Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Факультет математики та інформатики
кафедра математичного моделювання

Операції над матрицями та числові методи розв'язування матричних алгебраїчних рівнянь Ріккаті

Кваліфікаційна робота

Рівень вищої освіти – другий (магістерський)

Виконав:

студент 6 курсу, 607 групи

Тодоріко Ілля Сергійович

Керівник:

д.ф.-м.н., доцент Клевчук І.І

До захисту допущено
на засіданні кафедри
протокол № 5 від 26 листопада 2024 р.
Зав. кафедрою _____ проф. Черевко І.М.

Чернівці – 2024

АНОТАЦІЯ

В даній роботі розглядаються задачі, що приводять до розв’язування матричних алгебраїчних рівнянь Ріккаті, зокрема задачі аналітичного конструювання регуляторів та оптимальної стабілізації. Проведено дослідження та зроблена порівняльна характеристика числових методів, за допомогою яких можна розв’язувати матричні алгебраїчні рівняння Ріккаті та матричні рівняння Ляпунова. На основі цих методів розроблено програму в середовищі Visual Studio Code мовою JAVASCRIPT, яка в інтерактивному режимідозволяє знайти розв’язки цих рівнянь.

Ключові слова: матричні алгебраїчні рівняння Рікаті, числові методи, оптимальна стабілізація, Visual Studio Code, JavaScript, алгоритми розв’язування рівнянь.

ABSTRACT

In this work, problems that lead to a solution are considered of Riccati's matrix algebraic equations, in particular, analytical problems design of regulators and optimal stabilization. Conducted research and made comparative characteristics of numerical methods, according to which can be used to solve Riccati's matrix algebraic equations and matrix Lyapunov equations. Based on these methods, a program was developed in the Visual Studio Code environment in the JAVASCRIPT language, which allows youto search for these levels interactively.

Keywords: Riccati matrix algebraic equations, numerical methods, optimal stabilization, Lyapunov equations, analytical design of regulators, Visual Studio Code, JavaScript, equation-solving algorithms, dynamic systems, stability analysis.

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

_____ І.С. Тодоріко

(підпис)

ЗМІСТ

ВСТУП

Теорія керування – це одна із складових природничо-математичних дисциплін, в рамках якої вивчаються закони, принципи і методи управління різними системами, процесами, об’єктами, тощо. При цьому на перший план висуваються такі властивості систем як стійкість та оптимальність, тобто мається на увазі, що система має здатність зберігати свій стан при малих зовнішніх збуреннях та працювати без збоїв.

Різноманітні процеси, що відбуваюся в навколишньому середовищі найчастіше є керованими, тобто протікають різним чином в залежності від конкретного впливу на них керуючої сторони. При цьому природним є прагнення обрати в деякому розумінні оптимальний керуючий вплив, тобто найкращий, в порівнянні з усіма іншими можливими варіантами керування. Саме таким питанням присвячена дана магістерська робота.

В першому розділі дається визначення поняття керованої системи. Розглядаються задачі про аналітичне конструювання регуляторів та оптимальну стабілізацію.

Другий розділ присвячений розгляду методів розв’язування матричних алгебраїчних рівнянь Ріккаті. Розглянуто ітераційний метод Ньютона-Рафсона та метод матричної сигнум-функції для неперервного випадку. Самостійно модифіковано метод матричної сигнум-функції для дискретного рівняння Ріккаті.

В третьому розділі розглянуті оновні принципи JavaScript, React та функціонал веб-додатків

Четвертий розділ – це опис розробленого програмного забезпечення для розв’язування матричних алгебраїчних рівнянь

1. ОПТИМАЛЬНА СТАБІЛІЗАЦІЯ ЛІНІЙНИХ КЕРОВАНИХ СИСТЕМ

Розвиток теорії оптимального керування пов’язаний з ростом вимог до швидкодії і точності різних систем. Збільшити швидкодію можливо лише при правильному розподілі обмежених ресурсів управління, тому це стало одним із основних питань теорії оптимального керування. З іншого боку, побудова систем регулювання високої точності призвела до необхідності врахування при синтезі регуляторів взаємовпливу окремих частин (каналів) системи. Синтез таких складних багатовимірних систем також складає предмет теорії оптимального керування.[1]

На сьогодні побудована математична теорія оптимального керування. На її основі розроблені способи побудови оптимальних за швидкодією систем, і процедури аналітичного конструювання оптимальних регуляторів. Аналітичне конструювання регуляторів, разом з теорією оптимального спостереження (оптимальних фільтрів), створюють набір методів, які широко використовуються при проектуванні новітніх складних систем керування.

Коли система автоматично керована, оцінюється лише якість динамічних процесів і при цьому критерієм (мірою) цієї якості виступає інтегральний показник якості. Такий опис критеріїв якості дозволяє використовувати для знаходження оптимального керування добре розвинений в математиці апарат варіаційного числення.

В сучасній теорії керування значне місце займають питання стабілізації динамічних систем та побудови стабілізуючого закону керування зі зворотнім зв’язком. Задача стабілізації була повністю розв’язана для лінійних систем, які описуються звичайними диференціальними рівняннями. Показано, що в цьому випадку необхідною і достатньою умовою стабілізовності є асимптотична нуль-керованість. Стабілізуюче керування при цьому лінійно залежить від фазового вектора. Довгий час дослідники намагалися узагальнити цей факт на нелінійні автономні системи. Розроблено ряд методів аналізу нелінійних систем за лінійним наближенням.

1.1. Керованість лінійних стаціонарних систем

Розглядаємо лінійну систему

$$\dot{x}(t) = Ax(t) + Bu(t), \quad t \geq t_0, \quad x \in R^n, \quad u \in R^m, \tag{1.1.1}$$

де вектор $x(t)$ визначає стан системи, вектор $u(t) = (u_1,...,u_m)^T$ складається із управляючих сил або компонент керування $u(t)$, $A, B-$ сталі матриці, що мають розміри $n \times n$ і $n \times m$ відповідно.[2] Обмежень на управління не накладається.

Означення 1. Система (1.1.1) називається керованою на відрізку $[t_0, t_1]$, якщо для довільних векторів $\{x_0, x_1\} \subset R^n$ знайдеться таке управління $u(t)$, при якому існує розв'язок системи (1.1.1), що задовольняє умови:

$$x(t_0) = x_0, \; x(t_1) = x_1.$$

Іншими словами, керовану систему із довільного початкового стану x_0 в момент часу t_0 можна перевести в стан x_1 в момент часу t_1 при відповідному виборі управління.

Умови керованості системи (1.1.1) залежать від рангу матриці керованості, що має вигляд

$$K = \left(B, AB,..., A^{n-1}B\right). \tag{1.1.2}$$

Теорема 1. Для керованості системи (1.1.1) необхідно і досить, щоб ранг матриці керованості K був рівний n (розмірності вектора x).

1.2. Керованість лінійних нестаціонарних систем

Наведемо достатні умови керованості лінійних нестаціонарних систем

$$\dot{x}(t) = A(t)x(t) + B(t)u(t), \quad x \in R^n, \quad u \in R^m. \tag{1.2.1}$$

Означення керованості аналогічне означенню для стаціонарних систем. Припускається, що матриці $A(t)$ і $B(t)$ неперервно-диференційовні до $n-1$ порядку включно в околі деякої точки $\tau \in [t_0, t_1]$. [3]Нехай матриці $K_i(t)$ визначаються наступними співвідношеннями:

$$K_1(t) = B(t), \qquad K_i(t) = A(t)K_{i-1}(t) - \frac{dK_{i-1}(t)}{dt}, \qquad i = 2,...,n.$$

Теорема 2. Якщо існує така точка $\tau \in [t_0, t_1]$, в якій ранг матриці $K = (K_1,..., K_n)$ рівний n, то система (1.2.1) керована на інтервалі $[t_0, t_1]$.

1.3. Задачі про аналітичне конструювання регулятора і про оптимальну стабілізацію

Розглядаємо процес керування, який описується векторним диференціальним рівнянням

$$\dot{x} = A(t)x + B(t)u, \qquad t_0 < t < T, \tag{1.3.1}$$

де $x = \{x_1,..., x_n\}$ – фазовий вектор, $u = \{u_1,...,u_r\}$ – вектор компонент керування. Матриці $A(t)_{n\times n}$ і $B(t)_{n\times r}$ – неперервні. Припустимими керуваннями вважаються кусково-неперервні функції $u = u(t),$ причому всі точки розриву функцій $u = u(t)$ (якщо такі є) – першого роду.[4]

Згідно з припущенням кожному такому припустимому керуванню відповідає єдиний розв'язок задачі Коші

$$\dot{x} = A(t)x + B(t)u(t), \qquad t_0 < t < T, \tag{1.3.2}$$

$$x(t_0) = x_0. \tag{1.3.3}$$

Це значить, що кожна задача (1.3.2)-(1.3.3) однозначно визначає розв'язок $x(t),$ і при переході від одного управління $u = u_1(t)$ до іншого $u = u_2(t)$ будуть отримуватись різні задачі (1.3.1)-(1.3.3), тому що цим керуванням будуть відповідати різні праві частини рівняння (1.3.1). У прикладних задачах особливе значення мають припустимі керування у формі $u = u(t, x),$ коли існує залежність не тільки від часу, але і від поточного стану системи.

Щоб краще зрозуміти цей спосіб управління, можна розглянути рух велосипеда. Велосипедист доступними йому засобами (педалі і руль) діє на рух велосипеда, враховуючи поточний стан свого засобу пересування. З математичної точки зору, це означає, що управляюча дія залежить від фазового стану об'єкту, що рухається.

В результаті для опису процесу замість (1.3.2)-(1.3.3) отримується інша задача:

$$x = A(t)x + B(t)u(t,x), \quad t_0 < t < T, \tag{1.3.4}$$

$$x(t_0) = x_0, \tag{1.3.5}$$

в якій припустимим керуванням вважається функція $u = u(t,x)$.

Надалі припускається, що задача (1.3.4)-(1.3.5) має єдиний розв'язок, при довільному заданому векторі x_0. На таких рівняннях і відповідних їм розв'язках задачі (1.3.4)-(1.3.5) визначений функціонал

$$J(u) = x^T(T)Fx(T) + \int_{t_0}^{T}[x^T(t)Q(t)x(t) + u^T(t)C(t)u(t)]dt, \tag{1.3.6}$$

який розглядається як критерій якості системи (1.3.1). В ньому F, Q, C – симетричні матриці, причому F – невід'ємна стала матриця, $Q(t)$ – невід'ємна неперервна матриця, $C(t)$ – додатно-визначена неперервна матриця. Не порушуючи загальності, будемо вважати, що всі ці матриці симетричні. Даний функціонал обмежений знизу. Тому природно поставити задачу про оптимальне керування, яка називається задачею про аналітичне конструювання регулятора.

Задача 1. Потрібно знайти припустиме керування $u = u(t,x)$ таке, щоб на відповідному йому розв'язку рівняння (1.3.4) функціонал J досягав свого найменшого можливого значення при довільному початковому векторі x_0 в умові (1.3.5).

ЇЇ розв'язок можна отримати різними способами. Ми наведемо результат, що отримується при використанні методу динамічного програмування.

За допомогою цього методу отримується наступний результат. Якщо ввести допоміжну функцію

$$S(t,x) = \min_{u(s),t\le s\le T}\left(x^T(T)Fx(T) + \int_{t}^{T}[x^T(s)Q(s)x(s) + u^T(s)C(s)u(s)]ds \right), \tag{1.3.7}$$

то оптимальне управління $u = u_0(t, x)$ разом із цією функцією формують розв'язок рівняння Беллмана:

$$-\frac{\partial S(t,x)}{\partial t} = \min_u \left[x^T Q(t) x + u^T C(t) u + \left(\frac{\partial S(t,x)}{\partial x} \right)^T \{A(t)x + B(t)u\} \right], \tag{1.3.8}$$

де $t_0 \le t < T$, з додатковою умовою

$$S(T, x) = x^T F x, \tag{1.3.9}$$

де

$$\frac{\partial S(t,x)}{\partial x} = \left\{ \frac{\partial S(t,x)}{\partial x_1}, \dots, \frac{\partial S(t,x)}{\partial x_n} \right\},$$

Розв'язок задачі (1.3.8)-(1.3.9) шукають наступним чином.

Спочатку знаходять елемент $u = u_0$, при якому досягається мінімум в правій частині рівняння (1.3.8). Необхідну умову екстремуму можна подати у вигляді

$$u = -\frac{1}{2} C^{-1}(t) B^T(t) \frac{\partial S(t,x)}{\partial x}. \tag{1.3.10}$$

Підставивши знайдене значення u із (1.3.10) в праву частину рівняння (1.3.8), отримаємо рівняння

$$\begin{aligned} -\frac{\partial S(t,x)}{\partial t} = {} & x^T Q(t) x + \frac{1}{2} \left(\frac{\partial S(t,x)}{\partial x} \right)^T A(t) x + \frac{1}{2} x^T A^T(t) \frac{\partial S(t,x)}{\partial x} - \\ & - \frac{1}{4} \left(\frac{\partial S(t,x)}{\partial x} \right)^T B(t) C^{-1}(t) B^T(t) \frac{\partial S(t,x)}{\partial x}, \end{aligned} \tag{1.3.11}$$

з якого потрібно знайти функцію $S(t, x)$, яка задовольняла б ще й умову (1.3.9). Розв'язок такої задачі шукають у вигляді квадратичної форми

$$S(t, x) = x^T P(t) x, \tag{1.3.12}$$

де $P(t)$ – симетрична матриця, яку потрібно знайти.

Із (1.3.12) випливає, що

$$\frac{\partial S(t,x)}{\partial t} = x^T \dot{P}(t) x, \quad \frac{\partial S(t,x)}{\partial x} = 2P(t)x, \quad \left(\frac{\partial S(t,x)}{\partial x} \right)^T = 2x^T P(t). \tag{1.3.13}$$

Останні дві рівності із (1.3.13) випливають з того, що за припущенням матриця P симетрична. Підставляючи значення S і її похідних із (1.3.12) і (1.3.13) в рівняння (1.3.11), отримаємо

$$x^T \dot{P} x = x^T Q(t)x + x^T PA(t)x + x^T A(t)Px - x^T PB(t)C^{-1}(t)B^T(t)Px.$$

Дана рівність повинна виконуватись при довільних x із достатньо малого околу початку координат. Матриця P повинна задовольняти наступне рівняння Ріккаті:

$$\dot{P} + Q(t) + PA(t) + A^T(t)P - PB(t)C^{-1}(t)B^T(t)P = 0, \qquad (1.3.14)$$

Оскільки функція $S(t, x)$ повинна задовольняти умову (1.3.9), то

$$P(T) = F. \qquad (1.3.15)$$

Отже, для пошуку матриці $P = P(t)$ ми маємо задачу Коші (1.3.14)-(1.3.15). Ця матриця визначається допоміжною функцією Беллмана $S(t, x)$ за формулою (1.3.12), функція має бути невід'ємною. Тому матриця $P(t)$ також має бути невід'ємною. Потрібно знайти невід'ємну симетричну матрицю, яка задовольняла б рівняння (1.3.14), початкову умову (1.3.15) і була визначена на $[t_0, T]$.

Припустимо що існує розв'язок такої задачі, який визначений на проміжку $[t_0, T]$. Тоді за формулами (1.3.12) знаходимо функцію $S = S(t, x)$, а далі згідно з формулою (1.3.10) отримаємо керування

$$u(t, x) = -C^{-1}(t)B^T(t)P(t)x\,. \qquad (1.3.16)$$

Отримане керування, очевидно, задовольняє необхідну умову оптимальності (в задачі мінімізації функціоналу J), сформульовану у вигляді рівняння Беллмана.

Описана процедура побудови оптимального управління зводиться врешті до розв'язування рівняння Ріккаті. При цьому нас цікавить лише невід'ємний розв'язок рівняння. Всі інші етапи розв'язання задачі достатньо прості і не потребують певних спеціальних методів.

Розглянемо тепер задачу про оптимальну стабілізацію. Вона формулюється наступним чином.

Нехай керуючий процес описується рівнянням (1.3.1), в якому матриці $A(t)$ і $B(t)$ неперервні при $t_0 < t < \infty$. Припустимими керуваннями є функції $u = u(t)$, які визначені при $t_0 \le t < \infty$ і кусково-неперервні на кожному інтервалі часу $[\tau, T]$ при $t_0 \le \tau < T$.

Кожному такому керуванню поставимо в відповідність розв'язок $x = x(t)$, що задовольняє початкову умову (1.3.3) при достатньо малій величині $|x_0|$. Цей розв'язок може бути обмеженим і необмеженим на $t_0 \le \tau < T$. Може також виявитись, що виконана умова

$$\lim_{t\to\infty} |x(t)| = 0. \tag{1.3.17}$$

В останньому випадку на керуванні $u = u(t)$ визначений функціонал

$$J_0(u) = \int_{t_0}^{\infty} \left[x^T(t) Q(t) x(t) + u^T C(t) u(t) \right] dt, \tag{1.3.18}$$

в якому $Q(t)$ і $C(t)$ – неперервні обмежені симетричні матриці, причому $Q(t)$ припускається невід'ємною, а $C(t)$ – додатно-визначеною. На конкретному управлінні цей функціонал може набувати скінченних значень або бути необмеженим.

Так як і в задачі про аналітичне конструювання регулятора, особливий інтерес становлять управління у формі $u = u(t, x)$. Тоді відповідні фазові траєкторії визначаються задачею Коші

$$\dot{x} = A(t)x + B(t)u(t, x), \quad t_0 < t < \infty, \tag{1.3.19}$$

$$x(t_0) = x_0. \tag{1.3.20}$$

Для нас особливий інтерес становить випадок, коли на кожному керуванні $u = u(t, x)$ рівняння (1.3.19) має тривіальний розв'язок $x(t) \equiv 0$.

Задача 2. В класі припустимих керувань $u = u(t, x)$ потрібно знайти таке керування, щоб

1) відповідний йому тривіальний розв'язок рівняння (1.3.19) був асимптотично-стійкий;
2) на довільному розв'язку $x = x(t)$ задачі (1.3.19)-(1.3.20) при цьому керуванні і достатньо малих значеннях x_0 функціонал (1.3.18) набував найменших значень.

Розв'язок такої задачі отримується тим же методом, що й задачі про аналітичне конструювання регулятора. У підсумку задача зводиться до розв'язування того ж таки рівняння Ріккаті (1.3.4) з додатковою умовою (1.3.15). Єдина відмінність полягає в тому, що розв'язок має бути визначений на півплощині $t_0 \le t < \infty$.

Розглянемо частковий випадок системи (1.3.1), коли матриці A та B сталі. В критерії оптимальності (1.3.18) матриці $Q(t)$ і $C(t)$ також сталі. При розв'язуванні цієї задачі методом динамічного програмування ми знову отримаємо рівняння вигляду (1.3.11). Однак, всі відомі в ньому матриці будуть сталими. Тому його розв'язок можна шукати у вигляді

$$S(t, x) = S(x) = x^T Px, \tag{1.3.21}$$

де P – невідома, симетрична матриця, яку потрібно визначити.

Якщо тепер повторити наступні за формулою (1.3.12) міркування відносно невідомої матриці P, то отримаємо *алгебраїчне рівняння Ріккаті*

$$PA + A^T P - PBC^{-1}B^T P + Q = 0. \tag{1.3.22}$$

2. МЕТОДИ РОЗВ'ЯЗУВАННЯ МАТРИЧНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ РІККАТІ

Матричне диференціальне рівняння Ріккаті відіграє важливу роль в сучасній математиці і її застосуваннях до різних методів і задач природознавства і техніки. Зокрема, в так званій лінійній квадратичній гауссовій проблемі матричне диференціальне рівняння виникає при розв'язанні оптимальних задач управління і фільтрації в системах з неперервним і дискретним часом[5]. В даному розділі розглядаються числові методи розв'язування матричних алгебраїчних рівнянь Ріккаті для неперервного та дискретного випадків.

2.1. Метод матричної сигнум-функції. Неперервний випадок

Матричне алгебраїчне рівняння Ріккаті для неперервного випадку має вигляд

$$PA + A^T P - PBC^{-1}B^T P + Q = 0. \quad [6] \tag{2.1.1}$$

Якщо ввести позначення $H = BC^{-1}B^T$, то рівняння (2.1.1) набуде вигляду

$$PA + A^T P - PHP + Q = 0. \tag{2.1.2}$$

Розв'язок рівняння (2.1.1) або (2.1.2) можна отримати при розв'язанні двоточкової крайової задачі для наступної системи диференціальних рівнянь

$$\begin{pmatrix} \dot{x} \\ \dot{\alpha} \end{pmatrix} = Z \begin{pmatrix} x \\ \alpha \end{pmatrix}; \quad Z = \begin{pmatrix} A & -H \\ -Q & -A^T \end{pmatrix},$$

де α – n-вимірний вектор множників Лагранжа, а шукана матриця P визначає зв'язок між x та α, тобто $\alpha = Px$.

Матриця Z є гамільтоновою, що означає наступне: якщо J – матриця вигляду $J = \begin{pmatrix} 0 & E \\ -E & 0 \end{pmatrix}$, то $J^{-1} Z^T J = -Z$.

Спектр гамільтонової матриці має таку властивість: поряд із кожним власним числом λ міститься власне число $-\lambda$, причому із тією ж кратністю.

Оскільки матриця A має розмірність n, то матриця Z має розмірність $2n$. Матриця Z має n власних значень $-\lambda$ в лівій півплощині, їм відповідає інваріантний підпростір L_-, а також n власних значень λ в правій півплощині – їм відповідає L_+. Справджується така рівність:

$$(\operatorname{sgn} Z)x = \begin{cases} x, x \in L_+, \\ -x, x \in L_- . \end{cases}$$

Нехай $\det(\lambda E - Z)$ – характеристичний многочлен матриці Z. Подамо цей многочлен у вигляді добутку многочленів: $\det(\lambda E - Z) = (-1)^n \Delta(\lambda)\Delta(-\lambda)$, так що нулі многочлена $\Delta(-\lambda)$ лежать в лівій півплощині, а нулі многочлена $\Delta(\lambda)$ – у правій півплощині. Тоді розв'язок рівняння (2.1.1) визначається співвідношенням Басса :

$$\Delta(-Z)\begin{pmatrix} E \\ P \end{pmatrix} = 0. \qquad (2.1.3)$$

Матриця $\operatorname{sgn} Z$ має ті ж власні вектори, що й матриця Z, а власні значення матриці $\operatorname{sgn} Z$ рівні ± 1. Характеристичний многочлен матриці $\operatorname{sgn} Z$ має вигляд $\det(\lambda E - \operatorname{sgn} Z) = (\lambda + 1)^n (\lambda - 1)^n$.

Із співвідношення Басса (2.1.3) отримаємо

$$(E + \operatorname{sgn} Z)^n \begin{pmatrix} E \\ P \end{pmatrix} = 0,$$

звідси знаходимо

$$WP = -M, \qquad (2.1.4)$$

де $W = \begin{pmatrix} Z_{12} \\ Z_{22} + E \end{pmatrix}$, $M = \begin{pmatrix} Z_{11} + E \\ Z_{21} \end{pmatrix}$, $\operatorname{sgn} Z = \begin{pmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{pmatrix}$.

Розв'язок (2.1.1) має вигляд:

$$P = -W^+ M, \qquad (2.1.5)$$

де W^+ – псевдообернена матриця до W. Алгоритм розв'язування рівняння Ріккаті зводиться до виконання таких кроків:

1) Утворюємо матрицю Z і наближено знаходимо $\operatorname{sgn} Z = X_n$ за формулою

$$X_{n+1} = \frac{1}{2}\left(X_n + X_n^{-1}\right), \qquad X_0 = Z, \qquad n = 0,1,\ldots ,$$

обчислення зупиняємо, коли $\|X_n - X_{n-1}\| < \varepsilon$, де ε – задана точність.

За норму матриці можна взяти, наприклад, величину $\|A\| = \max\limits_{1 \le j \le m} \sum\limits_{i=1}^{n} |a_{ij}|$.

2) Виділяємо блоки матриці $\operatorname{sgn} Z$ і знаходимо W, M. За формулою (2.1.5) знаходимо розв'язок рівняння Ріккаті (2.1.1).

Для знаходження W^+ можна скористатись формулою $W^+ = \left(W^T W\right)^{-1} W^T$.

Приклад.

Розглянемо задачу оптимальної стабілізації для системи вигляду [4, с. 501]

$$\begin{gathered} \dot{x}_1 = x_2, \ \dot{x}_2 = u, \\ J(u) = \int_0^{\infty} \left[x_1^2(s) + 2x_2^2(s) + u^2(s) \right] ds \to \min. \end{gathered} \tag{2.1.6}$$

Матриці, що входять в співвідношення (1.3.1) та (1.3.18), для задачі (2.1.6) мають вигляд

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad Q = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}, \quad C = (1). \tag{2.1.7}$$

Неважко встановити, що матриця

$$H = BC^{-1}B^T = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.1.8}$$

Використаємо алгоритм матричної сигнум-функції:

Побудуємо матрицю Z

$$Z = \begin{pmatrix} A & -H \\ -Q & -A^T \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & -2 & -1 & 0 \end{pmatrix}. \tag{2.1.9}$$

Знаходимо $\operatorname{sgn} Z$

$$\operatorname{sgn} Z = \begin{pmatrix} 0 & 0{,}5 & -0{,}5 & 0 \\ 0{,}5 & 0 & 0 & -0{,}5 \\ -1{,}5 & 0 & 0 & -0{,}5 \\ 0 & -1{,}5 & -0{,}5 & 0 \end{pmatrix}. \tag{2.1.10}$$

Виділяємо блоки матриці $\operatorname{sgn} Z$ та знаходимо матриці W, M із співвідношення (2.1.4)

$$W = \begin{pmatrix} -0,5 & 0 \\ 0 & -0,5 \\ 1 & -0,5 \\ -0,5 & 1 \end{pmatrix}, \quad M = \begin{pmatrix} 1 & 0,5 \\ 0,5 & 1 \\ -1,5 & 0 \\ 0 & -1,5 \end{pmatrix}. \tag{2.1.11}$$

Знаходимо псевдообернену матрицю W^+

$$W^+ = (W^T W)^{-1} W^T = \begin{pmatrix} -0,6 & -0,4 & 0,8 & 0,2 \\ -0,4 & -0,6 & 0,2 & 0,8 \end{pmatrix}. \tag{2.1.12}$$

За формулою (2.1.5) знаходимо розв'язок рівняння Ріккаті

$$P = -W^+ M = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}. \tag{2.1.13}$$

Тепер знаходимо стабілізуюче керування (1.3.16):

$$u(x) = -C^{-1} B^T P x = \begin{pmatrix} 1 & 2 \end{pmatrix} x = x_1 + 2x_2 .$$

2.2. Метод матричної сигнум-функції. Дискретний випадок

Розглядаємо випадок, коли керована система описується скінченно-різницевим рівнянням

$$x(k+1) = \Phi(k)x(k) + \Gamma(k)u(k), \quad x(0) = x_0, \tag{2.2.1}$$

в якому $x(k)$ і $u(k)$ – вектори фазових координат і компонент керування, $\Phi(k)$, $\Gamma(k)$ – матриці відповідних розмірностей. Потрібно знайти послідовність значень вектора керування $u(k)$, $k \in \{0,1,...,N-1\}$, що при довільному x_0 мінімізує квадратичну форму

$$J(u) = x^T(N) F x(N) + \sum_{k=0}^{N-1} \left(x^T(k) Q(k) x(k) + u^T(k) C(k) u(k) \right). \tag{2.2.2}$$

Розв'язок цієї задачі (рівняння регулятора) має вигляд

$$u(k) = -\left[C(k) + \Gamma^T(k) P(k+1) \Gamma(k) \right]^{-1} \Gamma^T(k) P(k+1) \Phi(k) x(k). \tag{2.2.3}$$

В (2.2.3) симетрична матриця $P(k)$ визначається різницевим рівнянням

$$P(k) = \Phi^T(k) M(k+1) \Phi(k) + Q(k), \tag{2.2.4}$$

$$M(k+1) = P(k+1) - P(k+1)\Gamma(k)\left[C(k) + \Gamma^T(k) P(k+1) \Gamma(k) \right]^{-1} \Gamma(k) P(k+1).$$

і крайовою умовою $P(N) = F.$ Якщо існує $C^{-1}(k),$ то

$$M(k+1) = P(k+1)\left[E + \Gamma(k)C^{-1}(k)\Gamma^T(k)P(k+1)\right]^{-1}.$$

Якщо матриці, що входять до (2.2.1) і (2.2.2) не залежать від k, то при $N \to \infty$ можлива збіжність послідовності $P(k)$ до сталої матриці, що задовольняє рівняння

$$P = \Phi^T\left(P - P\Gamma\left(C + \Gamma^T P\Gamma\right)^{-1}\Gamma^T P\right)\Phi + Q. \qquad (2.2.5)$$

Рівняння вигляду (2.2.5) називають *дискретним алгебраїчним рівнянням Ріккаті*, його розв'язок будемо шукати за допомогою методу матричної сигнум-функції, аналогічно як для неперервного випадку.

Розв'язок P можна отримати при розв'язуванні двоточкової крайової задачі для наступної системи різницевих рівнянь

$$\begin{aligned} x(i+1) &= \Phi x(i) - \Gamma C^{-1}\Gamma^T\lambda(i+1), \\ \lambda(i) &= Qx(i) + \Phi^{-1}\lambda(i+1), i = 0,1,... \end{aligned} \qquad (2.2.6)$$

Тут $x(i)$ і $\lambda(i)$ − вектори фазових координат і множник Лагранжа відповідно, шукана матриця P визначає зв'язок між цими двома векторами: $\lambda(i) = Px(i).$

Нехай Φ^{-1} існує і, для загальності, припустимо, що матриця системи (2.2.6)

$$Z = \begin{vmatrix} \Phi + \Gamma C^{-1}\Gamma^T\left(\Phi^T\right)^{-1}Q & -\Gamma C^{-1}\Gamma^T\left(\Phi^T\right)^{-1} \\ -\left(\Phi^T\right)^{-1}Q & \left(\Phi^T\right)^{-1} \end{vmatrix} \qquad (2.2.7)$$

має різні власні значення, що розташовані всередині одиничного кола. Це гарантується асимптотичною стійкістю замкнутої системи "об'єкт + регулятор". Тоді можна провести діагоналізацію матриці Z у вигляді

$$Z = V\begin{bmatrix} M & 0 \\ 0 & M^{-1} \end{bmatrix}V^{-1}. \qquad (2.2.8)$$

Тут M − діагональна матриця, елементи якої за модулем менші одиниці. Матриця V побудована з власних векторів матриці Z, тобто стовпчики матриці V − це власні вектори $x_1, x_2, \ldots, x_{2n}$ матриці $Z,$ що відповідають власним значенням $\lambda_1, \lambda_2, \ldots, \lambda_{2n}$ цієї матриці. Власні вектори визначають з формули $Zx_i = \lambda_i x_i$ або $(Z - \lambda_i E)x_i = 0,$ xi − власний вектор, $x_i \neq 0.$ Розіб'ємо матриці V і $V^{-1} =: \Lambda$ на квадратні блоки

$$V = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}, \quad \Lambda = \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{bmatrix}. \tag{2.2.9}$$

Тоді відомо, що шуканий розв'язок P матричного рівняння (2.2.5) визначається із наступних матричних рівнянь

$$\Lambda_{21} + \Lambda_{22} P = 0, \tag{2.2.10}$$

або

$$V_{21} - P V_{11} = 0.$$

Факторизуємо характеристичний поліном Z наступним чином

$$\det\left| E\mu - Z \right| = \varphi(\mu)\psi(\mu) \tag{2.2.11}$$

так, щоб корені $\varphi(\mu)$ були за модулем менші одиниці (тоді корені $\psi(\mu)$ будуть за модулем більші одиниці). Використовуючи (2.2.8) і (2.2.9) запишемо матричний поліном $\varphi(Z)$ у вигляді

$$\varphi(Z) = V \begin{bmatrix} M & 0 \\ 0 & M^{-1} \end{bmatrix}_{\varphi} \Lambda = V_{\varphi}(J)\Lambda, \text{ де } J = diag\left(M, M^{-1}\right).$$

Розглянемо добуток матриць

$$\varphi(Z) \begin{bmatrix} E \\ P \end{bmatrix} = V_{\varphi}(J) \Lambda \begin{bmatrix} E \\ P \end{bmatrix}.$$

Із розбиття матриці Λ на блоки (2.2.9) і співвідношення (2.2.10) отримаємо

$$\Lambda \begin{bmatrix} E \\ P \end{bmatrix} = \begin{bmatrix} \Lambda_{11} + \Lambda_{12} P \\ 0 \end{bmatrix}. \tag{2.2.12}$$

Неважко переконатись, що матричний поліном $\varphi(J)$ має вигляд

$$\varphi(J) = \begin{bmatrix} 0 & 0 \\ 0 & \varphi(M^{-1}) \end{bmatrix}. \tag{2.2.13}$$

Отже, із (2.2.12) і (2.2.13) отримаємо рівність

$$\varphi(Z) \begin{bmatrix} E \\ S \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{2.2.14}$$

яку називають співвідношенням Басса для рівняння (2.2.5) .

У випадку, коли Φ^{-1} не існує, (2.2.13) не має змісту. Але є інші способи побудови співвідношення Басса (2.2.14).

Справді, використовуючи перетворення (2.2.9), можна показати, що власні вектори матриці

$$Z_1 = \begin{bmatrix} E+\Phi & \Gamma C^{-1}\Gamma^T \\ -Q & E+\Phi^T \end{bmatrix}^{-1} \begin{bmatrix} \Phi-E & -\Gamma C^{-1}\Gamma^T \\ -Q & E-\Phi^T \end{bmatrix} \tag{2.2.15}$$

і системи (2.2.6) збігаються, а власні значення Z_1 розташовані симетрично відносно уявної вісі. Тоді можна провести діагоналізацію матриці Z_1 наступним чином:

$$Z_1 = V \begin{bmatrix} M_1 & 0 \\ 0 & -M_1 \end{bmatrix} \Lambda, \tag{2.2.16}$$

де M_1 – діагональна матриця, власні значення якої лежать в лівій півплощині.

Факторизуємо характеристичний многочлен Z_1 у вигляді

$$\det|E\mu - Z_1| = \varphi_1(\mu)\varphi_1(-\mu), \tag{2.2.17}$$

де $\varphi_1(\mu)$ має строго від'ємні дійсні частини. Використовуючи (2.2.10) і (2.2.16), легко переконатись, що P задовольняє співвідношення Басса

$$\varphi_1(Z_1)\begin{bmatrix} E \\ P \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.2.18}$$

у випадку, коли Φ^{-1} не існує.

Нехай C^{-1} не існує. Тоді, як відомо [3], систему різницевих рівнянь (2.2.6) можна звести до вигляду

$$\begin{aligned} &Lx(i+1) = L\Phi x(i) - U\lambda(i+1), \\ &\lambda(i) = Qx(i) + \Phi^T\lambda(i+1), \quad i = 0,1,... \end{aligned} \tag{2.2.19}$$

Тут

$$\Gamma = \begin{bmatrix} \gamma \\ \gamma_+ \\ \gamma_- \end{bmatrix}, \; L = \begin{bmatrix} \left(\gamma^T\right)^{-1} C\gamma^{-1} & 0 \\ \pm\gamma\ \gamma^{-1}{}_+ & E \end{bmatrix}, \; U = \begin{bmatrix} E\left(\gamma^T\right)^{-1} & \gamma^T \\ 0 & 0 \end{bmatrix}, \tag{2.2.20}$$

де γ_+ – квадратна матриця, для якої існує обернена. Аналогічно наведеному вище доводиться, що власні вектори матриці

$$Z_2 = \begin{bmatrix} L\Phi + L & U \\ -Q & E+\Phi^T \end{bmatrix}^{-1} \begin{bmatrix} L\Phi - L & -U \\ -Q & E-\Phi^T \end{bmatrix} \tag{2.2.21}$$

в системі (2.2.19) співпадають, а власні значення Z_2 розташовані симетрично відносно уявної вісі.

Факторизуючи характеристичний многочлен Z_2 у вигляді

$$\det\left|E\mu - Z_2\right| = \varphi_2(\mu)\varphi_2(-\mu), \tag{2.2.22}$$

отримаємо співвідношення Басса, коли C^{-1} не існує

$$\varphi_2(Z_2)\begin{bmatrix} E \\ P \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{2.2.23}$$

Корені многочлена $\varphi_2(\mu)$ знаходяться в лівій півплощині.

Зауваження1. Для простоти ми вважали, що матриці Z_1, Z_2, Z_3 мають різні власні значення. Однак, отримані результати мають місце і у випадку, коли ці матриці мають кратні власні значення.

Зауваження2. Якщо існує обернена хоча б для однієї із матриць співмножників P в (2.2.23) (аналогічні міркування правильні для (2.2.18) і (2.2.14)), то в цьому випадку матриця P визначається відповідним рівнянням. В протилежному випадку використовуємо звичайний метод розв'язування систем, в яких кількість рівнянь перевищує кількість невідомих.

Сформулюємо тепер алгоритм матричної сигнум-функції, що базується на використанні співвідношення Басса, для дискретного випадку рівняння Ріккаті.

1) Будуємо матрицю Z за формулою (2.2.7), якщо Φ^{-1} не існує, то будуємо матрицю Z_1 за формулою (2.2.15), якщо C^{-1} не існує, то будуємо матрицю Z_2 за формулою (2.2.21).

2) Наближено знаходимо матрицю $\operatorname{sgn} Z \approx X_{n+1}$, X_{n+1} знаходимо з наступної ітераційної процедури:

$$X_0 = (Z - E)^{-1}(Z + E),$$

$$X_{n+1} = \frac{1}{2}\left(X_n + X_n^{-1}\right).$$

Процедуру продовжуємо, поки не буде виконуватись нерівність

$$\left\|X_{n+1} - X_n\right\| < \varepsilon,$$

де ε – задана точність.

Зазначимо, що якщо використовувались формули (2.2.15) чи (2.2.21), то початкове наближення вибирається $X_0 = Z_1$ чи $X_0 = Z_2$ відповідно.

3) Розбиваємо матрицю $\operatorname{sgn} Z$ на блоки:

$$\operatorname{sgn} Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix},$$

де Z_{11}, Z_{12}, Z_{21}, Z_{22} – квадратні матриці.

4) З блоків матриці $\operatorname{sgn} Z$ конструюємо матриці M та W наступним чином:

$$M = \begin{bmatrix} Z_{11} + E \\ Z_{21} \end{bmatrix}, \quad W = \begin{bmatrix} Z_{12} \\ Z_{22} + E \end{bmatrix}.$$

5) Розв'язок рівняння (2.2.5) знаходимо за формулою $P = -W^{+}M$, де W^{+} – псевдообернена до W, $W^{+} = \left(W^T W\right)^{-1} W^T$.

Приклад.

Нехай задані наступні матриці, що входять до рівняння (2.2.5)

$$\Phi = \begin{pmatrix} 1 & 0 \\ 2 & \\ 1 & -1 \end{pmatrix}, \ \Gamma = \begin{pmatrix} 0 \\ \\ 1 \end{pmatrix}, \ Q = \begin{pmatrix} 0 & 0 \\ & \\ 0 & 4 \end{pmatrix}, \ C = (3).$$

Φ^{-1} існує, $\det(\Phi) = -0.5$, тому побудуємо матрицю Z за формулою (2.2.7)

$$Z = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & -\frac{7}{3} & 0 & \frac{1}{3} \\ 0 & -8 & 2 & 2 \\ 0 & 4 & 0 & -1 \end{pmatrix}.$$

З точністю $\varepsilon = 0{,}0001$ знаходимо $\operatorname{sgn} Z$:

$$\operatorname{sgn} Z = \begin{pmatrix} -1 & 0 & 0 & 0 \\ -\frac{3}{7} & \frac{1}{2} & 0 & -\frac{1}{4} \\ -\frac{20}{7} & \frac{6}{7} & 1 & \frac{3}{7} \\ \frac{6}{7} & -3 & 0 & -\frac{1}{2} \end{pmatrix}.$$

Виділяємо блоки $\operatorname{sgn} Z$ та будуємо матриці M та W:

$$M = \begin{pmatrix} 0 & 0 \\ -\frac{3}{7} & \frac{3}{2} \\ -\frac{20}{7} & \frac{6}{7} \\ \frac{6}{7} & -3 \end{pmatrix}, \quad W = \begin{pmatrix} 0 & 0 \\ 0 & -\frac{1}{4} \\ 2 & \frac{3}{7} \\ 0 & \frac{1}{2} \end{pmatrix}.$$

Знаходимо псевдообернену до W

$$W^{+} = \begin{pmatrix} 0 & \frac{1}{6} & \frac{1}{2} & -\frac{1}{3} \\ 0 & -\frac{4}{5} & 0 & \frac{8}{5} \end{pmatrix}.$$

Розв'язок P знаходимо за формулою $P = -W^{+}M$:

$$P = \frac{1}{49} \begin{pmatrix} 88 & -84 \\ -84 & 294 \end{pmatrix}.$$

Знайдена матриця $P-$ симетричний і додатно-визначений розв'язок (2.2.3).

2.3. Метод Ньютона-Рафсона для неперервного матричного алгебраїчного рівняння Ріккаті

В даному пункті розглядається ітераційний метод пошуку розв'язку рівняння Ріккаті, що зводиться до послідовного розв'язування матричних рівнянь Ляпунова.

Розглядаємо матричне алгебраїчне рівняння Ріккаті (неперервний випадок)

$$PA + A^{T}P - PBC^{-1}B^{T}P + Q = 0.$$

Цей алгоритм включає наступні етапи:

1. Вибирається закон зворотного зв'язку $u = L_0 x$ (тобто матриця L_0) таким чином, щоб система (1.3.4) із цим зв'язком, була асимптотично стійкою (тобто, щоб всі власні значення матриці $A + BL_0$ знаходилися в лівій півплощині).
2. Знаходиться послідовність симетричних матриць P_j $j \in \{0,1,2,...\}$, що задовольняють матричні рівняння Ляпунова

$$P_j\left(A + BL_j\right) + \left(A^{T} + L_j^{T}B^{T}\right)P_j + Q + L_j^{T}CL_j = 0. \qquad (2.3.1)$$

В цій послідовності $L_j = -C^{-1}B^T P_{j-1}$ (при $j \neq 0$), а матриця L_0 вибирається у відповідності до пункту 1. Стабілізуючий розв'язок рівняння (1.3.22), тобто розв'язок, що забезпечує стійкість замкнутої системи "об'єкт + регулятор", є границею послідовності P_j: $P = \lim_{j\to\infty} P_j$, причому $P_{j+1} \le P_{j.}$

Цей метод (його називають методом Ньютона-Рафсона) має квадратичну збіжність. На думку багатьох дослідників, він є найбільш ефективним методом пошуку стабілізуючого розв'язку рівняння (1.3.22). Однак така оцінка методу буде правильною, якщо використовуються ефективні алгоритми стабілізації системи (1.3.4) (побудова матриці L_0), і розв'язку матричного рівняння Ляпунова. Ці дві задачі досить довго вивчались. Тому обмежимось викладенням найбільш вдалих варіантів розв'язання цих задач. Почнемо із визначення матриці L_0, а методи розв'язування рівнянь Ляпунова будуть наведені в наступному розділі. Нехай пара матриць *A, B*, що входять до рівняння (1.3.4), керована. Тоді матриця

$$L_0 = -B^T Z^{-1} \qquad (2.3.2)$$

стабілізує систему (1.3.4), якщо матриця $Z = Z^T > 0$ задовольняє матричне рівняння Ляпунова

$$-(A+\beta E)Z + Z[-(A+\beta E)]^T = -2BB^T. \qquad (2.3.3)$$

В (2.3.3) скаляр $\beta > \|A\|$ ($\|A\|$ – визначає будь яку норму матриці *A*, *E* – одинична матриця).

Якщо пара матриць *A, B* стабілізуюча (Z^{-1} може не існувати), то (2.3.2) слід замінити наступним співвідношенням

$$L_0 = -B^T Z^+. \qquad (2.3.4)$$

Тут Z^+ визначає псевдообернену матрицю для розв'язку рівняння (2.3.3).

Така процедура вибору L_0 фактично зводить розв'язання рівняння (1.3.22) до розв'язування послідовності рівнянь Ляпунова.

2.4. Порівняльна характеристика методів матричної сигнум-функції та Ньютона-Рафсона

Матричні алгебраїчні рівняння Ріккаті можна розв’язувати різними методами. Більшість з них були спеціально розроблені саме для алгебраїчних рівнянь і мають більшу ефективність ніж методи, розроблені для диференціальних рівнянь та модифіковані для алгебраїчних.

В даній роботі описані класичні ітераційні алгоритми, які зручно реалізуються на комп’ютері. Розглянутий алгоритм матричної сигнум-функції є досить складним в теоретичному плані, але досить простим в реалізації. Він містить один ітераційний процес, кожна ітерація якого вимагає пошуку оберненої матриці.Даний алгоритм з однією ітерацією пошуку має асимптотичну оцінку складності порядку $O(26n^3)$.

Проте, для отримання точного значення сигнум-функції рідко вистачає однієї ітерації, зазвичай їх 5-8, а інколи і 20, все залежить від вхідних даних та очікуваної точності результату. Крім того побудова сигнум-функції збільшує порядок методу вдвічі, оскільки матриця Z із (2.1.3) має порядок $2n$, де n − розмірність матриці A із рівняння (2.1.1). Однак, обчислювальна складність цього алгоритму не є суттєвою проблемою для сучасних комп’ютерів.

Алгоритм методу Ньютона-Рафсона на перший погляд може видатись простішим.Сказано, що даний метод має квадратичну збіжність, однак там враховано тільки кроки самого алгоритму без оцінки складності розв’язування рівнянь Ляпунова.Даний вираз поданий більш конкретизована оцінка методу разом із оцінкою методів розв’язування алгебраїчних рівнянь Ляпунова, середня складність цього алгоритму має порядок $O(16n^3)$.

Зазначимо, що алгоритм методу Ньютона-Рафсона передбачає дві ітераційні процедури, що вже збільшує його складність. Даний алгоритм доцільно використовувати, коли відоме початкове наближення, яке близьке до розв’язку. Тоді кількість ітерацій методу мінімальна і процес швидко збігається.

В розділі 5 описано розроблене програмне забезпечення, в якому реалізовані числові методи розв’язування алгебраїчних рівнянь Ріккаті та Ляпунова, що розглядаються в даній роботі. Був проведений числовий експеримент, тестовий приклад взятий з [4, с.501]. Було зафіксовано час виконання різних алгоритмів. За допомогою методу Ньютона-Рафсона розв’язок був знайдений за 15 мілісекунд, кількість ітерацій методу без врахування ітерацій для розв’язування рівнянь Ляпунова – 4. Метод матричної сигнум-функції показав кращий результат. Розв’язок був знайдений за 14 мілісекунд при всього двох ітераціях.

Як показав числовий експеримент, дані алгоритми дають результат приблизно за однаковий час. Зазначимо, що для пошуку розв’язків рівнянь Ляпунова використовувався модифікований метод матричної сигнум-функції [4, с. 492]. Часова різниця в одну секунду могла бути викликана особливостями реалізації. Для тестової програми вона не є важливою, однак для систем великого порядку, таких як системи керування великим космічними об’єктами, проблема обробки інформації в реальному часі досить актуальна.

3. Розробка веб-сайту з використанням JavaScript та React

Розробка веб-сайту з використанням JavaScript та React передбачає створення інтерактивного та динамічного інтерфейсу користувача. JavaScript забезпечує функціональність і логіку, тоді як React дозволяє ефективно створювати компоненти, управляти станом програми та повторно використовувати код. Такий підхід полегшує розробку масштабованих і сучасних веб-додатків.

3.1 Огляд JavaScript

JavaScript є однією з найпопулярніших мов програмування, що використовується для створення динамічних веб-сторінок та веб-додатків. Вперше випущений у 1995 році, JavaScript став необхідним інструментом для багатьох розробників, оскільки дозволяє створювати інтерактивні елементи, обробляти події, маніпулювати DOM та виконувати асинхронні запити.[7]

Основні особливості JavaScript включають:

Динамічна типізація: JavaScript є слабо типізованою мовою, що дозволяє змінювати типи змінних без явного оголошення типу.

Об'єктно-орієнтоване програмування: JavaScript підтримує об'єктно-орієнтований підхід, де об'єкти виконують роль основних будівельних блоків програми.

Функціональне програмування: JavaScript також має підтримку функціонального програмування, де функції вважаються першокласними об'єктами і можуть передаватись як аргументи та повертатись як значення.

JavaScript має широкий набір вбудованих об'єктів та методів, що спрощують роботу з рядками, масивами, датами, математичними обчисленнями та іншими операціями. Крім того, за допомогою JavaScript можна взаємодіяти з браузером, змінювати стиль та зміст елементів сторінки, обробляти події, виконувати валідацію даних та багато іншого.

JavaScript також є основою для багатьох популярних фреймворків та бібліотек, зокрема React, Angular і Vue.js, які спрощують процес розробки веб-додатків та дозволяють швидше створення складних інтерфейсів користувача.

Огляд React

React є одним з найпопулярніших фреймворків JavaScript для розробки користувацьких інтерфейсів. Він був розроблений компанією Facebook і

випущений у 2013 році.[8] Однією з основних ідей React є використання компонентного підходу, де інтерфейс розбивається на невеликі незалежні компоненти, які можуть бути повторно використані та легко управляються.

Основні переваги React включають:

Віртуальний DOM: React використовує віртуальний DOM для ефективного оновлення лише необхідних елементів на сторінці, що сприяє покращенню продуктивності та швидкості роботи додатків.

Односторінкова архітектура: React дозволяє створювати односторінкові додатки, де перезавантаження сторінки відбувається тільки при необхідності.

Розширюваність: Завдяки великій кількості сторонніх бібліотек та розширень, React надає можливість швидко розширювати функціональність додатків. React також надає можливість використовувати JSX (JavaScript XML), спеціальний синтаксис, який дозволяє описувати компоненти за допомогою коду, схожого на HTML. JSX спрощує розробку і підтримку коду, роблячи його більш зрозумілим та легко читабельним.

Крім того, React має широкий екосистему, яка включає різноманітні інструменти для тестування, маршрутизації, керування станом додатків (наприклад, Redux), анімацій, локалізації та багато іншого. Це робить React потужним інструментом для розробки сучасних веб-додатків.

JavaScript і React є двома важливими інструментами для розробки веб-додатків. JavaScript надає потужні можливості для маніпулювання сторінками, обробки подій та взаємодії з користувачем. React, з свого боку, спрощує процес розробки інтерфейсів, забезпечує ефективне управління станом та покращує продуктивність додатків.

Використання JavaScript та React дозволяє розробникам створювати потужні, масштабовані та ефективні веб-додатки, задовольняючи потреби сучасного інтернету. Знання цих технологій важливо для успішної кар'єри в сфері веб-розробки.

3.2 Архітектура веб-сайту

Архітектура веб-сайту є важливою складовою процесу розробки та визначає структуру, організацію та взаємозв'язки між різними компонентами веб-сайту. Вона визначає як веб-сторінки, функції, дані та інші ресурси будуть організовані та взаємодіяти між собою для досягнення поставлених цілей.

Основні принципи архітектури веб-сайту включають:

Розширюваність: Архітектура повинна бути гнучкою та розширюваною, щоб легко додавати нові функції та компоненти без значних змін усієї системи.

Масштабованість: Архітектура повинна бути здатною до масштабування, щоб забезпечити ефективну роботу веб-сайту навіть при збільшенні навантаження та обсягу даних.

Модульність: Система повинна бути розбита на незалежні модулі, що дозволяє легко управляти та підтримувати окремі компоненти без впливу на решту системи.

Безпека: Архітектура повинна передбачати заходи безпеки, щоб захистити веб-сайт від потенційних загроз та зловмисників.

Ефективність: Архітектура повинна бути ефективною з точки зору продуктивності, швидкості завантаження сторінок та відповіді на запити користувачів.

Один з популярних підходів до архітектури веб-сайту - це Model-View-Controller (MVC). В рамках цього підходу, компоненти веб-сайту розділені на три основні частини:

Модель (Model): Модель представляє дані та бізнес-логіку веб-сайту. Вона відповідає за отримання та збереження даних, виконання розрахунків та логічних операцій. Модель може включати базу даних, API, класи або структури даних, які представляють конкретні об'єкти.

Представлення (View): Представлення відповідає за відображення даних користувачу. Воно включає веб-сторінки, шаблони, графічні елементи та інші компоненти, які візуалізують інформацію для користувача. Представлення зазвичай взаємодіє з моделлю для отримання необхідних даних.

Контролер (Controller): Контролер діє як посередник між моделлю та представленням. Він обробляє запити користувача, взаємодіє з моделлю для отримання або оновлення даних та керує відображенням результатів у представленні.

Окрім MVC, існують й інші архітектурні підходи, які можуть бути використані в розробці веб-сайтів:

Шарова архітектура (Layered Architecture): Веб-сайт може бути розбитий на різні шари, такі як презентаційний шар, бізнес-логіка та доступ до даних. Кожен шар має свою відповідальність та взаємодіє з іншими шарами через добре визначені інтерфейси.

Сервісно-орієнтована архітектура (Service-Oriented Architecture, SOA): Веб-сайт може бути розбитий на набір сервісів, які надають певну функціональність та можуть бути використані багатьма компонентами. Комунікація між сервісами зазвичай відбувається через мережу за допомогою стандартів, таких як REST або SOAP.

Мікросервісна архітектура (Microservices Architecture): Веб-сайт може бути побудований як набір невеликих та незалежних мікросервісів, кожен з яких виконує конкретну функцію або надає певний сервіс. Мікросервіси можуть бути розгорнуті та масштабовані незалежно один від одного.

При розробці архітектури веб-сайту також слід враховувати такі аспекти:

Безпека: Забезпечення безпеки веб-сайту є критично важливим. Це включає захист від атак, шифрування даних, перевірку автентичності користувачів та забезпечення конфіденційності даних.

Масштабованість: Веб-сайт повинен бути здатним до масштабування, щоб витримувати зростаюче навантаження та кількість користувачів. Це може

включати використання розподіленої архітектури, кешування, горизонтального та вертикального масштабування та інші методи оптимізації продуктивності.

Доступність: Веб-сайт повинен бути доступним для різних користувачів, включаючи людей з обмеженими можливостями. Це включає використання стандартів веб-доступності, таких як WAI-ARIA, правильну семантику HTML та врахування потреб різних груп користувачів.

Архітектура веб-сайту є ключовим елементом успішної розробки та експлуатації веб-сайтів. Вона визначає, як компоненти веб-сайту організовані та взаємодіють між собою, забезпечуючи гнучкість, розширюваність, безпеку та продуктивність системи.

При розробці архітектури веб-сайту слід розглядати різні підходи, такі як MVC, шарова архітектура, сервісно-орієнтована архітектура та мікросервісна архітектура. Важливо також враховувати аспекти безпеки, масштабованості та доступності.

Правильно спроектована архітектура веб-сайту допоможе забезпечити стабільну та ефективну роботу веб-сайту, задовольняючи потреби користувачів та досягаючи поставлених бізнес-цілей.

3.3 Розробка функціональності веб-сайту

Розробка функціональності веб-сайту є процесом створення різноманітних функцій та можливостей, які веб-сайт надає користувачам. Це охоплює взаємодію з елементами інтерфейсу, обробку даних, реалізацію функцій, управління станом та багато іншого.

При розробці функціональності веб-сайту важливо враховувати потреби користувачів та бізнес-вимоги. Це включає аналіз та визначення функцій, їхнє проектування, реалізацію та тестування для забезпечення якісної та зручної веб-продуктивності.

Процес розробки функціональності веб-сайту може включати наступні етапи:

Аналіз вимог: Спілкування зі зацікавленими сторонами та збір вимог щодо функцій веб-сайту. Розуміння бізнес-потреб та користувацьких сценаріїв.

Проектування функціональності: Створення структури функціональних блоків та визначення їх залежностей та інтерфейсів. Розробка макетів та схем взаємодії між різними елементами.

Реалізація функціональності: Написання коду, що виконує функції веб-сайту. Це включає розробку клієнтської та серверної логіки, роботу з базою даних, взаємодію зі сторонніми сервісами тощо.

Управління станом: В деяких випадках, особливо при використанні фреймворків, важливо вести контроль за станом веб-сайту та керувати його змінами. Це може охоплювати збереження даних, синхронізацію стану між компонентами та керування подіями.

Тестування та відладка: Перевірка розробленої функціональності на правильність роботи та виявлення помилок. Це включає автоматизоване та ручне тестування, відлагодження помилок та вирішення проблем.

Інтеграція та впровадження: Після успішного розроблення та тестування функціональності, вона повинна бути інтегрована з веб-сайтом та впроваджена в роботу. Це може включати налаштування серверного середовища, розгортання коду та перевірку правильності роботи функцій на живому середовищі.

Підтримка та поновлення: Після впровадження функціональності веб-сайту важливо забезпечити її подальшу підтримку. Це включає виявлення та виправлення помилок, вдосконалення функцій, а також впровадження оновлень та нових можливостей.

Розробка функціональності веб-сайту є складним та багатоетапним процесом. Вона вимагає аналізу вимог, проектування, реалізації, тестування, інтеграції та підтримки функцій. Важливо враховувати потреби користувачів та

бізнес-вимоги, а також дотримуватися найкращих практик розробки програмного забезпечення.

Правильно розроблена функціональність веб-сайту дозволяє користувачам зручно та ефективно взаємодіяти з веб-сайтом, що сприяє досягненню бізнес-цілей та задоволенню користувацьких потреб.

4. Опис програми

Для здійснення операцій над матрицями та розв’язування матричних алгебраїчних рівнянь Рікатті було розроблено веб додаток в межах якого було реалізовано числові методи описані в даній роботі. При розробці було використано середовище Visual Studio Code, мова програмування JavaScript та React Framework.

Веб-додаток на React.js для розв'язування матричних алгебраїчних рівнянь - це програмний інструмент, розроблений з використанням React.js, який дозволяє користувачам ввести матричні рівняння і отримати їх розв'язки.

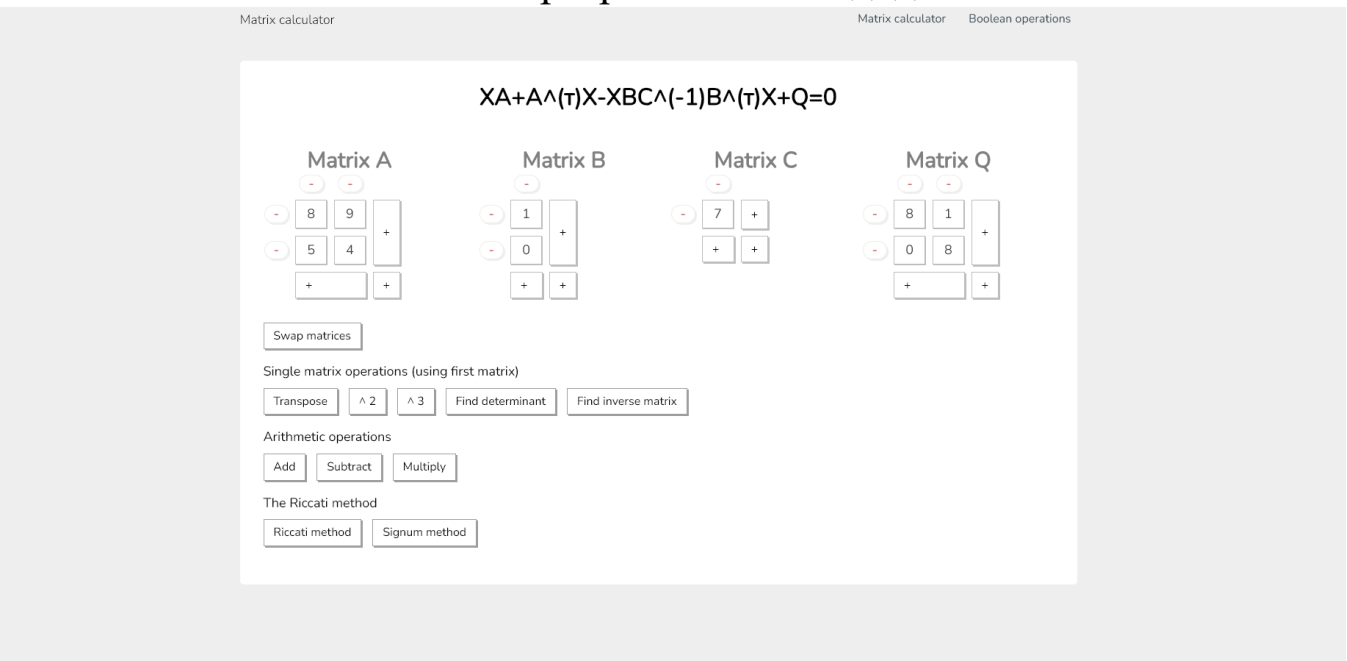
Коли ви відкриваєте додаток, ви побачите інтерфейс, де ви можете ввести матриці для вашого рівняння. Це будуть таблички, де кожен коефіцієнт із залежністю від змінних буде представлений у вигляді елементів матриці.

Після введення матриць ви натискаєте будь-яку операцію або подібний елемент керування. Додаток використовує матричні алгоритми та алгебраїчні методи для обчислення розв'язків рівнянь. Це може включати методи Рікатті та Ляпунова або інші числові методи, які застосовуються для розв'язання матричних рівнянь.

Після обчислення додаток виводить результати на екрані.Для зручності в сприйнятті це буде у вигляді таблички.

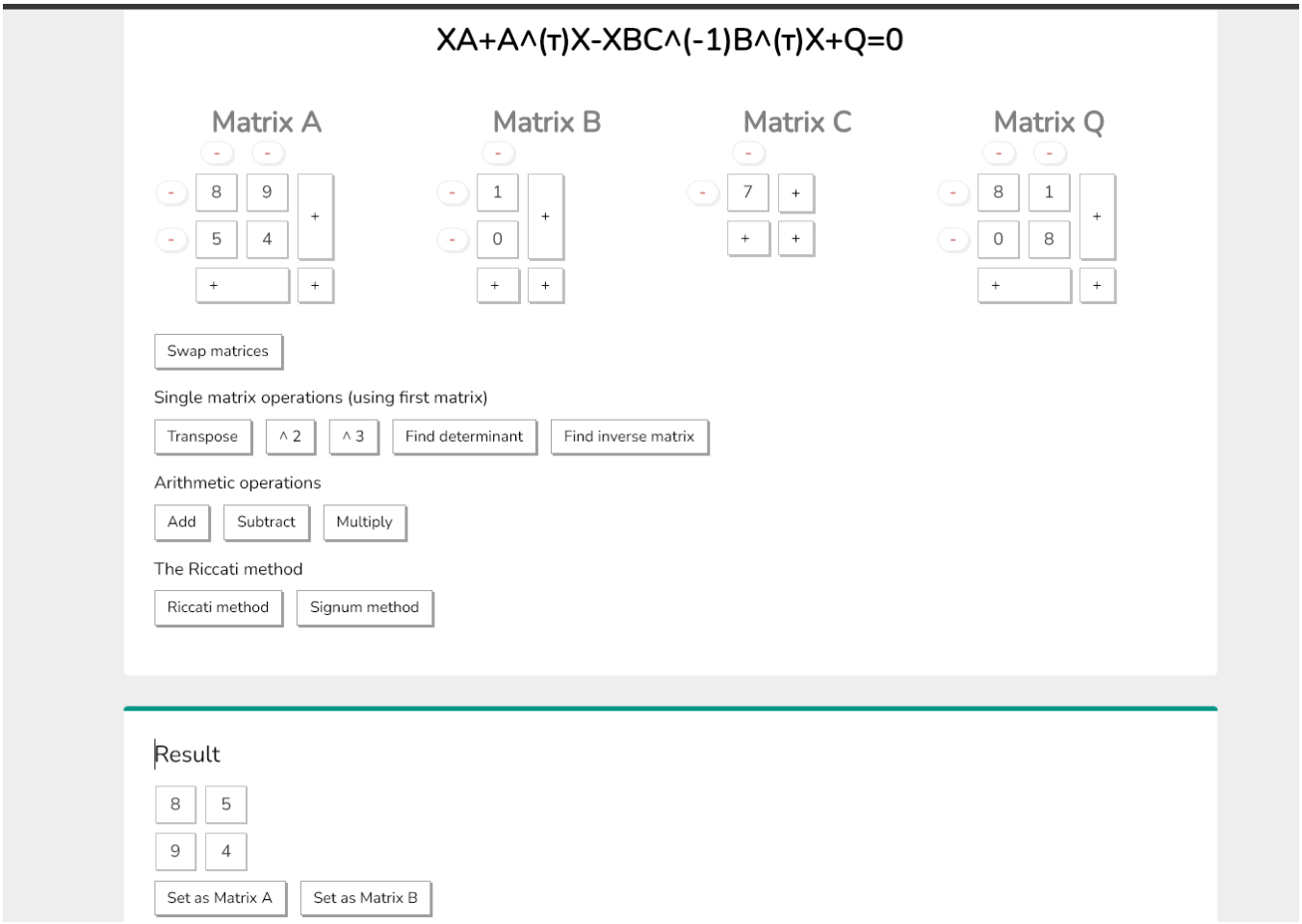
Загалом, веб-додаток на React.js для розв'язування матричних алгебраїчних рівнянь надає зручний інтерфейс користувача для введення рівнянь, обчислення розв'язків та візуалізації результатів. Він допомагає спростити процес розв'язання складних матричних рівнянь, що збільшує продуктивність і зручність для користувача.

Розглянемо розроблений веб додаток



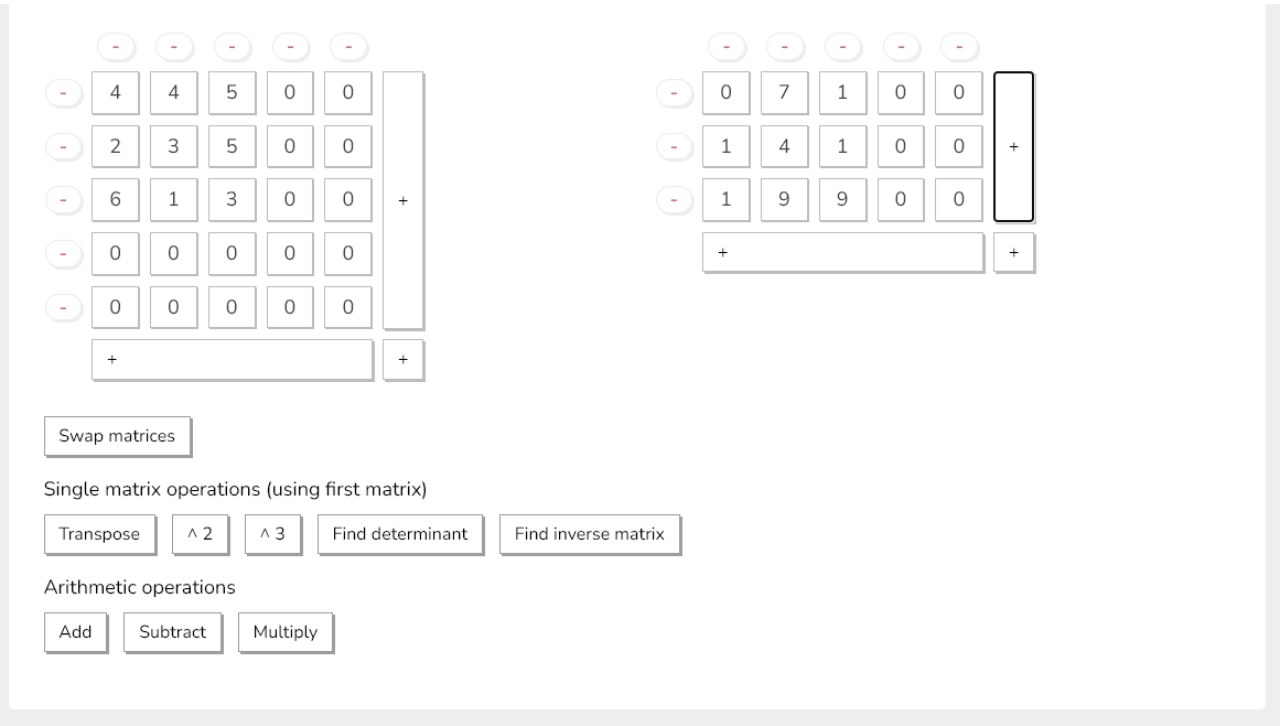
На даному зображенні ми маємо головну сторінку ношого сайту. Де зображено загальні можливості. Після того як ми потрапляємо сюда в нас автоматично заповняються всі матриці(це було зроблено для зручності тестування).

Після натискання операції(наприклад обернену матрицю) :



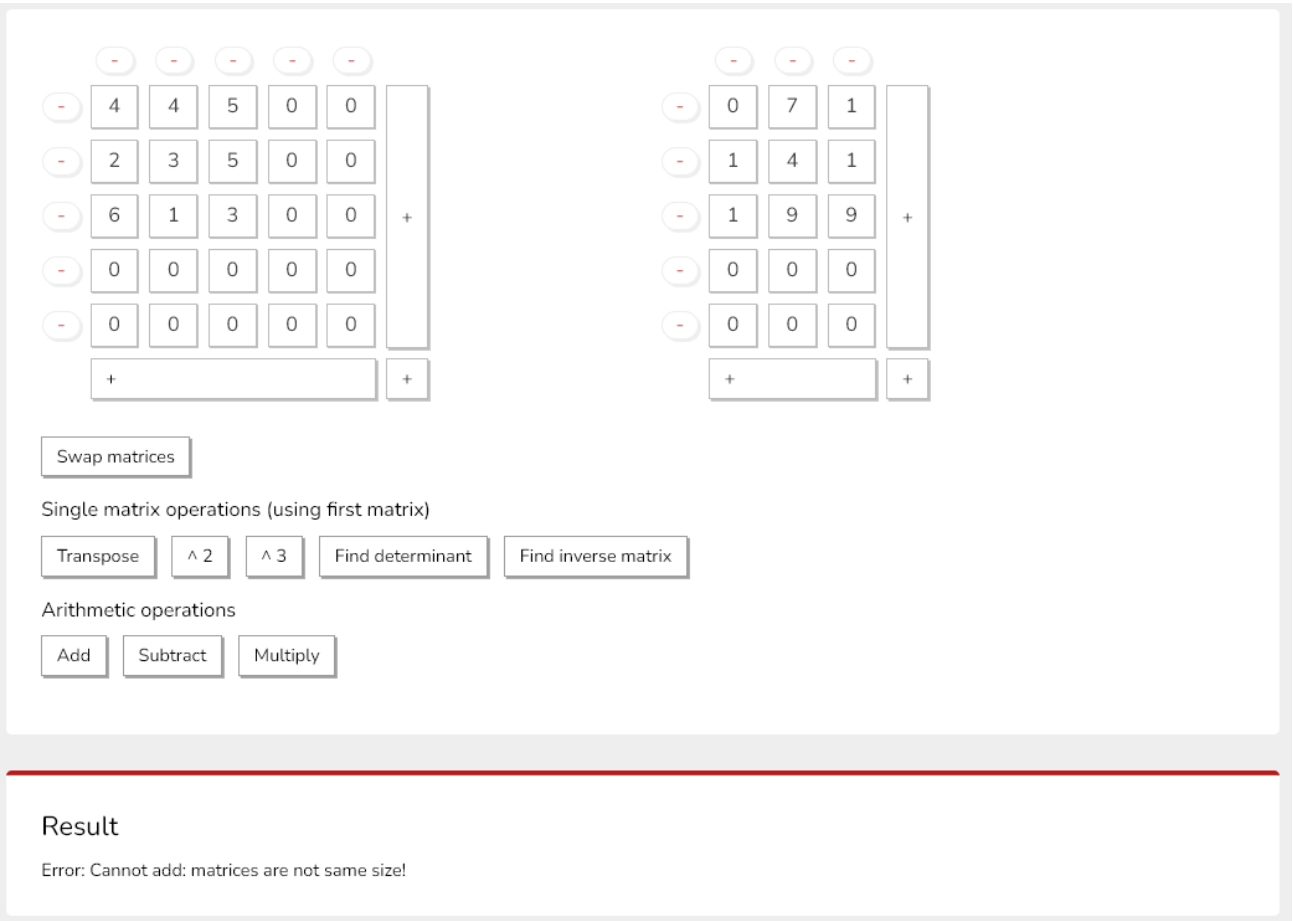
Буде виведено результат виконання який можна замінити в будь-яку з матриць(наприклад для здійснення однієї операції декілька разів або створити ланцюжок операцій)

Також є можливість додавання полів матриць

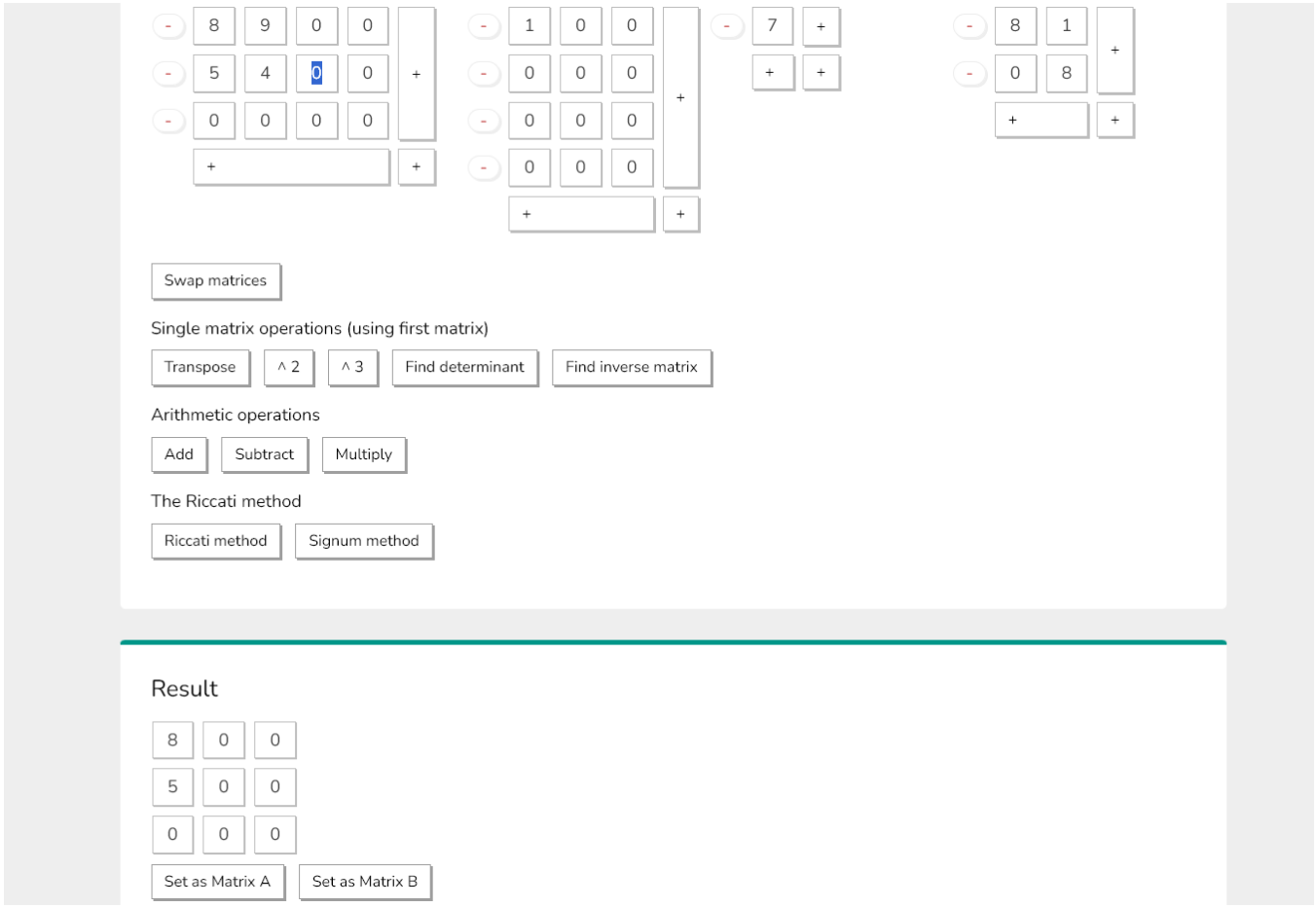


Але вони будуть автоматично всі 0.

Також є перевірки на розмірність для деяких операцій де повинна бути однакова розмірність матриць.

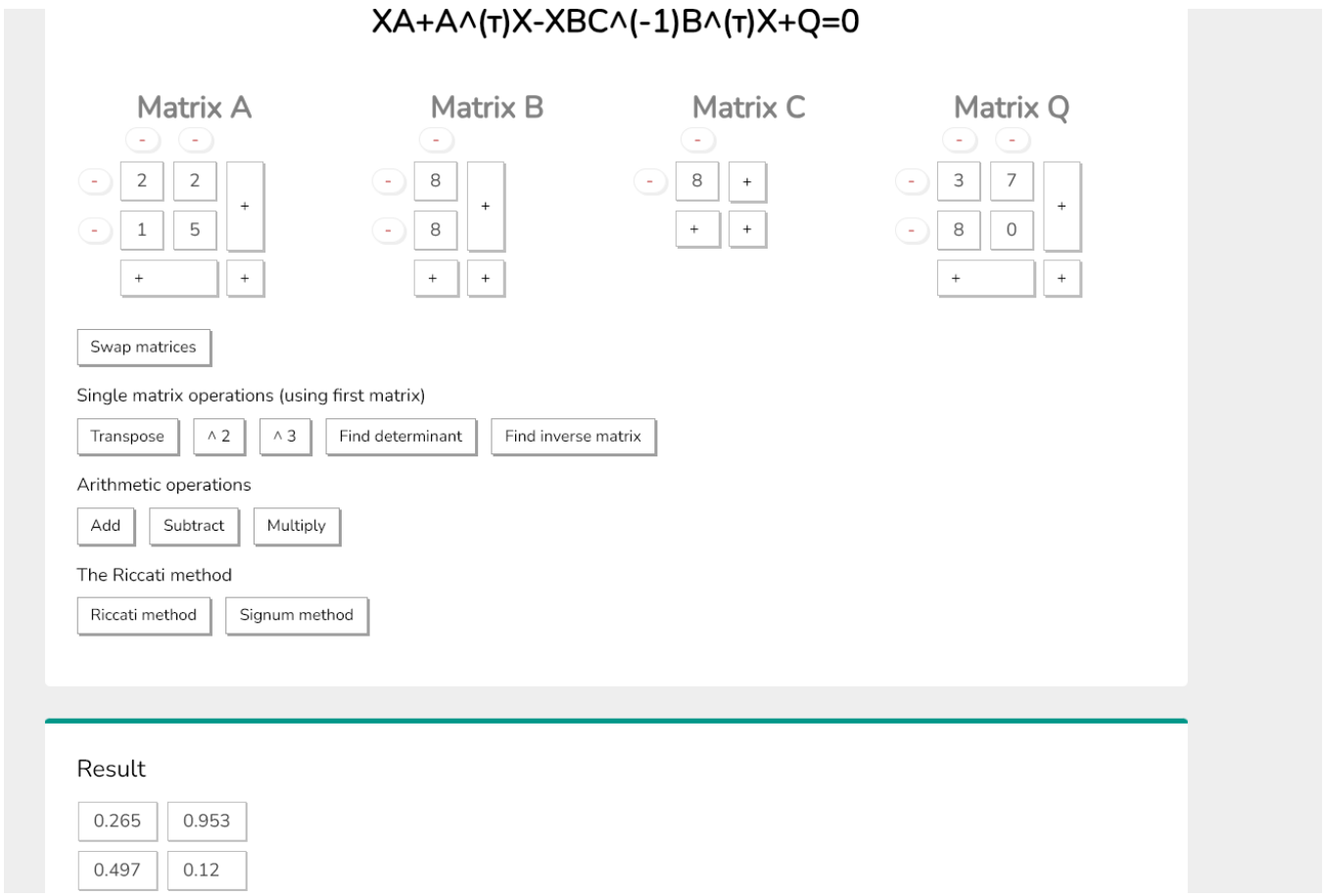


Наприклад додавання!

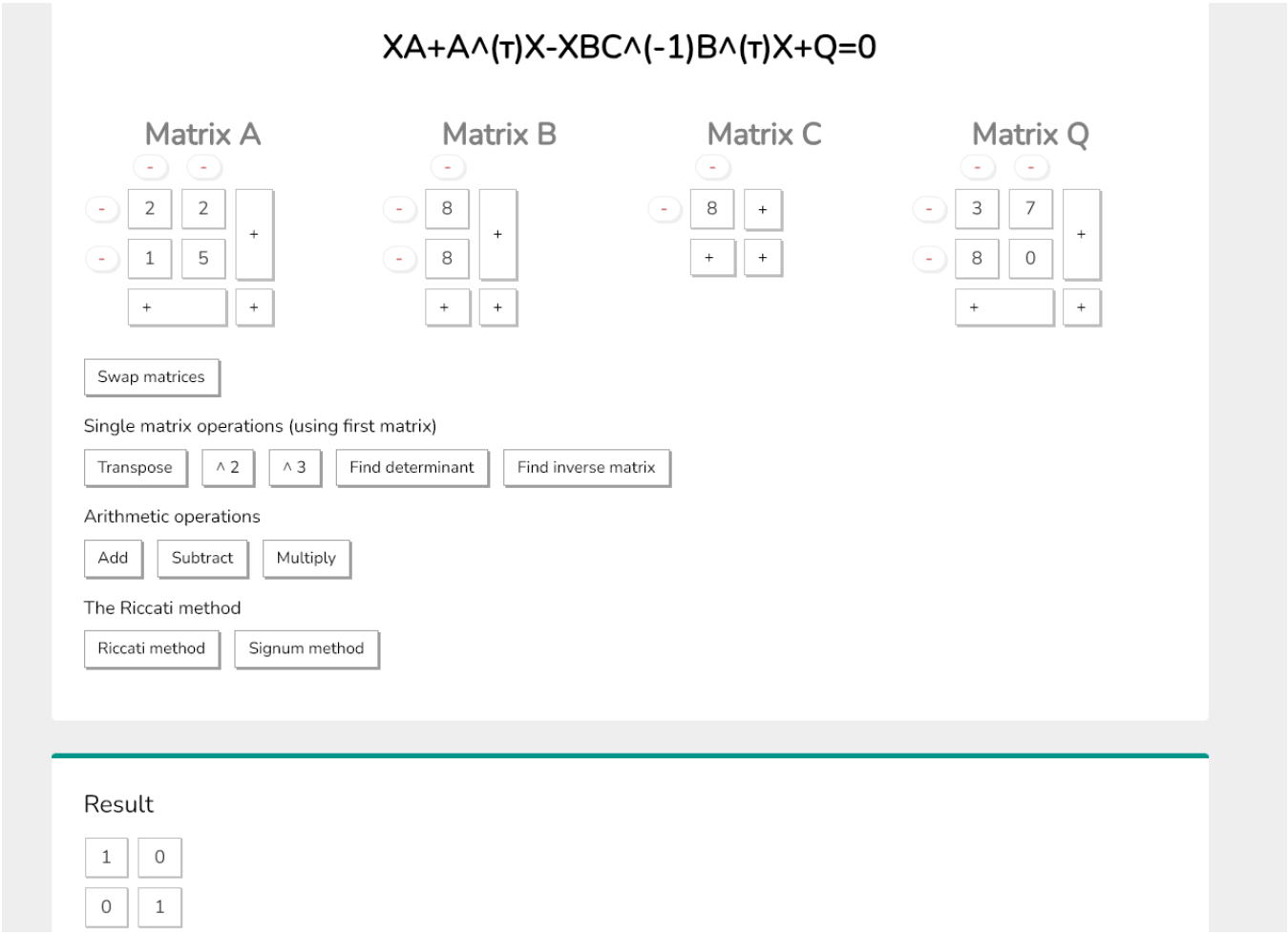


Якщо ж множення тоді результат буде. Так як розмірність рядка першої = розмірності стовбця другої:

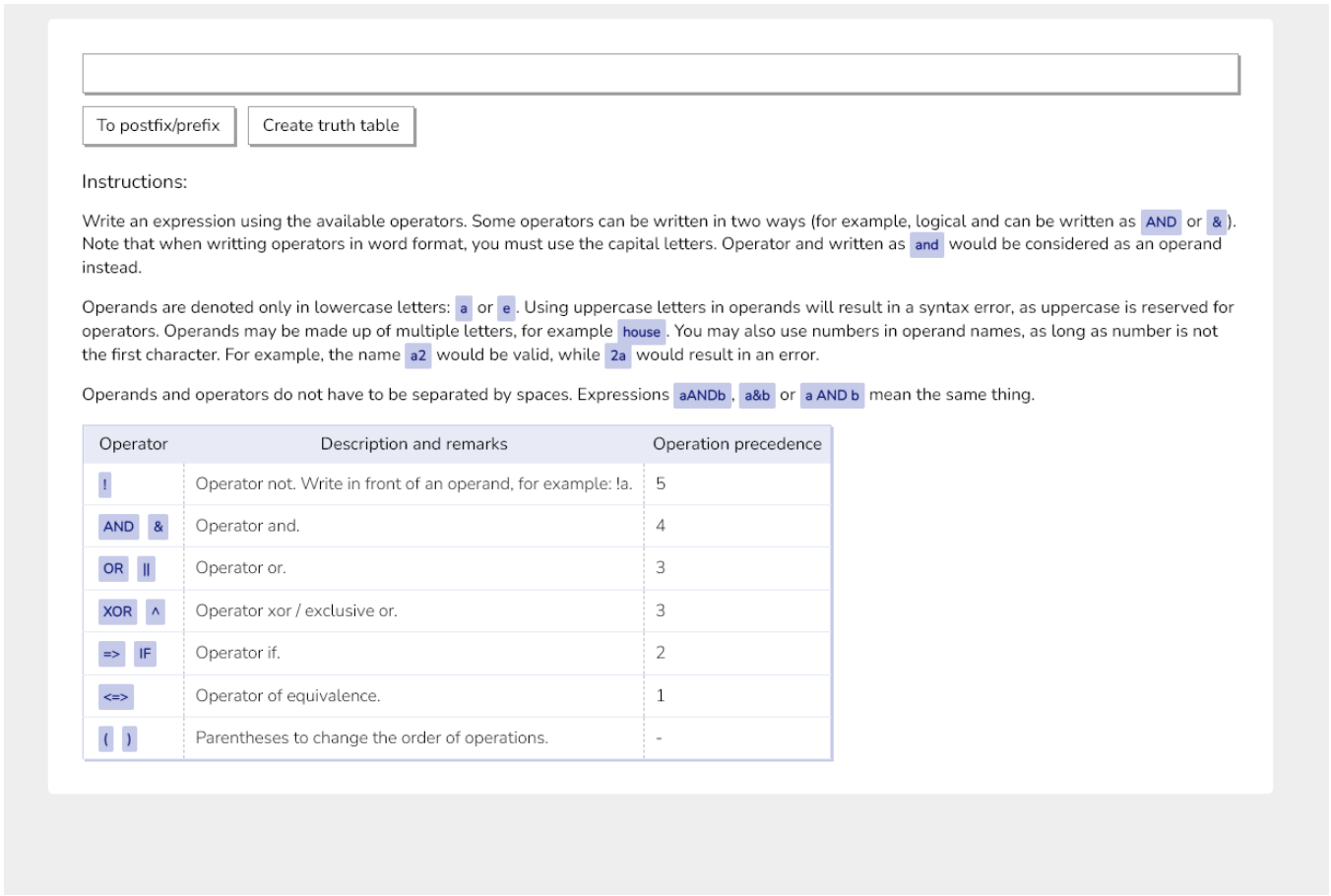
А також метод можна розв’язати рівняння методом Рікатті:



А також метод можна розв’язати рівняння метод сигнум функції:



Також є інформаційна панель:



To postfix/prefix | Create truth table

Instructions:

Write an expression using the available operators. Some operators can be written in two ways (for example, logical and can be written as `AND` or `&`). Note that when writting operators in word format, you must use the capital letters. Operator and written as `and` would be considered as an operand instead.

Operands are denoted only in lowercase letters: `a` or `e`. Using uppercase letters in operands will result in a syntax error, as uppercase is reserved for operators. Operands may be made up of multiple letters, for example `house`. You may also use numbers in operand names, as long as number is not the first character. For example, the name `a2` would be valid, while `2a` would result in an error.

Operands and operators do not have to be separated by spaces. Expressions `aANDb`, `a&b` or `a AND b` mean the same thing.

Operator	Description and remarks	Operation precedence
`!`	Operator not. Write in front of an operand, for example: !a.	5
`AND` `&`	Operator and.	4
`OR` `||`	Operator or.	3
`XOR` `^`	Operator xor / exclusive or.	3
`=>` `IF`	Operator if.	2
`<=>`	Operator of equivalence.	1
`(` `)`	Parentheses to change the order of operations.	-

5. Висновки

В кваліфікаційній роботі застосовується Web-додаток React.js для реалізації алгоритмів розв’язування матричних рівнянь.

Розроблено зручний інтерфейс користувача.

Розроблений додаток надає зручний інтерфейс користувача, що дозволяє ввести матричні рівняння та отримати їх розв'язки шляхом використання числових алгоритмів та методів, що забезпечують задану точність та ефективність обчислень.

Web-додаток можна рекомендувати науковцям ,які займаються дослідженнями матричних рівнянь для розробки ефективних додатків роботи з аналітичними обчисленнями.

6. Список літератури

1. Astrom, K.J., Murray, R.M. Feedback Systems: An Introduction for Scientists and Engineers. - Princeton University Press, 2008. - 408 p.
2. Boyd, S., Ghaoui, L.E., Feron, E., Balakrishnan, V. Linear Matrix Inequalities in System and Control Theory. - SIAM, 1994. - 232 p.
3. Khalil, H.K. Nonlinear Systems. - Prentice Hall, 2002. - 752 p.
4. Lewis, F.L., Vrabie, D.L., Syrmos, V.L. Optimal Control. - Wiley-Interscience, 2012. - 704 p.
5. "Matrix Analysis and Applied Linear Algebra" by Carl D. Meyer – 2000. -479c.
6. http://uk.wikipedia.org/wiki/Рівняння_Ріккаті
7. https://uk.javascript.info/
8. https://uk.legacy.reactjs.org/docs/getting-started.html

7. Додатки

MatrixOperation.js

```
import { createInitialMatrix } from './MatrixModifers'

export const OP_TRANSPOSE = "transpose"
export const OP_DETERMINANT = "determinant"
export const OP_SQUARE = "square"
export const OP_CUBE = "cube"
export const OP_ADD = "add"
export const OP_SUBTRACT = "subtract"
export const OP_MULTIPLY = "multiply"
export const OP_INVERSE = "inverse"
export const OP_RiccatiMatrixSygnum = "RiccatiMatrixSygnum"
export const OP_Solve = "solve"
export const OP_MINUS = "minus"

export default class MatrixOperations {

    static doOperation(operation, matrixA, matrixB, matrixC, matrixQ) {
        let resultMatrix = null

        try {

            switch (operation) {
                case OP_TRANSPOSE:
                    resultMatrix = MatrixOperations.transpose(matrixA)
                    break;
                case OP_ADD:
                    resultMatrix = MatrixOperations.add(matrixA, matrixB)
```

```
                break;


            case OP_RiccatiMatrixSygnum:
                resultMatrix = MatrixOperations.RiccatiMatrixSygnum(matrixA,
matrixB, matrixC, matrixQ)
                break;
            case OP_Solve:
                resultMatrix = MatrixOperations.solve(matrixA, matrixB, matrixC,
matrixQ)
                break;
            case OP_SUBTRACT:
                resultMatrix = MatrixOperations.subtract(matrixA, matrixB)
                break;
            case OP_MULTIPLY:
                resultMatrix = MatrixOperations.multiply(matrixA, matrixB)
                break;
            case OP_SQUARE:
                resultMatrix = MatrixOperations.multiply(matrixA, matrixA)
                break;
            case OP_CUBE:
                resultMatrix =
MatrixOperations.multiply(MatrixOperations.multiply(matrixA, matrixA), matrixA)
                break;
            case OP_DETERMINANT:
                resultMatrix = MatrixOperations.findDeterminant(matrixA)
                break;
            case OP_INVERSE:
                resultMatrix = MatrixOperations.matrixInverse(matrixA)
                break;
            case OP_MINUS:
                resultMatrix = MatrixOperations.findMinus(matrixA)
```

```
            default:
                return [null, "Unimplemented operation"]
        }
    }
    catch (error) {
        return [null, error || "Error while performing an operation with the provided matrices"]
    }

    return [resultMatrix, null]
}

static transpose(matrix) {
    let result = createInitialMatrix(getColCount(matrix), getRowCount(matrix))

    for (let row = 0; row < getRowCount(matrix); row++) {
        for (let col = 0; col < getColCount(matrix); col++) {
            result[col][row] = matrix[row][col]
        }
    }

    return result
}

static findMinus(matrix) {
    let result = createInitialMatrix(getColCount(matrix), getRowCount(matrix))
    let num = -1;
    for (let row = 0; row < getRowCount(matrix); row++) {
        for (let col = 0; col < getColCount(matrix); col++) {
```

```
            result[row][col] = num * matrix[row][col]
        }
    }
    return result;
}

static createMatrix(rows, columns) {
    const matrix = new Array(rows);

    for (let i = 0; i < rows; i++) {
        matrix[i] = new Array(columns).fill(0);
    }

    return matrix;
}
static scalarMatrixAddition(scalar, matrix1, matrix2) {
    const rows = matrix1.length;
    const columns = matrix1[0].length;
    const result = MatrixOperations.createMatrix(rows, columns);

    for (let i = 0; i < rows; i++) {
        for (let j = 0; j < columns; j++) {
            result[i][j] = scalar * (matrix1[i][j] + matrix2[i][j]);
        }
    }

    return result;
}

static matrixNorm(matrix) {
```

```
    const rows = matrix.length;
    const columns = matrix[0].length;
    let sum = 0;

    for (let i = 0; i < rows; i++) {
        for (let j = 0; j < columns; j++) {
            sum += Math.pow(matrix[i][j], 2);
            console.log(sum);
        }
    }

    return Math.sqrt(sum);
}

static createSubmatrix(matrix, startRow, startColumn, rowCount, columnCount) {
    const submatrix = MatrixOperations.createMatrix(rowCount, columnCount);

    for (let i = 0; i < rowCount; i++) {
        for (let j = 0; j < columnCount; j++) {
            submatrix[i][j] = matrix[startRow + i][startColumn + j];
        }
    }

    return submatrix;
}

static matrixAdditionWithIdentity(matrix) {
    const rows = matrix.length;
    const columns = matrix[0].length;
```

```
        for (let i = 0; i < rows; i++) {
            for (let j = 0; j < columns; j++) {
                matrix[i][j] += (i === j) ? 1 : 0;
            }
        }
    }
    static RiccatiMatrixSygnum(A, B, C, Q, epsilon = 0.0001) {
        const H = MatrixOperations.multiply(MatrixOperations.multiply(B,
MatrixOperations.matrixInverse(MatrixOperations.multiply(C, B))),
MatrixOperations.transpose(B));
        const n = A.length;
        const At = MatrixOperations.transpose(A);
        const Z = MatrixOperations.createMatrix(2 * n, 2 * n);



        for (let i = 0; i < n; i++) {
            for (let j = 0; j < n; j++) {
                Z[i][j] = A[i][j];
                Z[n + i][n + j] = -At[i][j];
                Z[i][j + n] = -H[i][j];
                Z[i + n][j] = -Q[i][j];
            }
        }


        let X0 = MatrixOperations.createMatrix(Z);
        let X = MatrixOperations.scalarMatrixAddition(0.5, X0,
MatrixOperations.matrixInverse(X0));
        let norma = 1;


        while (norma > epsilon) {
            X0 = X;
```

```
        X = MatrixOperations.scalarMatrixAddition(0.5, X0,
MatrixOperations.matrixInverse(X0));
        norma = MatrixOperations.matrixNorm(MatrixOperations.subtract(X, X0));
    }

    const Z11 = MatrixOperations.createSubmatrix(X, 0, 0, n, n);
    const Z21 = MatrixOperations.createSubmatrix(X, n, 0, n, n);
    const Z12 = MatrixOperations.createSubmatrix(X, 0, n, n, n);
    const Z22 = MatrixOperations.createSubmatrix(X, n, n, n, n);

    MatrixOperations.matrixAdditionWithIdentity(Z11);
    MatrixOperations.matrixAdditionWithIdentity(Z22);

    const W = MatrixOperations.createMatrix(2 * n, n);
    const M = MatrixOperations.createMatrix(2 * n, n);

    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++) {
            W[i][j] = Z12[i][j];
            W[n + i][j] = Z22[i][j];
            M[i][j] = Z11[i][j];
            M[n + i][j] = Z21[i][j];
        }
    }

    const Wpo = MatrixOperations.multiply(

MatrixOperations.multiply(MatrixOperations.matrixInverse(MatrixOperations.multip
ly(MatrixOperations.transpose(W), W)), MatrixOperations.transpose(W)),
        M
    );
```

```
    return MatrixOperations.scalarMatrixMultiplication(-1, Wpo);
}

static add(a, b) {
    if (!areMatricesSameSize(a, b))
        throw new Error("Cannot add: matrices are not same size!")

    let result = a.map((row, rowIndex) =>
        row.map((el, colIndex) => el + b[rowIndex][colIndex]))

    return result
}

static subtract(a, b) {
    if (!areMatricesSameSize(a, b))
        throw new Error("Cannot subtract: matrices are not same size!")

    let result = a.map((row, rowIndex) =>
        row.map((el, colIndex) => el - b[rowIndex][colIndex]))

    return result
}

static multiply(a, b) {
    if (!canMultiplyMatrices(a, b)) {
        throw new Error("Cannot multiply matrices")
    }

    let result = createInitialMatrix(getRowCount(a), getColCount(b))
```

```
    // Calculate a value for every element in the result matrix
    for (let row = 0; row < getRowCount(result); row++) {
        for (let col = 0; col < getColCount(result); col++) {

            let elementValue = 0

            for (let index = 0; index < getRowCount(b); index++) {
                elementValue += a[row][index] * b[index][col]
            }

            result[row][col] = elementValue
        }
    }

    return result
}

static multiplyByConst(matrix, number) {
    return matrix.map(
        row => row.map(el => (el * number))
    )
}

static findDeterminant(m) {
    if (getColCount(m) !== getRowCount(m)) {
        throw new Error("Cannot find determinant: matrix is not square")
    }

    if (getColCount(m) === 1) {
        return m[0][0]
```

```
        }
        else if (getColCount(m) === 2) {
            return m[0][0] * m[1][1] - m[0][1] * m[1][0]
        }

        // Pick the first row of the matrix
        let firstRow = m[0]
        let result = 0
        firstRow.forEach((val, col) => {
            result += val * Math.pow(-1, col + 1 + 1) * MatrixOperations.getMinor(m, 0,
col)
        })

        return result
    }

    static getMinor(m, rowIndex, colIndex) {

        // Remove the requested row
        let result = m.filter((val, r) => r !== rowIndex)
        // Remove the requested column
        result = result.map(
            matrixRow => matrixRow.filter((el, col) => col !== colIndex)
        )

        return MatrixOperations.findDeterminant(result)
    }



    static matrixInverse(matrix) {
```

```
    const n = matrix.length;
    const augmentedMatrix = MatrixOperations.createAugmentedMatrix(matrix);

    // Apply Gauss-Jordan elimination
    for (let i = 0; i < n; i++) {
      // Find the pivot row
      let pivotRow = i;
      for (let j = i + 1; j < n; j++) {
        if (Math.abs(augmentedMatrix[j][i]) >
Math.abs(augmentedMatrix[pivotRow][i])) {
          pivotRow = j;
        }
      }

      // Swap rows if necessary
      if (pivotRow !== i) {
        MatrixOperations.swapRows(augmentedMatrix, pivotRow, i);
      }

      // Perform row operations to eliminate elements below the pivot
      for (let j = i + 1; j < n; j++) {
        const factor = augmentedMatrix[j][i] / augmentedMatrix[i][i];
        for (let k = i; k < 2 * n; k++) {
          augmentedMatrix[j][k] -= factor * augmentedMatrix[i][k];
        }
      }
    }

    // Perform backward substitution
    for (let i = n - 1; i >= 0; i--) {
```

```
            const pivot = augmentedMatrix[i][i];
            for (let j = i - 1; j >= 0; j--) {
                const factor = augmentedMatrix[j][i] / pivot;
                for (let k = i; k < 2 * n; k++) {
                    augmentedMatrix[j][k] -= factor * augmentedMatrix[i][k];
                }
            }
            // Divide the row by the pivot to make the diagonal element equal to 1
            for (let k = i; k < 2 * n; k++) {
                augmentedMatrix[i][k] /= pivot;
            }
        }

        // Extract the inverse matrix from the augmented matrix
        const inverseMatrix = [];
        for (let i = 0; i < n; i++) {
            inverseMatrix[i] = augmentedMatrix[i].slice(n);
        }

        return inverseMatrix;
    }
static diag(values) {
    const size = values.length;
    const matrix = Array.from({ length: size }, (_, i) =>
        Array.from({ length: size }, (_, j) => (i === j ? values[i] : 0))
    );
    return matrix;
    }

    static qrDecomposition(matrix) {
    const n = matrix.length;
    let Q = this.identityMatrix(n);
    let R = matrix.map(row => [...row]);

    for (let k = 0; k < n - 1; k++) {
        const x = R.slice(k).map(row => row[k]);
```

```
        const alpha = Math.sqrt(x.reduce((sum, val) => sum + val * val, 0));
        const e = Array(x.length).fill(0);
        e[0] = alpha;
        const u = x.map((val, i) => val - e[i]);
        const uNorm = Math.sqrt(u.reduce((sum, val) => sum + val * val, 0));
        const v = u.map(val => val / uNorm);

        const H = this.identityMatrix(n - k).map((row, i) =>
            row.map((_, j) => (i === j ? 1 : 0) - 2 * v[i] * v[j])
        );

        const HFull = this.identityMatrix(n).map((row, i) =>
            row.map((_, j) => (i < k || j < k ? (i === j ? 1 : 0) : H[i - k][j - k]))
        );

        R = this.multiply(HFull, R);
        Q = this.multiply(Q, this.transpose(HFull));
    }

    return { Q, R };
}
    static eigenDecomposition(matrix, iterations = 100) {
    let A = matrix.map(row => [...row]);
    let QFinal = this.identityMatrix(matrix.length);

    for (let i = 0; i < iterations; i++) {
        const { Q, R } = this.qrDecomposition(A);
        A = this.multiply(R, Q);
        QFinal = this.multiply(QFinal, Q);
    }

    const eigenvalues = A.map((row, i) => row[i]);
    const eigenvectors = QFinal;

    return { eigenvalues, eigenvectors };
}

static identityMatrix(size) {
    return Array.from({ length: size }, (_, i) =>
        Array.from({ length: size }, (_, j) => (i === j ? 1 : 0))
    );
}
    static SignumMethod(A) {

        const n = A.length;
        if (n !== A[0].length) throw new Error("Matrix must be square");

        // 1. Знаходимо власні значення і власні вектори
        const { eigenvalues, eigenvectors } = this.eigenDecomposition(A);
```

```
    // 2. Створюємо діагональну матрицю сигнумів
    const signDiag = this.diag(eigenvalues.map(value => Math.sign(value)));

    // 3. Відновлюємо сигнум-матрицю
    const eigvecsInv = this.findInverse(eigenvectors);
    const signMatrix = this.multiply(
        this.multiply(eigenvectors, signDiag),
        eigvecsInv
    );

    return signMatrix;
}


 // Helper functions


 static createAugmentedMatrix(matrix) {
     const n = matrix.length;
     const augmentedMatrix = new Array(n);
     for (let i = 0; i < n; i++) {
         augmentedMatrix[i] = matrix[i].slice();
         augmentedMatrix[i][n + i] = 1; // Augment each row with an identity matrix
```

```
        }
        return augmentedMatrix;
    }

    static swapRows(matrix, row1, row2) {
        const temp = matrix[row1];
        matrix[row1] = matrix[row2];
        matrix[row2] = temp;
    }



    static findInverse(m) {
        if (getColCount(m) !== getRowCount(m)) {
            throw new Error("Cannot find the inverse of a non square matrix")
        }

        let determinant = MatrixOperations.findDeterminant(m)
        let result = createInitialMatrix(getRowCount(m), getColCount(m))

        for (let i = 0; i < getRowCount(m); i++) {
            for (let j = 0; j < getColCount(m); j++) {
                result[j][i] = MatrixOperations.getMinor(m, i, j) * m[i][j] * Math.pow(-1, i
+ j + 2)
            }
        }

        return MatrixOperations.multiplyByConst(result, 1 / determinant)
    }
```

```
}

export const getColCount = matrix => matrix[0].length
export const getRowCount = matrix => matrix.length
export const areMatricesSameSize = (a, b) =>
  getRowCount(a) === getRowCount(b) && getColCount(a) === getColCount(b)
export const canMultiplyMatrices = (a, b) =>
  getColCount(a) === getRowCount(b)
```

Calculator.js

```
import React from 'react';

import MatrixGridController from '../components/Matrix/MatrixGridController'
import SelectButton from '../components/SelectButton'
import { createInitialMatrix } from '../lib/MatrixModifers'
import MatrixOperations, {
  OP_TRANSPOSE,
  OP_DETERMINANT,
  OP_SQUARE,
  OP_CUBE,
  OP_ADD,
  OP_SUBTRACT,
  OP_MULTIPLY,
  OP_INVERSE,
  OP_RiccatiMatrixSygnum,
  OP_Solve
} from '../lib/MatrixOperations'
import { size } from 'lodash';
var x = document.getElementById("f1.png");
```

```
class Calculator extends React.Component {

    constructor( props ) {
        super( props )

        this.state = {
            matrixA: createInitialMatrix( 2, 2 ),
            matrixB: createInitialMatrix(2, 2),
            matrixC: createInitialMatrix(2, 2),
            matrixQ: createInitialMatrix(2, 2),

            resultMatrix: null,
            calculationError: null,
        }

        this.createOpSelectHandler = this.createOpSelectHandler.bind( this )
    }

    swapMatrices() {
        let matrixA = this.state.matrixA
        this.setState({
            matrixA: this.state.matrixB,
            matrixB: matrixA
        })
    }

    selectOperation( operation ) {
        const [result, error] = MatrixOperations.doOperation(operation,
this.state.matrixA, this.state.matrixB, this.state.matrixC, this.state.matrixQ )
```

```
        this.setState({
            resultMatrix: result,
            calculationError: error
        })
    }

    createOpSelectHandler = ( operation ) => {
        return () => this.selectOperation(operation)
    }
    render() {
        return (
            <div>



                <div className='container boxContainer'>
                    <div className='sm:justify-center items-start  flex flex-wrap'>
                        <h1>XA+A^(т)X-XBC^(-1)B^(т)X+Q=0</h1>
                    </div>
                    <div className='sm:justify-center items-start  flex flex-wrap'>
                        <div className='pr-8 mb-8' style={{ flex: 1 }}>
                            <div>
                                <h1 style={{marginTop:'50px', marginLeft: '60px',
fontSize:'30px', color:'gray' }}>Matrix A</h1>
                            </div>
                            <MatrixGridController
                                onChange={(mat) => this.setState({ matrixA: mat })}
                                matrix={this.state.matrixA}>
                            </MatrixGridController>
```

```
                </div>
                <div className='mb-8' style={{ flex: 1 }}>
                    <div className='sm:justify-center items-start  flex flex-wrap'>
                        <h1 style={{ marginTop: '50px', marginLeft: '-30px', fontSize: '30px', color: 'gray' }}>Matrix B</h1>
                    </div>
                    <MatrixGridController
                        onChange={(mat) => this.setState({ matrixB: mat })}
                        matrix={this.state.matrixB}>
                    </MatrixGridController>
                </div>
                <div className='mb-8' style={{ flex: 1 }}>
                    <div className='sm:justify-center items-start  flex flex-wrap'>
                        <h1 style={{ marginTop: '50px', marginLeft: '-30px', fontSize: '30px', color: 'gray' }}>Matrix C</h1>
                    </div>
                    <MatrixGridController
                        onChange={(mat) => this.setState({ matrixC: mat })}
                        matrix={this.state.matrixC}>
                    </MatrixGridController>
                </div>
                <div className='mb-8' style={{ flex: 1}}>
                    <div className='sm:justify-center items-start  flex flex-wrap'>
                        <h1 style={{ marginTop: '50px', marginLeft: '-30px', fontSize: '30px', color: 'gray' }}>Matrix Q</h1>
                    </div>
                    <MatrixGridController
                        onChange={(mat) => this.setState({ matrixQ: mat })}
                        matrix={this.state.matrixQ}>
                    </MatrixGridController>
                </div>
```

```
            </div>


            <ButtonRow>
                <SelectButton onSelect={this.swapMatrices.bind( this )}>Swap
matrices</SelectButton>
            </ButtonRow>


            <ButtonRow title="Single matrix operations (using first matrix)">
                <SelectButton onSelect={this.createOpSelectHandler(
OP_TRANSPOSE )}>Transpose</SelectButton>
                <SelectButton onSelect={this.createOpSelectHandler( OP_SQUARE
)}>^ 2</SelectButton>
                <SelectButton onSelect={this.createOpSelectHandler( OP_CUBE
)}>^ 3</SelectButton>
                <SelectButton onSelect={this.createOpSelectHandler(
OP_DETERMINANT )}>Find determinant</SelectButton>
                <SelectButton onSelect={this.createOpSelectHandler( OP_INVERSE
)}>Find inverse matrix</SelectButton>
            </ButtonRow>


            <ButtonRow title="Arithmetic operations">
                <SelectButton onSelect={this.createOpSelectHandler( OP_ADD
)}>Add</SelectButton>
                <SelectButton onSelect={this.createOpSelectHandler(
OP_SUBTRACT )}>Subtract</SelectButton>
                <SelectButton onSelect={this.createOpSelectHandler(
OP_MULTIPLY )}>Multiply</SelectButton>
            </ButtonRow>
            <ButtonRow title="The Ricatti method">
                <SelectButton
onSelect={this.createOpSelectHandler(OP_RiccatiMatrixSygnum)}> Riccati
method</SelectButton>
            </ButtonRow>
```

```
            </div>
            {/* <ResultContainer matrix={this.state.resultMatrix}
error={this.state.calculationError}></ResultContainer> */}
            {this.createResultContainer()}
        </div>
    )
  }
  createResultContainer() {
    const matrix = typeof( this.state.resultMatrix ) == "number" ?
      [[ this.state.resultMatrix ]] : this.state.resultMatrix
    const error = this.state.calculationError

    if ( error ) {
      return (
        <div className='container boxContainer' style={{ borderTop: '5px solid
#b71c1c'}}>
          <div className='font-2xl mb-4'>Result</div>
          {error.toString()}
        </div>
      )
    }
    else if ( matrix ) {
      return (
        <div className='container boxContainer' style={{ borderTop: '5px solid
#009688'}}>
          <div className='font-2xl mb-4'>Result</div>
          <MatrixGridController
            matrix={matrix}
            readonly>
          </MatrixGridController>
          <ButtonRow>
```

```
                    {/* When setting matrix, we need to copy it first */}
                    <SelectButton onSelect={() => this.setState({ matrixA: [...matrix] })}>Set as Matrix A</SelectButton>
                    <SelectButton onSelect={() => this.setState({ matrixB: [...matrix] })}>Set as Matrix B</SelectButton>
                    <SelectButton onSelect={() => this.setState({ matrixC: [...matrix] })}>Set as Matrix C</SelectButton>
                    <SelectButton onSelect={() => this.setState({ matrixQ: [...matrix] })}>Set as Matrix Q</SelectButton>
                </ButtonRow>
            </div>
        )
    }
    return null;
  }
}

function ButtonRow({ title, children }) {

  return (
    <>
      {!!title ? <div className='font-lg mb-3'>{title}</div> : null}

      <div className='flex flew-wrap mb-5'>
        {children}
      </div>
    </>
  )
}

export default Calculator;
```