МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра прикладної математики і механіки

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«АЛГОРИТМІЗАЦІЯ ПРОЦЕСУ МОДЕЛЮВАННЯ ПОВЕДІНКИ ПОЗИТИВНИХ ДИНАМІЧНИХ СИСТЕМ»**

Виконав(ла): студент(ка) 2 курсу, групи 8.1139

спеціальності 113 прикладна математика
(шифр і назва спеціальності)

освітньої програми прикладна математика
(назва освітньої програми)

О. С. Андрієнко
(ініціали та прізвище)

Керівник доцент кафедри прикладної математики і механіки, 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

2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра прикладної математики і механіки

Рівень вищої освіти магістр

Спеціальність 113 прикладна математика

(шифр і назва)

Освітня програма Прикладна математика

ЗАТВЕРДЖУЮ

Завідувач кафедри прикладної математики і механіки, д.т.н., професор

_____ Грищак В.З.

(підпис)

« 01 » 12 2020 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ(СТУДЕНТЦІ)

Андрієнко Олександру Сергійовичу

(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Алгоритмізація процесу моделювання позитивних динамічних систем

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

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

затверджені наказом ЗНУ від « 29 » 05 2020 року № 576-с

2. Строк подання студентом роботи 14.12.2020

3. Вихідні дані до роботи 1. Розробка алгоритму позитивної динамічної системи
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аніліз об'єкта і апкдметної області дослідження, алгоритмізація моделювання поведінки позитивних динамічних систем, проектування програмного забезпечення

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

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	29.05.2020	виконано
2.	Збір вихідних даних.	01.07.2020	виконано
3.	Обробка методичних та теоретичних	17.07.2020	виконано
	джерел.		
4.	Розробка першого розділу.	05.08.2020	виконано
5.	Розробка другого розділу.	23.09.2020	виконано
6.	Розробка третього розділу	19.11.2020	виконано
7.	Оформлення та нормоконтроль		
	кваліфікаційної роботи.	20.11.2020	виконано
8.	Захист кваліфікаційної роботи.	17.12.2020	виконано

Студент _____ О. С. Андрієнко
(підпис) (ініціали та прізвище)

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

Нормоконтроль пройдено

Нормоконтролер _____ В.В. Леонтьєва
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Алгоритмізація процесу моделювання поведінки позитивних динамічних систем»: 96 с., 10 рис., 1 таб., 21 джерел, 1 додатки.

ДИНАМІЧНА СИСТЕМА, ПОЗИТИВНА ДИНАМІЧНА СИСТЕМА, АЛГОРИТМ, МАТЕМАТИЧНА МОДЕЛЬ, ПРОГРАМА.

Об’єкт дослідження – позитивна динамічна система.

Мета роботи: розробка алгоритму та його реалізування в програмі, визначення й дослідження вихідних характеристик позитивної динамічної системи.

Метод дослідження – аналітичний.

У роботі розглядається математична модель позитивної динамічної системи, описуються основні обмеження, які накладаються на вхідні, та вихідні характеристики та початкові умови моделі, наводиться аналіз моделі та отримуваних за нею результатів, зазначаються можливі керування та керуючі впливи для досліджуваної системи та розглядається їх застосування до розв’язання задачі керування в залежності від поставленої мети задачі.

SUMMARY

Master’s Qualification Thesis «Algorithmization of the process of modeling the behavior of positive dynamic system»: 96 pages, 10 figures, 1 tables, 21 references, 1 supplements.

DYNAMICAL SYSTEM, POSITIVE DYNAMICAL SYSTEM, ALGORITHM, MATHEMATICAL MODEL, PROGRAM.

The object of the study is positive dynamic system.

The aim of the study is development of the algorithm and its implementation in the program, definition and research of initial characteristics of the positive dynamic system.

The methods of research is analytical.

The qualification work considers the mathematical model of a positive dynamical system, describes the main limitations imposed on the input and output characteristics and initial conditions of the model, provides an analysis of the model and its results, indicates possible controls and control effects for the studied system and considers their application to solving the control problem depending on the purpose of the problem.

ЗМІСТ

ВСТУП

Актуальність теми обумовлюється використанням методу математичного моделювання, при описі різноманітних об'єктів реально світу, бо за його допомогою маємо можливість описати поведінку об'єкта дослідження, вивчити його структуру, властивості, закони розвитку і взаємодії з навколишнім світом.

Метою роботи є розробка алгоритму для втілення його в програму з можливістю дослідження позитивної динамічної системи балансного типу. На даний момент, побудова алгоритму складна і потребує багато часу на реалізацію.

Об'єктом дослідження є позитивна динамічна система балансового типу. Такими системами є системи, які описують поведінку складних економічних, екологічних, біологічних та інших об’єктів.

При роботі був розроблений програмний продукт на мові C++ з використанням бібліотек QT. Необхідність створення даної програми, обумовлене складністю алгоритмізування процесу математичного моделювання досліджуваної позитивної динамічної системи, виконання аналізу отримуваних результатів, визначення потрібних дій та здійснення відповідного дій з урахуванням різних цілей дослідника. На даний момент реалізацію цих алгоритмів складно знайти, саме тому моя тема є актуальною і в ній є потреба.

Кваліфікаційна робота включає в себе три розділи.

У першому розділі розглядаються теоретичні аспекти динамічної системи, складної позитивної системи з ентропійним оператором та складної позитивної системи балансованого типу. Також розглядається побудова та схема функціювання позитивної динамічної системи.

У другому розділі проектується, структурування раніше наданої інформації для впровадження алгоритму у програмний продукт, крім цього в цьому розділі буде обґрунтовано вибір мови і середовища програмування.

У третьому розділі буде продемонстрована програмна реалізація алгоритму, її кінцевий вигляд для користувача з детальною інструкцією по використанню створеного продукту. Також будуть представлені приклади роботи програмного продукту.

1 РОЗГЛЯД ОСНОВНИХ ПОНЯТЬ ПРО ДИНАМІЧНУ СИСТЕМУ

Динамічна система — це математична модель будь-якого об'єкта, процесу або явищ, в яких нехтують випадковими відхиленнями будь-якої величини і іншими статичними явищами.

Динамічну систему можна представити, як систему, що володіє станом. У цьому випадку, динамічна система буде описувати динаміку обраного процесу, а саме, динаміку процесу переходу системи з одного стану в інший. Із вище наданого зрозуміло, що динамічна система, характеризується своїм початковим станом і законом, за яким система переходить з початкового стану в інше [1].

Розрізняють системи з дискретним часом і системи з неперервним часом.

У перших системах, які традиційно називаються каскадами, поведінка системи або, траєкторія системи в фазовому просторі описується послідовністю станів. У других системах, які традиційно називаються потоками, стан системи визначено для кожного моменту часу на речовій або комплексної осі. Каскади і потоки є основним предметом розгляду в символічній і топологічної динаміці.

Сучасна теорія динамічних систем, це збірна назва для досліджень, де широко використовуються і поєднуються методи з різних розділів математики, геометрії, алгебри, теорії міри, теорії особливостей, теорії диференціальних форм топології, теорії особливостей і катастроф.

Модель може вважатися корисною, якщо з її допомогою вдається теоретично виявити нові особливості поведінки, залежності і закономірності, які потім підтверджуються експериментально.

Завдяки своїй варіотивності і гнучкості застосування, динамічна система користується популярністю в розрахунках.

Наступним кроком ми розглянемо позитивні динамічні системи

1.1 Математичне моделювання позитивних динамічних систем

У цьому розділі ми розглянемо підкласи позитивної динамічної системи.

Позитивну динамічну систему характеризують вхідні і вихідні об'єкти, які залишаються невід'ємними в часі.

У цьому розділі ми розглянемо декілька підкласів позитивних динамічних систем, а саме:

а) позитивна динамічна система з ентропійним оператором;

б) позитивна динамічна система балансового типу.

1.1.1 Математична модель динамічної системи з ентропійним оператором

Для прикладу, візьмемо динамічну систему в якій є два процеси, детермінований (повільний $x(t) \in \mathbb{R}^n$) і випадкові (швидкий $y(t) \in \mathbb{R}^m$) з різницею у часі релаксації, тобто $T_{slow} \gg T_{fast}$, де T_{fast} — середній час релаксації.

Ця умова означає, що в шкалі "повільного" часу процес (y), в інтервалі $[t, t + \Delta t]$ (де Δt - мала величина) швидко затухає. Його локальний стан ($y^*(t)$), залежить від стану "повільного" процесу (x) в момент часу (t), тобто $y^*(t) = y^*(x(t))$. Ця ситуація повністю відповідає класичним принципом локальної рівноваги.

У кожний фіксований момент часу вектор $y^*(t) = \{y_1^*(t), \dots, y_n^*(t)\}$ має розподіл віртуального або реального ресурсу по (m) скринькам. Нехай, цим ресурсом буде квантован. У цьому процесі кожен ресурсний квант, може випадкова і незалежно від інших потрапляти у скриню (i). Зумовленими

характеристиками цього процесу є зумовлені вірогідності ω_i попадання ресурсного кванта в і-ту скриню, його ємкість S_i і особливості внутрішньої структури.

Вірогідність і ємкість залежать від (x(t)) повільного процесу, тобто:

$$\omega_j = \omega_j(x),\ S_j = S_j(x).$$

Структура скринь передбачає собою структуру комірок, один квант ресурсу може зайняти лише одну комірку.

Таким чином моделі можна характеризувати трьома ентропіями:

а) Ферми:

$$H_F(x,y) = -\sum_{j=1}^{m}\left(y_j \ln {y_j}\Big/_{\omega_j(x)} + \left(S_j(x) - y_j\right)\ln(S_j(x) - y_j\ )\ )\right). \quad (1.1)$$

б) Енштейна:

$$H_F(x,y) = -\sum_{j=1}^{m}\left(y_j \ln {y_j}\Big/_{\omega_j(x)} - \left(S_j(x) + y_j\right)\ln(S_j(x) + y_j\ )\ )\right). \quad (1.2)$$

в) Больцмана:

$$H_F(x,y) = -\sum_{j=1}^{m} y_j \ln {y_j}\Big/_{e\omega_j(x)}\ . \quad (1.3)$$

Зазвичай у задачах можна побачити обмеження на розподіл (y(t)). Вони зазначають допустиму множину D(x), яка в загальному випадку має наступний вигляд:

$$D(x) = \{y : \phi_{r+1+s}(x, y) \leq 0, k = 1, \dots, r; s = 1, \dots, l\}. \quad (1.4)$$

Рівність стану і умовна максимальна ентропія виглядає наступним чином:

$$y^*\big(x(t)\big) = arc\max(H(x(t), y) | \, y \in D\big(x(t)\big)). \quad (1.5)$$

Оператор який визначає рівність (1.5), будемо називати ентропіним оператором. Із цього визначення можна зробити висновок, що ентропійний оператор (y(x(t))) відноситься до класу позитивних операторів.

Також, треба відокремити загальну умову позитивної динамічної системи, а саме, якщо її траєкторія $x(t) \geq 0$ для всіх $t \geq 0$. Тепер розглянемо важливий клас позитивних динамічних систем математична модель яких будується термінами темпу росту:

$$1/x \otimes dx/dt = g\big(x, y^*(x)\big), \quad (1.6)$$

де $\otimes$ — координатор оператора множення

У рівнянні (1.6) функція $g(x, y^*)$ характеризує темп росту $g \in R^n$, а також виконує умову $g(0, 0) \neq \infty$ і для всіх $x(0) > 0$ векторів стану $x(t) \geq 0$.

Далі розглянемо функцію росту $g(x, y^*)$ наступного типу:

$$g(x) = a - x \otimes Q(x), a, Q(x, y^*(x)) \in R_+^n). \quad (1.7)$$

У випадку з функцією (1.7) позитивна динамічна система буде описана наступним диференційним рівнянням:

$$dx/_{dt} = x \otimes (a - x \otimes Q(x, y^*(x))). \quad (1.8)$$

Розглянемо випадок, коли:

$$Q\big(x, y^*(x)\big) = Py^*(x), \quad (1.9)$$

де (m * n) — це матриця $P = [p_{ij} \geq 0]$. При підстановці цього виразу в рівняння (1.8), отримуємо рівняння позитивної динамічної системи з ентропійним оператором який має вигляд:

$$dx/_{dt} = x \otimes (a - x \otimes Py(x)), \quad (1.10)$$

де y(x) є ентропійним оператором (1.5)

Розглянемо ентропійний оператор (1.5) для котрого допустима множина (1.4) описується системою рівності:

$$D(x) = \{y : \phi_k(x, y) = 0, k = 1, \dots, r; y \in R^m, x \in R^n\}. \quad (1.11)$$

Оператори (1.5), (1.11), будемо називати умовно оптимальними ентропійними операторами.

Продемонструємо на прикладі раніше наведеної моделі (1.3), загальний підхід дослідження даного класу ентропійних операторів, який заснований на моделі Лагранжа, а також зробимо зауваження стосовно (1.1)(1.2).

Розглянемо підмножину $X \in R^n$ і точку $x \in X$. Введемо функцію Лагранжа:

$$L(x, y, \lambda) = H_B(x, y) - \sum_{k=1}^{r} \lambda_k \phi_k(x, y). \quad (1.12)$$

Оптимальні умови для (1.3) представлені в наступній формі:

$$\psi_j(x,y,\lambda) = \ln {}^{\omega_j(\mathrm{x})}\Big/_{y_j} - \sum_{k=1}^{r} \lambda_k \, {}^{\partial \phi_k(y,x)}\Big/_{\partial y_j} = 0, j = 1, \dots, m; \quad (1.13)$$

$$\phi_k(y,x) = 0, k = 1, \dots r. \quad (1.14)$$

Аналогічні результати будуть для (1.1) і (1.2), якщо ввести додаткову умову сепарабельності функції ϕ_k:

$$\phi_k(y,x) = \sum_l \phi_{kl}(y,x). \quad (1.15)$$

Таким чином були розглянуті основні поняття математичної моделі динамічної системи з ентропійним оператором, наступним кроком розглянемо другий підклас позитивної динамічної системи, а саме математичну модель динамічної системи балансового типу.

1.1.2 Математична модель динамічної системи балансового типу

Розглянемо математичну модель динамічної системи балансового типу із [2, 4, 5] знаємо, що для даного підкласу характерно:

а) балансові співвідношення в моделях, що описують поведінку складних систем;

б) одержання невід’ємних розв’язків на нескінченному інтервалі часу при невід’ємності вхідних параметрів і початкових станів системи, забезпечує наявність у моделях обмежень на відповідні матриці коефіцієнтів.

Візьмемо для прикладу, математичну модель позитивної динамічної системи балансового типу [2, 3 ,5, 6]:

$$(I-B)X_{t+1}=(A-B)X_t+C_t, \tag{1.16}$$

де t — час, X_t, X_{t+1} — n-мірні вектори стану позитивної системи, C_t — n-мірні вектори керування системи, A = $(a_{ij})_{n*n}$, B = $(b_{ij})_{n*n}$ — сталі матриці коефіцієнтів, n — кількість агрегованих взаємозв'язаних між собою підсистем досліджуваної складної системи.

Математична модель, яка була описана системою (1.16) є розімкненою, тобто вхідна характеристика X_{t+1} при своїй зміні не впливає на вхідні характеристики X_t і C_t позитивної динамічної системи балансного типу. Ця модель залежить від значень вхідних характеристик (минулі і поточні) і не може врівноважити зміни вхідної характеристики системи, яка є неминучою [2, 6].

Модель позитивної динамічної системи балансового типу, яка описується системою (1.16), може бути зведена до моделі вигляду:

$$X_{t+1}=(I-B)^{-1}(A-B)X_t+(I-B)^{-1}C_t, \tag{1.17}$$

або

$$X_{t+1}=\tilde{A}X_t+D_t. \tag{1.18}$$

Згідно з [7, 8], для моделі рівняння (1.17), можуть бути справедливими наступні теорема і лема.

Теорема 1. Модель (1.18) продуктивна тоді і тільки тоді, коли спектральний радіус матриці $\rho(\tilde{A}) < 1$.

Лема 1. Якщо модуль (1.18) матриці A і $\tilde{A}$ є продуктивними, то для будь-якого вектора $D_t \geq 0$ будуть продуктивними і матриці $B, (A - B)$.

З огляду на те, що система рівнянь, яка описується системою (1.17) описує реальні процеси, в яких суттєве позицію мають лише невід'ємні значення векторів стану, матриці $A, B, (A - B), (I - B)^{-1}(A - B)$ можуть бути продуктивними.

Згідно з [9] невід'ємну матрицю A можна назвати продуктивною, якщо існує невід'ємний вектор X, що:

$$X - AX \geq 0. \tag{1.19}$$

Критерієм продуктивності для матриці A, яка описана системою (1.17), є окремий випадок умови Брауера-Соло, що формулюється в термінах сум коефіцієнтів матриці. Для продуктивності матриці A достатнє виконання одного із стверджень:

$$\sum_{j=1}^{n} a_{ij} < 1, i = \overline{1,n}; \tag{1.20}$$

$$\sum_{j=1}^{n} a_{ij} < 1, j = \overline{1,n}. \tag{1.21}$$

Данні умови продуктивності матриці A поширюються також на матриці $B, (A - B), (I - B)^{-1}(A - B)$.

Для забезпечення невід'ємності одержуваних розв'язків X_t, до моделі яка описана системою (1.17) або (1.18) діють умови невід'ємності й продуктивності

матриць коефіцієнтів $A, B, (A-B), (I-B)^{-1}(A-B)$, що в свою чергу забезпечує позитивність системи й асимптотичну стійкість розв'язків [4, 6].

Якщо всі умови виконуються, то система (1.17) є асимптотично стійкою, а при $t \to \infty$ отримана модель збігається до рівноважного значення X^*, яке визначається при $\Delta X_t = X_{t+1} - X_t = \Theta$, де $\Theta = (\theta_{ij})_{n*n}, \theta_{ij} = 0$ для $i, j = \overline{1,n}$ і $C_t = C^0$ – const.

На прикладі проведеного аналізу [7] динамічних властивостей системи (1.17) є можливість описати не тільки якісний характер траєкторій руху системи, але й розрахувати значення змінних моделі, тобто визначити стан системи в будь-який момент часу.

Припустимо, що у моделі (1.17) значення $C(t) = C^0 > 0$ $C^0 = (C_0^1, C_0^2, \dots, C_0^n)$ є сталими, тоді вона прийме вигляд:

$$X_{t+1} = (I-B)^{-1}(A-B)X_t + (I-B)^{-1}C_t, \qquad (1.22)$$

або

$$X_{t+1} = \tilde{A}X_t + \tilde{B}C^0. \qquad (1.23)$$

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

Точки рівноваги для системи (1.22) знаходяться $\Delta X_t = X_{t+1} - X_t = \Theta$, та приймають вигляд:

$$X^* = (I-B)^{-1}C^0. \qquad (1.24)$$

Для кращого розуміння знаходження розв’язку системи рівнянь (1.23) згідно з [10, 11] приймають вигляд:

$$X_{t+1} = \widetilde{A^{t+1}}X_0 + (\widetilde{A^t} + \widetilde{A^{t-1}} + \ldots + \tilde{A} + I)\,\tilde{B}C^0, \qquad (1.25)$$

де вираз в дужках є сумою членів геометричної прогресії для квадратних матриць.

Зазначимо, що:

$$\left(I - \tilde{A}\right)\left(I + \tilde{A} + \widetilde{A^2} + \ldots + \widetilde{A^{t-1}} + \widetilde{A^t}\right) = I - \widetilde{A^{t+1}},$$

звідки отримуємо, що сума членів геометричної прогресії для квадратних матриць має наступний вигляд:

$$I + \tilde{A} + \widetilde{A^2} + \ldots + \widetilde{A^{t-1}} + \widetilde{A^t} = \left(I - \tilde{A}\right)^{-1}\left(I - \widetilde{A^{t+1}}\right). \qquad (1.26)$$

Тоді, з огляду на (1.26), розв’язок (1.25) системи рівнянь (1.23), прийме вигляд:

$$X_{t+1} = \widetilde{A^{t+1}}X_0 + \left(I - \tilde{A}\right)^{-1}\left(I - \widetilde{A^{t+1}}\right)\tilde{B}C^0,$$

або

$$X_{t+1} = \widetilde{A^{t+1}}X_0 + \left(I - \tilde{A}\right)^{-1}\tilde{B}C^0 - \left(I - \tilde{A}\right)^{-1}\widetilde{A^{t+1}}\tilde{B}C^0. \qquad (1.27)$$

Враховуючи у (1.27) (при $\tilde{A} = \tilde{B}(A - B), \tilde{B} = (I - B)^{-1}$), маємо розв'язок системи різницевих рівнянь (1.23) у вигляді:

$$X_{t+1} = \widetilde{A^{t+1}} X_0 + X^* - (I - A)^{-1}(A - B)\widetilde{A^t}\tilde{B}C^0, \qquad (1.28)$$

де $X_0 = \mathrm{X}(0)$ — початковий стан системи; $X^* = (I - A)^{-1}C^0$ — рівноважний стан системи.

Якщо відбувається зміна значень розв'язків $X^* = (I - A)^{-1}C^0$, вона супроводжується і зміною X_{t+1}, які відображені у формулі (1.28), також якщо $C^0 \geq 0$ є сталим, тоді на елементи вектору $X^* = (I - A)^{-1}C^0$ впливає значення елементів матриці A [5, 7].

У цьому розділі були розглянуті основні поняття математичну модель динамічної системи балансового типу, далі розглянемо методи керування й регулювання моделі динамічної системи балансового типу.

1.1.3 Керування й регулювання моделі динамічної системи балансового типу

Розглянемо питання керування й регулювання позитивних динамічних систем балансового типу, які описуються лінійними різницевими рівняннями [8].

Керування і регулювання позитивних динамічних систем балансового типу застосовується тоді, коли відсутня можливість редагування вхідних параметрів моделі, які відповідають за безліч змінних основні з яких:

а) поведінку складних систем;

б) стабілізування нестійких систем;

в) покращення динамічних властивостей системи.

Необхідність їх використання обумовлюється випадками, коли потрібно змінити досліджуваний процес таким чином, щоб показники, які його характеризують, задовольняли певним вимогам [10, 12, 13].

Для прикладу розглянемо динамічну систему балансового типу системою виду [7]:

$$X_{t+1} = \tilde{A}X_t + \tilde{B}C_t, \qquad (1.29)$$

де X_t — n – мірний вектор стану системи; C_t — n – мірний вектор керування системи в дискретний момент часу; $\tilde{A}, \tilde{B}$ — сталі матриці розмірності $n \times n$

Основне значення, надається принципу проектування в якому говориться, що система керування має бути асимптотично стійкою [13].

Значимість цього принципу обумовлюється тим, що дискретні системи керування характеризуються нестійким об'єктом який може бути стабілізований за допомогою замкненого керування й не може бути стабілізований при розімкненому керуванні [12, 13].

Принцип розімкненого керування або керування за розімкненим контуром полягає в тому, що на основі заданого алгоритму функціонування виробляється алгоритм керування, який ніяк не може корегуватися у процесі при змінах характеристик або параметрів об’єкта керування. В свою чергу принцип замкненого керування або керування за замкненим контуром полягає в тому, що на основі вимірювання зворотного зв’язку керованої змінної, в так званому, замкненому контурі формується алгоритм керування, що дозволяє в будь-який момент порівнювати бажану величину з дійсною й миттєво реагувати на будь-які відхилення.

Систему, яка функціонує на принципі розімкненого курування, тобто систему, що безпосередньо не використовує кінцеві результати керування

об’єктом, будемо називати розімкненою системою керування. Замкненою системою керування або системою керування зі зворотним зв’язком, будемо називати систему у якій процес керування об’єктом залежить від результату керування. Слід зазначити, що при розмиканні зворотного зв’язку, система яка функціонує за замкненим контуром, автоматично перетворюється в систему розімкнену [5].

Згідно з [10, 11], для того, щоб з одного стану об’єкт керування можна було перевести в інший з використанням інформації про всі його змінні стану, він повинен характеризуватися властивістю керованості.

Згідно з [10, 11], об’єкт керування, поведінка якого описується векторно-матричним рівнянням (1.29) може бути названим повністю керованим тільки тоді, коли його можна перевести з будь-якого початкового стану $X(0) = X_0$ в будь-який кінцевий стан $X(T) = X_T$ за кінцевий час за кінцевий час T при дотриманні заданих обмежень.

На цьому завершується огляд математичного моделювання позитивних динамічних систем, наступним кроком, йде проектування програмного забезпечення для розрахунку математичної моделі балансованого типу.

1.1.4 Технологічне керування в розімкнених і замкнених моделях позитивної динамічної системи балансового типу

Для прикладу, розглянемо розімкнену дискретну математичну модель позитивної динамічної системи балансового типу вигляду [4, 6]:

$$(I - B)X_{t+1} = (A - B)X_t + C_t,\ X(0) = X_0, \qquad (1.30)$$

та її неперервний аналог (час t припускається неперервним)

$$X^* = (I - B)^{-1}(a - x)X(t) + (I - B)^{-1}C(t),\ X(0) = X_0, \qquad (1.31)$$

де $X_t, X(t) -$ $n-$мірні вектори станів; $A, B -$ матриці розмірності $n \times n$ сталих коефіцієнтів; C_t, $C(t) -$ $n-$мірні вектори керування відповідно для дискретної й неперервної моделі; $I -$ одинична матриця розмірності $n \times n$; $X(0) = X_0 -$ $n-$ мірний вектор-стовпець початкових умов системи.

Припустимо, що елементи вектору $C^0 \geq 0$ не піддаються зміні, тоді значення елементів матриці A істотно впливають на елементи вектора $X^* = (I - A)^{-1}C^0$.

Отже маємо, що на значення елементів X_t (для дискретного випадку) або $X(t)$ (для неперервного випадку) буде впливати матриця сталих коефіцієнтів A. Тому цю матрицю вважатимемо керуючим впливом, що дозволяє покращувати в деякому смислі зазначені характеристики об'єкта.

Для застосування формули $X^* = (I - A)^{-1}C^0 \geq 0$, елементи матриці A повинні задовольняти нерівності $0 \leq a_{ij} < 1,\ i, j = \overline{1,n}$ (або $0 \leq A < 1$ при $n = 1$). Тому покращення вихідних характеристик об’єкта здійснюється безпосереднім вибором елементів матриці A так, щоб матриця A залишалася продуктивною.

У випадку коли матриця $A = (a_{ij})_{n \times n}$ є непродуктивною або ж продуктивною без можливості зміни, можливо шляхом переходу від розімкненої моделі до замкненої моделі можливо зробити матрицю A продуктивною.

Шляхом введення зворотного зв’язку яка описується системою (1.30) побудуємо замкнену дискретну математичну модель позитивної динамічної системи балансового типу, яка матиме вигляд [8]:

$$C_t = C_t^0 - kX_t. \tag{1.32}$$

Відповідно до (1.32), замкнена система керування описується наступним векторно-матричним рівнянням стану:

$$X_{t+1} = (I-B)^{-1}(A-k-B)X_t + (I-B)^{-1}C_t^0. \tag{1.33}$$

Вигляд (1.33), ідентичний вигляду (1.30), Відмінність полягає в тому, що матриця A системи (1.30), замінена в системі (1.34) матрицею $(A-k)$. Тому, щоб у системі (1.33) був невід'ємний розв'язок, необхідно, щоб матриці коефіцієнтів A, B, $(A-k)$, $(A-k-B)$, $(I-B)^{-1}(A-k-B)$ були невід'ємними й продуктивними.

Згідно з [8, 10] система (1.33) приймає вигляд:

$$\Delta X_t = (I-B)^{-1}(A-k-I)X_t + (I-B)^{-1}C^0. \tag{1.34}$$

Розглянемо аналіз внутрішніх динамічних властивостей складної системи (1.35) при сталому $C_t^0 = C^0$, $C^0 \geq 0$, розв'язок такого рівняння шукається відповідно до підходу, описаному в пункті 1.1.2 розділу 1, і надається у вигляді:

$$X_{t+1} = \tilde{A}^{t+1}X_0 + X^* - (I-A+k)^{-1}(A-k-B)\tilde{A}^t\tilde{B}C^0, \tag{1.35}$$

де $X_0 = X(t_0)$ – початковий стан системи; $X^* = (I-A+k)^{-1}C^0$ – рівноважний стан системи.

На основі описаного вище підходу, для замкненої дискретної моделі, на основі розімкненої неперервної моделі, описуваною системою диференціальних рівнянь (1.31), шляхом введення в систему (1.31) зворотного зв'язку, вираз набуває вигляду:

$$C(t) = C_0(t) - k\,X(t). \qquad (1.36)$$

Отримана в (1.43) вектор-функція є лінійним регулятором досліджуваної системи. На підставі цього вважатимемо, що цей вектор-функції здійснює регулювання позитивною динамічною системою балансового типу, описуваною системою (1.31).

З урахуванням (1.36) замкнена система керування запишеться у вигляді:

$$X^*(t) = (I - B)^{-1}(A - k - I)X(t) + (I - B)^{-1}C_0(t)\ . \qquad (1.37)$$

Якщо порівнювати отримані умови невід'ємності й продуктивності матриць коефіцієнтів для замкненої неперервної моделі коефіцієнтів з умовами для замкненої дискретної моделі, то можна побачити, що ці умови не накладаються на матриці коефіцієнтів $(A-k-B)$ і $(I-B)^{-1}(A-k-B)$, що йде зі структури одержуваних рівнянь (1.37).

У (1.37) для аналізу внутрішніх динамічних властивостей будемо вважати вектор $C_0(t)$ сталим, тобто будемо розглядати систему (1.44) у вигляді:

$$X^*(t) = (I - B)^{-1}(A - k - I)X(t) + (I - B)^{-1}C_0(t) \qquad (1.45)$$

Розглянемо систему (1.45) при сталому $C_0(t) = C^0$, $C^0 \geq 0$, тобто систему лінійних неоднорідних диференціальних рівнянь першого порядку зі сталими коефіцієнтами. Розв'язок системи має вигляд:

$$X(t) = e^{\tilde{A}t}(X_0 - X*) + X*, \tag{1.47}$$

де $X_0 = X(t_0)$ – початковий стан системи; $e^{\tilde{A}t}$ – матрична експонента; $X^* = (I - A + k)^{-1} C^0$ – рівноважний стан системи.

Аналогічно дискретній моделі, можливі наступні випадки:

а) при $X_0 > X^*$ значення $X(t)$ поелементно наближуються до елементів рівноважного X^*, монотонно експоненційно спадаючи з початкового стану $X(0) = X_0$;

б) при $X_0 = X^*$ елементи вектора $X(t)$ залишаються для всього наступного часу сталими й рівними елементам вектора X^*;

в) при $X_0 < X^*$ значення $X(t)$ поелементно наближуються до елементів рівноважного X^*, монотонно експоненційно зростаючи з початкового стану $X(0) = X_0$.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРАХУНКУ МАТЕМАТИЧНОЇ МОДЕЛІ БАЛАНСОВОГО ТИПУ

У даному розділі будуть організовані знання, які були надані раніше для побудови алгоритму який буде використовуватись у програмному забезпеченні, визначимо вхідні и вихідні дані системи, а також виберемо мову на якій буде реалізований наш проект.

2.1 Алгоритм побудови програмного забезпечення

Алгоритм побудови програмного забезпечення буде проходити у декілька етапів.

Для початку нам потрібно задати початкові умови $X(0) = X_0$, а також вхідні параметри A, B, C_0.

Наступним кроком йде аналіз компонентів, який включає в себе дослідження, як на невід'ємність, так и на продуктивність вхідних параметрів матриць коефіцієнтів $A, B, (A - B), \tilde{A} = (I - B)^{-1}(A - B)$. Продуктивність перевіряється з використання необхідних і достатніх умов, котрі були описані у підрозділі 1.1.1 розділу 1. Зберіганням цих умов обумовлюється позитивність системи та асимптотичну стійкість одержуваних розв'язків.

Наступним кроком є порівняння вихідних характеристиками об'єкта моделювання, які повинні задовольняти користувача, з всіма вхідними характеристиками об'єкта, якщо вихідні характеристики об'єкта моделювання не збігаються, тим самим не задовольняють користувача тоді із підрозділу 1.1.2

розділу 1 обирається вплив на систему, який забезпечує досягнення поставленої мети.

В кінці повинно бути отримано графічне представлення чисельних результатів.

2.2 Вхідні та вихідні дані системи

Вхідними даними буде розімкнена модель об’єкту дослідження з дискретним часом вигляду:

$$X_{t+1} = (I - B)^{-1}(A - B)X_t + (I - B)^{-1}C_t, X(0) = X_0, \qquad (2.1)$$

де X_t, X_{t+1} — n – мірний вектори стану позитивної системи відповідно в моменти часу t і $t + 1$; C_t — n – мірний вектор керування системи в момент часу t; A, B — сталі матриці розмірності $n \times n$; n — кількість агрегованих взаємозв'язаних між собою підсистем досліджуваної складної системи; I — одинична матриця розмірності $n \times n$; $X(0) = X_0$ — n – мірний вектор-стовпець початкових умов системи.

Вхідними даними системи є початковий вектор C_t (для дискретної моделі) або $C(t)$ (для неперервної моделі) та початковий стан системи X_0 у момент часу t = 0.

Визначимо основні компоненти моделі об’єкта дослідження.

n – мірна вектор-функція X_t, вона відповідає за визначення фазових станів об’єкта в дискретний момент часу.

C_t — це вхідна змінна моделі з дискретним часом.

Система (2.1) надає можливість виміряти спостережуваною змінною та позначається для системи з дискретним часом у вигляді $Y_t = DX_t$, де $D = (d_{ij})$ — відома матриця сталих коефіцієнтів відповідної розмірності.

Матриці A, B — це сталі матриці моделі значення яких встановлюється емпірично. Зазначені матриці відображають взаємозв'язки між підсистемами досліджуваної складної системи і є звичайно заданими. Якщо допускається зміна елементів матриць A, B, тоді надається можливість з їх допомогою впливати і покращувати динамічні властивості системи.

Також для систем (2.1) може бути заданий бажаний стан об'єкта дослідження, відповідно X_t^* (для дискретної моделі) або $X^*(t)$ (для неперервної моделі), за допомогою яких визначаються програмні керування рухом систем.

Крім того, передбачається, що для даної системи (2.1) можна виміряти деяку лінійну комбінацію змінних станів, яка називається спостережуваною (вихідною) змінною та позначається для системи з дискретним часом у вигляді $Y_t = DX_t$, де $D = (d_{ij})$ – відома матриця сталих коефіцієнтів відповідної розмірності.

2.3 Мова програмування

Для якісної реалізації поставленої задачі, потрібно порівняти найбільш актуальні мови програмування, а саме C++, Java, C#[15, 16].

C++ це мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблений у 1979 році та початково отримала назву «Сі з класами». У 1983 перейменована у C++ [15, 16].

Java це об’єктно-оріентована мова програмування випущена у 1995 році як основний компонент платформи Java. Синтаксис мови багато в чому схож на C++ і C# [17].

C# відноситься до сім’ї з C-подібним синтаксисом, з цих мов його синтаксис найбільш близький до C++ і Java. Мова має статичну типізацію, підтримує поліморфізм та перезавантаження операторів [18].

C# і Java є чистими об’єктно-оріентованими мовами програмування, C++ також підтримує процедурне програмування. В C# та Java для створення глобальних змінних, треба створити фіктивний клас та оголосити їх з модифікатором static.

В C# та Java збирач сміття використовується для керування пам'яттю, яке автоматично вивільняє невикористовувану пам'ять. Це продуктивно, але викликає велике споживання пам’яті та низьку продуктивність. Обумовлено це тим, що період виклику збирач сміття залежить від використовуваної реалізації, а у моменти поки він не буде діяти, блоки не будуть звільнені, також цей процес потребує додаткові системні ресурси, що погіршує ефективність виконання програм.

В C++ керування пам'яттю повністю віддається програмісту, він в свою чергу може у будь-який момент виділити або звільнити пам'ять. Це як перевага, так і недолік, бо з однієї сторони робить програми швидкими і не вимогливими до пам’яті, а з іншої може призводити до «витоків пам’яті». Виявлення місця «витоку пам’яті» є досить складною задачею і потребує багато часу. Тому це залежить тільки від програміста.

Мови C# та Java не підтримують множинного наслідування класів на відміну від C++, тому в них були введені інтерфейси, які підтримують множинне наслідування. C++, в свою чергу відсутня сама конструкція для інтерфейсів, але їх механізм може бути реалізован за допомогою абстрактних класів.

Код, який написаний на C# та Java компілюється спочатку у байт-код, а потім інтерпритується JVM (Java Virtual Machine) та CLR (Common Language Runtime) відповідно.

Код написаній на C++ компілюється одразу в машинний код, що в свою чергу робить виконання програм швидшими. Цей код може компілюватись під кожну платформу спеціальним компілятором, але кросплатформеність залежить основним чином від бібліотек, тому що бібліотеки є як кросплатформених, так і платформо-залежні.

Основним критерієм вибору мови є не тільки її потужність і гнучкість в використанні, а й кількістю бібліотек, які написні для неї. Адже чим більша кількість бібліотек, тим більш широкий спектр задач можна вирішити за її допомогою. Якщо порівнювати стандартні бібліотеки, тоді C# та Java мають набагато більше ніж C++ і їх спектр можливостей значно ширший. Тому при програмуванні на C++ , потрібно залучати сторонні бібліотеки. В свою чергу, якщо порівняти кількість сторонніх бібліотек, то можна побачити, що у C++ їх набагато більше ніж у C# та Java. Але багата кількість обумовлюється спецефічною архітектурою багатьох з ніх. Крім того багато C++ бібліотек створює і визначає нові базові типи, заводить свої типи контейнерів, рядків, шо в свою чергу ускладнює орієнтування і вносить додаткову заплутаність, як наслідок, зменшує зручність використання.

Порівняння C++, Java, C# наведено у таблиці 2.1.

Таблиця 2.1 — Порівняння C++, Java, C#

Особливість	C++	Java	C#
Явне приведення типів	+	-	-
Параметричний поліморфізм	+	-	+\-
Блок finally	+	-	+
Блок else	+	-	+\-
Контроль границь масивів	+	+\-	+
Множинне наслідування	-	+	-
Значення параметрів по замовчуванню	+	+	+

Можна перерахувати переваги мови C++ над іншими, а саме:

а) використовує проміжний код і тому може виконуватись на будь-яких комп’ютерах;

б) є повністю об’єктно-оріентованим;

в) має багато доповнень які полегшують програмування;

г) програма на C++ може складатися як з одного, так і з декількох файлів , що містять вихідний текст на мові C++.

Інша перевага в використанні мови програмування C++ – це велика кількість інструментів та бібліотек для роботи з різними типами даних та об’єктів що зменшують час роботи.

Одним із таких інструментів для розробки програмного забезпечення є QT. Використання цієї бібліотеки зумовлене великою кількістю модулів та класів, що полегшують написання програм із графічним інтерфейсом.

2.4 Основні можливості та переваги динамічної бібліотеки QT

Відмінна особливість Qt від інших бібліотек це використання Meta Object Compiler (MOC) - попередньої системи обробки вихідного коду. MOC дозволяє у багато разів збільшити потужність бібліотек, вводячи такі поняття, як слоти і сигнали. Крім того, це дозволяє зробити код більш лаконічним. Утиліта MOC шукає в заголовних файлах на C ++ опису класів, що містять макрос Q_OBJECT, і створює додатковий вихідний файл на C ++, що містить мета-об’єктний код [19].

Qt є фундаментом популярної робочого середовища KDE, що входить до складу багатьох дистрибутивів Linux.

Qt дозволяє створювати власні плагіни і розміщувати їх безпосередньо в панелі візуального редактора.

Qt дає можливість, створення переносних програм для роботи з базами даних використовуючи стандартні СУБД такі як:

а) Microsoft SQL Server;

б) Sybase Adaptive Server;

в) IBM DB2 ;

г) PostgreSQL;

д) MySQL;

е) ODBC.

є) Oracle;

Qt надає програмісту не тільки зручний набір бібліотек класів, а й певну модель розробки додатків, певний каркас їх структури. Дотримання принципів і правил «гарного стилю програмування на C ++ / Qt» істотно знижує частоту таких важко відловлювати помилок в додатках, як витоку пам'яті (memory leaks), необроблені виключення, незакриті файли або зайняті дескриптори ресурсних

об'єктів, чим нерідко страждають програми, написані лише на C ++ без використання бібліотеки Qt.

Однією з важливих переваг Qt є добре продуманий, логічний і стрункий набір класів, що надає програмісту дуже високий рівень абстракції. Завдяки цьому, при використанні Qt, доводиться писати значно менше коду, ніж при використанні бібліотеки класів MFC. Код написаний у Qt виглядає стрункіше, простіше, логічніше і зрозуміліше, ніж аналогічний за функціональністю код MFC. Qt дозволяє створювати власні плагіни і розміщувати їх безпосередньо в панелі візуального редактора. Також існує можливість розширення звичної функціональності віджетів, пов'язаної з розміщенням їх на екрані, відображенням, перемальовуванням при зміні розмірів вікна [19, 20].

В Qt вбудована візуальнє середовище розробки графічного інтерфейсу «Qt Designer», що дозволяє створювати діалоги і форми в режимі WYSIWYG. Також в Qt є «Qt Linguist» - це графічна утиліта, яка спрощую локалізацію і переклад програми на інші мови.

«Qt Assistant» - довідкова система Qt, що спрощує роботу з документацією по бібліотеці, а також дозволяє створювати крос-платформну довідку для розроблювального на основі Qt програмного забезпеченяя. За допомогою величезної кількості всіляких класів можна написати додаток, що працює тільки через інтерфейс командного рядка (CLI). Але в основному, звичайно, Qt використовується для побудови графічних додатків, головна особливість яких – крос платформність [20, 21].

Крос-платформність у Qt важлива перевага тому, що у разі використання Qt для цього знадобиться всього лише перекомпіляція вихідного коду. У разі ж використання, наприклад, MFC або системних API знадобиться багато важкої роботи по портуванню, адаптації та налагодженні, або і переписування коду для

іншої ОС або апаратної платформи. Qt є повністю об'єктно-орієнтованим, легко розширюваним і підтримує техніку компонентного програмування [21].

Програми на Qt можуть розширювати свою функціональність за допомогою модулів і динамічних бібліотек. Модулі включають додаткові кодеки, драйвера баз даних, формати зображень, стилі та віджети. Бібліотеки розділена на декілька модулів:

а) QtOpenGL – набір класів для роботи з OpenGL;

б) QtSql – набір класів для роботи з базами даних з використанням мови структурованих запитів SQL. Основні класи даного модуля в версії 4.2.x: QSqlDatabase – клас для надання з'єднання з базою, для роботи з якою-небудь конкретною базою даних вимагає об'єкт, успадкований від класу QSqlDriver – абстрактного класу, який реалізується для конкретної бази даних і може вимагати для компіляції SDK бази даних. Наприклад, для складання драйвера під базу даних FireBird / InterBase вимагає .h файли і бібліотеки статичної лінковки, що входять в комплект поставки даної БД [20, 21];

в) QtScript – класи для роботи з Qt Scripts;

г) QtNetwork – набір класів для мережевого програмування. Підтримка різних високорівневих протоколів може змінюватися від версії до версії. У версії 4.2.x присутні класи для роботи з протоколами FTP і HTTP. Для роботи з протоколами TCP / IP призначені такі класи, як QTcpServer, QTcpSocket для TCP і QUdpSocket для UDP;

д) QtGui – компоненти графічного інтерфейсу;

е) QtCore – класи ядра бібліотеки, які використовуються іншими модулями;

є) QtSvg – класи для відображення і роботи з даними Scalable Vector Graphics (SVG);

ж) QtDesigner – класи створення розширень QtDesigner для своїх власних віджетів;

з) QtDeclarative – модуль, що надає декларативний фреймворк для створення динамічних, що настроюються призначених для користувача інтерфейсів.

и) QtXmlPatterns – модуль для підтримки XQuery 1.0 і XPath 2.0;

і) QtWebKit – модуль WebKit, інтегрований в Qt і доступний через її класи;

к) QtTest – модуль для роботи з UNIT тестами;

л) QtAssistant – довідкова система;

м) QtUiTools – класи для обробки в додатку форм Qt Designer.

Однією з вагомих переваг проекту Qt є наявність якісної документації. Статті документації забезпечені великою кількістю прикладів. Вихідний код самої бібліотеки добре форматований, докладно коментований і легко читається, що також спрощує вивчення Qt.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ

3.1 Реалізація програмного продукту

Для створення програмного продукту, котрий призначений для побудови та керування складної динамічної системи балансового типу, був використаний алгоритм який описувався у підрозділі 2.1 розділу 2.

На рис. 3.1 зображено зовнішній вигляд розробленого продукту, він включає в себе наступні елементи (номер пункту у наступному списку відповідає номеру на рис. 3.1):

а) (1) — область введення кількості підсистем у моделі;

б) (2) — область для графічного представлення даних;

в) (3) — таблиця результатів вхідних характеристик;

г) (4) — область введення матриці A;

д) (5) — область введення матриці B;

е) (6) — область введення вектору C_0;

є) (7) — область введення вектору X_0;

ж) (8) — кнопка виконання операції побудови динамічної системи;

з) (9) — функція збереження результатів розрахунків у файл;

и) (10) — керування за допомогою матриці A;

і) (11) — керування за допомогою матриці B.

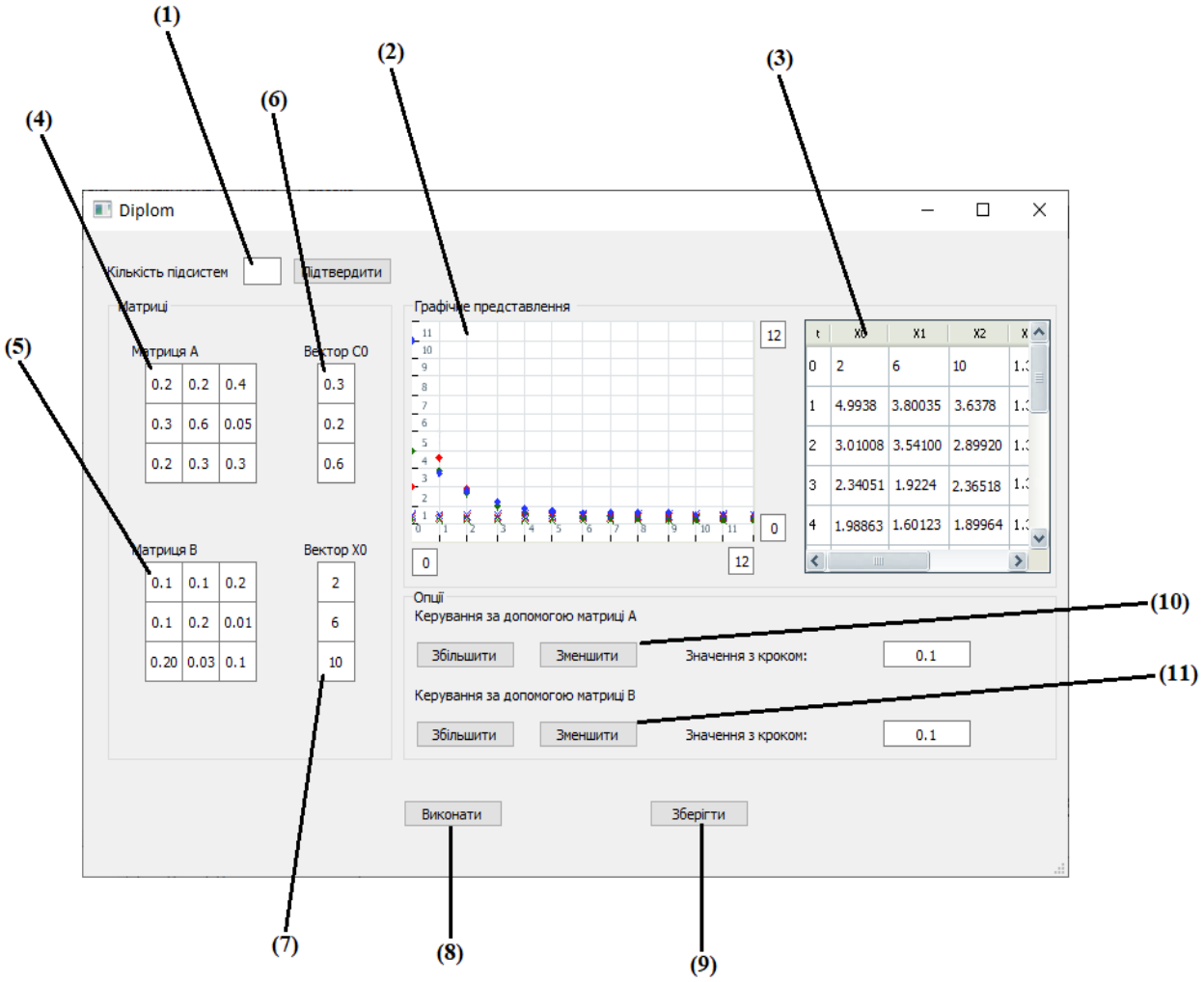

Рисунок 3.1 — Зовнішній вигляд розробленого продукту

Щоб полегшити роботу кінцевого користувача, у програмний продукт був введений захист у вигляді перевірки на недопустимість символів матриць A і B а також векторів X_0 і C_0, на рис. 3.2 зображена ділянка коду яка відповідає за перевірку.

```
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, "Ошибка", "Введены не все
значения матрицы. Заполните все ячейки.", "Продолжить");
                        return 1;
                  }
                  sNum = matrix->item(exI,exJ)->text();

                  for(int strI=0;strI<sNum.length();strI++)
                  {
                        if(!(sNum[strI]<=QChar('9') && sNum[strI]>=QChar('0'))
&& !(sNum[strI]<=QChar('.') && sNum[strI]>=QChar(',')))
                        {
                              QMessageBox::critical(this, "Ошибка", "В
матрицах обнаружены недопустимые символы.", "Продолжить");
                              return 1;
                        }
                  }
            }
      return 0;
}
```

Рисунок 3.2 — Перевірка на недопустимість символів

Програма має зрозумілий інтерфейс в якому легко орієнтуватися, що значно полегшує роботу з ним і дозволяє швидко розв’язувати задачі і наочно побачити результати моделювання позитивної динамічної системи балансового типу.

3.2 Інструкція по використанню для користувача

Зробимо детальний розбір та порядку використання програмного забезпечення.

Перш за все, перед роботою з програмою потрібно вказати необхідну розмірність системи, це можна зробити вказавши у полі позначеному на рис. 3.1 під цифрою 1 потрібне значення і натиснути на кнопку «Підтвердити».

Слідом необхідно заповнити матриці A, B (їх розмірність буде варіюватися згідно з тим яку розмірність ви призначили у 1) та векторів C_0 і X_0, їх можна побачити на рис. 3.1 під цифрами 10, 11, 6, 7 відповідно. На ці вхідні данні стоїть декілька перевірок, а саме:

а) у поля таблиці можуть бути вписані тільки цифри і символи «.» (крапка) і «,» (кома). Якщо у полі буде вписана латинська літера або інший службовий знак програма сповістить про помилку при заповненні таблиці (рис.3.3);

б) запис «0.001», або «0,001» сприймаються однаково і приводять до однакового результату;

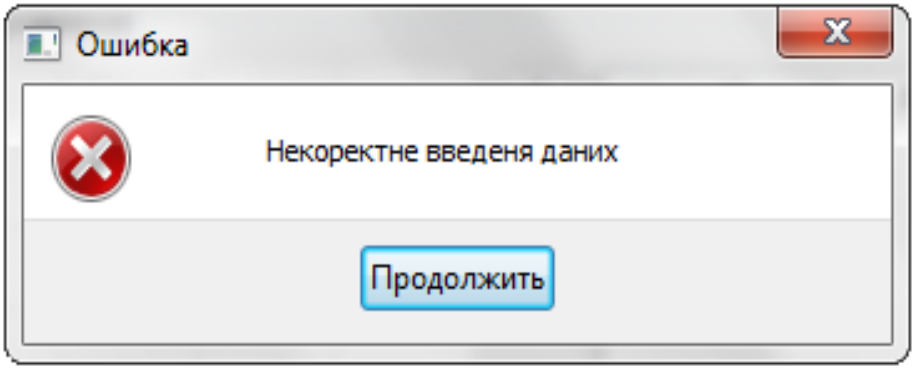

Рисунок 3.3 — Повідомлення про невірне введення даних

Після коректного вводу даних потрібно обрати необхідний масштаб графіка, у цьому допоможе пункт 2 на рис. 3.1 Знизу та з правого боку графічної області присутні чотири поля для задання інтервалу відображення розв’язків системи (1.15).

На цю область діють такі ж самі перевірки як і на матриці A, B та векторів C_0 і X_0. Крім них накладаються додаткові обмеження, а саме:

а) інтервал повинен бути цілочисельним;

б) обмеження на задання інтервалу рендерингу графіка. Неможливість вводу інтервалу виду (1; -3) або (1; 1) ;

У випадку порушень цих обмежень користувач побачить повідомлення аналогічне повідомленню на рис. 3.3.

Далі, для отримання розв’язків складної системи без використання керування потрібно натиснути на кнопку під цифрою 8 на рис. 3.1. Після цього можливі два випадки:

а) у випадку непродуктивності матриці, умови якої описані у підрозділі 1.1.1 розділу 1, буде надіслано повідомлення (рис. 3.4) про непродуктивність матриці (на рис. 3.5 зображена ділянка коду яка відповідає за перевірку непродуктивності матриці) та можливість перейти до замкненої моделі при використанні матриці k за допомогою якої стабілізувати нестійку систему(на рис. 3.6 зображена ділянка коду яка відповідає за перехід до замкненної моделі);

б) у випадку успішного виконання в графічній області під цифрою 7 на рис. 3.1 та в таблиці під цифрою 6 на рис. 3.1 будуть виведені результати обчислення;

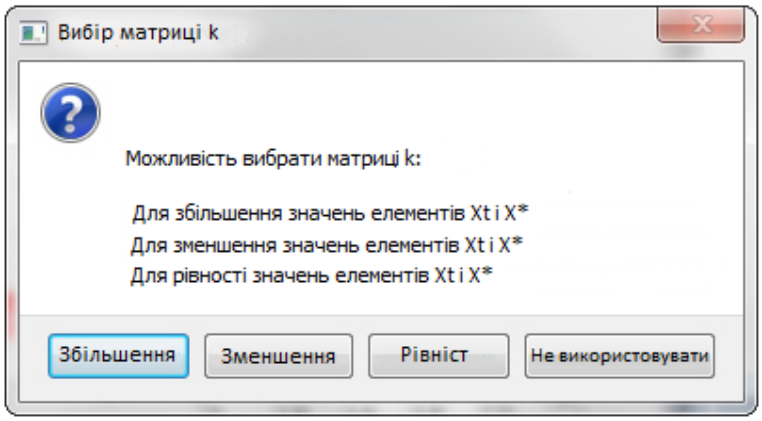

Рисунок 3.4 — Попередження у випадку непродуктивності матриць

```
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);

	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;

}
```

Рисунок 3.5 — Перевірка на непродуктивність матриці

```
                                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;
                                // Заменяем значения в области ввода
                                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;
                          }
```

Рисунок 3.6 — Перехід до замкненої моделі при використанні матриці k

Після отримання результату користувачу надається можливість перейти до замкненої моделі та використання матриці k посилення зворотнього зв'язку. Користувачу доступно чотири варіанти на вибір (рис. 3.7):

а) зменшити значення елементів вектора X_t та перейти до замкненої моделі;

б) збільшити значення елементів вектора X_t та перейти до замкненої моделі;

в) забезпечити рівність значень елементів вектора X_t елементам вектору C_0 та перейти до замкненої моделі;

г) відмовитись від переходу до замкненої моделі.

```
QMessageBox msgProd;
                        QPushButton *incButton =
msgProd.addButton("Збільшити", QMessageBox::ActionRole);
                        QPushButton *decButton =
msgProd.addButton("Зменшити", QMessageBox::ActionRole);
                        QPushButton *eqvButton =
msgProd.addButton("Рівність", QMessageBox::ActionRole);
                        QPushButton *notButton =
msgProd.addButton("Не використовувати",
QMessageBox::ActionRole);
                        msgProd.setIcon(QMessageBox::Question);
                        msgProd.setWindowTitle("Вибір матриці k");
                        msgProd.setInformativeText("\n\nl. Для
збільшення значень елементів вектору Xt и X*\n2. Для меншення
значень елементів вектору Xt и X*\n3. Для рівності значень
елементів вектору Xt и X*");
                        msgProd.exec();

                        // Обработка возможных событий (увеличение
\уменьшение\равенство)
                        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;
                        }
```

Рисунок 3.7 — Дії при переході до замкненої моделі при використанні матриці *k*

У разі потреби технологічного керування шляхом беспосереднього вибору елементів матриці *A*, будемо мати вибір із:

а) зміна елементів матриці у області під цифрою 5 на рис. 3.1;

б) збільшення або зменшення всієї матриці за допомогою області під цифрою 10 на рис. 3.1.

Збереження розрахунків здійснюється за допомогою кнопки під номером 9 на рис. 3.1. Збереження здійснюється у файлі формату *.txt, де будуть знаходитись табличні значення з області, позначеної цифрою 3 на рис. 3.1, у вигляді таблиці.

3.3 Приклади роботи програмного забезпечення

3.3.1 Визначення основних характеристик системи без керування

Проведемо обчислювальний експеримент при наступних умовах:

$$n = 2, A = \begin{pmatrix} 0.6 & 0.2 \\ 0.3 & 0.4 \end{pmatrix}, B = \begin{pmatrix} 0.35 & 0.1 \\ 0.2 & 0.25 \end{pmatrix},$$
$$C_0 = \begin{pmatrix} 0.1 \\ 0.6 \end{pmatrix}, X_0 = \begin{pmatrix} 8 \\ 5 \end{pmatrix}, t_0 = 0 \qquad (3.1)$$

Результати роботи програми наведено на рис. 3.8.

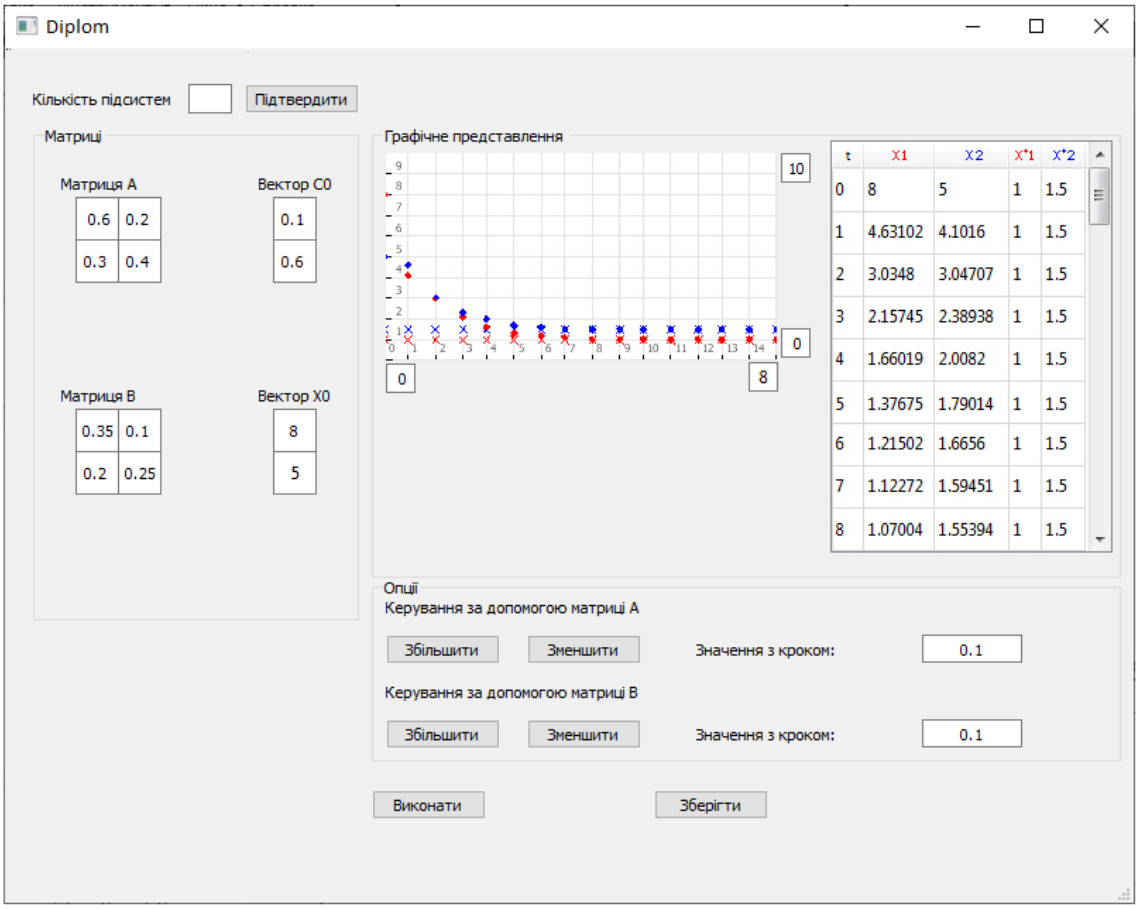

Рисунок 3.8 — Визначення основних характеристик без керування

Далі розглянемо приклад використання технологічного керування для замкненої динамічної системи.

3.3.2 Визначення основних характеристик системи з технологічним керуванням

Проведемо обчислювальний експеримент при наступних умовах:

$$n = 2, A = \begin{pmatrix} 1{,}2 & 0.2 \\ 0.3 & 0.4 \end{pmatrix}, B = \begin{pmatrix} 0.35 & 0.1 \\ 0.2 & 0.25 \end{pmatrix},$$

$$C_0 = \begin{pmatrix} 0.1 \\ 0.6 \end{pmatrix}, X_0 = \begin{pmatrix} 8 \\ 5 \end{pmatrix}, t_0 = 0 \quad (3.2)$$

В цьому випадку матриця A непродуктивна і щоб розв'язки системи були невід'ємними і система була асимптотично стійкою потрібне введення матриці k посилення зворотного зв'язку та переходу до замкненої моделі. Внутрішніми функціями розрахунку матриці посилення зворотного зв'язку створеного програмного продукту матриця k була обрана наступним чином:

$$k = \begin{pmatrix} 0{,}6 & 0{,}01 \\ 0{,}01 & 0{,}01 \end{pmatrix}. \quad (3.3)$$

Результати цієї дії наведено у рис.3.9

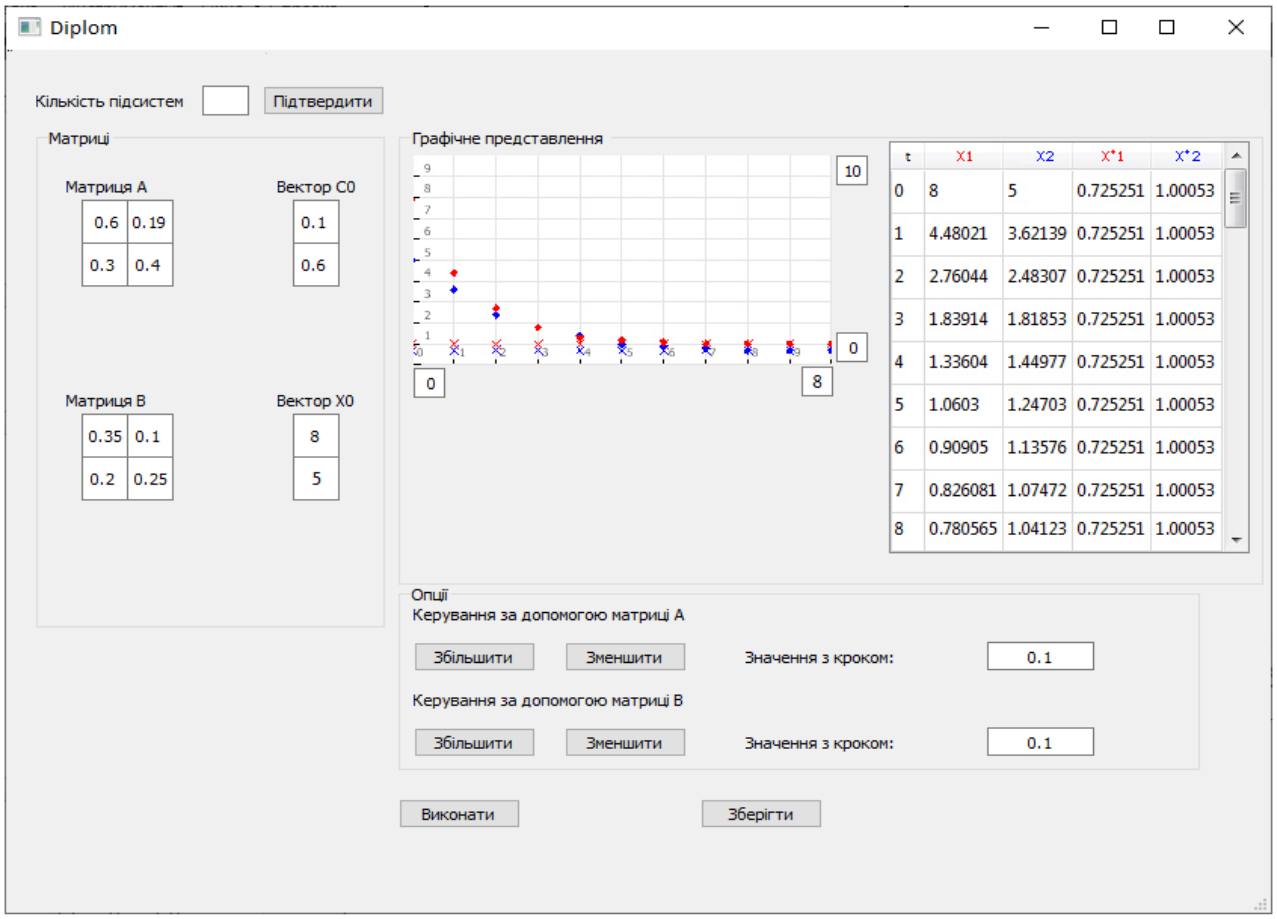

Рисунок 3.9 — Графічне представлення результатів з початковими умовами

Щоб покращити отримані значення застосуємо знов зворотній зв’язок, але вже при продуктивній матриці A обравши внутрішніми функціями розрахунку наступну матрицю k:

$$k = \begin{pmatrix} -0.2 & -0,01 \\ -0,01 & -0,01 \end{pmatrix} \quad (3.4)$$

У результаті отримаємо збільшені траєкторії X_t і X^* (рис. 3.10)

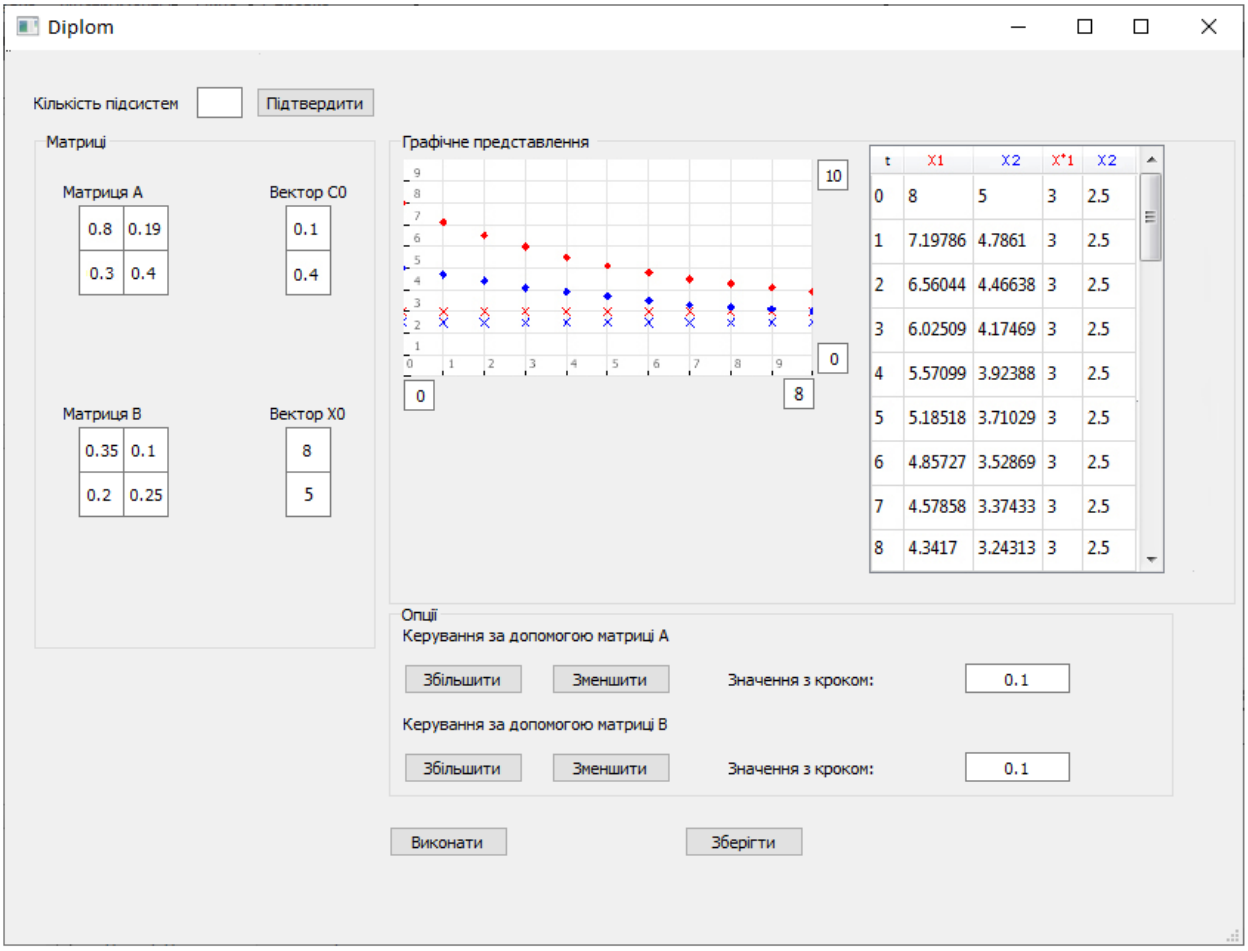

Рисунок 3.10 — Графічне представлення результатів для продуктивної моделі з використанням матриці посилення зворотного зв’язку

Таким чином за використовуючи зворотний зв’язок, непродуктивна матриця A була покращена так само як і вхідні характеристики X_t і X^* які з початку не відповідали належним їм властивостям.

ВИСНОВКИ

Метою роботи є розробка алгоритму для втілення його в програму з можливістю дослідження позитивної динамічної системи балансного типу. На даний момент, побудова алгоритму складна і потребує багато часу на реалізацію.

Кваліфікаційна робота включає в себе три розділи.

У першому розділі були розглянуті теоретичні аспекти динамічної системи, основні етапи побудови позитивної системи балансового типу і були розглянуті усі типи необхідних видів керувань із множин можливих для даної системи.

У другому розділі було спроектовано структурування позитивної системи балансованого типу і спираючись на нього був розроблений алгоритм котрий ліг в основу створеного програмного забезпечення визначення й дослідження вихідних характеристик позитивної динамічної системи балансового типу.

У третьому розділі була продемонстрована програмна реалізація алгоритму, її кінцевий вигляд для користувача з детальною інструкцією по використанню створеного продукту. Також були представлені приклади роботи програмного продукту.

Мета кваліфікаційної роботи була досягнута в повному обсязі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Одномерные модели сложного поведения простых динамических систем. URL: https://cyberleninka.ru/article/n/odnomernye-modeli-slozhnogo-povedeniya-prostyh-dinamicheskih-sistem (Дата звернення: 05.08.2020)

2. Леонтьева В.В., Кондратьева Н.А. О математической модели динамики функционирования взаимозависимых производств. *IX Крымская Международная математическая школа «Метод функций Ляпунова и его приложения»* (15-20 сентября 2008 г., г. Алушта). Тезисы докладов. Симферополь: Таврический национальный университет, 2008. С. 100.

3.Гельфонд А.О. Исчисление конечных разностей. Москва: Физматгиз, 1959. 400 с.

4. Леонтьева В.В., Кондратьева Н.А. Управление позитивной динамической системой балансового типа. *Воронежская зимняя математическая школа «Современные методы теории функций и смежные проблемы»* (27 января-2 февраля 2009 г., г. Воронеж). Материалы конференции. Воронеж: Воронежский государственный университет, 2009. С. 102

5. Леонтьева В.В., Кондратьева Н.А. Управление позитивной динамической системой балансового типа. *Воронежская зимняя математическая школа «Современные методы теории функций и смежные проблемы»* (27 января-2 февраля 2009 г., г. Воронеж). Материалы конференции. Воронеж: Воронежский государственный университет, 2009. С. 102

6. 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.

7. Леонтьева В.В. Математическая модель динамики функционирования позитивных систем балансового типа. *Зб. наук. праць. Вісник ЗНУ*. Запоріжжя: Запорізький національний університет, 2008. №1 С. 118-124.

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

9. Стрейц В. Метод пространства состояний в теории дискретных линейных систем управления: Пер. с англ. Э.Д. Аведьяна / Под ред. Я.З.Цыпкина. Москва: Наука, Главная редакция физико-математической литературы, 1985. 296 с.

10. Ашманов С.А. Введение в математическую экономику. Москва: Наука, 1984. 296 с.

11. Квакернаак Х., Сиван Р. Линейные оптимальные системы управления. Москва: Мир, 1977. 656 с.4

12. Стрейц В. Метод пространства состояний в теории дискретных линейных систем управления: Пер. с англ. Э.Д. Аведьяна / Под ред. Я.З.Цыпкина. Москва: Наука, Главная редакция физико-математической литературы, 1985. 296 с.

13. Андреев Ю.Н. Управление конечномерными линейными объектами. Москва: Наука, Главная редакция физико-математической литературы, 1976. 424 с

14. Красовский Н.Н. Теория управления движением. Москва: Наука, 1968. 476 с.

15. C++ URL: https://ru.wikipedia.org/wiki/C%2B%2B (Дата звернення: 15.09.2020)

16. Язик программирования C++. URL: https://learn-code.ru/yazyki-programmirovaniya/cpp (Дата звернення: 15.09.2020)

17. Java краткое руководство для начинающих URL: https://tproger.ru/translations/java-intro-for-beginners/ (Дата звернення: 15.09.2020)

18. Введение в C# Язык C# и платформа .NET Core URL: https://metanit.com/sharp/tutorial/1.1.php (Дата звернення: 15.09.2020)

19. Краткое развитие Qt. URL: https://www.opennet.ru/docs/RUS/qt3_prog/f71.html (Дата звернення: 18.09.2020)

20. Введение в Qt Creator URL: http://doc.crossplatform.ru/qtcreator/2.0.1/creator-overview.html (Дата звернення: 20.09.2020)

21. Часть 1. Введение. Инструменты разработчика и объектная молель. URL: https://www.ibm.com/developerworks/ru/library/l-qt_1/index.html (Дата звернення: 11.10.2020)

Додаток А

Лістинг вихідного коду програмного продукту для визначення й дослідження вихідних характеристик позитивної динамічної системи балансового типу

Файл dynamicSystem.pro:

```
CONFIG     +=    qt
QT         +=    opengl
HEADERS    +=    dynamicSystem.h \
                 glwidget.h \
                 matrix.h
SOURCES    +=    main.cpp \
                 dynamicSystem.cpp \
                 glwidget.cpp \
                 matrix.cpp
```

Файл main.cpp:

```
#include <QApplication>
#include <QTextCodec>
#include "dynamicSystem.h"

int main(int argc, char *argv[])
{
      QApplication app(argc, argv);
      QTextCodec::setCodecForCStrings(QTextCodec::codecForName("windows-
      1251"));
      DynamicSystem system;
      system.setWindowTitle("Моделирование и управление сложной позитивной
динамической системой");
      system.setMinimumSize(600, 650);
      system.setMaximumSize(600, 650);
      system.show();
      return app.exec();
}
```

Файл dynamicSystem.h:

```
#ifndef DSYS_H
#define DSYS_H

#include "matrix.h"
#include "glwidget.h"

#include <QWidget>
```

```
class QPushButton;
class QCheckBox;
class QComboBox;
class QGroupBox;
class QLabel;
class QLineEdit;

class QTableWidget;
class QTableWidgetItem;
class QTableView;

class QFont;

class GLWidget;

class DynamicSystem : public QWidget
{
     Q_OBJECT

public:
     DynamicSystem(QWidget *parent = 0);
     ~DynamicSystem();

     matrix               A;    // Матрица A
     matrix               B;    // Матрица B
     matrix               C0;   // Начальный вектор C0
     matrix               X0;   // Начальный вектор X0
     matrix               X;    // Значение функции во времени t
     matrix               Xst;  // Вектор-столбец желаемого состояния
(изменяется на каждой итерации)
     matrix               k;    // Матрица усиления связи

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

     int                  systemSize; // Размерность системы

     int                  x0;   // Начальная координата по X
     int                  x1;   // Конечная координата по X
     int                  y0;   // Начальная координата по Y
     int                  y1;   // Конечная координата по Y
     int                  tableRowCount;  // Кол-во записей

private slots:
     void toggleFuncList();     // Активатор\дезактиватор комбо-бокса с
доступными функциями
     void toggleEditA();        // Активатор\дезактиватор комбо-бокса
редактирования матрицы A
```

```
    void toggleSave();         // Активатор\дезактиватор комбо-бокса
сохранения предыдущей операции
    bool isFuncEnabled();// Признак активированного комбо-бокса с
доступными функциями

    void resizeX(); // Слот для проверки валидности ввода значений по оси
X
    void resizeY(); // Слот для проверки валидности ввода значений по оси
Y

    void incMatrixA();   // Инкремент всех элементов матрицы A.
    void decMatrixA();   // Декремент всех элементов матрицы A.
    void incMatrixB();   // Инкремент всех элементов матрицы B.
    void decMatrixB();   // Декремент всех элементов матрицы B.

    void resizeData();   // Изменение исходного вида системы
    void takeNewData();  // Получение данных после нажатия кнопки
"Выполнить"
    void saveData();     // Сохранение графика и всех состояний

private:

    QLineEdit* editSize;
    QPushButton*    resize;

    QCheckBox* checkChangeA;   // Чек-бокс для матрицы A
    QCheckBox* checkIsSetX;    // Чек-бокс для заданного X*
    QCheckBox* checkPrevIter;  // Чек-бокс для сохранения предыдущей
итерации

    QComboBox* functionList;   // Комбо-бокс с выбором доступных функций

    QTableWidget*   matrixA;   // Область для ввода матрицы A
    QTableWidget*   matrixB;   // Область для ввода матрицы B
    QTableWidget*   matrixC0;  // Область для ввода матрицы C0
    QTableWidget*   matrixX0;  // Область для ввода матрицы X0

    GLWidget*       graphArea; // Виджет графика
    QTableWidget*   tableArea; // Виджет таблицы, где будут записаны все
результаты

    QLineEdit* editX0;         // Начальная координата для масштабирования
по X
    QLineEdit* editX1;         // Конечная координата для масштабирования
по X
    QLineEdit* editY0;         // Начальная координата для масштабирования
по Y
    QLineEdit* editY1;         // Конечная координата для масштабирования
по Y

    QPushButton*    decA;      // Декремент прирощения значений матрицы A
    QPushButton*    incA;      // Инкремент прирощения значений матрицы A
    QPushButton*    decB;      // Декремент прирощения значений матрицы B
```

```
    QPushButton*    incB;      // Инкремент прирощения значений матрицы B

    QLineEdit*editA;     // Значение, на которое будет
увеличено\уменьшено значение матрицы A
    QLineEdit*editB;     // Значение, на которое будет
увеличено\уменьшено значение матрицы B

    QPushButton*launch;        // Запустить построение графиков и
получение численных данных

    QPushButton*save;          // Сохранить в файл все состояния и график
функции

    bool isFirstStart;   // Признак первого запуска программы
    bool isFirstStartSave;
    bool canSave;

    int  funcIndex;      // Индекс функции для X_st
    bool isSave;
    bool isChangable;    // Матрицу A можно изменять (также активируется
если не продуктивна матрица)
    bool isUsedFeedback; // Признак использования обратной связи (перейти
до замкненої моделі при використанні матриці kk по метке)

    int  pharseMatrix(int n, int m, QTableWidget* matrix);
// Парсер для матриц.

    bool isAllProductive();    // Проверка всех матриц на продуктивность

    void addNewLine(matrix tmp, matrix tmp_old, int t);
// Добавление новой записи в tableArea

    void getX_st(int t, int index);
// Получение функции X* в зависимости от funcIndex
    void getX(int t);
// Получение значений функции X с коорд. (X0, X1, X2)
    void getX_st_old(int t, int index);
// Получение функции X* в зависимости от funcIndex
    void getX_old(int t);

    void takeIncK(); // Функция для нахождения матрицы K в случае
увеличения знач. элем. вектора X
    void takeDecK(); // Функция для нахождения матрицы K в случае
уменьшения знач. элем. вектора X
    void takeEqvK(); // Функция для нахождения матрицы K в случае
равенства знач. элем. вектора X
    int  takeAttr; // DELETE LATER //

    void saveOldData();
    void resizeSystem();
};
```

```
#endif
```

Файл dynamicSystem.cpp:

```
#include "dynamicSystem.h"
#include "glwidget.h"
#include "matrix.h"

#include <QtGui>
#include <QDialog>

#include <math.h>

DynamicSystem::DynamicSystem(QWidget *parent)
    : QWidget(parent)
{
      funcIndex = -1;
      tableRowCount = 0;
      isChangable = true;
      isSave = false;
      isUsedFeedback = false;
      isFirstStart = true;
      isFirstStartSave = true;
      canSave = false;
      // Начальный размер = 3
      systemSize = 3;

      graphArea = new GLWidget;
      graphArea->setMaximumSize(278, 147);
      graphArea->setMinimumSize(278, 147);
      graphArea->systemSize = 3;

      tableArea = new QTableWidget(this);
      tableArea->verticalHeader()->hide();
      QFont headerF("Tahoma", 7);
      tableRowCount = 0;

      // Построение окна и расположение всех элементов внутри последнего
      QVBoxLayout* mainInfo = new QVBoxLayout(this);
      QHBoxLayout* topSize = new QHBoxLayout;
      QGridLayout* topInfoM = new QGridLayout;
      QGridLayout* topInfoC = new QGridLayout;
      QGridLayout* bottomInfoS   = new QGridLayout;
      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("Матрицы", this);
```

```
    QGroupBox* graphInfo = new QGroupBox("Графическое представление",
this);
    QGroupBox* changeInfo = new QGroupBox("Опции", this);

    QLabel* labelA = new QLabel("Матрица A:", this);
    QLabel* labelB = new QLabel("Матрица B:", this);
    QLabel* labelC0 = new QLabel("Вектор C0:", this);
    QLabel* labelX0 = new QLabel("Вектор X0:", this);

    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);

    resizeSystem();

    matrixA->verticalHeader()->hide();
    matrixA->horizontalHeader()->hide();
    matrixB->verticalHeader()->hide();
    matrixB->horizontalHeader()->hide();
    matrixC0->verticalHeader()->hide();
    matrixC0->horizontalHeader()->hide();
    matrixX0->verticalHeader()->hide();
    matrixX0->horizontalHeader()->hide();

    // Если задано X* то можно выбрать 1 из допустимых функций
    functionList->setEnabled(FALSE);
    functionList->addItem("X*1=4, X*2=3, X*3=2");
    functionList->addItem("X*1=4*t/5, X*2=2+t/2, X*3=3*t/6");
    functionList->addItem("X*1=4+sin(t), X*2=2*sqrt(t), X*3=8+2*cos(t)");

    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->addWidget(checkIsSetX, 0, 1);
    topInfoC->addWidget(functionList, 0, 2);
    bottomInfoS->addWidget(checkPrevIter, 0, 0);
```

```
	matrixInfo->setLayout(topInfoM);

	tableArea->setColumnCount((2*systemSize)+1);	// Кол-во столбцов в
таблице результатов
	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";
	//tableLabels << "t" << "X0" << "X1" << "X2" << "X*1" << "X*2" <<
"X*3";

	tableArea->setHorizontalHeaderLabels(tableLabels);

	editX0 = new QLineEdit(this);
	editX1 = new QLineEdit(this);
	editY0 = new QLineEdit(this);
	editY1 = new QLineEdit(this);

	// Задаем горизонтальные и ветикальные отступы между элементами
	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("Изменить", this);

    QLabel* labelSize = new QLabel("Количество подсистем:", 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("Управление с помощью матрицы A:",
this);
    decA = new QPushButton("Уменьшить", this);
    incA = new QPushButton("Увеличить", this);
    QLabel* labelStepA = new QLabel("значения с шагом:", this);
    editA = new QLineEdit(this);

    QLabel* labelChangeB = new QLabel("Управление с помощью матрицы B:",
this);
    decB = new QPushButton("Уменьшить", this);
    incB = new QPushButton("Увеличить", this);
    QLabel* labelStepB = new QLabel("значения с шагом:", this);
    editB = new QLineEdit(this);

    launch = new QPushButton("Выполнить", this);
    save = new QPushButton("Сохранить", 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(matrixInfo);
mainInfo->addLayout(topInfoC);
mainInfo->addWidget(graphInfo);
mainInfo->addWidget(changeInfo);
mainInfo->addLayout(bottomInfoS);
mainInfo->addLayout(bottomInfoL);
setLayout(mainInfo);

// Объединяем сигналы со слотами (обработка событий)
connect(resize, SIGNAL(clicked()), SLOT(resizeData()));

connect(checkIsSetX, SIGNAL(clicked()), SLOT(toggleFuncList()));
connect(checkChangeA, SIGNAL(clicked()), SLOT(toggleEditA()));

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(launch, SIGNAL(clicked()), SLOT(takeNewData()));
connect(save, SIGNAL(clicked()), SLOT(saveData()));
```

```
}

DynamicSystem::~DynamicSystem()
{
}

// Парсер для матриц. Проверяет на присутствие пустых ячеек матрицы и на
недопустимые символы.
//    Входные параметры:
//    n, m - размерность матрицы. matrix - указатель на матрицу\массив
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, "Ошибка", "Введены не все
значения матрицы. Заполните все ячейки.", "Продолжить");
                        return 1;
                  }
                  sNum = matrix->item(exI,exJ)->text();

                  for(int strI=0;strI<sNum.length();strI++)
                  {
                        if(!(sNum[strI]<=QChar('9') && sNum[strI]>=QChar('0'))
&& !(sNum[strI]<=QChar('.') && sNum[strI]>=QChar(',')))
                        {
                              QMessageBox::critical(this, "Ошибка", "В
матрицах обнаружены недопустимые символы.", "Продолжить");
                              return 1;
                        }
                  }
            }
      return 0;
}

// Добавление новой записи в таблицу
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 == false)
                  sTmp_old.setNum(tmp_old.getElement(0,i));
            if(isFirstStartSave == true)
                  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));
      }
}

// Реализация функции X*
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));
}

void DynamicSystem::getX_st_old(int t, int index)
{
      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));
}

// Получение координат X0, X1, X2
void DynamicSystem::getX(int t)
{
      matrix pM1(systemSize,systemSize);
      matrix pM2(systemSize,systemSize);

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

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

```
	tmp1 = tmp1 - B;
	tmp1.getInverse();
	tmp2 = tmp2 - B;
	tmp1 = tmp1*tmp2;
	tmp5 = tmp1;
	tmp1.getPow(t);
	pM1 = tmp1*X0;
	tmp3 = tmp3 - A;
	tmp3.getInverse();
	tmp4 = tmp4 - B;
	tmp3 = tmp3*tmp4;
	tmp5.getPow(t-1);
	tmp6 = tmp6 - B;
	tmp6.getInverse();
	tmp6 = tmp6*C0;
	pM2 = tmp3*tmp5;
	pM2 = pM2*tmp6;

	for(int i=0; i<systemSize; i++)
		X.setElement(i, 0, pM1.getElement(i, 0)+Xst.getElement(i, 0)-
pM2.getElement(i, 0));
}

void DynamicSystem::getX_old(int t)
{
	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));
}

// Проверка, являются ли все матрицы продуктивными
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);

	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;
}

// Сохранение старых данных
void DynamicSystem::saveOldData()
```

```
{
      //matrix tmp(*this);
      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()
{
      // Задаем размер основных матриц
      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);
      // Задаем размер резервных матриц для сохранения пред. операции
      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);
      // Задаем кол-во строк и столбцов в виджетах матриц
      matrixA->setColumnCount(systemSize);
      matrixA->setRowCount(systemSize);
      matrixB->setColumnCount(systemSize);
      matrixB->setRowCount(systemSize);
      matrixC0->setRowCount(systemSize);
      matrixX0->setRowCount(systemSize);
      // Задаем размеры секций (ячеек) после ресайза
      QFont matrixFont("Tahoma", 8); //24/systemSize);
      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);
      }
      // Очищаем секции (ячейки) от значений
      matrixA->clear();
      matrixB->clear();
      matrixC0->clear();
      matrixX0->clear();
      // Очищаем графическую область
      graphArea->isGraphBuild = true;
      graphArea->updateGL();
      // Очищаем таблицу результатов
      tableRowCount = 0;
      tableArea->setRowCount(tableRowCount);
      tableArea->setColumnCount((2*systemSize)+1);
      // Создаем новые столбцы в случае изменения размера
      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);
      // Возвращаем все регистры в начальное состояние
      isChangable = true;
      isSave = false;
      isUsedFeedback = false;
      isFirstStart = true;
      isFirstStartSave = true;
      canSave = false;


}
```

```
// Слот обработки вкл\выкл комбобокса с выбором функций
void DynamicSystem::toggleFuncList()
{
	if(functionList->isEnabled() == true)
		functionList->setEnabled(false);
	else
		functionList->setEnabled(true);
}

bool DynamicSystem::isFuncEnabled()
{
	if(functionList->isEnabled() == true)
		return true;
	else return false;
}

// Слот обработки вкл\выкл редактирования матрицы A
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;
	}
}

// Инкремент значений матрицы A
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));
                  }
      }
}

// Декремент значений матрицы A
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));
                  }
      }
}

// Инкремент значений матрицы B
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));
                  }
      }
}

// Декремент значений матрицы A
```

```
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));
			}
	}
}

// Обработка события ввода значения координаты по оси X
void DynamicSystem::resizeX()
{
	QString tmp;
	int X0, X1;
	bool ok = TRUE;
	X0 = editX0->text().toInt(&ok, 10);
	if(ok == FALSE)
	{
		QMessageBox::warning(this, "Ошибка", "Начальная координата не
является целым числом", "Продолжить");
		return;
	}
	X1 = editX1->text().toInt(&ok, 10);
	if(ok == FALSE)
	{
		QMessageBox::warning(this, "Ошибка", "Конечная координата не
является целым числом", "Продолжить");
		return;
	}
	if((X0 < 0) || (X1 < 0))
	{
		QMessageBox::warning(this, "Ошибка", "Ввиду неотрицательности
решений невозможно задать отрицательную координату\n(должна быть больше
0)", "Продолжить");
		return;
	}
	if(X0 == X1)
	{
		QMessageBox::critical(this, "Ошибка", "Начальная и конечная
координаты по оси X совпадают. Невозможно построить график", "Продолжить");
	   return;
	}
	if(X0 > X1)
	{
```

```
            QMessageBox::warning(this, "Предупреждение", "Координаты будут
инвертированы (Начальная координата больше конечной)", "Продолжить");
            tmp = editX0->text();
            editX0->setText(editX1->text());
            editX1->setText(tmp);
      }
}

// Обработка события ввода значения координаты по оси Y
void DynamicSystem::resizeY()
{
      QString tmp;
      int Y0, Y1;
      bool ok = TRUE;
      Y0 = editY0->text().toInt(&ok, 10);
      if(ok == FALSE)
      {
            QMessageBox::warning(this, "Ошибка", "Начальная координата не
является целым числом", "Продолжить");
            return;
      }
      Y1 = editY1->text().toInt(&ok, 10);
      if(ok == FALSE)
      {
            QMessageBox::warning(this, "Ошибка", "Конечная координата не
является целым числом", "Продолжить");
            return;
      }
      if(Y0 == Y1)
      {
            QMessageBox::critical(this, "Ошибка", "Начальная и конечная
координаты по оси Y совпадают. Невозможно построить график", "Продолжить");
         return;
      }
      if(Y0 > Y1)
      {
            QMessageBox::warning(this, "Предупреждение", "Координаты будут
инвертированы (Начальная координата больше конечной)", "Продолжить");
            tmp = editY0->text();
            editY0->setText(editY1->text());
            editY1->setText(tmp);
      }
}


// Обработка события после нажатия кнопки "Выполнить". Получение всех
данных и решение задачи
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);

	// Очищаем таблицу от любых записей
	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);

	// Проверка на валидность матриц A, B, C0 и X0
	if((pharseMatrix(systemSize, systemSize, matrixA)==0) &&
(pharseMatrix(systemSize, systemSize, matrixB)==0) &&
(pharseMatrix(systemSize, 1, matrixC0)==0) && (pharseMatrix(systemSize, 1,
matrixX0)==0))
	{
		// Проверка на валидность введенных координат рендеринга
		if(!((ok0 == FALSE) || (ok1 == FALSE) || (ok2 == FALSE) || (ok3
== FALSE)))
		{
			// Перенос элементов входных матриц из виджетов в систему
			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());
				}
			}
			// Перенос значений масштабирования в систему
			bool ok = TRUE;
```

```
                x0 = editX0->text().toInt(&ok, 10);
                x1 = editX1->text().toInt(&ok, 10);
                if((x0<0) || (x1<0))
                {
                     QMessageBox::critical(this, "Ошибка", "Ввиду
неотрицательности решений невозможно задать отрицательную
координату\n(должна быть больше 0)", "Продолжить");
                     return;
                }
                y0 = editY0->text().toInt(&ok, 10);
                y1 = editY1->text().toInt(&ok, 10);
                if((x0==x1) || (y0==y1))
                {
                     QMessageBox::critical(this, "Ошибка", "Начальная и
конечная координаты совпадают. Невозможно построить график", "Продолжить");
                     return;
                }

                // Если активирован чекбокс - определение номера функции
(0, 1, и т.д.)
                if(isFuncEnabled() == TRUE)
                     funcIndex = functionList->currentIndex();
                else
                     funcIndex = -1;

                // Вектор-столбец, через который передаются значения t, X и
X*
feedbackLabel: matrix tempM(1, (2*systemSize)+1);
                matrix tempM_old(1, (2*systemSize)+1);

                // Задание кол-ва записей в таблице
                tableRowCount = (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];
                }

                // Устанавливаем текущий масштаб для графика
                graphArea->isGraphBuild = true;
                graphArea->setLines(x0, x1, y0, y1);

                //В случае непродуктивности матрицы A, (A-B), (I-B)^(-
1)*(A-B) вводим регулятор и т.о. делаем матрицу продуктивной
                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,
"Предупреждение", "Система асимптотически неустойчива!\nСтроим систему с
матрицей K усиления обратной связи.", "Продолжить");
                              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;
                              // Заменяем значения в области ввода
                              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(isAllProductive() == true)
                  {
                        // Выполнение операций в случае продуктивности матриц
                        int line = 0; // Значение текущей строки
                        for(int t=x0;t<=x1;t++)
                        {
                              tempM.setElement(0,0, float(t));
                              // Получение значений X*
                              getX_st(t, funcIndex);
                              if(isFirstStartSave == false) getX_st_old(t,
funcIndex);
                              // Получение значений X = (X0,X1,X2 ...)
                              getX(t);
                              if(isFirstStartSave == false) getX_old(t);

                              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);
                               // Передаем значения всех точек для визуализации
на графике
                               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
                     {
                          // В случае непродуктивности указываем какая именно
матрица непродуктивна
                          matrix tmpAB(systemSize,systemSize);
                          matrix tmpA(A);
                          matrix tmpB(B);
                          if(tmpA.isProductive() == true)
```

```
                        {
                              if(tmpB.isProductive() == true)
                              {
                                    tmpAB = tmpA - tmpB;
                                    if(tmpAB.isProductive() == true)
                                    {
                                          matrix tmp1(systemSize,systemSize);
                                          matrix tmp2(systemSize,systemSize);
                                          matrix E;
                                          E.setSingle(systemSize);
                                          tmp1 = E - B;
                                          tmp1.getInverse();
                                          tmp2 = tmpAB;
                                          tmp2 = tmp1*tmpAB;
                                          if(!(tmp2.isProductive() == true))
QMessageBox::critical(this, "Ошибка", "Матрица (E-B)^(-1)*(A-B) не
продуктивна!\nПостроение системы недоступно.", "Продолжить");
                                    }
                                    else QMessageBox::critical(this, "Ошибка",
"Матрица A-B не продуктивна!\nПостроение системы недоступно.",
"Продолжить");
                              }
                              else QMessageBox::critical(this, "Ошибка",
"Матрица B не продуктивна!\nПостроение системы недоступно.", "Продолжить");
                        }
                        else QMessageBox::critical(this, "Ошибка", "Матрица A
не продуктивна!\nПостроение системы недоступно.", "Продолжить");
                  }

                  if(isUsedFeedback == false)
                  {
                        // Сообщение о возможности манипуляций с помощью
матрицы K
                        QMessageBox msgProd;
                        QPushButton *incButton =
msgProd.addButton("Збільшити", QMessageBox::ActionRole);
                        QPushButton *decButton = msgProd.addButton("Зменшити",
QMessageBox::ActionRole);
                        QPushButton *eqvButton = msgProd.addButton("Рівність",
QMessageBox::ActionRole);
                        QPushButton *notButton = msgProd.addButton("Не
використовувати", QMessageBox::ActionRole);
                        msgProd.setIcon(QMessageBox::Question);
                        msgProd.setWindowTitle("Вибір матриці k");
                        msgProd.setInformativeText("\n\n1. Для збільшення
значень елементів вектору Xt и X*\n2. Для меншення значень елементів
вектору Xt и X*\n3. Для рівності значень елементів вектору Xt и X*");
                        msgProd.exec();

                        // Обработка возможных событий
(увеличение\уменьшение\равенство)
                        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;
                }
            }
            else
            {
                // Сообщение о продолжении цикла
                QMessageBox msgContinue;
                QPushButton *yesButton = msgContinue.addButton("Да",
QMessageBox::ActionRole);
                QPushButton *noButton = msgContinue.addButton("Нет",
QMessageBox::ActionRole);
                msgContinue.setIcon(QMessageBox::Question);
                msgContinue.setWindowTitle("Матрица усиления обратной
связи k");
                msgContinue.setInformativeText("Продолжить с выбором
матрицы k по заданому условию?");
                msgContinue.exec();

                if(msgContinue.clickedButton() == yesButton)
                {
                    // Повторение выбранной функции (уменьшение-
увеличение-равенство)
                    switch(takeAttr)
                    {
                    case 1:
                        takeIncK();
                        break;
                    case 2:
                        takeDecK();
                        break;
                    case 3:
                        takeEqvK();
                        break;
                    goto feedbackLabel;
                }
            }
            isUsedFeedback = false;
        }
        else
```

```
                QMessageBox::critical(this, "Ошибка", "Координаты заданы
неправильно", "Продолжить");
      }
      isFirstStart = false; // Передаем сигнал о том что все операции были
выполнены хотя-бы один раз
}

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

      if(!(ok == FALSE))
      {
            // Вывод сообщения о подтверждении
            QMessageBox msgClear;
            QPushButton *yesClear = msgClear.addButton("Да",
QMessageBox::ActionRole);
            QPushButton *noClear = msgClear.addButton("Нет",
QMessageBox::ActionRole);
            msgClear.setIcon(QMessageBox::Question);
            msgClear.setWindowTitle("Предупреждение");
            msgClear.setInformativeText("При изменении кол-ва подсистем все
ранее введенные значения будут утеряны.\nПродолжить?");
            msgClear.exec();
            // Если выбрано "ДА" то очистить все области
            if(msgClear.clickedButton() == yesClear)
            {
                  systemSize = editSize->text().toInt(&ok, 10);
                  graphArea->systemSize = systemSize;
                  resizeSystem();
            }
      }
      else
      {
            QMessageBox::critical(this, "Ошибка", "Недопустимые символы при
задании кол-ва подсистем", "Продолжить");
            return;
      }

}

// Обработка события клика и последующего сохранения
void DynamicSystem::saveData()
{
      if(isFirstStart == true)
      {
            QMessageBox::critical(this, "Ошибка", "Нет информации для
сохранения", "Продолжить");
            return;
      }
```

```
	if((isSave == true) && (canSave == true))
	{
		// Операции по сохранению в файл при активном сохранении двух
состояний системы
		QMessageBox::critical(this, "Ошибка", "Сохранение ДВУХ
состояний", "Продолжить");
	}
	else
		// Операции по сохранению в файл при дезактивированом сохранении
предыдущего состояния
		QMessageBox::critical(this, "Ошибка", "Сохранение ОДНОГО
состояния", "Продолжить");
}
```

Файл glwidget.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 **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(int x, int y, float R, float G, float B);
	void drawPointX(int x, int y, float R, float G, float B);
	void drawPointXst_old(int x, int y, float R, float G, float B);
	void drawPointX_old(int x, int y, float R, float G, float B);

protected:
	void initializeGL();
	void paintGL();
```

```
};

#endif
```

Файл glwidget.cpp:

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

GLWidget::GLWidget(QWidget *parent)
      : QGLWidget(parent)
{
      //makeCurrent();
      tX0 = 0;
      tX1 = 5;
      tY0 = 0;
      tY1 = 5;
      //isFirstStart = true;
      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::black);
}

void GLWidget::paintGL()
{
      if(isGraphBuild == true)
            glClear(GL_COLOR_BUFFER_BIT);
      if(isGraphBuild == true)
      {
            drawInfoLinesX();
            drawInfoLinesY();
      }
      if(isGraphBuild == false)
            for(int i=0;i<=tX1-tX0;i++)
            {
                  for(int ix=0;ix<systemSize;ix++)
                  {
```

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

// 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(int x, int y, float R, float G, float B)
{
	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(int x, int y, float R, float G, float B)
{
	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::drawPointXst_old(int x, int y, float R, float G, float B)
{
      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(int x, int 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();
}
```

Файл matrix.h:

```
#ifndef MATRIX_H
#define MATRIX_H

class matrix
{
private:
      float** array;  //элементы матрицы
      int sizeRow;    //кол-во строк
      int sizeCol;    //кол-во столбцов

public:
      matrix();                        //конструктор по умолчанию
      matrix(matrix &m);               //конструктор копирования
      matrix(int row, int col);  //конструктор с заданием размера
      ~matrix();

      void init(float c);
      //инициализация числом
      void initStat(float *p, int size_r, int size_c);      //инициализ.
статич. массивом
      void initDynam(float **p, int size_r, int size_c);
      //инициализ. динамич. массивом

      void setElement(int row, int col, float k);
      float getElement(int row, int col);
```

```
	void setSize(int row, int col); //установка размера
	int getSizeRow();                    //кол-во строк
	int getSizeCol();                    //кол-во столбцов

	float& operator () (int row, int col);     //элемент матрицы
	float& operator [] (int k);     //элемент матрицы-вектора
	matrix& operator = (matrix &m); //присваивание
	matrix& operator + (matrix &m); //сложение матриц
	matrix& operator - (matrix &m); //вычитание матриц
	matrix& operator * (matrix &m); //умножение матриц

	matrix& operator + (float c);   //операции со скалярами
	matrix& operator - (float c);
	matrix& operator * (float c);
	matrix& operator / (float c);

	float getRowSum(int n);    //получить сумму элементов строки
	matrix  getInverse(); //обратная матрица
	matrix     getTranspose(); //транспонирование
	matrix  getPow(int p); //возведение в степень
	matrix& setSingle(int n); //сделать единичной

	bool isNotNegative(); //проверка на неотрицательность
	bool matrix::isDiagonalLower(); //проверка на то, являются ли
диагональные элементы матрицы < 1
	bool isProductive(); //проверка на продуктивность
};

#endif
```

Файл matrix.cpp:

```
#include "matrix.h"

#include <math.h>
#include <stddef.h>

// конструктор по умолчанию
matrix::matrix()
{
	array=NULL;
	sizeRow=0;
	sizeCol=0;
}

// конструктор копирования
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];
     }
     else
          return array[0][0];
}

// перегрузка оператора '[]'
float& matrix::operator [] (int k)
{
     if(sizeRow==1 && k>0 && k<=sizeCol)
          return array[0][k-1];
     else
     {
          if(sizeCol==1 && k>0 && k<=sizeRow)
               return array[k-1][0];
          else
               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::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)
{
	//matrix tmp(sizeRow, sizeCol);
	if(sizeRow==m.sizeRow && sizeCol==m.sizeCol)
		for(int i=0; i<sizeRow; i++)
			for(int j=0; j<sizeCol; j++)
				//tmp.array[i][j]=array[i][j]+m.array[i][j];
				array[i][j]=array[i][j]+m.array[i][j];
	return *this;
	//return tmp;
}

// вычитание матриц
matrix& matrix::operator - (matrix &m)
{
	//matrix tmp(sizeRow, sizeCol);
	if(sizeRow==m.sizeRow&&sizeCol==m.sizeCol)
		for(int i=0; i<sizeRow; i++)
```

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

// 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)
{
        //matrix tmp(*this);
        if(sizeRow==sizeCol)
                for(int i=0; i<sizeRow; i++)
                        //tmp.array[i][i]+=c;
                        array[i][i]+=c;
        return *this;
        //return tmp;
}

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

matrix& matrix::operator * (float c)
{
        //matrix tmp(sizeRow, sizeCol);
```

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

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

/*
matrix operator + (float c, matrix &m)
{
	return m+c;
}

matrix operator - (float c, matrix &m)
{
	return m*(-1)+c;
}

matrix operator * (float c, matrix &m)
{
	return m*c;
}
*/

// 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;
	float A[10][10], L[4][4];
	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(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)
			{
				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() == true)
      {
            if(tmp.isDiagonalLower() == true)
            {
                  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;
            }
            else return false;
      }
      else return false;
}

// 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;
}
```