> setLines(x0, x1, y0, y1);
        if (isAllProductive() == true) {
          int line = 0;
          bool isBug = false;
          Ct.initMatrix(C0, systemSize, 1);
          getC(0);
          getX_st(x0);
          if (isFirstStartSave == false) {
              getX_st_old(x0);
          }

          for (float t = x0; t <= x1; t += stepIndex) {
            tempM.setElement(0, 0, t);
            if (isBug == false) {
              getX(t);
              if (isFirstStartSave == false) {
                  getX_old(t);
              }
            }

            if (isBug == false && systemType == false) {
              for (int b = 0; b < systemSize; b++) {
                  if (((X.getElement(b, 0) - Xst.getElement(b, 0)) <
0.001) && systemType == false) {
                      isBug = true;
                  }
              }

            }

            if (isBug == true && systemType == false) {
```

```
              for (int bf = 0; bf < systemSize; bf++) {
                X.setElement(bf, 0, Xst.getElement(bf, 0));
                if (isFirstStartSave == false) {
                    X_old.setElement(bf, 0, Xst_old.getElement(bf,
0));
                }
              }
            }

            for (int xe = 0; xe < systemSize; xe++) {
              tempM.setElement(0, xe + systemSize + 1,
Xst.getElement(xe, 0));
              tempM.setElement(0, xe + 1, X.getElement(xe, 0));
            }

            if (isFirstStartSave == false) {
              for (int xe_old = 0; xe_old < systemSize; xe_old++) {
                tempM_old.setElement(0, xe_old + systemSize + 1,
Xst_old.getElement(xe_old, 0));
                tempM_old.setElement(0, xe_old + 1,
X_old.getElement(xe_old, 0));
              }
            }

            addNewLine(tempM, tempM_old, line);
            for (int pa = 0; pa < systemSize; pa++) {
              graphArea -> pArray[line][pa] = X.getElement(pa, 0);
              graphArea -> pArray[line][pa + systemSize] =
Xst.getElement(pa, 0);
            }

            if (isFirstStartSave == false) {
              for (int pa_old = 0; pa_old < systemSize; pa_old++) {
                graphArea -> pArray_old[line][pa_old] =
X_old.getElement(pa_old, 0);
                graphArea -> pArray_old[line][pa_old + systemSize] =
Xst_old.getElement(pa_old, 0);
              }
            }
            line++;
          }
          graphArea -> isGraphBuild = false;
          graphArea -> updateGL();

          if (isSave == true) {
            saveOldData();
            if (isFirstStartSave == false) {
                graphArea -> isSavedOld = true;
            }
            isFirstStartSave = false;
          }

        } else {
          matrix tmpAB(systemSize, systemSize);
          matrix tmpA(A);
          matrix tmpB(B);
          if (tmpA.isProductive() == true) {
            if (tmpB.isProductive() == true) {
```

```
                matrix tmp1(systemSize, systemSize);
                matrix tmp2(systemSize, systemSize);
                matrix tmp3(systemSize, systemSize);
                matrix E;
                E.setSingle(systemSize);
                tmpAB = tmpA - tmpB;
                if (tmpAB.isProductive() == true) {

                  tmp1 = E - B;
                  tmp1.getInverse();
                  tmp2 = tmpAB;
                  tmp2 = tmp1 * tmpAB;
                  if (!(tmp2.isProductive() == true)) {
                      QMessageBox::critical(this, "Error", "The matrix
(E-B)^(-1)*(A-B) is not productive!\nSystem construction is not
available.", "Continue");
                  }

                } else {
                    QMessageBox::critical(this, "Error", "The A-B matrix
is not productive!\nSystem building is not available.", "Continue");
                }
              } else {
                  QMessageBox::critical(this, "Error", "Matrix B is not
productive!\nSystem construction is not available.", "Continue");
              }
            } else {
                QMessageBox::critical(this, "Error", "Matrix A is not
productive!\nSystem construction is not available.", "Continue");
            }

            if (tmpB.isProductive() == true) {
              for (int prodI = 0; prodI < systemSize; prodI++) {
                  for (int prodJ = 0; prodJ < systemSize; prodJ++) {
                      if (A.getElement(prodI, prodJ) <
B.getElement(prodI, prodJ)) {
                          QMessageBox::warning(this, "Warning", "The
system is asymptotically unstable!\nWe build a system with a feedback
gain matrix K.", "Continue");

                          k.init(0.0);
                          for (int i = 0; i < systemSize; i++) {
                              for (int j = 0; j < systemSize; j++) {
                                  if (A.getElement(i, j) < B.getElement(i,
j)) {
                                      if (A.getElement(i, j) < 0) {
                                          k.setElement(i, j,
A.getElement(i, j) - B.getElement(i, j) - 0.01);
                                      } else {
                                          k.setElement(i, j, -
(B.getElement(i, j) - A.getElement(i, j)) - 0.01);
                                      }
                                  }
                              }
                          }

                          A = A - k;
                          QString sTmp;
```

```
                        for (int x = 0; x < systemSize; x++) {
                            for (int y = 0; y < systemSize; y++) {
                              sTmp.setNum(A.getElement(x, y));
                              matrixA -> setItem(x, y, new
QTableWidgetItem(sTmp));
                            }
                        }
                        break;
                    }
                }
            }
          }
        }

        if ((isUsedFeedback == false) && (isAllProductive() == true))
{
          QMessageBox msgProd;
          QPushButton * incButton = msgProd.addButton("Increase",
QMessageBox::ActionRole);
          QPushButton * decButton = msgProd.addButton("Decrease",
QMessageBox::ActionRole);
          QPushButton * eqvButton = msgProd.addButton("Equality",
QMessageBox::ActionRole);
          QPushButton * notButton = msgProd.addButton("Do not use",
QMessageBox::ActionRole);
          msgProd.setIcon(QMessageBox::Question);
          msgProd.setWindowTitle("Selection of matrix k");
          msgProd.setInformativeText("Management:\n\n1. To increase
the values of the vector elements Xt and X*\n2. To reduce the values
of the vector elements Xt and X*\n3. To ensure equality of vector
elements Xt and X*");
          msgProd.exec();
          if (msgProd.clickedButton() == incButton) {
            takeIncK();
            takeAttr = 1;
            goto feedbackLabel;
          }
          if (msgProd.clickedButton() == decButton) {
            takeDecK();
            takeAttr = 2;
            goto feedbackLabel;
          }
          if (msgProd.clickedButton() == eqvButton) {
            takeEqvK();
            takeAttr = 3;
            goto feedbackLabel;
          }
        }
        if ((isUsedFeedback == true) && (isAllProductive() == true)) {
          QMessageBox msgContinue;
          QPushButton * yesButton = msgContinue.addButton("Yes",
QMessageBox::ActionRole);
          QPushButton * noButton = msgContinue.addButton("No",
QMessageBox::ActionRole);
          msgContinue.setIcon(QMessageBox::Question);
          msgContinue.setWindowTitle("Feedback gain matrix k");
          msgContinue.setInformativeText("Continue with the choice of
matrix k according to the given condition?");
```

```
          msgContinue.exec();
          if (msgContinue.clickedButton() == yesButton) {
            switch (takeAttr) {
            case 1:
              takeIncK();
              break;
            case 2:
              takeDecK();
              break;
            case 3:
              takeEqvK();
              break;
            }
            goto feedbackLabel;
          }
        }
        isUsedFeedback = false;
      } else
        QMessageBox::warning(this, "Error", "It is impossible to build
a system. Matrix C(t) < 0.\nTry using other parameters in the
matrix.", "Continue");
    } else
      QMessageBox::critical(this, "Error", "The coordinates are set
incorrectly", "Continue");
  }
  isFirstStart = false;
  isProgControl = false;
}

void DynamicSystem::calculateInPoint() {
  if ((pharseMatrix(systemSize, systemSize, matrixA) == 0) &&
(pharseMatrix(systemSize, systemSize, matrixB) == 0) &&
(pharseMatrix(systemSize, 1, matrixC0) == 0) &&
(pharseMatrix(systemSize, 1, matrixX0) == 0)) {
    for (int i = 0; i < systemSize; i++) {
      C0.setElement(i, 0, matrixC0 -> item(i, 0) ->
text().toDouble());
      X0.setElement(i, 0, matrixX0 -> item(i, 0) ->
text().toDouble());
      for (int j = 0; j < systemSize; j++) {
        A.setElement(i, j, matrixA -> item(i, j) ->
text().toDouble());
        B.setElement(i, j, matrixB -> item(i, j) ->
text().toDouble());
      }
    }
    double point;
    if (systemType == true) {
      bool ok = TRUE;
      point = editCalc -> text().toInt( & ok, 10);
      if (ok == FALSE) {
        QMessageBox::critical(this, "Error", "The specified time t is
not an integer.", "Continue");
        return;
      }
    } else {
      QString sTmp;
      sTmp = editCalc -> text();
```

```
      for (int stI = 0; stI < sTmp.length(); stI++) {
        if (!(sTmp[stI] <= QChar('9') && sTmp[stI] >= QChar('0')) &&
!(sTmp[stI] <= QChar('.') && sTmp[stI] >= QChar(','))) {
          QMessageBox::critical(this, "Error", "Invalid characters
detected at specified time t", "Continue");
          return;
        }
      }
      point = editCalc -> text().toDouble();
    }

    if (point < 0) {
      QMessageBox::critical(this, "Error", "Due to the non-negativity
of solutions, it is impossible to specify a negative moment in time
t", "Continue");
      return;
    } else {
      if (isAllProductive() == true) {
        QString str("Function value at point t = ");
        QString sRes, sNum, sPoint;
        str += sPoint.setNum(point) + "\n";
        getX_st(point);
        getX(point);
        for (int sI = 0; sI < systemSize; sI++) {
          sRes.setNum(X.getElement(sI, 0));
          sNum.setNum(sI);
          str += "X" + sNum + " = " + sRes + "; ";
        }
        str += "\n Equilibrium state: \n";
        for (int stI = 0; stI < systemSize; stI++) {
          sRes.setNum(Xst.getElement(stI, 0));
          sNum.setNum(stI);
          str += "X*" + sNum + " = " + sRes + "; ";
        }
        QMessageBox::information(this, "Function value", str,
"Continue");
      }
    }
  }

}

void DynamicSystem::resizeData() {
  bool ok = true;
  editSize -> text().toInt( & ok, 10);

  if (!(ok == FALSE)) {
    QMessageBox msgClear;
    QPushButton * yesClear = msgClear.addButton("Yes",
QMessageBox::ActionRole);
    QPushButton * noClear = msgClear.addButton("No",
QMessageBox::ActionRole);
    msgClear.setIcon(QMessageBox::Question);
    msgClear.setWindowTitle("Warning");
    msgClear.setInformativeText("When changing the number of
subsystems, all previously entered values will be lost.\nContinue?");
    msgClear.exec();
```

```
    if (msgClear.clickedButton() == yesClear) {
      systemSize = editSize -> text().toInt( & ok, 10);
      graphArea -> systemSize = systemSize;
      graphArea -> stepIndex = stepIndex;
      resizeSystem(true);
    }
  } else {
    QMessageBox::critical(this, "Error", "Invalid characters when
specifying the number of subsystems", "Continue");
    return;
  }

}

void DynamicSystem::saveData() {
  FILE * fTxt, * fImage;

  if (isFirstStart == true) {
    QMessageBox::critical(this, "Error", "No information to save",
"Continue");
    return;
  }

  QDateTime dateTime = QDateTime::currentDateTime();
  QString dateTimeT = dateTime.toString("dd-MM-yyyy-hh-mm-ss") +
".txt";
  QString dateTimeI = dateTime.toString("dd-MM-yyyy-hh-mm-ss") +
".bmp";
  dateTimeT.toLocal8Bit();
  dateTimeI.toLocal8Bit();

  char * fileTextName = dateTimeT.toAscii().data();
  char * fileImgName = dateTimeI.toAscii().data();

  if ((isSave == true) && (canSave == true)) {
      QMessageBox::critical(this, "Error", "Storing two states",
"Continue");
  } else {
      QMessageBox::critical(this, "Error", "Saving one state",
"Continue");
  }

  QImage scr(278, 147, QImage::Format_RGB16);
  scr = graphArea -> grabFrameBuffer();
  scr.save(dateTimeI, "BMP");
}

void DynamicSystem::progDialogInit() {
  progDialog = new QDialog(this);
  progDialog -> setFixedSize(500, 300);
  progDialog -> setWindowTitle("Program control");
  progDialog -> setModal(true);

  progDialogProceed = new QPushButton("Continue", this);
  progDialogCancel = new QPushButton("Cancel", this);

  progDialogSelect = new QComboBox(this);
  progDialogSelect -> addItem("Power");
```

```
  progDialogSelect -> addItem("Power (extended)");
  progDialogSelect -> addItem("Trigonometric");

  progSelectFunc = new QComboBox(this);
  progSelectFunc -> addItem("sin");
  progSelectFunc -> addItem("cos");
  progSelectFunc -> addItem("tg");

  progOmega = new QLineEdit(this);
  progK = new QLineEdit(this);
  progOmega -> setMaxLength(4);
  progOmega -> setAlignment(Qt::AlignCenter);
  progK -> setMaxLength(4);
  progK -> setAlignment(Qt::AlignCenter);
  progOmega -> setMaximumSize(35, 20);
  progK -> setMaximumSize(35, 20);
  progMatrixC0 = new QTableWidget(systemSize, 1, this);
  progMatrixa0 = new QTableWidget(systemSize, 1, this);
  progMatrixd0 = new QTableWidget(systemSize, 1, this);
  progMatrixa0_1 = new QTableWidget(systemSize, 1, this);
  progMatrixC0 -> setMinimumSize(42, 122);
  progMatrixa0 -> setMinimumSize(42, 122);
  progMatrixd0 -> setMinimumSize(42, 122);
  progMatrixa0_1 -> setMinimumSize(42, 122);
  progMatrixC0 -> setMaximumSize(42, 122);
  progMatrixa0 -> setMaximumSize(42, 122);
  progMatrixd0 -> setMaximumSize(42, 122);
  progMatrixa0_1 -> setMaximumSize(42, 122);
  for (int i = 0; i < systemSize; i++) {
    progMatrixC0 -> horizontalHeader() -> resizeSection(i, 120 /
systemSize);
    progMatrixC0 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
    progMatrixa0 -> horizontalHeader() -> resizeSection(i, 120 /
systemSize);
    progMatrixa0 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
    progMatrixd0 -> horizontalHeader() -> resizeSection(i, 40);
    progMatrixd0 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
    progMatrixa0_1 -> horizontalHeader() -> resizeSection(i, 120 /
systemSize);
    progMatrixa0_1 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
  }
  progMatrixC0 -> verticalHeader() -> hide();
  progMatrixC0 -> horizontalHeader() -> hide();
  progMatrixa0 -> verticalHeader() -> hide();
  progMatrixa0 -> horizontalHeader() -> hide();
  progMatrixd0 -> verticalHeader() -> hide();
  progMatrixd0 -> horizontalHeader() -> hide();
  progMatrixa0_1 -> verticalHeader() -> hide();
  progMatrixa0_1 -> horizontalHeader() -> hide();

  QVBoxLayout * mainProgDialog = new QVBoxLayout;
  QGridLayout * progDialogGrid = new QGridLayout;
  QHBoxLayout * progDialogUpperLayout = new QHBoxLayout;
  QHBoxLayout * progDialogLowerLayout = new QHBoxLayout;
```

```
  QGroupBox * progDialogGroup = new QGroupBox("Program control",
this);
  QLabel * progDialogSelectLabel = new QLabel("Type of program
control:");
  progDialogSelectLabel -> setAlignment(Qt::AlignRight |
Qt::AlignCenter);

  QLabel * progGroupX0 = new QLabel("C0:", this);
  QLabel * progGroupa0 = new QLabel("a*:", this);
  QLabel * progGroupa0_1 = new QLabel("a*:", this);
  progGroupd0 = new QLabel("d0:", this);

  QLabel * progGroupTrig = new QLabel("func:", this);
  QLabel * progGroupOmega = new QLabel("Omega:", this);
  QLabel * progGroupK = new QLabel("k:", this);
  // Group matrixes and forms in layouts
  QLabel * progLabel11 = new QLabel(" + ", this);
  QLabel * progLabel11_1 = new QLabel(" + ", this);
  QLabel * progLabel13 = new QLabel(" + ( ", this);
  QLabel * progLabel15 = new QLabel(" ) * ", this);
  QLabel * progLabel17 = new QLabel(" ( ", this);
  QLabel * progLabel19 = new QLabel(" * t ^ ", this);
  QLabel * progLabel111 = new QLabel(" ) ", this);
  progGroupX0 -> setAlignment(Qt::AlignCenter);
  progGroupa0 -> setAlignment(Qt::AlignCenter);
  progGroupd0 -> setAlignment(Qt::AlignCenter);
  progGroupTrig -> setAlignment(Qt::AlignCenter);
  progGroupOmega -> setAlignment(Qt::AlignCenter);
  progGroupK -> setAlignment(Qt::AlignCenter);

  QHBoxLayout * progDialogCenterHLayout = new QHBoxLayout;
  QVBoxLayout * progDialogCenterVLayout[14];
  for (int vstep = 0; vstep < 14; vstep++) {
      progDialogCenterVLayout[vstep] = new QVBoxLayout;
  }

  progDialogCenterVLayout[0] -> addWidget(progGroupX0);
  progDialogCenterVLayout[0] -> addWidget(progMatrixC0);
  progDialogCenterVLayout[0] -> addSpacing(20);
  progDialogCenterVLayout[1] -> addWidget(progLabel11);
  progDialogCenterVLayout[2] -> addWidget(progGroupa0);
  progDialogCenterVLayout[2] -> addWidget(progMatrixa0);
  progDialogCenterVLayout[2] -> addSpacing(20);
  progDialogCenterVLayout[3] -> addWidget(progLabel13);
  progDialogCenterVLayout[4] -> addWidget(progGroupd0);
  progDialogCenterVLayout[4] -> addWidget(progMatrixd0);
  progDialogCenterVLayout[4] -> addSpacing(20);
  progDialogCenterVLayout[5] -> addWidget(progLabel11_1);
  progDialogCenterVLayout[6] -> addWidget(progGroupa0_1);
  progDialogCenterVLayout[6] -> addWidget(progMatrixa0_1);
  progDialogCenterVLayout[6] -> addSpacing(20);
  progDialogCenterVLayout[7] -> addWidget(progLabel15);
  progDialogCenterVLayout[8] -> addSpacing(60);
  progDialogCenterVLayout[8] -> addWidget(progGroupTrig);
  progDialogCenterVLayout[8] -> addWidget(progSelectFunc);
  progDialogCenterVLayout[8] -> addSpacing(80);
  progDialogCenterVLayout[9] -> addWidget(progLabel17);
  progDialogCenterVLayout[10] -> addSpacing(60);
```

```
  progDialogCenterVLayout[10] -> addWidget(progGroupOmega);
  progDialogCenterVLayout[10] -> addWidget(progOmega);
  progDialogCenterVLayout[10] -> addSpacing(80);
  progDialogCenterVLayout[11] -> addWidget(progLabel19);
  progDialogCenterVLayout[12] -> addSpacing(40);
  progDialogCenterVLayout[12] -> addWidget(progGroupK);
  progDialogCenterVLayout[12] -> addWidget(progK);
  progDialogCenterVLayout[12] -> addSpacing(100);
  progDialogCenterVLayout[13] -> addWidget(progLabel111);

  for (int vset = 0; vset < 14; vset++) {
      progDialogCenterHLayout ->
addLayout(progDialogCenterVLayout[vset]);
  }

  progDialogGroup -> setLayout(progDialogCenterHLayout);
  progDialogUpperLayout -> addWidget(progDialogSelectLabel);
  progDialogUpperLayout -> addWidget(progDialogSelect);
  progDialogLowerLayout -> addSpacing(250);
  progDialogLowerLayout -> addWidget(progDialogProceed);
  progDialogLowerLayout -> addWidget(progDialogCancel);
  progDialogLowerLayout -> addSpacing(250);
  mainProgDialog -> addLayout(progDialogUpperLayout);
  mainProgDialog -> addWidget(progDialogGroup);
  mainProgDialog -> addLayout(progDialogLowerLayout);
  progDialog -> setLayout(mainProgDialog);
  progCurrentDialogSelect = progDialogSelect -> currentIndex();
  progSelect(progCurrentDialogSelect);
}

void DynamicSystem::progDialogResize() {
  C0_prog.setSize(systemSize, 1);
  a0.setSize(systemSize, 1);
  a0_1.setSize(systemSize, 1);
  d0.setSize(systemSize, 1);
  Ct.setSize(systemSize, 1);

  progMatrixC0 -> setRowCount(systemSize);
  progMatrixa0 -> setRowCount(systemSize);
  progMatrixd0 -> setRowCount(systemSize);
  progMatrixa0_1 -> setRowCount(systemSize);

  int fSize = 8;
  if (systemSize > 4) {
    fSize = 7;
    if (systemSize > 6) {
      fSize = 5;
      if (systemSize > 8) {
        fSize = 4;
      }
    }
  } else {
    fSize = 8;
  }

  QFont matrixFont("Tahoma", fSize);
  progMatrixC0 -> setFont(matrixFont);
  progMatrixa0 -> setFont(matrixFont);
```

```
  progMatrixd0 -> setFont(matrixFont);
  progMatrixa0_1 -> setFont(matrixFont);
  for (int i = 0; i < systemSize; i++) {
    progMatrixC0 -> horizontalHeader() -> resizeSection(i, 40);
    progMatrixC0 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
    progMatrixa0 -> horizontalHeader() -> resizeSection(i, 40);
    progMatrixa0 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
    progMatrixd0 -> horizontalHeader() -> resizeSection(i, 40);
    progMatrixd0 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
    progMatrixa0_1 -> horizontalHeader() -> resizeSection(i, 40);
    progMatrixa0_1 -> verticalHeader() -> resizeSection(i, 120 /
systemSize);
  }
}

void DynamicSystem::progProceed() {
  bool ok1 = true;
  bool ok2 = true;
  progOmega -> text().toDouble( & ok1);
  progK -> text().toDouble( & ok2);

  auto isValidMatrixC0 = pharseMatrix(systemSize, 1, progMatrixC0) ==
0;
  auto isValidMatrixa0 = pharseMatrix(systemSize, 1, progMatrixa0) ==
0;
  auto isValidMatrixd0 = pharseMatrix(systemSize, 1, progMatrixd0) ==
0;
  auto isValidMatrix0_1 = pharseMatrix(systemSize, 1, progMatrixa0_1)
== 0;

  if (isValidMatrixC0 && isValidMatrixa0 && isValidMatrixd0 &&
isValidMatrix0_1) {
    if (ok1 == true)
      if (ok2 == true) {
        QString sTmp;

        for (int i = 0; i < systemSize; i++) {
          switch (progDialogSelect -> currentIndex()) {
          case 0:
            C0.setElement(i, 0, progMatrixd0 -> item(i, 0) ->
text().toDouble());
            a0.setElement(i, 0, progMatrixa0_1 -> item(i, 0) ->
text().toDouble());
            break;
          default:
            C0.setElement(i, 0, progMatrixC0 -> item(i, 0) ->
text().toDouble());
            a0.setElement(i, 0, progMatrixa0_1 -> item(i, 0) ->
text().toDouble());
            d0.setElement(i, 0, progMatrixd0 -> item(i, 0) ->
text().toDouble());
            break;
          }

          sTmp.setNum(C0.getElement(i, 0));
```

```
          matrixC0 -> item(i, 0) -> setText(sTmp);
        }

        omega = progOmega -> text().toDouble();
        powK = progK -> text().toDouble();
        isProgControl = true;

        progCancel();
        takeNewData();
      }
    else
      QMessageBox::critical(this, "Error", "Field k contains invalid
characters", "Continue");
    else
      QMessageBox::critical(this, "Error", "The omega field contains
invalid characters", "Continue");
  }

}

void DynamicSystem::progCancel() {
  progDialog -> hide();
  progDialog -> close();
}

void DynamicSystem::progSelect(int option) {
  QTableWidgetItem * itemTmp = new QTableWidgetItem("1");
  progCurrentDialogSelect = option;
  QString sTmp;
  double dTmp;
  switch (option) {
  case 0:
    progMatrixC0 -> setEnabled(false);
    progMatrixa0 -> setEnabled(false);
    progMatrixd0 -> setEnabled(true);
    progMatrixa0_1 -> setEnabled(true);
    progSelectFunc -> setEnabled(false);
    progOmega -> setEnabled(false);
    progK -> setEnabled(true);
    progGroupd0 -> setText("C0:");
    break;
  case 1:
    for (int j = 0; j < systemSize; j++) {
      progMatrixd0 -> item(j, 0) -> setText("0.001");
    }
    progMatrixC0 -> setEnabled(true);
    progMatrixa0 -> setEnabled(true);
    progMatrixd0 -> setEnabled(true);
    progMatrixa0_1 -> setEnabled(false);
    progSelectFunc -> setEnabled(false);
    progOmega -> setEnabled(false);
    progK -> setEnabled(true);
    progGroupd0 -> setText("d0:");
    break;
  case 2:
    for (int k = 0; k < systemSize; k++) {
      progMatrixd0 -> item(k, 0) -> setText("0.001");
    }
```

```
    progMatrixC0 -> setEnabled(true);
    progMatrixa0 -> setEnabled(true);
    progMatrixd0 -> setEnabled(true);
    progMatrixa0_1 -> setEnabled(false);
    progSelectFunc -> setEnabled(true);
    progOmega -> setEnabled(true);
    progK -> setEnabled(false);
    progGroupd0 -> setText("d0:");
    break;
  }
}

bool DynamicSystem::progFiNegative() {
  if (isProgControl) {
    matrix firstSum(systemSize, systemSize);
    firstSum.setSingle(systemSize);
    matrix secondSum(A);
    firstSum = firstSum - B;
    firstSum.getInverse();
    secondSum = secondSum - B;

    matrix fi(systemSize, 1);
    matrix fi1(systemSize, 1);
    matrix tmpLin1(C0);
    matrix tmpLin2(C0);
    matrix tmpStep(C0);
    matrix tmpStep1(d0);
    matrix tmpStep2(C0);
    matrix tmpStep3(d0);
    matrix tmpHarm(C0);
    matrix tmpHarm1(d0);
    matrix tmpHarm2(C0);
    matrix tmpHarm3(d0);

    matrix result(systemSize, 1);
    matrix tmp_result1(systemSize, 1);
    matrix tmp_result2(systemSize, systemSize);

    switch (progDialogSelect -> currentIndex()) {
    case 0:
      fi = tmpLin1 * pow(0, powK);
      tmpLin2 = tmpLin2 + a0;
      fi1 = tmpLin2 * pow(1, powK);
      break;
    case 1:
      tmpStep1 = tmpStep1 * pow(0, powK);
      tmpStep = tmpStep + tmpStep1;
      fi = tmpStep;
      tmpStep3 = tmpStep3 * pow(1, powK);
      tmpStep2 = tmpStep2 + a0;
      tmpStep2 = tmpStep2 + tmpStep3;
      fi1 = tmpStep2;
      break;
    case 2:
      switch (progSelectFunc -> currentIndex()) {
      case 0:
        tmpHarm1 = tmpHarm1 * sin(omega * 0);
```

```
          tmpHarm3 = tmpHarm3 * sin(omega * (1));
          break;
        case 1:
          tmpHarm1 = tmpHarm1 * cos(omega * 0);
          tmpHarm3 = tmpHarm3 * cos(omega * (1));
          break;
        case 2:
          tmpHarm1 = tmpHarm1 * tan(omega * 0);
          tmpHarm3 = tmpHarm3 * tan(omega * (1));
          break;
        }
        tmpHarm = tmpHarm + tmpHarm1;
        fi = tmpHarm;
        tmpHarm2 = tmpHarm2 + a0;
        tmpHarm2 = tmpHarm2 + tmpHarm3;
        fi1 = tmpHarm2;
        break;
      }

      tmp_result1.initMatrix(fi1, systemSize, 1);
      tmp_result2.initMatrix(firstSum, systemSize, systemSize);

      tmp_result2 = tmp_result2 * secondSum;
      tmp_result2 = tmp_result2 * fi;

      for (int k = 0; k < systemSize; k++) {
        auto result = tmp_result1.getElement(k, 0) -
tmp_result2.getElement(k, 0);
        if (result < 0) {
          return false;
        }
      }
    }
    return true;
}
```

Файл src/matrix/matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

class matrix
{
private:
    float** array;
    int sizeRow;
    int sizeCol;
public:
    matrix();
    matrix(matrix &m);
    matrix(int row, int col);
    ~matrix();
    void init(float c);
    void initStat(float *p, int size_r, int size_c);
    void initDynam(float **p, int size_r, int size_c);
    void initMatrix(matrix &m, int size_r, int size_c);
    void setElement(int row, int col, float k);
    float getElement(int row, int col);
```

```
    void setSize(int row, int col);
    int getSizeRow();
    int getSizeCol();
    float& operator () (int row, int col);
    float& operator [] (int k);
    matrix& operator = (matrix &m);
    matrix& operator + (matrix &m);
    matrix& operator - (matrix &m);
    matrix& operator * (matrix &m);
    matrix& operator + (float c);
    matrix& operator - (float c);
    matrix& operator * (float c);
    matrix& operator / (float c);
    float getRowSum(int n);
    matrix    getExp(float dt);
    matrix  getInverse();
    matrix    getTranspose();
    matrix  getPow(int p);
    matrix& setSingle(int n);
    bool isNotNegative();
    bool isDiagonalLower();
    bool isProductive();
};
#endif // MATRIX_H
```

Файл src/matrix/matrix.cpp

```
#include "matrix.h"
#include <math.h>
#include <stddef.h>

#define EPS 0.1e-15
#define Max(A, B) (((A) > (B)) ? (A) : (B))
#define Min(A, B) (((A) < (B)) ? (A) : (B))
#define Abs(A) (((A) > 0.) ? (A) : -(A))

// default constructor
matrix::matrix()
{
    array = NULL;
    sizeRow = 0;
    sizeCol = 0;
}

// copy constructor
matrix::matrix(matrix &m)
{
    array = NULL;
    sizeRow = 0;
    sizeCol = 0;

    if(m.sizeRow > 0 && m.sizeCol > 0) {
        sizeRow = m.sizeRow;
        sizeCol = m.sizeCol;
        array = new float * [sizeRow];

        for(int ic = 0; ic < sizeRow; ic++) {
            array[ic] = new float[sizeCol];
```

```
        }

        for(int i = 0; i < sizeRow; i++) {
            for(int j = 0; j < sizeCol; j++) {
                array[i][j] = m.array[i][j];
            }
        }
    }
}

// конструктор с заданием размера
matrix::matrix(int row, int col)
{
    array = NULL;
    sizeRow = 0;
    sizeCol = 0;

    if(row > 0 && col > 0) {
        sizeRow = row;
        sizeCol = col;
        array = new float*[sizeRow];

        for(int i = 0; i < sizeRow; i++) {
            array[i] = new float[sizeCol];
        }
    }
}

// деструктор
matrix::~matrix()
{
    if(array != NULL) {
        for(int i = 0; i < sizeRow; i++) {
             if(array[i] != NULL) {
                 delete [] array[i];
             }
        }
        delete [] array;
    }
}

// перегрузка оператора '='
matrix &matrix::operator = (matrix &m)
{
    if(this != &m) {
        if(sizeRow == 0 && sizeCol == 0) {
            sizeRow = m.sizeRow;
            sizeCol = m.sizeCol;
            array = new float*[sizeRow];

            for(int i = 0; i < sizeRow; i++) {
                array[i] = new float[sizeCol];
            }
        }

        if(sizeRow == m.sizeRow && sizeCol == m.sizeCol)
        {
            for(int i = 0; i < sizeRow; i++)
```

```
                for(int j = 0; j < sizeCol; j++)
                    array[i][j] = m.array[i][j];
            }
        }
        return *this;
}

// перегрузка оператора '()'
float& matrix::operator () (int row, int col)
{
        if(row > 0 && row <= sizeRow && col > 0 && col <= sizeCol) {
            return array[row-1][col-1];
        }
        return array[0][0];
}

// перегрузка оператора '[]'
float& matrix::operator [] (int k)
{
        if(sizeRow==1 && k>0 && k<=sizeCol) {
            return array[0][k-1];
        }

        if(sizeCol == 1 && k > 0 && k <= sizeRow) {
            return array[k-1][0];
        }

        return array[0][0];
}

// получение кол-ва строк
int matrix::getSizeRow()
{
        return sizeRow;
}

// получение кол-ва столбцов
int matrix::getSizeCol()
{
        return sizeCol;
}

// установка размера
void matrix::setSize(int row, int col)
{
        if(row == 0 && col == 0) {
            if(array != NULL) {
                for(int i = 0; i < sizeRow; i++) {
                    if(array[i] != NULL) {
                        delete [] array[i];
                    }
                }

                delete [] array;
                array=NULL;
            }

            sizeRow=0;
```

```
        sizeCol=0;
        return;
    }

    if(row > 0 && col > 0) {
        if(array != NULL) {
            for(int i=0; i<sizeRow; i++) {
                if(array[i]!=NULL) {
                    delete [] array[i];
                }
            }
            delete [] array;
        }

        sizeRow = row;
        sizeCol = col;
        array = new float*[sizeRow];

        for(int i=0; i<sizeRow; i++) {
            array[i] = new float[sizeCol];
        }
    }
}

// сделать единичной
matrix & matrix::setSingle(int n)
{
    if(n > 0) {
        setSize(n, n);
        init(0);

        for(int i = 0; i < n; i++) {
            array[i][i] = 1;
        }
    }
    return *this;
}

// инициализация числом
void matrix::init(float c)
{
    for(int i = 0; i < sizeRow; i++) {
        for(int j = 0; j < sizeCol; j++) {
            array[i][j] = c;
        }
    }
}

// инициализ. статич. массивом
void matrix::initStat(float *p, int size_r, int size_c)
{
    this->setSize(size_r,size_c);
    for(int i = 0; i < sizeRow; i++) {
        for(int j = 0; j < sizeCol; j++) {
            array[i][j] = p[i*size_c+j];
        }
    }
}
```

```
// инициализ. динамич. массивом
void matrix::initDynam(float **p, int size_r, int size_c)
{
    this->setSize(size_r,size_c);
    for(int i = 0; i < sizeRow; i++) {
        for(int j = 0; j < sizeCol; j++) {
            array[i][j] = p[i][j];
        }
    }
}

void matrix::initMatrix(matrix &m, int size_r, int size_c)
{
    this->setSize(size_r, size_c);
    for(int i = 0; i < sizeRow; i++) {
        for(int j = 0; j < sizeCol; j++) {
            array[i][j] = m.array[i][j];
        }
    }
}

void matrix::setElement(int row, int col, float k)
{
    array[row][col] = k;
}

float matrix::getElement(int row, int col)
{
    return array[row][col];
}

// сложение матриц
matrix& matrix::operator + (matrix &m)
{
    if(sizeRow == m.sizeRow && sizeCol == m.sizeCol) {
        for(int i = 0; i < sizeRow; i++) {
            for(int j = 0; j < sizeCol; j++) {
                array[i][j] = array[i][j] + m.array[i][j];
            }
        }
    }

    return *this;
}

// вычитание матриц
matrix& matrix::operator - (matrix &m)
{
    if(sizeRow == m.sizeRow && sizeCol == m.sizeCol)
        for(int i = 0; i < sizeRow; i++)
            for(int j = 0; j < sizeCol; j++)
                array[i][j] = array[i][j] - m.array[i][j];

    return *this;
}

// умножение матриц
```

```
matrix& matrix::operator * (matrix &m)
{
    matrix tmp(sizeRow, m.sizeCol);
    if(sizeCol == m.sizeRow) {
        for(int i=0; i<tmp.sizeRow; i++) {
            for(int j=0; j<tmp.sizeCol; j++) {
                float sum = 0;
                for(int k = 0; k < sizeCol; k++)
                    sum += array[i][k] * m.array[k][j];
                tmp.array[i][j] = sum;
            }
        }
    }

    for(int ir = 0; ir < tmp.sizeRow; ir++) {
        for(int jr = 0; jr < tmp.sizeCol; jr++) {
            array[ir][jr] = tmp.array[ir][jr];
        }
    }

    return *this;
}

//операции со скалярами
matrix& matrix::operator + (float c)
{
    if(sizeRow == sizeCol) {
        for(int i = 0; i < sizeRow; i++) {
            array[i][i] += c;
        }
    }

    return *this;
}

matrix& matrix::operator - (float c)
{
    if(sizeRow == sizeCol) {
        for(int i = 0; i < sizeRow; i++) {
            array[i][i] -= c;
        }
    }

    return *this;
}

matrix& matrix::operator * (float c)
{
    for(int i = 0; i < sizeRow; i++) {
        for(int j = 0; j < sizeCol; j++) {
            array[i][j] = array[i][j] * c;
        }
    }

    return *this;
}

matrix& matrix::operator / (float c)
```

```
{
    for(int i = 0; i < sizeRow; i++) {
        for(int j = 0; j < sizeCol; j++) {
            array[i][j] = array[i][j] / c;
        }
    }

    return *this;
}

// транспонирование
matrix matrix::getTranspose()
{
    matrix tmp(*this);

    for(int i = 0; i < sizeRow; i++) {
        for(int j = 0; j < sizeCol; j++) {
            array[j][i] = tmp.array[i][j];
        }
    }

    return *this;
}

// Получить сумму по строке
float matrix::getRowSum(int n)
{
    float sum = 0;
    for(int i = 0; i < sizeRow; i++) {
        sum += array[n][i];
    }

    return sum;
}

// обратная матрица
matrix matrix::getInverse()
{
    matrix tmp(*this);
    setSingle(sizeRow);

    int n;
    int j = 0;
    int m = 0;
    float prm = 0;
    int tmpSize = sizeRow * 4;
    float A[tmpSize][tmpSize], L[tmpSize][tmpSize];

    for(int it = 0; it < sizeRow; it++) {
        for(int jt = 0; jt < sizeRow; jt++) {
            A[it][jt] = tmp.array[it][jt];
        }
    }

    for(int ia = 0; ia < sizeRow; ia++) {
        for(int ja = 0; ja < sizeRow; ja++) {
            L[ia][ja] = array[ia][ja];
        }
```

```
    }

    do {
        m = j;

        for(int nj = j+1; nj < sizeRow; nj++) {
            if(fabs(A[m][j]) < fabs(A[nj][j])) {
                m = nj;
            }
        }

        for(int n = 0; n < sizeRow; n++) {
            prm = A[m][n];
            A[m][n] = A[j][n];
            A[j][n] = prm;

            prm = A[m][n];
            A[m][n] = A[j][n];
            A[j][n] = prm;
        }

        prm = A[j][j];
        for(int np = 0; np < sizeRow; np++) {
            A[j][np] = float(A[j][np] / prm);
            L[j][np] = float(L[j][np] / prm);
        }

        for(int in=0; in<=sizeRow; in++) {
            if (in == j) {
                continue;
            }

            prm = A[in][j];
            for(int n1 = j; n1 <= sizeRow; n1++) {
                A[in][n1] = A[in][n1] - float(prm * A[j][n1]);
            }

            for(int n2 = 0; n2 <= sizeRow; n2++) {
                L[in][n2] = L[in][n2] - float(prm * L[j][n2]);
            }
        }

        j++;
    } while (j < sizeRow);

    for(int ir = 0; ir < sizeRow; ir++) {
        for(int jr = 0; jr < sizeRow; jr++) {
            array[ir][jr] = L[ir][jr];
        }
    }

    return *this;
}

// проверка на неотрицательность
bool matrix::isNotNegative()
{
    for(int i = 0; i < sizeRow; i++) {
```

```
        for(int j = 0; j < sizeCol; j++) {
            if(array[i][j] < 0) {
                return false;
            }
        }
    }

    return true;
}

bool matrix::isDiagonalLower()
{
    for(int i = 0; i < sizeRow; i++) {
        if(array[i][i] > 1) {
            return false;
        }
    }

    return true;
}

//проверка на продуктивность
bool matrix::isProductive()
{
    matrix tmp(*this);
    matrix E;
    E.setSingle(sizeRow);

    if (!tmp.isNotNegative()) {
        return false;
    }

    if (!tmp.isDiagonalLower()) {
        return false;
    }

    tmp = E - tmp;
    tmp.getInverse();

    for(int i = 0; i < tmp.sizeCol; i++) {
        for(int j = 0; j < tmp.sizeRow;j++) {
            if(tmp.array[i][j] < 0) {
                return false;
            }
        }
    }

    return true;
}

// Возведение в степень
matrix matrix::getPow(int p)
{
    matrix tmp1(*this);
    matrix tmp2(*this);
    matrix tmp3(*this);

    if(p == -1) {
```

```
            getInverse();
        }

        if(p == 0) {
            if(sizeRow = sizeCol) {
                 setSingle(sizeRow);
            }
        }

        if(p > 1) {
            matrix mult(sizeRow, sizeCol);
            for(int i = 1; i < p; i++) {
                if(i != 1) {
                     tmp1 = mult;
                }

                tmp1 = tmp1 * tmp2;
                mult = tmp1;
                tmp1 = tmp2;
            }

            for(int ir = 0; ir < sizeRow; ir++) {
                for(int jr = 0; jr < sizeRow; jr++) {
                    array[ir][jr] = mult.array[ir][jr];
                }
            }
        }

        return *this;
    }

    matrix matrix::getExp(float dt)
    {
        matrix a(*this);
        matrix ea(sizeRow, sizeCol);
        matrix aa(sizeRow, sizeCol);
        matrix wm(sizeRow, sizeCol);
        matrix w(sizeRow, 1);

        double am, em, emi;
        int i, j, k, ii;
        em = 0;
        for(int ia = 0; ia < sizeRow; ia++) {
            for(int ja = 0; ja < sizeRow; ja++) {
                ea.setElement(ia, ja, 0);
                aa.setElement(ia, ja, 0);
                wm.setElement(ia, ja, 0);
                a.setElement(ia, ja, a.getElement(ia, ja)*dt);
                am = Abs(a.getElement(ia, ja));
                em = Max(am, em);
            }

            ea.setElement(ia, ia, 1);
            aa.setElement(ia, ia, 1);
            wm.setElement(ia, ia, 1);
        }

        emi = 1;
```

```
    ii = 0;
    while(emi > EPS) {
        ii++;
        if(ii >= 40) {
            break;
        }

        emi = 0;
        for(int jb = 0; jb < sizeRow; jb++) {
            for(int ib = 0; ib < sizeRow; ib++) {
                w.setElement(ib, 0, wm.getElement(ib, jb));
            }

            for(int ic = 0; ic < sizeRow; ic++) {
                wm.setElement(ic, jb, 0);
                for(int k = 0; k < sizeRow; k++) {
                    wm.setElement(ic, jb, wm.getElement(ic, jb) +
(a.getElement(ic, k) * w.getElement(k, 0)));
                }
            }
        }

        for(int id = 0; id < sizeRow; id++) {
            for(int jd = 0; jd < sizeRow; jd++) {
                wm.setElement(id, jd, wm.getElement(id, jd) /
(double)ii);
                ea.setElement(id, jd, ea.getElement(id, jd) +
wm.getElement(id, jd));
                aa.setElement(id, jd, aa.getElement(id, jd) +
(wm.getElement(id, jd) / (double)(ii + 1)));
                am = Abs(wm.getElement(id, jd));
                emi = Max(am, emi);
            }
        }

    }

    for(int it = 0; it < sizeRow; it++) {
        for(int jt = 0; jt < sizeRow; jt++) {
            array[it][jt] = ea.array[it][jt];
        }
    }

    return *this;
}
```

Файл src/widget/widget.h

```
#ifndef GLWIDGET_H
#define GLWIDGET_H

#include <QGLWidget>

class GLWidget: public QGLWidget {
  Q_OBJECT

  public:
    GLWidget(QWidget * parent = 0);
```

```
  ~GLWidget();
  int tX0;
  int tX1;
  int tY0;
  int tY1;
  int systemSize;
  float stepIndex;
  float ** pArray;
  float ** pArray_old;
  bool isGraphBuild;
  bool isSavedOld;
  void setLines(int cx0, int cx1, int cy0, int cy1);
  private:
    int curT;
  void drawInfoLinesX();
  void drawInfoLinesY();
  void drawPointXst(float x, float y, float R, float G, float B);
  void drawPointX(float x, float y, float R, float G, float B);
  void drawLineX(float x_old, float y_old, float x, float y, float R,
float G, float B);
  void drawPointXst_old(float x, float y, float R, float G, float B);
  void drawPointX_old(float x, float y, float R, float G, float B);
  void drawLineX_old(float x_old, float y_old, float x, float y, float
R, float G, float B);
  protected:
    void initializeGL();
  void paintGL();

};

#endif // GLWIDGET_H
```

Файл src/widget/widget.cpp

```
#include <QtGui>
#include <QtOpenGL>
#include <math.h>
#include "glwidget.h"
#include "dynamicSystem.h"

GLWidget::GLWidget(QWidget *parent)
    : QGLWidget(parent)
{
    tX0 = 0;
    tX1 = 5;
    tY0 = 0;
    tY1 = 5;
    isGraphBuild = true;
    isSavedOld = false;
}

GLWidget::~GLWidget()
{
}

void GLWidget::initializeGL()
{
```

```
    qglClearColor(QColor(255, 255, 255, 0));
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, 100, 0, 100, -1, 1);
    qglColor(Qt::white);
}

void GLWidget::paintGL()
{
    if(isGraphBuild == true)
        glClear(GL_COLOR_BUFFER_BIT);
    if(isGraphBuild == true)
    {
        drawInfoLinesX();
        drawInfoLinesY();
    }
    if(isGraphBuild == false)
    {
        int step = 0; /**/
        for(float i=0;i<=tX1-tX0;i+=stepIndex)
        {
            for(int ix=0;ix<systemSize;ix++)
            {
                if(stepIndex < 1)
                {
                    // Нанесение линий в непрерывном случае системы
                    if(i == 0)
                        drawPointX(i*100/(tX1-tX0), (pArray[step][ix]-
tY0)*100/(tY1-tY0), 1.0, 0.0, 0.0);
                    else
                        drawLineX((i-stepIndex)*100/(tX1-tX0),
(pArray[step-1][ix]-tY0)*100/(tY1-tY0), i*100/(tX1-tX0),
(pArray[step][ix]-tY0)*100/(tY1-tY0), 1.0, 0.0, 0.0);
                }
                else
                {
                    // Нанесение точек в дискретном случае системы
                    drawPointX(i*100/(tX1-tX0), (pArray[step][ix]-
tY0)*100/(tY1-tY0), 1.0, 0.0, 0.0);
                }
                drawPointXst(i*100/(tX1-tX0),
(pArray[step][ix+systemSize]-tY0)*100/(tY1-tY0), 1.0, 0.0, 0.0);
            }
            if(isSavedOld == true)
            {
                for(int ix_old=0;ix_old<systemSize;ix_old++)
                {
                    if(stepIndex < 1)
                    {
                        if(i == 0)
                            drawPointX_old(i*100/(tX1-tX0),
(pArray_old[step][ix_old]-tY0)*100/(tY1-tY0), 0.0, 0.0, 1.0);
                        else
                            drawLineX_old((i-stepIndex)*100/(tX1-tX0),
(pArray_old[step-1][ix_old]-tY0)*100/(tY1-tY0), i*100/(tX1-tX0),
(pArray_old[step][ix_old]-tY0)*100/(tY1-tY0), 0.0, 0.0, 1.0);

                    }
```

```
                    else
                    {
                        drawPointX_old(i*100/(tX1-tX0),
(pArray_old[step][ix_old]-tY0)*100/(tY1-tY0), 0.0, 0.0, 1.0);
                    }
                    drawPointXst_old(i*100/(tX1-tX0),
(pArray_old[step][ix_old+systemSize]-tY0)*100/(tY1-tY0), 0.0, 0.0,
1.0);
                }
            }
            step++;
        }
    }
}

// Установка необходимого масштаба
void GLWidget::setLines(int cx0, int cx1, int cy0, int cy1)
{
    tX0 = cx0;
    tX1 = cx1;
    tY0 = cy0;
    tY1 = cy1;
    updateGL();
}

// Внутренняя функция отрисовки линий
void GLWidget::drawInfoLinesX()
{
    int fsize = 6;
    for(int i=tX0;i<=tX1;i++)
    {
        glLineWidth(1);
        glBegin(GL_LINES);
        glColor3f(0.9, 0.9, 0.9);
        glVertex2i((i-tX0)*100/(tX1-tX0), 0);
        glVertex2i((i-tX0)*100/(tX1-tX0), 100);
        glEnd();
        glLineWidth(1);
        glBegin(GL_LINES);
        glColor3f(0.0, 0.0, 0.0);
        glVertex2i((i-tX0)*100/(tX1-tX0), 0);
        glVertex2i((i-tX0)*100/(tX1-tX0), 3);
        glEnd();
        QString sTmp;
        QFont fTmp("Tahoma", fsize);
        sTmp.setNum(i, 10);
        renderText((i-tX0)*100/(tX1-tX0)+1.0, 4.0, 0.0, sTmp, fTmp);
    }
}

void GLWidget::drawInfoLinesY()
{
    int fsize = 6;
    for(int i=tY0;i<=tY1;i++)
    {
        glLineWidth(1);
        glBegin(GL_LINES);
        glColor3f(0.9, 0.9, 0.9);
```

```
            glVertex2i(0, (i-tY0)*100/(tY1-tY0));
            glVertex2i(100, (i-tY0)*100/(tY1-tY0));
            glEnd();
            glLineWidth(1);
            glBegin(GL_LINES);
            glColor3f(0.0, 0.0, 0.0);
            glVertex2i(0, (i-tY0)*100/(tY1-tY0));
            glVertex2i(2, (i-tY0)*100/(tY1-tY0));
            glEnd();
            if(i != tY0)
            {
                QString sTmp;
                QFont fTmp("Tahoma", fsize);
                sTmp.setNum(i, 10);
                renderText(3.0, (i-tY0)*100/(tY1-tY0)+2.0, 0.0, sTmp,
fTmp);
            }
        }
}

void GLWidget::drawPointXst(float x, float y, float R, float G, float
B)
{
    // В случае, если система непрерывная
    if(stepIndex < 1)
    {
        glColor3f(R, G, B);
        glLineWidth(1);
        glBegin(GL_POINTS);
        glVertex2f(x, y);
        glEnd();
    }
    // В случае, если система дискретная
    else
    {
        glColor3f(R, G, B);
        glLineWidth(1);
        glBegin(GL_LINES);
        glVertex2i(x+1, y+2);
        glVertex2i(x-1, y-2);
        glVertex2i(x+1, y-2);
        glVertex2i(x-1, y+2);
        glEnd();
    }
}

void GLWidget::drawPointX(float x, float y, float R, float G, float B)
{
    if(stepIndex < 1)
    {
        glColor3f(R, G, B);
        glLineWidth(1);
        glBegin(GL_POINTS);
        glVertex2f(x, y);
        glEnd();
    }
    else
    {
```

```
            glColor3f(R, G, B);
            glBegin(GL_POLYGON);
            glVertex2i(x, y+2);
            glVertex2i(x-1, y);
            glVertex2i(x, y-2);
            glVertex2i(x+1, y);
            glEnd();
        }
    }

    void GLWidget::drawLineX(float x_old, float y_old, float x, float y,
    float R, float G, float B)
    {
        glColor3f(R, G, B);
        glLineWidth(1);
        glBegin(GL_LINES);
        glVertex2f(x_old, y_old);
        glVertex2f(x, y);
        glEnd();
    }

    void GLWidget::drawLineX_old(float x_old, float y_old, float x, float
    y, float R, float G, float B)
    {
        glColor3f(R, G, B);
        glLineWidth(2);
        glBegin(GL_LINES);
        glVertex2f(x_old, y_old);
        glVertex2f(x, y);
        glEnd();
    }

    void GLWidget::drawPointXst_old(float x, float y, float R, float G,
    float B)
    {
        // В случае, если система непрерывная
        if(stepIndex < 1)
        {
            glColor3f(R, G, B);
            glLineWidth(2);
            glBegin(GL_POINTS);
            glVertex2f(x, y);
            glEnd();
        }
        // В случае, если система дискретная
        else
        {
            glColor3f(R, G, B);
            glLineWidth(1);
            glBegin(GL_LINES);
            glVertex2i(x+1, y);
            glVertex2i(x-1, y);
            glVertex2i(x, y+2);
            glVertex2i(x, y-2);
            glEnd();
        }
    }
```

```
void GLWidget::drawPointX_old(float x, float y, float R, float G,
float B)
{
    glColor3f(R, G, B);
    glBegin(GL_POLYGON);
    glVertex2i(x-0.5, y+1);
    glVertex2i(x-0.5, y-1);
    glVertex2i(x+0.5, y-1);
    glVertex2i(x+0.5, y+1);
    glEnd();
}
```

Додаток Б

Вміст файлу README.md для розробленого програмного продукту

dynamic-system

Based on the conducted research, two types of dynamic systems are considered: discrete and continuous. The developed algorithm is aimed at determining and investigating the output characteristics of a complex controlled positive dynamic system in both cases. The result of this work is a software product implemented in the C++ programming language and utilizing the Qt libraries. This product automates the process of obtaining solutions for the tasks of defining and researching output characteristics for both discrete and continuous positive dynamic systems.

Feature

- C++11
- QT 4.7.3
- QT Creator
- GIT
- Single file for each class

Installation

Clone repository

git clone https://github.com/stas-polos/dynamic-system.git

Install tools

You need to install the necessary tools described here

Run program

To start the program, at Qt Creator press F5

Program overview

The choice of the C++ programming language and the Qt library version 4.7.3 for developing this program may be influenced by several factors:

- C++11 Support:
    - The C++11 standard introduces many new features and improvements that facilitate development, making the code more readable and secure. Utilizing C++11 functionality can significantly simplify and enhance the program's development, rendering the code more modern and efficient.

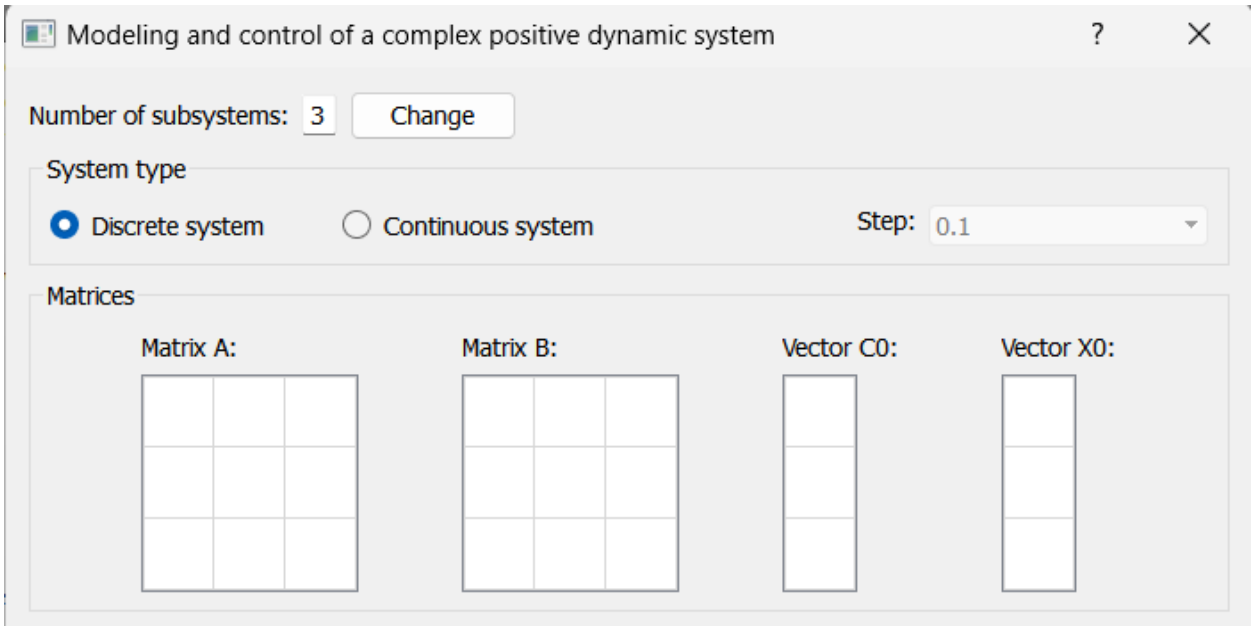
- Qt Library:
  - Qt is a powerful library for developing graphical interfaces and cross-platform applications. Version 4.7.3, while not the latest, is stable and might be chosen for its reliability and extensive capabilities.
  - If there are specific features or capabilities in Qt 4.7.3 that are necessary for the project, and there is no need or possibility to migrate to newer versions, this could be a crucial factor in selecting a particular library version.
- Stability:
  - If stability and compatibility with previous versions were crucial during development, choosing a less recent but already proven version might be justified.

Thus, the selection of C++11 and Qt 4.7.3 for writing the program could be motivated by the project's requirements, including functionality, stability, and the existing experience of the development team in using these technologies.

The choice also considered alternatives such as Python and PyQt, but the decision leaned towards developing in the language that the framework itself was written in. While Python and PyQt were viable options, it was deemed more advantageous to align with the language in which the underlying framework was originally implemented. This decision ensures better integration with the core functionalities of the framework, potentially leveraging existing features and optimizations specific to the chosen programming language, in this case, C++.

UI overview

The user interface (UI) of the program features the following elements:



- Input Fields for Matrices A and B, as well as Vectors C_0 and X_0: Text input areas allowing the user to input values for matrices A and B, as well as vectors C_0 and X_0

- Functionality for Changing Dimensions of Matrices and Vectors:
  - Controls or buttons enabling the user to dynamically change the dimensions (size) of matrices A and B and vectors C0 and X0. This ensures flexibility in handling different system configurations.

- System Type Selection: Discrete and Continuous:
  - A dropdown or selection mechanism allowing the user to choose between two types of systems: discrete and continuous. This selection influences the subsequent calculations and checks performed on the input matrices and vectors.
- Calculation Step Size Selection:
  - Input or selection mechanism for the user to specify the step size for calculations. This parameter determines the granularity of the computations and is especially relevant in dynamic systems.

After entering the values for matrices and vectors, the program performs checks on matrix properties based on the specified system type:

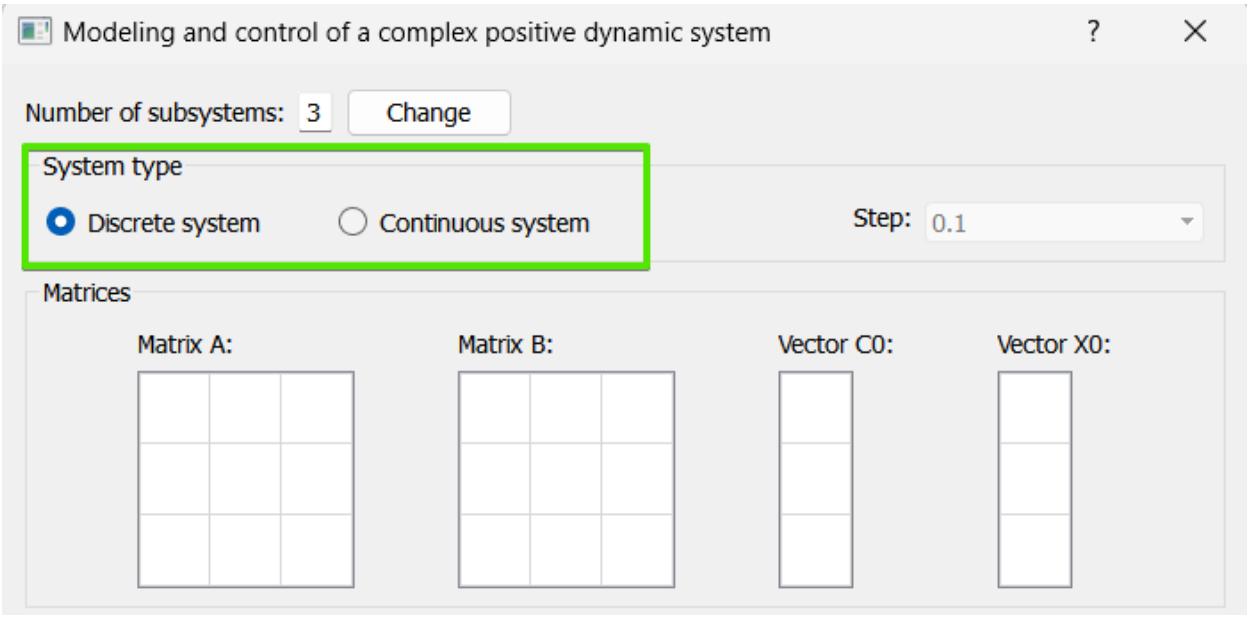
- For Discrete Systems:
  - Check for non-negativity (all elements of the matrix should be non-negative) and productivity. Checks matrices A, B, $(A-B)$, $(I-B)^{-1}(A-B)$

For Continuous Systems:

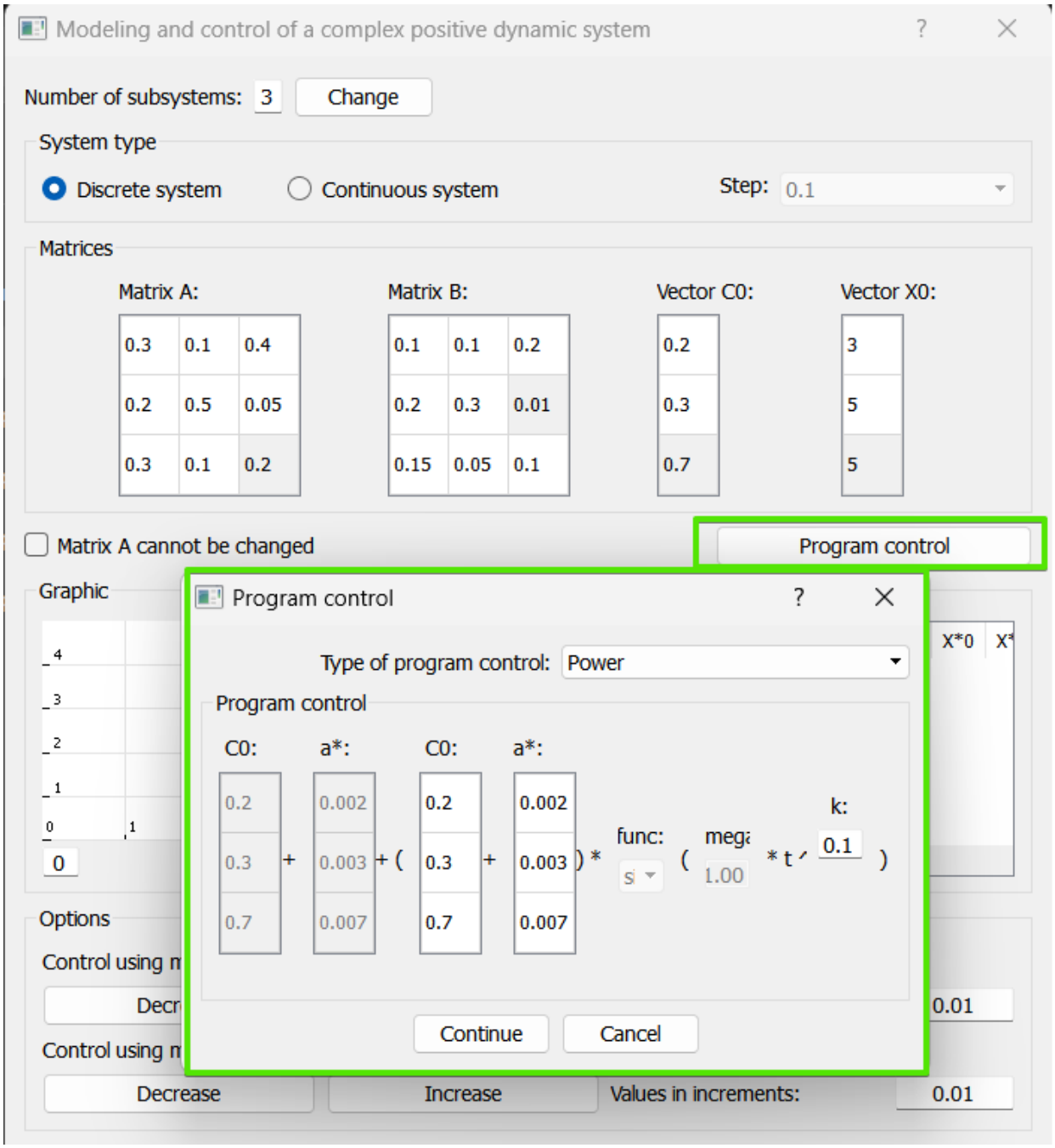
- Check for non-negativity and productivity:
  - Matrices A and B are checked for non-negativity.
  - In the matrix $(I-B)^{-1}(A-I)$, elements on the main diagonal must be negative, and off-diagonal elements must be non-negative.

These checks ensure that the input matrices and vectors adhere to specific properties required for the analysis of discrete and continuous systems, providing a robust and accurate foundation for subsequent calculations in the program.

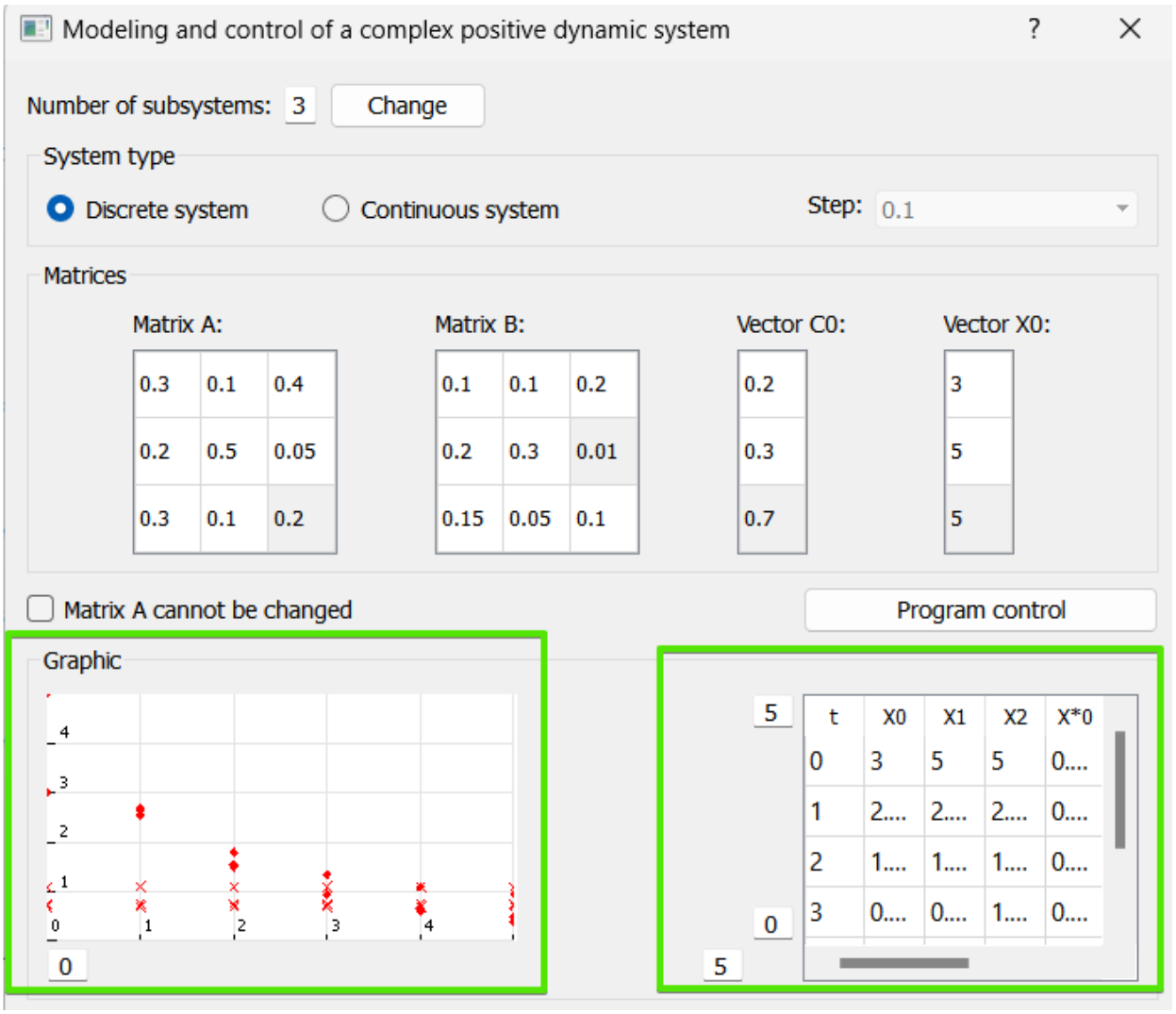
To select the system type, use the following buttons

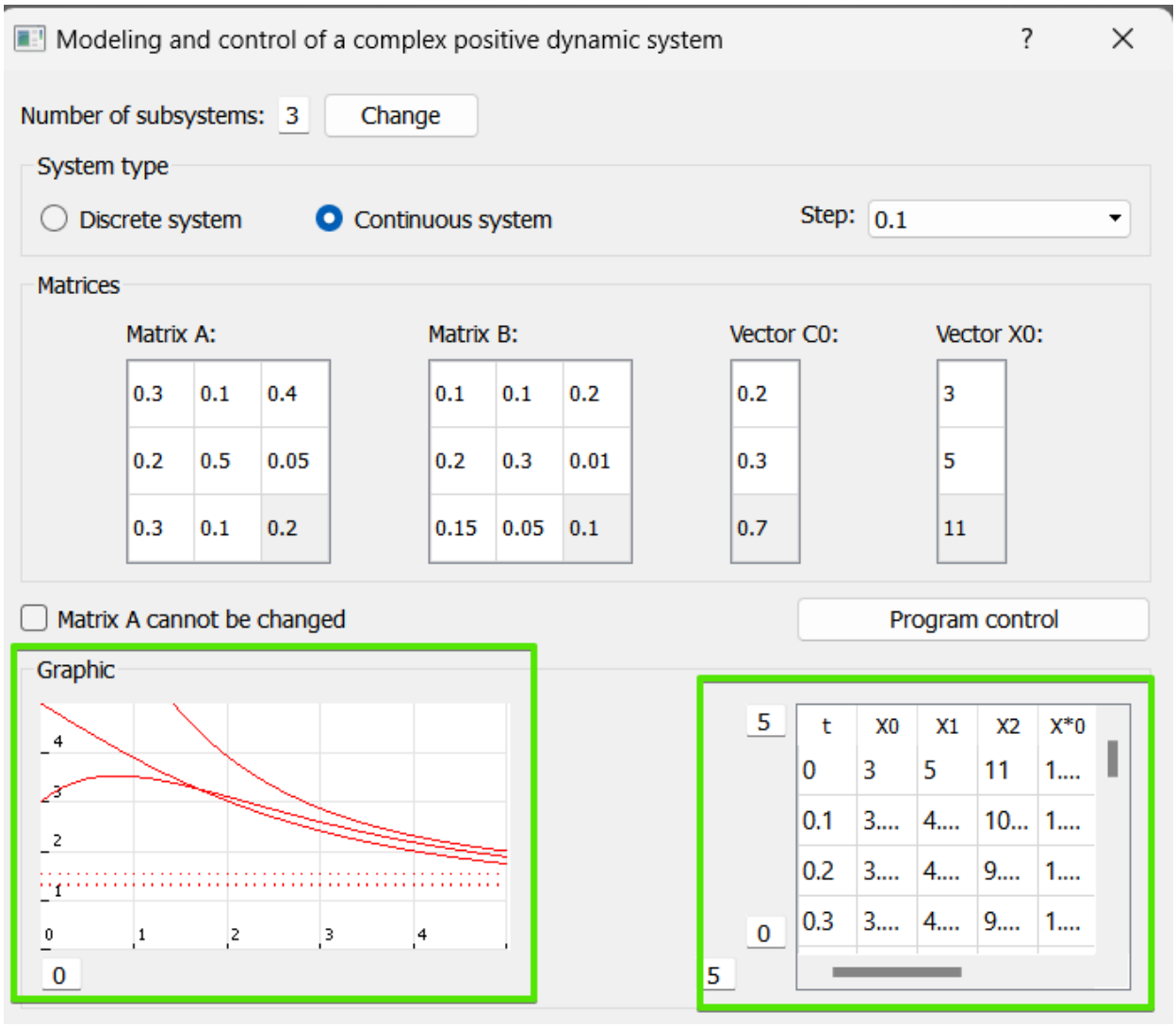


As an option you can choose software control by the system, with the choice of the type of function with which the calculations will be performed, with a choice of functions for each type and approximation (omega).

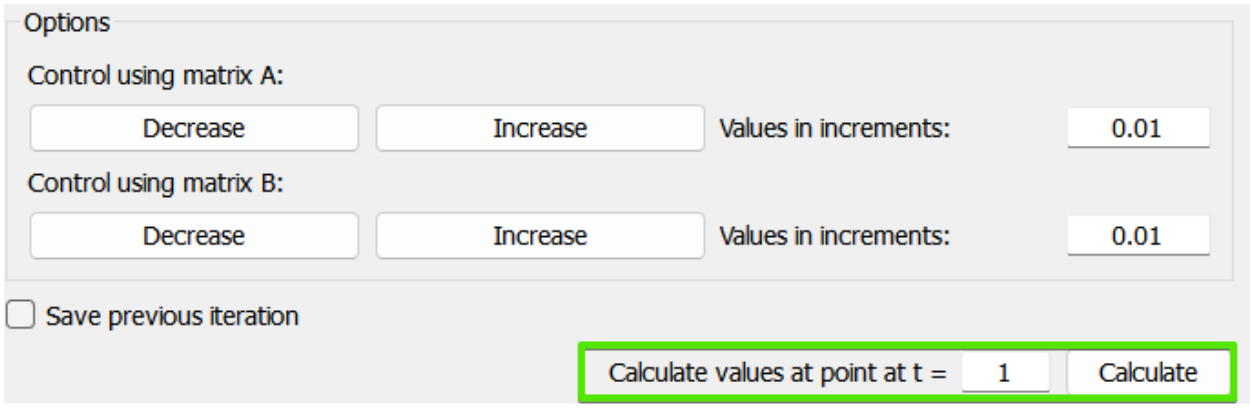


As a result of the solutions, a graphical representation of the solution of the system for the selected type (discrete or continuous), as well as numerical solutions are displayed.


Modeling and control of a complex positive dynamic system
Number of subsystems: 3
Change
System type
Discrete system
Continuous system
Step: 0.1
Matrices
Matrix A:
0.3 0.1 0.4
0.2 0.5 0.05
0.3 0.1 0.2
Matrix B:
0.1 0.1 0.2
0.2 0.3 0.01
0.15 0.05 0.1
Vector C0:
0.2
0.3
0.7
Vector X0:
3
5
5
Matrix A cannot be changed
Program control
Graphic
0
5
0
5
t X0 X1 X2 X*0
0 3 5 5 0....
1 2.... 2.... 2.... 0....
2 1.... 1.... 1.... 0....
3 0.... 0.... 1.... 0....



As additional functionality, matrix management, namely incrementing, decrementing elements of matrices *A* and *B*, as well as calculating the value at point *t*



Файл README.MD із наведеним вмістом розташований за посиланням https://github.com/stas-polos/dynamic-system/blob/master/README.md.

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Полос Станіслав Сергійович

студент 2 курсу магістратури, денної форми навчання, математичного факультету, спеціальності прикладна математика,
адреса електронної пошти polosstas1118@gmail.com,

– підтверджую, що написана мною кваліфікаційна робота магістра на тему «Автоматизація процесу математичного моделювання, аналізу та керування складною детермінованою позитивною динамічною системою» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений;

– заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

– згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи а також на архівування моєї роботи в базі даних цієї системи.

Студент

		Полос С.С.
(дата)	(підпис)	(прізвище, ініціали)

Науковий керівник

		Леонтьєва В.В.
(дата)	(підпис)	(прізвище, ініціали)