Міністерство освіти та науки України

Львівський національний університет імені Івана Франка

Факультет електроніки та комп’ютерних технологій

Кафедра радіофізики та

комп’ютерних технологій

Курсова робота

на тему «Розв’язання рівнянь математичних моделей динамічних систем методом Дорнмана-Принса зі змінним кроком інтегрування»

Виконав:

ст. гр. ФЕІм-11с

_____ Ангелов М.С.

Науковий керівник:

_____доц. Хвищун І. О.

Львів – 2023

Анотація

Реалізовано метод Дорнмана-Принса мовою програмування високого рівня Java версії 1.8. Для створення візуального інтерфейсу використано додаток мови JavaFX, що забезпечує повну крос-платформність через належність пакету до стандартної бібліотеки мови.

Функціональна частина програми полягає в інтегруванні математичних моделей фізичних систем методом Дорнмана-Принса, з автоматичним вибором кроку інтегрування методом трьох зон, який використовує задану користувачем точність для обчислення кожного наступного значення величини кроку.

Annotation

The Dornman-Prince method is implemented in the high-level programming language Java version 1.8. To create a visual interface, i used the JavaFX language application, which provides full cross-platform through the package's affiliation to the standard language library.

The functional part of the program is to integrate mathematical models of physical systems using the Dornman-Prince method, with automatic selection of the integration step by the three-zone method, which uses user-defined accuracy to calculate each subsequent value of the step value.

ЗМІСТ

ВСТУП **3**

1. МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ **5**

1.1. Особливості ММ фізичних систем 5
1.2. Дискретизація рівнянь ММ 7
2. Явний метод Ейлера 10
3. Методи Рунге-Кутта 12
3.1. Оцінка похибки на основі формул різної точності 13
3.2. Комбінація незалежних формул 13
3.3. Комбінація спеціальних формул 14
3.4. Контрольні члени методів Рунге-Кутта 15
3.5. Модифікація Мерсона 15
3.6. Модифікація Фельберга 16
4. Жорсткі системи диференціальних рівнянь 17

2. ПРОГРАМНЕ СЕРЕДОВИЩЕ ECLIPSE ТА МЕТОД ДОРНМАНА-ПРИНСА **18**

2.1. Мова програмування Java 18
2.2. Графічний інтерфейс JavaFX 18
2.3. Середовище розробки Eclipse 19
2.4. Метод Дорнмана-Принса 19
2.5. Алгоритми автоматичного вибору величини кроку h 20

3. АРХІТЕКТУРА ПРОЕКТУ **23**

3.1. Реалізований алгоритм вибору кроку 23
3.2. Створення графічного інтерфейсу 23
3.3. Реалізація проекту мовою Java 25
3.3.1. Клас «Root» 25
3.3.2. Клас «MathematicModel» 25
3.3.3. Клас «Integrator» 26

4. ТЕСТУВАННЯ ПРОГРАМИ **27**

4.1. Однопівперіодний випрямляч 27
4.2. Результати моделювання випрямляча 28
4.3. Візок із демпферами 29
4.4. Результати моделювання візочка 30

ВИСНОВКИ **31**

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ **32**

ДОДАТОК **33**

Вступ

Моделювання слугує методом пізнання людиною навколишнього світу. Здійснюючи експерименти, теоретичні дослідження, навіть обговорювання власних дій і намірів, ми практично займаємось моделюванням. Цілі, завдання, засоби й методи моделювання у цих випадках значно відрізняються один від одного, проте загальна спрямованість залишається єдиною – одержання нового знання шляхом випробування (дослідження) моделі, що слугує замінником реального об’єкта дослідження [1].

Спрощено моделювання можна розглядати як певний експеримент, об’єктом якого слугує або матеріальний аналог досліджуваного об’єкта, який має ту саму фізичну природу, що й досліджуваний об’єкт, або його знакова (математична) модель у формі математичних формул, співвідношень, рівнянь, де завданням моделювання є встановлення нових знань про об’єкти, що описуються цими співвідношеннями.

Результатом розв’язування інженерних (прикладних) задач будь-якого рівня слугують, зазвичай, чисельні оцінки (параметрів пристроїв, процесів, технічних і економічних характеристик тощо), які є наслідком розрахунків, що здійснюються з наближеними початковими даними. Здебільшого, прикладні задачі зводяться до математичних задач, які розв’язують різноманітними чисельними методами. Послідовність розв’язування таких задач можна подати наступними етапами:

1) постановка задачі;
2) створення математичної моделі (формулювання задачі);
3) перевірка моделі на адекватність;
4) побудова розрахункової (обчислювальної) моделі, яка від-повідає обраній математичній моделі;
5) виконання розрахунків за обраною обчислювальною моделлю за заданих (відомих) значень початкових даних;
6) аналіз одержаних результатів.

Під час розробки будь-якої моделі необхідно чітко визначити об’єкт моделювання, ціль моделювання і засоби моделювання.

Метою роботи було розробити мовою Java програмний проект для чисельного інтегрування математичних моделей, які представлені у нормальній формі Коші [1].

У процесі виконання роботи зроблено реалізацію інтегратора, який використовує метод Дорнмана-Принса з автоматичним вибором кроку інтегрування на основі алгоритму трьох зон [1].

1. Математичне моделювання

Необхідність у підвищенні узагальненості та універсальності моделей і методів моделювання вимагає підвищення рівня абстрагування. Щодо цього найабстрактнішим, а, отже, найбільше узагальненим є символьний опис. Тут для опису системи вводять спеціальні символи і встановлюють правила оперування з ними. Сукупність символів і правил їхнього використання (граматика) є абстрактною мовою. Деяке твердження, записане на абстрактній мові, утворює формулу. Прикладами абстрактних мов слугують хімічні формули та математичні формули. Щодо цього математику можна трактувати як цілий клас абстрактних мов з різними граматиками і різними рівнями абстрагування.

Відповідно до сказаного, математична модель – це система математичних співвідношень, яка описує певний фізичний процес чи явище. Для побудови ММ використовують будь-які математичні засоби – теорію множин, диференціальне та інтегральне числення, математичну логіку, теорію ймовірностей, теорію ігор і т. д.

Вибір виду ММ визначають за метою моделювання, оскільки розгляд моделі на певному рівні дає змогу отримати відповіді на певну групу питань, а щоб отримати іншу інформацію – необхідна ММ іншого виду чи іншого рівня абстрагування.

1.1. Особливості ММ фізичних систем

Термін **математичне моделювання** означає, що ми вчимось досліджувати не сам фізичний об’єкт, а його модель. Здебільшого, поведінку фізичних процесів і фізичних систем можна описати ММ у формі **систем звичайних диференціальних рівнянь** (ЗДР) або рівнянь у частинних похідних. Розглядатимемо лише перший випадок. Диференціальними рівняннями називають рівняння, у яких взаємозв’язані похідні від шуканої функції, сама шукана функція та незалежна змінна [1].

Розв’язком диференціального рівняння називають функцію, що має необхідну ступінь гладкості, і яка, за умови підставлення її у диференціальне рівняння, перетворює його в тотожність. Процес визначення розв’язку диференціального рівняння називають його **інтегруванням**.

Якщо в диференціальному рівнянні невідомі функції є функціями однієї змінної $F(x, y, y', y'', \ldots, y^{(n)}) = 0$, то його називають **звичайним диференціальним рівнянням**.

Порядком диференціального рівняння називають максимальний порядок похідної від невідомої функції, що входить у диференціальне рівняння.

Процес математичного моделювання певного фізичного об'єкта можна умовно поділити на три етапи:

1. Побудова його ММ як системи ЗДР у канонічній формі Коші:

$$\dot{x} = \frac{dx}{dt} = f(x,t) \quad , \tag{1.1}$$

або у неявній формі: $F(x, \dot{x}, t) = 0$. (1.2)

Тут x, f, F – вектор-функції.

2. Розв'язання (інтегрування) рівняння (1.1) чи (1.2) на певному проміжку часу $[t_0, t_k]$, тобто визначення сім'ї кривих. Якщо задати початкові умови $x(0) = x_0$, то матимемо розв'язання так званої **задачі Коші** [2, 3, 4].
3. Дослідження (аналіз) розв'язку з метою отримання певних характеристик поведінки об'єкта.

Оскільки існує чимало об'єктів, які досліджують фізики, то кожна предметна область має свій математичний апарат і свою методику побудови ММ своїх об'єктів.

Сформулюємо головні вимоги до процесу побудови ММ фізичних об'єктів:

1. ММ повинна максимально чітко і вичерпно описувати закони взаємодії його елементів між собою, а також правильно враховувати реакцію об'єкта на зовнішні збурення.

2. До ММ застосовують вимоги точності, економічності та універсальності:

- **точність** – це характеристика моделі, яка полягає у найвичерпнішому якісному і кількісному відображенні нею дійсних властивостей об'єкта;

- **економічнішою** буде та модель, яка для досягнення заданої точності вимагає менших затрат ресурсів комп'ютера;
- під **універсальністю** ММ розуміють її застосовність до широкого класу об'єктів і зовнішніх збурень.

Ми розглядаємо математичне моделювання процесів, які описуються системою рівнянь типу (1). Щодо цього функціональна залежність $f(x,t)$, зазвичай, є нелінійною. Задачі механіки, гідравліки, теорії пружності, астрономії, радіо-електроніки зводяться до розв'язання системи виду (1.1).

1.2. Дискретизація рівнянь ММ

Сформовану ММ досліджуваного об'єкта можна розв'язувати як аналітично, так і чисельно. Аналітично розв'язати задачу Коші можна лише у найпростіших випадках. Проінтегрувати систему (1.1), здебільшого, можна тільки чисельними методами. Отож, ми вивчатимемо лише чисельні методи розв'язання задачі Коші. Розглянемо спочатку простий випадок, коли математичною моделлю фізичного процесу слугує одне диференціальне рівняння.

Нехай потрібно розв'язати **задачу про радіоактивний розпад**. Закон радіоактивного розпаду стверджує, що відношення кількості атомів, які розпались за будь-який малий проміжок часу, до кількості атомів, які існували на початку цього проміжку, не залежить від загальної кількості атомів (якщо ця кількість є доволі великою). Адже радіоактивний розпад – це розпад ядер, які у звичайному стані не взаємодіють між собою, оскільки взаємодіють лише електронні оболонки. Отож, імовірність розпаду атома не залежить від того, скільки є атомів. Очевидно, що кількість атомів, які розпались за малий проміжок часу Δt, є пропорційною Δt. Позначимо масу речовини, що ще не розпалася в момент часу t, через $x(t)$. За час Δt розпадеться $x(t)-x(t+\Delta t)$ речовини.

Головний закон радіоактивного розпаду можна записати так: $\frac{x(t)-x(t+\Delta t)}{x(t)} \approx k\Delta t$.

Ця рівність стає все точнішою зі зменшенням проміжку Δt. Коефіцієнт k є постійним і характеризує радіоактивну речовину. Він дорівнює імовірності розпаду окремого атома за одиницю часу за умови вибору цієї одиниці достатньо

малою порівняно з періодом піврозпаду цієї речовини. Перепишемо нашу рівність: $\frac{x(t+\Delta t)-x(t)}{\Delta t} \approx -kx(t)$. (1.3)

Оскільки точність цього рівняння зростає при $\Delta t \to 0$, то здійснюючи граничний перехід, отримаємо точну рівність: $\frac{dx}{dt} = x'(t) = -kx(t)$, (1.4)

яка є диференціальним рівнянням радіоактивного розпаду.

Позначимо через x_0 кількість радіоактивної речовини на момент початку спостереження, тобто $x(0) = x_0$. Це буде початковою умовою для рівняння (1.2). Якщо воно має єдиний розв'язок, то визначаючи цей розв'язок на проміжку часу $[0, t_k]$, де t_k – кінцевий час дослідження, ми промоделюємо процес радіоактивного розпаду.

З теорії диференціальних рівнянь відомо, що розв'язком рівняння (1.4) слугуватиме множина функцій. Обираючи певну початкову умову, ми знайдемо одну функцію $x(t)$ з цієї множини. Проілюструємо, як це можна зробити.

Розділимо рівняння (1.1) на $x(t)$. Отримаємо: $\frac{x'(t)}{x(t)} = -k$. (1.5)

За правилом диференціювання складної функції і, враховуючи, що $(\ln x)' = \frac{1}{x}$, можна переписати рівняння (1.5) у вигляді: $(\ln x(t))' = -(kt)'$.

Оскільки дві первісні однієї функції відрізняються на постійну, отримаємо:

$\ln x(t) = -kt + C_1$.

Звідси, $x(t) = e^{-kt+C_1} = e^{C_1}e^{-kt} = Ce^{-kt}$, де $C > 0$. Підстановка $t = 0$ дає: $C = x_0$. Ми отримали розв'язок диференціального рівняння (1.4) з початковою умовою $x(0) = x_0$: $x(t) = x_0 e^{-kt}$. (1.6)

Розглянемо найпростіший **чисельний метод** (ЧМ) розв'язан-ня **задачі Коші**, яка формулюється так:

Знайти розв'язок диференціального рівняння на інтервалі $[t_0, t_k]$, який задовольняє початкову умову $x(t_0) = x_0$: $\dot{x} = \frac{dx}{dt} = f(x(t), t)$, $x(t_0) = x_0$, $[t_0, t_k]$. (1.7)

Загалом **чисельний розв'язок** задачі Коші полягає у заміні цієї задачі її дискретним (різницевим) аналогом:

$$x_{m+1} = x_m + \varphi(h_{m+1}, h_m, ..., h_{m-k}, x_{m+1}, x_m, ..., x_{m-k}, \dot{x}_{m-k}, \dot{x}_{m+1}, \dot{x}_m, ..., \dot{x}_{m-k}) \quad (1.8)$$

Тут $h_{m+1} = t_{m+1} - t_m$ – **крок інтегрування**. Дискретний аналог (1.8) задачі Коші (1.7) отримуємо, застосовуючи до формули (1.1) один із методів чисельного інтегрування. У виразі (1.8) символом *m* позначено номер кроку інтегрування, а символ *k* визначає "**кроковість**" чисельного методу. Ця величина вказує на те, що інформацію *k* попередніх кроків інтегрування використовують у цьому ЧМ для обчислення значення x_{m+1} . Відповідно, розрізняють однокрокові та багатокрокові чисельні методи.

При заміні (дискретизації) виразом (1.8) задачі (1.7) з певною точністю повинні виконуватись умови якісної відповідності неперервної та дискретної моделей. Ці умови полягають у тому, що задачі (1.7) і (1.8) повинні мати однакові стаціонарні режими та аналогічну структуру фазового простору.

При розробці програм, які реалізують чисельні методи розв'я-зання задачі Коші, використовують поняття точності, стійкості, узгодженості ЧМ, локальної і глобальної похибки дискретизації, жорсткості, обумовленості.

Поняття **точність** і **стійкість** різницевих схем стосуються вимоги досягти малого відхилення значень розв'язків рівнянь математичних моделей від їхніх точних рішень. Точність пов'язана з локальними похибками. Є два джерела виникнення похибок:

- **похибки заокруглення** виникають внаслідок скінченної довжини комірок пам'яті комп'ютера, у які записують двійкові представлення дійсних чисел;
- **похибки дискретизації** полягають у тому, що під час заміни неперервної функції її дискретним аналогом у вигляді рядів ми беремо не безмежну, а скінченну кількість членів ряду.

Стійкість взаємозв'язана із процесом накопичення похибок. Навіть якщо похибки заокруглення і похибки дискретизації є малими, від обчислень буде мало користі, якщо вплив цих малих похибок з часом наростає, тобто розв'язок є нестійким.

2. Явний метод Ейлера

Розглянемо найпростіший ЧМ, який ми можемо застосувати для розв'язання ММ задачі про радіоактивний розпад. Для зручності запишемо його диференціальне рівняння ще раз: $\frac{dx}{dt} = x'(t) = -kx(t)$. (2.1)

Якщо розв'язок цього рівняння має неперервні вищі похідні (хоча другу), то його можна розкласти у ряд Тейлора:

$$x(t+h) = x(t) + \frac{h\,\dot{x}(t)}{1!} + \frac{h^2\,\ddot{x}(t)}{2!} + \frac{h^3\,\dddot{x}(t)}{3!} + \ldots \qquad (2.2)$$

Якщо всі члени ряду (2.2), які містять похідні, що вищі за першу, відкинути, ми отримаємо чисельний метод розв'язання задачі Коші під назвою **явний метод Ейлера** (ЯМЕ). Перейшовши до дискретних відліків часу, одержимо $x(t_1) = x(t_0) + hf(x(t_0), t_0)$ або у загальноприйнятому записі:

$$x_{m+1} = x_m + hf(x_m, t_m), \qquad (2.3)$$

де $h = t_{m+1} - t_m$, m – номер дискретної точки.

З метою дослідження властивостей ЯМЕ розглянемо інтегру-вання цим методом простого модельного диференціального рівняння:

$$\dot{x} = \frac{dx}{dt} = \lambda x(t), \; x(t_0) = x_0. \qquad (2.4)$$

Вище ми зазначали, що аналітичним розв'язком цієї задачі слугує функція $x(t) = x_0 e^{\lambda t}$. Для аналізу впливу похибок на числовий розв'язок рівняння (2.4) проінтегруємо це тестове рівняння ЯМЕ з *h*=Const. Щоб відрізнити точний розв'язок рівняння (2.4) від чисельного розв'язку, який містить похибки, позначимо останній через $u(t)$, а наближене значення початкових умов у точці $t = t_0$ позначимо u_0 [5]. Тоді:

$$u_1 = u_0 + hf(x,t) = u_0 + h\lambda u_0 = u_0(1 + h\lambda);$$

$$u_2 = u_1 + h\lambda u_1 = u_1(1 + h\lambda) = u_0(1 + h\lambda)^2;$$

$$u_{m+1} = u_0(1 + h\lambda)^{m+1}.$$

Нехай початкову умову x_0 введено у пам'ять комп'ютера з похибкою. Позначимо це наближене значення початкової умови через u_0. Тоді величина $e_0 = x_0 - u_0$ слугуватиме похибкою введення початкової умови. Вона може з'явитись або внаслідок "ручного" заокруглення числа при введенні x_0, або внаслідок заокруглення під час запису числа у пам'ять машини, або внаслідок обох причин. Якщо порівняти вирази:

$$u_1 = u_0(1 + h\lambda) \text{ і } x = x_0 e^{\lambda t_1} = x_0 e^{\lambda h};$$

$$u_{m+1} = u_0(1 + h\lambda)^{m+1} \text{ і } x = x_0 e^{(m+1)h\lambda},$$

то можна зробити висновок, що тут експонента апроксимована виразом $(1 + h\lambda)$, тобто у розкладенні експоненти у ряд Тейлора $e^{h\lambda} = 1 + \frac{h\lambda}{1!} + \frac{(h\lambda)^2}{2!} + \frac{(h\lambda)^3}{3!} + \ldots$ було відкинуто всі доданки, починаючи з третього.

Запишемо вираз для глобальної похибки, яка накопичиться до $(m+1)$-го кроку інтегрування. Оскільки

$$u_{m+1} = u_0(1 + h\lambda)^{m+1} = (x_0 - e_0)(1 + h\lambda)^{m+1},$$

то глобальна похибка $(m+1)$-го кроку:

$$E_{m+1} = x_{m+1} - u_{m+1} = x_0 e^{(m+1)h\lambda} - (x_0 - e_0)(1 + h\lambda)^{m+1} =$$
$$= x_0(e^{(m+1)h\lambda} - (1 + h\lambda)^{m+1}) + e_0(1 + h\lambda)^{m+1}.$$

З останнього виразу бачимо, що у випадку застосування ЯМЕ до розв'язання задачі (8.2.4) глобальна похибка E_{m+1} має дві складові:

1) похибка, яка виникає внаслідок апроксимації експоненти $e^{h\lambda}$ виразом $(1 + h\lambda)$;
2) похибка, зумовлена неточністю введення початкових умов e_0.

У випадку $|1 + h\lambda| > 1$ друга частина похибки E_{m+1}: $e_0(1 + h\lambda)^{m+1}$ зростатиме зі збільшенням кількості кроків m і, врешті-решт, зведе нанівець весь процес інтегрування.

Отже, якщо $\lambda < 0$, то для того, щоб нейтралізувати частину похибки E_{m+1}, зумовленої наявністю e_0, необхідно обирати такий крок h, при якому забезпечується виконання умови $|1 + h\lambda| < 1$, тобто: $h < \frac{2}{|\lambda|}$. (2.5)

Це **умова стійкості** явного методу Ейлера**!**

Оскільки значення λ характеризує задачу, то можна стверджувати, що оптимальний крок інтегрування h необхідно обирати конкретно для кожної задачі. Величину, обернену до λ, називають **постійною часу**. Її позначають τ. Чим більше значення λ, тим менше τ і експонента розв'язку швидше загасає. Отож, для правильного відтворення інтегральної кривої необхідно, щоб значення величини кроку h було меншим.

3. Методи Рунге-Кутта

На початку XX століття Рунге, Хойн і Кутта запропонували підхід, заснований на побудові формули дискретизації рівняння (1.1) у вигляді:

$$x_{n+1} = x_n + h \cdot \Phi(t_n, x_n, H). \tag{3.1}$$

Тут функція Ф є близькою до *f*, однак, не містить похідних від функції правої частини рівняння (1.1). Одержано сім'ю методів, що вимагають s-кратного обчислення функції правої частини на кожному кроці інтегрування (s-етапні методи).

Формули цих методів добре пристосовані для практичних розрахунків: вони дають змогу легко змінювати крок інтегрування, є однокроковими, достатньо економічні (принаймні, до формул четвертого порядку включно). Явний метод Ейлера можна розглядати як одноетапний метод Рунге-Кутта.

Найвідомішою є формула чотириетапного явного методу четвертого порядку точності. Її називають РК4:

$$\begin{aligned}
&x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),\\
&k_1 = h \cdot f(t_n, x_n),\\
&k_2 = h \cdot f(t_n + \tfrac{h}{2}, x_n + \tfrac{1}{2}k_1),\\
&k_3 = h \cdot f(t_n + \tfrac{h}{2}, x_n + \tfrac{1}{2}k_2),\\
&k_4 = h \cdot f(t_n + h, x_n + k_3).
\end{aligned} \tag{3.2}$$

3.1. Оцінка похибки на основі формул різної точності

Цей спосіб базується на використанні двох наближених значень розв'язку в одній точці [5]. Проте ці наближення, на відміну від правила Рунге, обчислюють не за однією, а за двома формулами різних порядків точності p і s з однаковим кроком. Розглянемо загальний випадок, коли застосування формули Рунге-Кутта не взаємозв'язане одне з одним, а потім перейдемо до розгляду спеціально підібраних формул.

3.2. Комбінація незалежних формул

$$x_1 = x_0 + \sum_{i=1}^{q} p_{q,i} k_i(h), \tag{3.3}$$

$$k_1(h) = hf(t_0, x_0),$$

$$k_2(h) = hf\left(t_0 + \alpha_2 h, x_0 + \beta_{2,1} k_1(h)\right),$$

$$k_3(h) = hf\left(t_0 + \alpha_q h, x_0 + \beta_{q,1} k_1(h) + \ldots + \beta_{q,q-1} k_{q-1}(h)\right),$$

де α_i , $\beta_{i,j}$, $p_{q,i}$ числа [5], які обираються так, щоб вираз (3.3) збігався з розкладом у ряд Тейлора до максимально можливої степені за довільних правих частин $f(x,t)$ та довільного кроку h.

Такий спосіб засновано на комбінації двох формул виду (3.3) різних порядків точності p і s:

$$x_1^p = x_0 + \sum_{i=1}^{r} p_{r,i} k_i ,$$

$$k_1 = hf(t_0, x_0), \tag{3.4}$$

$$k_i = hf\left(t_0 + \alpha_i h, x_0 + \sum_{j=1}^{i-1} \beta_{i,j} k_j\right), \text{ та} \qquad x_1^s = x_0 + \sum_{i=1}^{\tilde{r}} \tilde{p}_{r,i} \tilde{k}_i ,$$

$$\tilde{k}_1 = hf(t_0, x_0), \tag{3.5}$$

$$\tilde{k}_i = hf\left(t_0 + \tilde{\alpha}_i h, x_0 + \sum_{j=1}^{i-1} \tilde{\beta}_{i,j} \tilde{k}_j\right).$$

Нехай $p > s$, $r \geq \tilde{r}$. Тоді локальні похибки набувають вигляду:

$$\rho^p = x(t_0 + h) - x_1^p = O\left(h^{p+1}\right),$$

$$\rho^s = x(t_0 + h) - x_1^s = O\left(h^{s+1}\right).$$

Звідси випливає оцінка локальної похибки формули (3.5)

$$\rho^s = x_1^p - x_1^s + O\left(h^{p+1}\right). \qquad (3.6)$$

Знехтуємо членами вищого порядку малості, отримаємо

$$\rho^s \cong x_1^p - x_1^s \ . \qquad (3.7)$$

Отримана оцінка вимагає $r + \tilde{r} - 1$ обчислень правої частини.

3.3. Комбінація спеціальних формул

Якщо коефіцієнти у формулах (3.4) і (3.5) такі, що

$$\alpha_i = \tilde{\alpha}_i \ , \ \beta_{i,j} = \tilde{\beta}_{i,j} \ , \ i = 1, 2, ..., \tilde{r} \ , \qquad (3.8)$$

то $k_i = \tilde{k}_i$, $i = 1, 2, ..., \tilde{r}$,

і для локальної похибки (3.7) формули (3.5) отримаємо такий вираз [5]:

$$\rho^s \cong x_1^p - x_1^s = \sum_{i=1}^{r} q_i k_i \ , \qquad (3.9)$$

де $q_i = p_i - \tilde{p}_i$, $i = 1, 2, ..., \tilde{r}$, (3.10)

$$q_i = p_i \ , \ i = \tilde{r} + 1, 2, ..., r \ .$$

Оцінка (3.9), окрім тих значень правої частини, які обчислюють на поточному кроці h і входять до (3.5), містить додаткові значення k_i , $i = \tilde{r} + 1, ..., r$. Це дає змогу зменшити кількість обчислень правої частини порівняно з правилом Рунге та оцінкою (3.9).

Оцінка (3.9), а також точніша оцінка (3.7), є асимптотичними, оскільки вони враховують лише члени головного порядку, і справедливі за достатньо малих величин кроку інтегрування.

На практиці наближеним значенням розв’язку вважають x_1^p як таке, що має дещо вищий порядок точності.

Величина $E = \sum_{i=1}^{r} q_i k_i$, де коефіцієнти q_i визначають з допомогою (8.3.10), називають **контрольним членом** [5]. У випадку системи диференціальних рівнянь контрольний член можна записати у формі:

$$E = \sum_{i=1}^{r} q_i k_i^j \ , \ j = 1, 2, ..., M \ . \qquad (3.11)$$

3.4. Контрольні члени методів Рунге-Кутта

Якщо у виразі (3.3) прийняти $\alpha_2 = \frac{1}{2}$, $\alpha_3 = 1$, одержимо метод із сім'ї Рунге-Кутта [5]:

$$x_1 = x_0 + \frac{1}{6}(k_1 + 4k_2 + k_3); \tag{3.12}$$

$$k_1 = hf(t_0, x_0);$$

$$k_2 = hf\left(t_0 + \frac{1}{2}h, x_0 + \frac{1}{2}k_1\right);$$

$$k_3 = hf(t_0 + h, x_0 - k_1 + 2k_2),$$

а при $\alpha_2 = \frac{1}{2}$ – метод:

$$x_1 = x_0 + k_2; \tag{3.13}$$

$$k_1 = hf(t_0, x_0);$$

$$k_2 = hf\left(t_0 + \frac{1}{2}h, x_0 + \frac{1}{2}k_1\right).$$

Обидва методи (3.12) та (3.13) задовольняють умову (3.8).

Контрольний член $E = \frac{1}{6}(k_1 - 2k_2 + k_3)$ має порядок $o(h^3)$. Тут $p = 3$, $s = 2$, $r = 3$, $\tilde{r} = 2$. Якщо оцінку для локальної похибки методу (3.13) здійснити за правилом Рунге, то це потребуватиме п'ять обчислень правої частини замість трьох.

3.5. Модифікація Мерсона

$$x_1 = x_0 + \frac{1}{6}(k_1 + 4k_4 + k_5);$$

$$k_1 = hf(t_0, x_0);$$

$$k_2 = hf\left(t_0 + \frac{1}{3}h, x_0 + \frac{1}{3}k_1\right);$$

$$k_3 = hf\left(t_0 + \frac{1}{3}h, x_0 + \frac{1}{6}k_1 + \frac{1}{6}k_2\right); \tag{3.14}$$

$$k_4 = hf\left(t_0 + \frac{1}{2}h, x_0 + \frac{1}{8}k_1 + \frac{3}{8}k_3\right);$$

$$k_5 = hf\left(t_0 + h, x_0 + \frac{1}{2}k_1 - \frac{3}{2}k_3 + 2k_4\right).$$

Формули (3.14) і формула третього порядку точності:

$$x_1 = x_0 + \frac{1}{10}\left(k_1 + 3k_3 + 4k_4 + 2k_5\right) \tag{3.15}$$

задовольняють умову (8.3.8). Контрольний член:

$$E = \frac{1}{30}\left(2k_1 - 9k_3 + 8k_4 - k_5\right)$$

має порядок $o(h^4)$. Тут $p = 4$, $s = 3$, $r = 5$, $\tilde{r} = 5$. Якщо оцінку для похибки методу третього порядку виконувати за правилом Рунге, то це потребуватиме вісім обчислень правої частини замість п'яти за методом Мерсона.

На більш вузькому класі лінійних рівнянь виду:

$$x' = f\left(x,t\right) = at + bx + c \tag{3.16}$$

формула (3.15) має не третій, а п'ятий порядок точності. Порядок точності формули (3.14) залишається четвертим. Тому величина контрольного члена слугує головним членом локальної похибки формули (3.14) і має порядок $o(h^5)$. Отже, метод Мерсона має ту особливість, що на класі рівнянь (3.16) головний член локальної похибки виражається з допомогою лінійної комбінації лише тих значень правої частини, які безпосередньо входять у формулу (3.14). Відповідно, для отримання оцінки похибки зникає необхідність у додаткових обчисленнях.

3.6. Модифікація Фельберга

Фельберг розробив множину методів, що задовольняють умову (3.8). Наведемо формули четвертого і п'ятого порядків:

$$x_1^{(4)} = x_0 + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5;$$

$$x_1^{(5)} = x_0 + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6; \tag{3.17}$$

$$k_1 = hf\left(t_0, x_0\right);$$

$$k_2 = hf\left(t_0 + \frac{1}{4}h, x_0 + \frac{1}{4}k_1\right);$$

$$k_3 = hf\left(t_0 + \frac{3}{8}h, x_0 + \frac{3}{32}k_1 + \frac{9}{32}k_2\right);$$

$$k_4 = hf\left(t_0 + \frac{12}{13}h, x_0 + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right);$$

$$k_5 = hf\left(x_0 + h, x_0 + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right);$$

$$k_6 = hf\left(t_0 + \frac{1}{2}h, x_0 - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right).$$

Контрольний член:

$$E = \frac{1}{360}k_1 - \frac{128}{4275}k_3 + \frac{127}{6840}k_4 + \frac{1}{50}k_5 + \frac{2}{55}k_6$$

має порядок $o(h^5)$ [5]. Тут $p = 5$, $s = 4$, $r = 6$, $\tilde{r} = 5$. Якщо оцінку для похибки методу четвертого порядку виконувати за правилом Рунге, то це потребуватиме одинадцять обчислень правої частини замість шести за методом Фельберга.

4. Жорсткі системи диференціальних рівнянь

У випадку **систем лінійних** диференціальних рівнянь порядку N: $\frac{dx}{dt} = Ax(t)$, де A – квадратна матриця розміру $(N \times N)$, існує N чисел, що обмежують величину кроку h. Цими числами є корені характеристичного рівняння $\det(A - \lambda_i E) = 0$, де E – одинична матриця; λ_i – власні значення матриці A.

Для **систем нелінійних** диференціальних рівнянь порядку N:

$\frac{dx}{dt} = f(x(t), t)$. Роль матриці A відіграє матриця Якобі системи $Ja = \frac{\partial f_i}{\partial x_j}$.

Властивості системи диференціальних рівнянь залежать від того, на скільки власні значення її матриці відрізняються одне від одного. Якщо вони різняться між собою на декілька порядків, то математичну модель системи називають **жорсткою** (stiff) [2, 3]. Жорсткість – це властивість математичної моделі системи, а не самої системи.

Для подібних задач розроблено сім'ю чисельних методів – **неявні методи інтегрування**. Здебільшого, тут застосовують методику прогнозу-корекції. Вона полягає в тому, що дискретне значення вектора розв'язку в $m+1$-й точці спочатку прогнозують, наприклад, за лінійною формулою. Унаслідок цього одержують систему нелінійних алгебричних рівнянь. Процедура корекції – це розв'язання цієї системи відомим нам методом Ньютона для уточнення значень вектора розв'язку із заданою точністю ε.

2. Програмне середовище Eclipse та метод Дорнмана-Принса

2.1. Мова програмування Java

Для створення цього проекту було використано середовище (IDE) «**Eclipse**» для роботи з мовою програмування **Java** [6].

Java — об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

2.2. Графічний інтерфейс JavaFX

JavaFX - платформа на основі Java для створення додатків з насиченим графічним інтерфейсом. Може використовуватися як для створення настільних додатків, що запускаються безпосередньо з-під операційних систем, так і для інтернет-додатків (RIA), що працюють в браузерах, і для додатків на мобільних пристроях. JavaFX була створена, щоб замінити використовувану раніше бібліотеку Swing. Платформа JavaFX конкурує з Microsoft Silverlight, Adobe Flash і аналогічними системами.

Починаючи з версії Java 11 більше не входить в Java SE і не розробляється компанією Oracle (як окремий модуль підтримується компанією Gluon). Але

Oracle буде вносити необхідні зміни до 2022 року як для частини Java SE 8, яку ми і використовуємо для проекту.

2.3. Середовище розробки Eclipse

Eclipse - це інтегроване середовище розробки (IDE), яке використовується в комп'ютерному програмуванні. Воно містить базову робочу область та розширювану систему плагінів для налаштування середовища. Eclipse написаний здебільшого на Java, і його основне використання використовується для розробки програм Java, але він також може бути використаний для розробки програм на інших мовах програмування за допомогою плагінів, включаючи Ada, ABAP, C, C ++, C #, Clojure, COBOL, D , Erlang, Fortran, Groovy, Haskell, JavaScript, Julia, Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby, Rust, Scala та Scheme. Він також може бути використаний для розробки документів з LaTeX (через плагін TeXlipse) та пакетів для програмного забезпечення Mathematica. Середовища розробки включають інструменти розробки Eclipse Java (JDT) для Java та Scala, Eclipse CDT для C / C ++ та Eclipse PDT для PHP.

Комплект для розробки програмного забезпечення Eclipse (SDK) - це вільне програмне забезпечення з відкритим кодом, яке випускається згідно з умовами публічної ліцензії Eclipse. Це одна з перших IDE, яка запускалася під GNU Classpath, і без проблем працює під IcedTea.

2.4. Метод Дорнмана-Принса

У чисельному аналізі метод Дорнмана-Принса [7, 8, 9] - це явний метод вирішення звичайних диференціальних рівнянь. Метод є членом сімейства Рунге-Кутта. Більш конкретно, він використовує шість оцінок функцій для обчислення точних рішень четвертого та п'ятого порядку. Потім різницю між цими рішеннями приймають за помилку рішення (четвертого порядку). Ця оцінка помилок дуже зручна для алгоритмів адаптивного інтеграційного кроку. Іншими аналогічними методами інтеграції є Фельберг (RKF) та Cash-Karp (RKCK).

Метод Дорнмана-Принса має сім етапів, але він використовує лише шість оцінок функції за крок, оскільки він має властивість FSAL (Перше таке ж, як і останнє): останній етап оцінюється в тій же точці, що і перший етап наступного

кроку. Дорманд і Прінс вибрали коефіцієнти свого методу, щоб мінімізувати похибку рішення п'ятого порядку. У цьому головна відмінність методу Фельберга, який був побудований так, що рішення четвертого порядку має невелику помилку. З цієї причини метод Дорманд-Прінс є більш підходящим, коли для продовження інтеграції використовується рішення вищого порядку, практика, відома як локальна екстраполяція (Shampine 1986; Hairer, Nørsett & Wanner 2008, pp. 178–179).

Як і усі інші методи класу Рунге-Кутта, цей метод має так звану «Таблицю Батчера», яка містить у собі усі використані для методи коефіцієнти.

0							
1/5	1/5						
3/10	3/40	9/40					
4/5	44/45	−56/15	32/9				
8/9	19372/6561	−25360/2187	64448/6561	−212/729			
1	9017/3168	−355/33	46732/5247	49/176	−5103/18656		
1	35/384	0	500/1113	125/192	−2187/6784	11/84	
	35/384	0	500/1113	125/192	−2187/6784	11/84	0
	5179/57600	0	7571/16695	393/640	−92097/339200	187/2100	1/40

2.5. Алгоритми автоматичного вибору величини кроку h

Для високоточних програм чисельного інтегрування необхідно завжди дотримуватися вимог точності до обчислень. Отож, практично в усіх подібних програмах встановлюють максимально допустиму локальну похибку обчислень або діапазон зміни похибки на всьому інтервалі інтегрування. Оскільки надто жорсткі вимоги до точності суттєво збільшують затрати машинного часу, то виникає необхідність визначення максимально можливої величини кроку. Отже, необхідно здійснювати інтегрування з максимально великим кроком, але таким, який задовольнятиме вимоги точності. Від ефективності алгоритму автоматичного вибору кроку значною мірою залежить швидкість виконання усієї програми. Ми величину кроку h у неявному методі Ейлера обирали за емпіричним алгоритмом. На практиці використовують інші, можливо ефективніші методи. Наведемо деякі з них:

1. **Метод Брайтона**:

$$h_{m+1} = h_m \sqrt[k+1]{\varepsilon / e_m}\,, \qquad (6.1)$$

де h_m, h_{m+1} – величина m-го та m+1-го кроку, відповідно;

ε – допустима локальна похибка обчислень на один крок;

e_m – вирахувана локальна похибка апроксимації на m-му кроці. Локальну похибку e_m апроксимації на m-му кроці вирахо-вують програмним чином за результатами обчислень у попередніх точках згідно з таким алгоритмом:

З літератури відомо, що похибку апроксимації явного методу Ейлера можна записати так [1]: $\varepsilon_m^{HME} = +0.5 h_m^2 x''(\bar{t})$. (6.2)

Для неявного методу Ейлера: $\varepsilon_m^{HME} = -0.5 h_m^2 x''(\bar{t})$, (6.3)

де $t_m < \bar{t} < t_{m+1}$.

Для визначення ε_m необхідно оцінити $x''(\bar{t})$ для $t_m < \bar{t} < t_{m+1}$. З цією метою можна використати значення вектора змінних стану x_{m+1}, x_m, x_{m-1}, віддалених один відносно одного по осі t на h_m, h_{m-1}. За цими трьома точками можна побудувати апроксимаційний поліном Ньютона для нерівновіддалених точок:

$$x(t) = x_{m-1} + (t - t_{m-1})\frac{x_m - x_{m-1}}{t_m - t_{m-1}} + (t - t_{m-1})(t - t_m)\frac{\dfrac{x_{m+1} - x_m}{t_{m+1} - t_m} - \dfrac{x_m - x_{m-1}}{t_m - t_{m-1}}}{t_{m+1} - t_{m-1}}. \qquad (6.4)$$

Якщо з (6.4) знайти другу похідну $x''(t)$ і підставити у вирази (6.2) та (6.3), то можна отримати узагальнену формулу для оцінки величини похибки апроксимації для явного та неявного методів Ейлера [1]:

$$\varepsilon_m = \frac{h_m}{h_m + h_{m-1}}\left[\left(x_{m+1} - x_m\right) - \frac{h_m}{h_{m-1}}\left(x_m - x_{m-1}\right)\right]. \qquad (6.5)$$

Формула (6.5) має простий геометричний зміст: у випадку рівновіддалених точок – це відстань між середньою точкою x_m та середньою алгебричною двох крайніх точок x_{m-1} та x_{m+1}.

Допустима локальна похибка апроксимації на один крок ε або обчислюється програмою за формулою $\varepsilon = 0.001 \times \|x_{\max}\|$, де $\|x_{\max}\|$ – максимально можливе абсолютне значення норми вектора змінних на всьому інтервалі інтегрування, або задається користувачем. Якщо після вирахування e_m виявилося, що $e_m > \varepsilon$, то такий крок вважається хибним і відкидається, величина кроку зменшується за формулою $h_{m+1} = h_m \sqrt[k+1]{\varepsilon / e_m}$, а розрахунки на m-му кроці повторюють з новим кроком.

2. **Метод трьох зон**:

У цьому методі наступний (m+1)-й крок визначають так:

$$h_{m+1} = \begin{cases} h_m / 2, \text{ якщо } |e_m| > \varepsilon; \\ h_m, \text{ якщо } \left[\varepsilon / k^2\right] < |e_m| < \varepsilon; \\ k h_m, \text{ якщо } |e_m| < \left[\varepsilon / k^2\right], \end{cases} \tag{6.6}$$

де k – порядок точності методу [1].

Результати дослідження засвідчили, що використання методу Брайтона збільшує середню максимальну величину кроку загалом на відрізку інтегрування, зменшує діапазон зміни локальної похибки апроксимації, розташований в околі ε, проте спричинює до збільшення загальної кількості відкинутих кроків, які не задовольняють вимогам точності.

Метод трьох зон позбавлений таких недоліків, однак спричиняє деяку інертність у реагуванні на зміну величини e_m , що передбачає зменшення середньої величини кроку, а, отже, сумарні затрати машинного часу.

3. Архітектура проекту

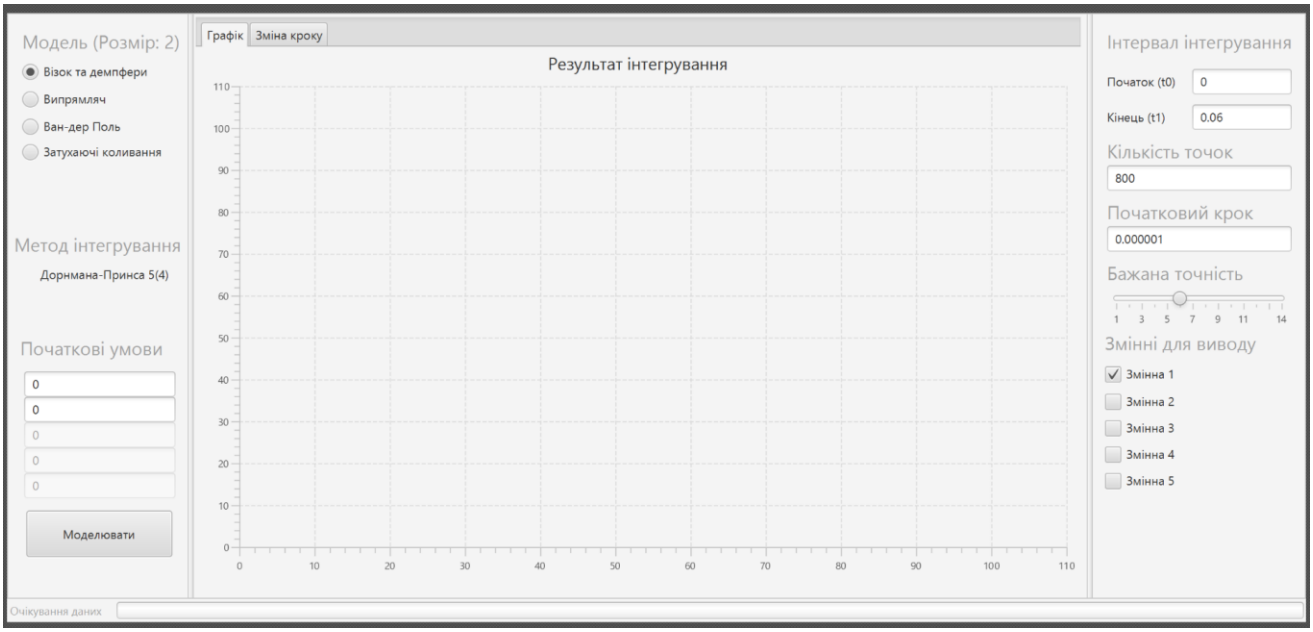
3.1. Реалізований алгоритм вибору кроку

У проекті реалізовано так званий «Метод трьох зон». В методі Дорнмана-Принса для обрахунку локальної похибки використовуються рішення четвертого та п’ятого порядку, після чого в залежності від її розміру можна обрахувати наступний крок за формулою метода трьох зон, де h_m це розмір кроку, k це порядок точності методу (для Дорнмана-Принса це 5), ε це бажана точність інтегрування, а e_m це обрахована локальна похибка.

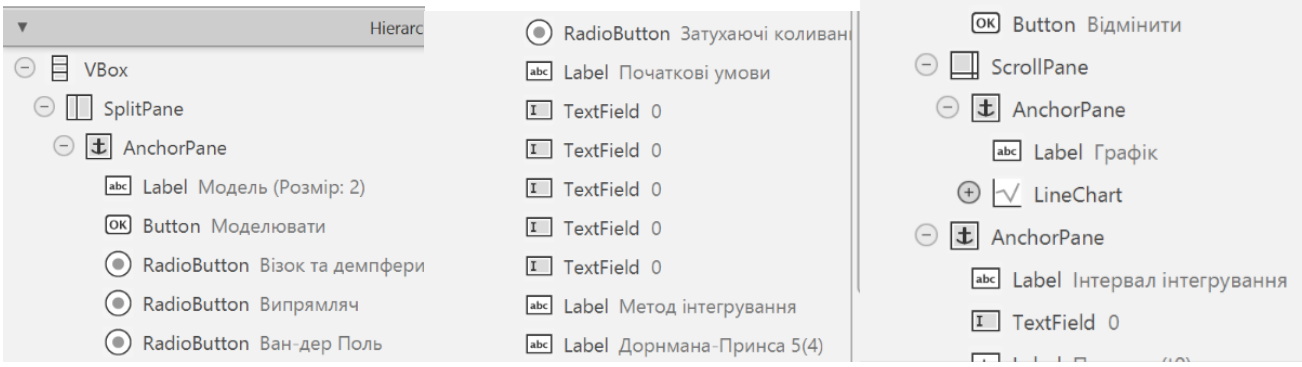
3.2. Створення графічного інтерфейсу

Для початку роботи з програмою-інтегратором користувачу необхідний графічний інтерфейс. Тому для його створення було використано програму-редактор JavaFX файлів «JavaFX Scene Builder».

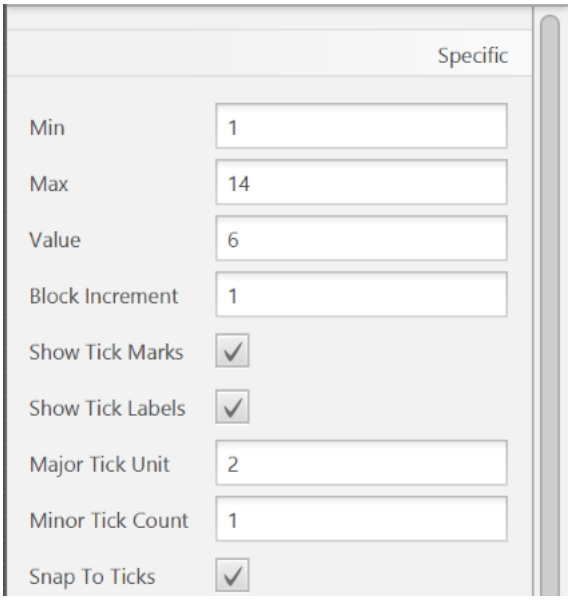
Вигляд панелі проекту у редакторі:



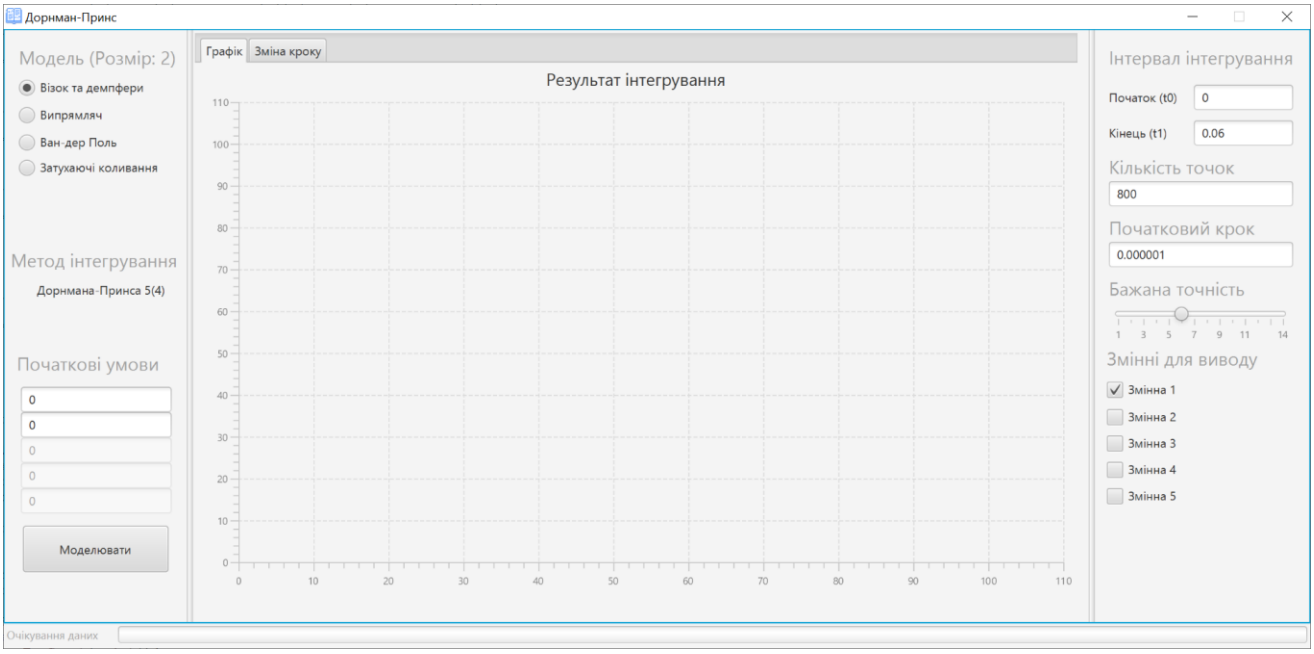
Список усіх компонент, які були використані для побудови інтерфейсу (усі елементи – об’єкти мови Java, а саме компоненти JavaFX Controls):



Редагування компонент (на прикладі слайдера для вибору бажаної точності):



Повне вікно програми, яке використовує шаблон, створений у графічному редакторі (компоненти пов’язані з кодом завдяки унікальному id):



3.3. Реалізація проекту мовою Java

Програма містить декілька класів, кожен з яких відповідає за певну частину роботи. Це є відповідним до парадигми об’єктно-орієнтованого програмування, де кожен об’єкт класу виконує свою роботу.

3.3.1. Клас «Root»

Містить у собі всю роботу з інтерфейсом програми (обробка вводу користувача), перевірки на некоректні дані, є головним керуючим потоком (який створює нові, або припиняє їх).

```
public class Root extends Application {

    private TextField t0, t1, points, starth;
    private List<RadioButton> radio = new ArrayList<>();
    private Integrator integr;
    private List<TextField> tableValues = new ArrayList<>();
    private List<CheckBox> cbVariables = new ArrayList<>();
```

3.3.2. Клас «MathematicModel»

Відповідає за обрахунок функцій математичних моделей, наданих у проекті. Створений у вигляді **enum**, оскільки у мові Java цей тип об’єктів має більше можливостей ніж в інших схожих мовах, та гарно підходить саме у даному випадку (фіксована кількість математичних моделей, яка не змінюється від час виковання програми).

```
public enum MathematicModel {

    CHAIN(2, 3),
    VAN_DER_POLL(2, 3),
    NORMALIZER(4, 6),
    SILENCER(5, 3);

    @Getter
    private final int size, power;

    private MathematicModel(int m, int pow) {
        size = m;
        power = pow;
    }
```

3.3.3. Клас «Integrator»

Головний клас програми, який містить логіку алгоритму Дорнмана-Принса та усі необхідні змінні для його правильної роботи. Два масиви з заздалегідь прорахованими коефіцієнтами алгоритму для того, щоб при інтегруванні не витрачався час на ділення чисел з плаваючою комою (вони статичні).

```
public class Integrator {

    private static final double[][] COEFF_K = {
            { 1.0 / 5.0 },
            { 3.0 / 40.0, 9.0 / 40.0 },
            { 44.0 / 45.0, -56.0 / 15.0, 32.0 / 9.0 },
            { 19372.0 / 6561.0, -25360.0 / 2187.0, 64448.0 / 6561.0, -212.0 / 729.0 },
            { 9017.0 / 3168.0, -355.0 / 33.0, 46732.0 / 5247.0, 49.0 / 176.0, -5103.0 / 18656.0 },
            { 35.0 / 384.0, 0.0, 500.0 / 1113.0, 125.0 / 192.0, -2187.0 / 6784.0, 11.0 / 84.0 }
            };
    private static final double[] COEFF_M = { 1.0 / 5.0, 3.0 / 10.0, 4.0 / 5.0, 8.0 / 9.0, 1.0, 1.0 };
    /** Прогрес-бар інтегрування */
    private final ProgressBar pb;
    /** Математична модель для обчислень */
    @Setter
    @Getter
    private MathematicModel model = MathematicModel.CHAIN;
    /** Кількість точок, які необхідно рахувати */
    @Setter
    @Getter
    private int points = 800;
    /** Крок інтегрування */
    @Setter
    @Getter
    private double step = 0.000001;
    /** Очікувана точність для автоматичного вибору кроку */
    @Setter
    @Getter
    private double epsilon = 0.000001;
```

4. Тестування програми

Програма має розширювальний інтерфейс, який дозволяє додавати нові математичні моделі. Тестування працездатності розробленої системи було проведено на основі двої математичних моделей різної розмірності.

4.1. Однопівперіодний випрямляч

Схему випрямляча подано на рис. 4.1. Математичну модель випрямляча було побудовано на основі використання законів Ома та Кірхгофа [1].

Оберемо напрями напруг і струмів кожної гілки схеми так, як це проілюстровано на рисунку. Оскільки схема налічує 3 ємності і 1 індуктивність, її математичною моделлю слугуватиме система чотирьох ЗДР з чотирма невідомими, які ми позначили, відповідно, x_1, x_2, x_3, x_4

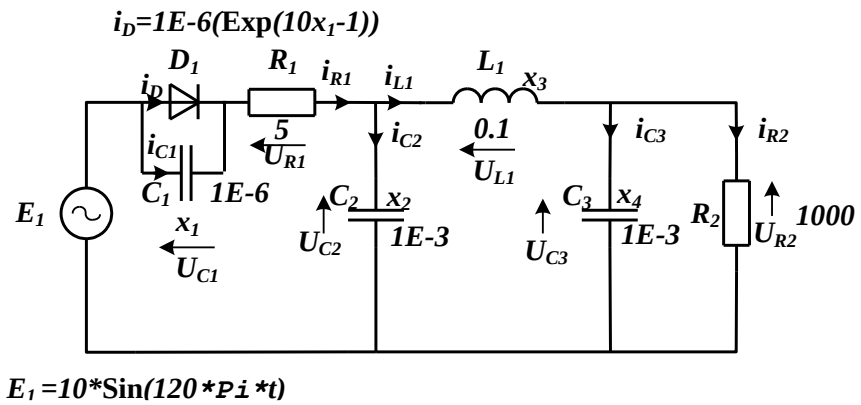


Рис. 4.1. Однопівперіодний випрямляч

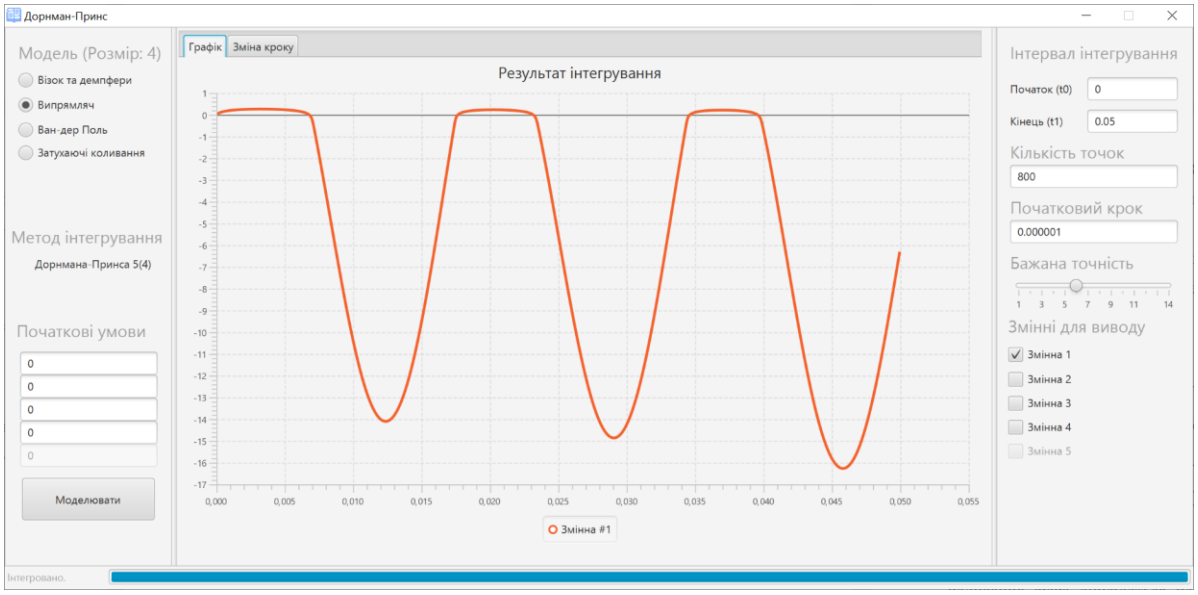
Побудуємо математичну модель схеми, використовуючи закони Кірхгофа й Ома. Одержані рівняння запишемо як систему у нормальній формі Коші, що й слугуватиме математичною моделлю нашого випрямляча.

$$\begin{cases} \dfrac{dx_1}{dt} = 0.2*10^6(-x_1 - x_2 + 10*\mathrm{Sin}(120*\pi*t)) - i_D \\ \dfrac{dx_2}{dt} = 0.2*10^3(-x_1 - x_2 + 10*\mathrm{Sin}(120*\pi*t) - x_3) \\ \dfrac{dx_3}{dt} = 10(x_2 - x_4) \\ \dfrac{dx_4}{dt} = 10^3 x_3 - x_4 \end{cases}$$

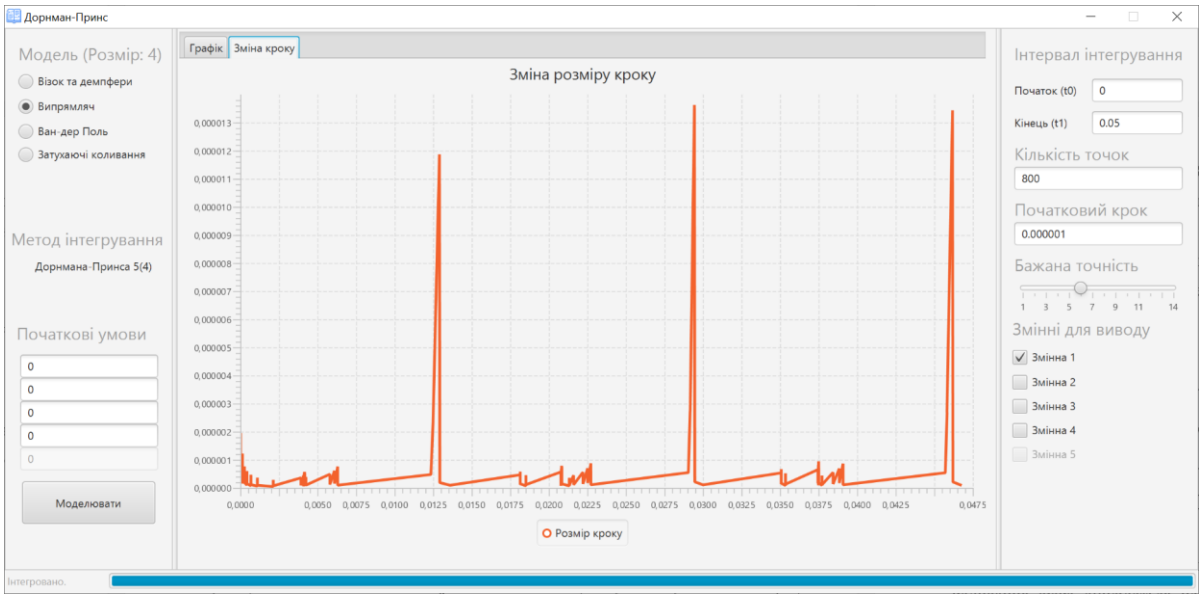
$$i_D = 10^{-6}(e^{10x_1} - 1).$$

4.2. Результати моделювання випрямляча

Межі інтегрування: [0-0,05]. Кількість точок – 10000. Бажана точність – 0,000001



На прикладі цієї моделі можемо побачити роботу програми. Ліва панель дозволяє вводити дані для обраної моделі (початкові умови) та обирати саму модель. Панель справа містить у собі редактор меж інтегрування (*початок* і *кінець*), поле для максимальної кількості точок (які будуть будувати графік), розмір початкового кроку (який буде автоматично перераховано в процесі інтеграції), слайдер порядку точності (по формулі 10^{-x}) та чекбокси для вибору змінних, які необхідно відобразити на фінальному графіку.



Також програма має графік зміни кроку протягом процесу інтегрування, тобто відображає його модифікацію методом трьох зон для досягнення максимальної ефективності та швидкодії.

4.3. Візок із демпферами

Моделювання візочка масою m, жорстко приєднаного до стіни трьома пристроями для гасіння удару (демпферами) [1]. Це – лінійна пружина з коефіцієнтом жорсткості *k*, нелінійна “кубічна” пружина, яка створює гальмівну силу, пропорційну кубові відхилення з постійним коефіцієнтом k^*, і спеціальний демпфер, гальмівна сила якого пропорційна швидкості візка з коефіцієнтом пропорційності c (рис. 8.12).

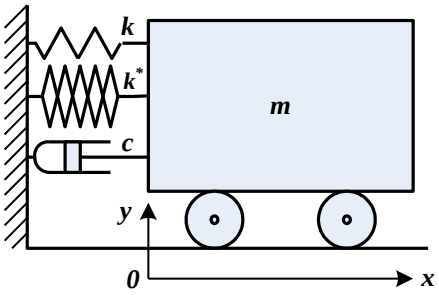


Рис. 8.12. Візок з потрійним демпфуванням

Якщо початок системи координат розмістити так, як на рис. 8.12, то загасаючі коливання візка описуються нелінійним диференціальним рівнянням другого порядку: $m\frac{d^2x}{dt^2}+c\frac{dx}{dt}+kx+k^*x^3=0$.

Змоделюємо динаміку цієї системи на проміжку часу $[t_0, t_k]$, тривалістю 1с. Виберемо такі фізичні параметри системи:

$$k=2.0\frac{H}{см};\quad k^*=2.0\frac{H}{см^3};\quad c=0.15\frac{H\cdot c}{см};\quad m=1.0\,кг.$$

Перетворимо диференціальне рівняння другого порядку в систему двох диференціальних рівнянь першого порядку:

$$\frac{dx}{dt}=x_1;\quad \frac{dx_1}{dt}=x_2;$$

$$\begin{cases}\dfrac{dx_1}{dt}=x_2;\\ \dfrac{dx_2}{dt}=-\dfrac{c}{m}x_2-\dfrac{k}{m}x_1-\dfrac{k^*}{m}x_1{}^3.\end{cases}$$

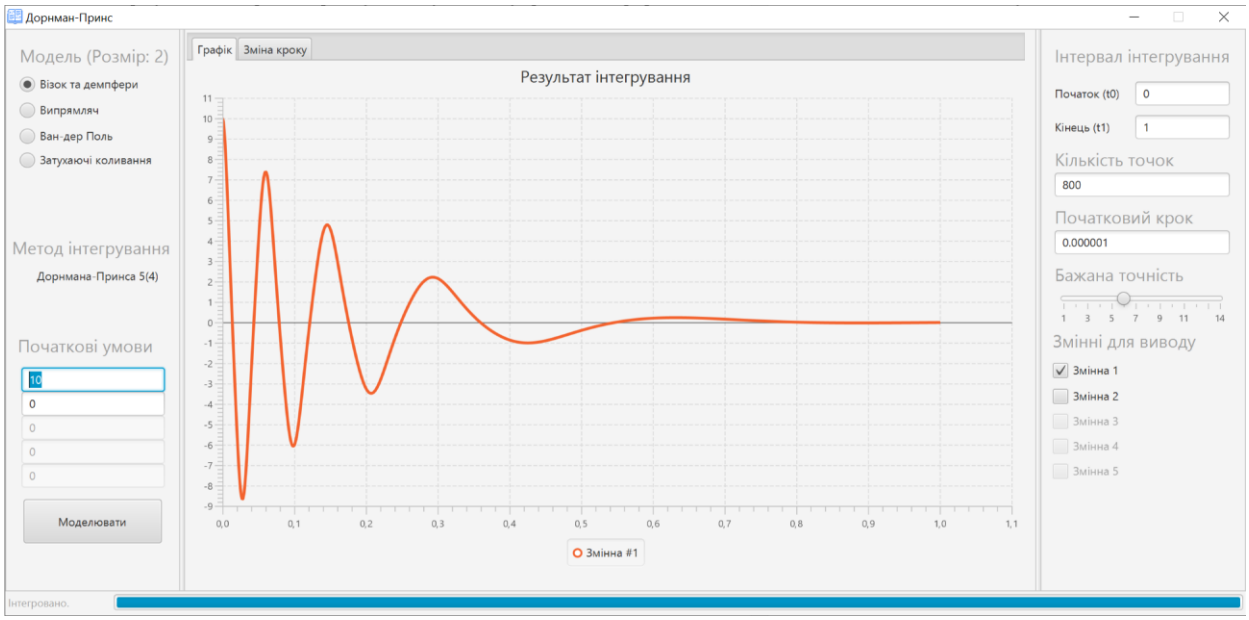
Підставляючи у цю систему числові значення параметрів у такій системі одиниць, щоб зміщення x виражалось у сантиметрах, одержимо готову для моделювання систему в нормальній формі Коші:

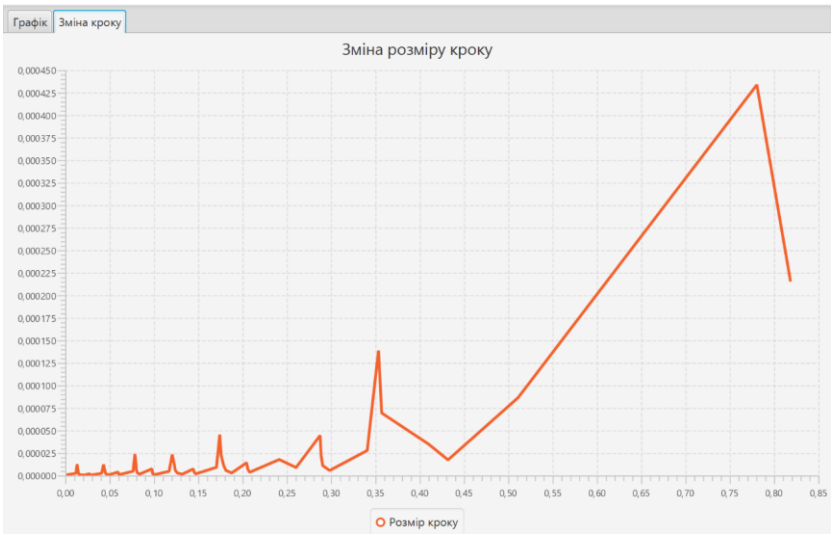
$$\begin{cases} \dfrac{dx_1}{dt} = x_2 \\ \dfrac{dx_2}{dt} = -15.0x_2 - 200x_1 - 200x_1^3 \end{cases}$$

Обираючи початкові умови: $x(0) = 10\,см$; $\dfrac{dx(0)}{dt} = 0$, одержимо графіки загасання коливань візка.

4.4. Результати моделювання візочка

Межі інтегрування: [0-1]. Кількість точок – 10000.Бажана точність – 0,000001. Початкові умови [10,0]





Висновки

Під час виконання курсової роботи розроблено проект мовою Java для інтегрування математичних моделей фізичних систем.

Реалізовано метод серії Рунге-Кутта, а саме алгоритм Дорнмана-Принса.

У результаті створення проекту я набув досвіду роботи з графічним редактором компонент JavaFX, дізнався про способи об’єднання інтерфейсу з головним кодом програми, та використав багатопоточність для виконання інтеграції окремим потоком.

Також у роботі мною були дослідженні методи автоматичного вибору кроку при інтегруванні, та у результаті міркувань і тестування було обрано метод трьох зон, як базу для реалізації автоматичного вибору кроку для методу Дорнмана-Принса.

Результатом курсової роботи стало програмне середовище з власним графічним інтерфейсом, здатне обраховувати задані математичні моделі залежно від наданих користувачем даних. Цю програму можна використовувати для візуалізації обчислень і інших математичних моделей з деякими модифікаціями коду. Тобто, цей проект можна вважати корисним вкладом у наукову спільноту, наприклад як інструмент для перевірки поведінки функцій потребуючих досить великої точності на певних інтервалах з заданими умовами.

Список використаної літератури

1. Хвищун І.О. Програмування і математичне моделювання: Підручник. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2007. – 544 с.
2. Ракитский Ю.В., Устинов С.М. и др. Численные методы решения жестких систем. М.: Наука, ГРФМЛ, 1979. 208 с.
3. Современные численные методы решения обыкновенных дифференциальных уравнений / Под ред. Дж. Холла и Дж. Уатта. М.: Мир, 1979. 312 с.
4. Чуа Л.О., Пен-Мин Лин. Машинный анализ электронных схем: Алгоритмы и вычислительные методы. М.: Энергия, 1980. 640 с.
5. Арашунян О.Б., Залеткин С.Ф. Численное решение обыкновенных дифференциальных уравнений на Фортране. М.: Изд-во МГУ, 1990. 336 с.
6. Эккель Б. Философия Java. 4-е изд. – СПб.: Питер. 2015. 1168 с.
7. Решение обыкновенных дифференциальных уравнений. Жесткие и дифференциально-алгебраические задачи (Э. Хайрер, Г. Ваннер), 1999
8. Numerical recipes in C. The art of scientific computing (William H. Press), 1992
9. Harder D. W. 4th-order Runge Kutta and the Dormand-Prince Methods (LEL Department of Electrical and Computer Engineering University of Waterloo), 2012
10. https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods
11. https://commons.apache.org/proper/commons-math/javadocs/api-3.6/src-html/org/apache/commons/math3/ode/nonstiff/DormandPrince54Integrator.html

Додаток
Код програми

```
public class Root extends Application {

	private TextField t0, t1, points, starth;
	private List<RadioButton> radio = new ArrayList<>();
	private Integrator integr;
	private List<TextField> tableValues = new ArrayList<>();
	private List<CheckBox> cbVariables = new ArrayList<>();
	private LineChart chart;
	private Button runIntegrate, runCancel;
	private Scene scene;
	private Parent root;
	private Thread activeIntegrator;

	@Override
	public void start(Stage stage) throws Exception {
		root = FXMLLoader.load(Root.class.getResource("/panel.fxml"));
		stage.getIcons().add(new Image("/favicon.png"));
		stage.setResizable(false);
		scene = new Scene(root, 1280, 600);
		stage.setTitle("Дорнман-Принс");
		stage.setScene(scene);
		stage.show();
		t0 = (TextField) root.lookup("#inputt0");
		t1 = (TextField) root.lookup("#inputt1");
		t0.textProperty().addListener((observable, oldValue, newValue) -> {
			if (doubleFieldChange(t0))
				integr.setT0(parseSilentDouble(newValue));
		});
		t1.textProperty().addListener((observable, oldValue, newValue) -> {
			if (doubleFieldChange(t1))
				integr.setT1(parseSilentDouble(newValue));
		});
		points = (TextField) root.lookup("#inputpoints");
		points.textProperty().addListener((observable, oldValue, newValue) -> {
			if (intFieldChange(points))
				integr.setPoints(parseSilentInt(newValue));
		});
		starth = (TextField) root.lookup("#starth");
		starth.textProperty().addListener((observable, oldValue, newValue) -> {
			if (doubleFieldChange(starth))
				integr.setStep(parseSilentDouble(newValue));
		});
		RadioButton m1 = (RadioButton) root.lookup("#model1");
		m1.selectedProperty().addListener((o, old, value) -> {
			if (!value)
				return;
			radio.stream().filter(r -> !r.getId().equals(m1.getId())).forEach(r ->
r.setSelected(false));
			integr.setModel(MathematicModel.CHAIN);
			applyModel();
		});
		RadioButton m2 = (RadioButton) root.lookup("#model2");
		m2.selectedProperty().addListener((o, old, value) -> {
			if (!value)
				return;
			radio.stream().filter(r -> !r.getId().equals(m2.getId())).forEach(r ->
r.setSelected(false));
			integr.setModel(MathematicModel.NORMALIZER);
			applyModel();
		});
		RadioButton m3 = (RadioButton) root.lookup("#model3");
		m3.selectedProperty().addListener((o, old, value) -> {
			if (!value)
				return;
			radio.stream().filter(r -> !r.getId().equals(m3.getId())).forEach(r ->
r.setSelected(false));
			integr.setModel(MathematicModel.VAN_DER_POLL);
			applyModel();
		});
		RadioButton m4 = (RadioButton) root.lookup("#model4");
		m4.selectedProperty().addListener((o, old, value) -> {
			if (!value)
				return;
			radio.stream().filter(r -> !r.getId().equals(m4.getId())).forEach(r ->
r.setSelected(false));
			integr.setModel(MathematicModel.SILENCER);
			applyModel();
		});
```

```
		runIntegrate = ((Button)root.lookup("#buttonIntegrate"));
		runIntegrate.setOnMouseClicked(c -> prepare());
		runCancel = ((Button)root.lookup("#btnCancel"));
		runCancel.setOnMouseClicked(c -> cancel());
		chart = (LineChart) root.lookup("#graph");
		radio.add(m1);
		radio.add(m2);
		radio.add(m3);
		radio.add(m4);
		tableValues.add((TextField)root.lookup("#td1"));
		tableValues.add((TextField)root.lookup("#td2"));
		tableValues.add((TextField)root.lookup("#td3"));
		tableValues.add((TextField)root.lookup("#td4"));
		tableValues.add((TextField)root.lookup("#td5"));
		cbVariables.add((CheckBox)root.lookup("#cbv1"));
		cbVariables.add((CheckBox)root.lookup("#cbv2"));
		cbVariables.add((CheckBox)root.lookup("#cbv3"));
		cbVariables.add((CheckBox)root.lookup("#cbv4"));
		cbVariables.add((CheckBox)root.lookup("#cbv5"));
		tableValues.get(0).textProperty().addListener((observable, oldValue, newValue) -> {
			if (doubleFieldChange(tableValues.get(0)))
				integr.getInitialX()[0] = parseSilentDouble(newValue);
		});
		tableValues.get(1).textProperty().addListener((observable, oldValue, newValue) -> {
			if (doubleFieldChange(tableValues.get(1)))
				integr.getInitialX()[1] = parseSilentDouble(newValue);
		});
		tableValues.get(2).textProperty().addListener((observable, oldValue, newValue) -> {
			if (doubleFieldChange(tableValues.get(2)))
				integr.getInitialX()[2] = parseSilentDouble(newValue);
		});
		tableValues.get(3).textProperty().addListener((observable, oldValue, newValue) -> {
			if (doubleFieldChange(tableValues.get(3)))
				integr.getInitialX()[3] = parseSilentDouble(newValue);
		});
		tableValues.get(4).textProperty().addListener((observable, oldValue, newValue) -> {
			if (doubleFieldChange(tableValues.get(4)))
				integr.getInitialX()[4] = parseSilentDouble(newValue);
		});
		integr = new Integrator((ProgressBar)root.lookup("#progressbar"));
	}

	public static void main(String[] args){
		launch(args);
	}

	private void applyModel(){
		((Label)scene.lookup("#labelModelSize")).setText(String.format("Модель (Розмір: %d)",
integr.getModel().getSize()));
		for (TextField n : tableValues){
			n.setDisable(Integer.parseInt(n.getId().substring(2)) >
integr.getModel().getSize());
		}
		for (CheckBox n : cbVariables){
			n.setDisable(Integer.parseInt(n.getId().substring(3)) >
integr.getModel().getSize());
		}
	}

	private void prepare(){
		runIntegrate.setDisable(true);
		radio.forEach(n -> n.setDisable(true));
		t0.setDisable(true);
		t1.setDisable(true);
		points.setDisable(true);
		starth.setDisable(true);
		setState("Перевірка даних..");
		if (integr.getPoints() < 100){
			points.setText("100");
			integr.setPoints(100);
		} else{
			integr.setPoints(parseSilentInt(points.getText()));
		}
		if (integr.getStep() > Math.pow(10, -integr.getModel().getPower())){
			starth.setText(String.format("%f", Math.pow(10, -
integr.getModel().getPower())).replace(",", "."));
			integr.setStep(Math.pow(10, -integr.getModel().getPower()));
		} else {
			integr.setStep(parseSilentDouble(starth.getText()));
		}
		integr.setEpsilon(Math.pow(10, -
((Slider)root.lookup("#slepsilon")).valueProperty().intValue()));
```

```
            if (integr.getT0() > integr.getT1()){
                double dt0 = integr.getT1(), dt1 = integr.getT0();
                t0.setText(String.valueOf(dt0));
                t1.setText(String.valueOf(dt1));
                integr.setT0(dt0);
                integr.setT1(dt1);
            }
            setState("Інтегрування...");
            activeIntegrator = new Thread(){
                public void run(){
                    process();
                }
            };
            runCancel.setVisible(true);
            activeIntegrator.start();
        }

        private void cancel(){
            if (activeIntegrator == null)
                return;
            try {
                activeIntegrator.interrupt();
                activeIntegrator = null;
            } catch (Exception e ){return;}
            setState("Відміна.");
            radio.forEach(n -> n.setDisable(false));
            t0.setDisable(false);
            t1.setDisable(false);
            points.setDisable(false);
            starth.setDisable(false);
            runIntegrate.setDisable(false);
            runCancel.setVisible(false);
        }

        private void process(){
            List<List<Pair<Double, Double>>> data = integr.integrate();
            activeIntegrator = null;
            Platform.runLater(() -> {
                setState("Інтегровано.");
                ObservableList<XYChart.Series<Double,Double>> lineChartData =
FXCollections.observableArrayList(
                                    data.stream().map(list -> new
LineChart.Series<Double,Double>("Змінна #"+(1 + data.indexOf(list)),
FXCollections.observableArrayList(
                                            list.stream().map(p -> new
XYChart.Data<Double,Double>(p.getLeft(), p.getRight())).collect(Collectors.toList())
                                        )).collect(Collectors.toList())
                        );
                if (lineChartData.size() >= 5 && !cbVariables.get(4).isSelected())
                    lineChartData.remove(4);
                if (lineChartData.size() >= 4 && !cbVariables.get(3).isSelected())
                    lineChartData.remove(3);
                if (lineChartData.size() >= 3 && !cbVariables.get(2).isSelected())
                    lineChartData.remove(2);
                if (lineChartData.size() >= 2 && !cbVariables.get(1).isSelected())
                    lineChartData.remove(1);
                if (lineChartData.size() >= 1 && !cbVariables.get(0).isSelected())
                    lineChartData.remove(0);
                chart.dataProperty().setValue(lineChartData);
                radio.forEach(n -> n.setDisable(false));
                t0.setDisable(false);
                t1.setDisable(false);
                points.setDisable(false);
                starth.setDisable(false);
                runIntegrate.setDisable(false);
                runCancel.setVisible(false);
            });
        }

        private void setState(String str){
            ((Label)scene.lookup("#labelStatus")).setText(str);
        }

        private static boolean doubleFieldChange(TextField f){
            if (f.getText().matches("[0-9\\,\\.]+"))
                return true;
            f.setText(f.getText().replaceAll("[^0-9\\,\\.]", ""));
            return false;
        }

        static double parseSilentDouble(String s){
            try {
```

```
                        return Double.parseDouble(s);
                } catch (Exception e){
                }
                return 0;
        }

        private static boolean intFieldChange(TextField f){
                if (f.getText().matches("[0-9]+"))
                        return true;
                f.setText(f.getText().replaceAll("[^0-9]", ""));
                return false;
        }

        static int parseSilentInt(String s){
                try {
                        return Integer.parseInt(s);
                } catch (Exception e){}
                return 0;
        }

}

public enum MathematicModel {

        CHAIN(2, 3),
        VAN_DER_POLL(2, 3),
        NORMALIZER(4, 6),
        SILENCER(5, 3);

        @Getter
        private final int size, power;

        private MathematicModel(int m, int pow) {
                size = m;
                power = pow;
        }

        /**
         * Значення моделі з зсувом t для значень x
         * @param t зсув по часу
         * @param x масив значень
         * @return значення моделі
         */
        public double[] func(double t, double[] x) {
                double[] f = new double[size];
                switch (this) {
                case CHAIN: // Візок
                        f[0] = x[1];
                        f[1] = -15.0 * x[1] - 200 * x[0] - 200 * x[0] * x[0] * x[0];
                        return f;
                case VAN_DER_POLL: // Ван дер Поль
                        f[0] = x[1];
                        f[1] = 100.0 * x[1] * (1.0 - Math.pow(x[0], 2.0)) - x[0];
                        return f;
                case NORMALIZER: // Випрямляч
                        f[0] = 200000 * (-x[0] - x[1] + 10 * Math.sin(120 * Math.PI * t)) -
(Math.exp(10 * x[0]) - 1) / (10e-6);
                        f[1] = 200 * (-x[0] - x[1] + 10 * Math.sin(120 * Math.PI * t) - x[2]);
                        f[2] = 10 * (x[1] - x[3]);
                        f[3] = 1000 * x[2] - x[3];
                        return f;
                case SILENCER: // Затухаючі к-ня
                        f[0] = 1000.0 * x[3];
                        f[1] = 1000.0 * x[4];
                        f[2] = x[3] - x[4];
                        f[3] = (-x[0] - x[2] - x[3]) / 997.0;
                        f[4] = (-x[1] + x[2] - x[4]) / 1003.0;
                        return f;
                }
                return null;
        }

}
public class Integrator {

        private static final double[][] COEFF_K = {
                        { 1.0 / 5.0 },
                        { 3.0 / 40.0, 9.0 / 40.0 },
                        { 44.0 / 45.0, -56.0 / 15.0, 32.0 / 9.0 },
                        { 19372.0 / 6561.0, -25360.0 / 2187.0, 64448.0 / 6561.0, -212.0 / 729.0 },
                        { 9017.0 / 3168.0, -355.0 / 33.0, 46732.0 / 5247.0, 49.0 / 176.0, -5103.0 /
18656.0 },
```

```
                        { 35.0 / 384.0, 0.0, 500.0 / 1113.0, 125.0 / 192.0, -2187.0 / 6784.0, 11.0 /
84.0 }
                        };
        private static final double[] COEFF_M = { 1.0 / 5.0, 3.0 / 10.0, 4.0 / 5.0, 8.0 / 9.0, 1.0,
1.0 };
        /** Прогрес-бар інтегрування */
        private final ProgressBar pb;
        /** Математична модель для обчислень */
        @Setter
        @Getter
        private MathematicModel model = MathematicModel.CHAIN;
        /** Кількість точок, які необхідно рахувати */
        @Setter
        @Getter
        private int points = 800;
        /** Крок інтегрування */
        @Setter
        @Getter
        private double step = 0.000001;
        /** Очікувана точність для автоматичного вибору кроку */
        @Setter
        @Getter
        private double epsilon = 0.000001;
        /** Межі інтегрування */
        @Setter
        @Getter
        private double t0, t1 = 0.06;
        /** Початкове наближення */
        @Setter
        @Getter
        private double[] initialX = new double[5];
        /** Історія зміни кроку */
        @Getter
        private List<Pair<Double, Double>> stepChanges = new ArrayList<>();

        public Integrator(ProgressBar pb) {
                this.pb = pb;
        }

        public List<List<Pair<Double, Double>>> integrate() {
                double[][] functionValuesGraph = new double[model.getSize()][points + 1];
                double[] functionValues = new double[model.getSize()],
                solutionsX = new double[model.getSize()],
                stepArray = new double[model.getSize()],
                stepsForGraph = new double[points + 2],
                k1 = new double[model.getSize()],
                k2 = new double[model.getSize()],
                k3 = new double[model.getSize()],
                k4 = new double[model.getSize()],
                k5 = new double[model.getSize()],
                k6 = new double[model.getSize()],
                y = new double[model.getSize()];
                stepChanges.clear();
                stepChanges.add(new Pair<>(t0, step));
                System.arraycopy(initialX, 0, solutionsX, 0, model.getSize());
                double stepGraph = (t1 - t0) / points,
                nextProgressBarUpdate = stepGraph / 4,
                t = t0;
                stepsForGraph[0] = t0;
                stepsForGraph[1] = stepGraph;
                for (int i = 0; i < model.getSize(); i++) {
                        functionValuesGraph[i][0] = initialX[i];
                        stepArray[i] = step;
                }
                int point = 1;
                do {
                        if (Thread.interrupted())
                                return Collections.emptyList();
                        functionValues = model.func(t + step * COEFF_M[0], solutionsX);
                        for (int i = 0; i < model.getSize(); i++) {
                                k1[i] = step * functionValues[i];
                                y[i] = solutionsX[i] + k1[i] * COEFF_K[0][0];
                        }
                        functionValues = model.func(t + step * COEFF_M[1], y);
                        for (int i = 0; i < model.getSize(); i++) {
                                k2[i] = step * functionValues[i];
                                y[i] = solutionsX[i] + k1[i] * COEFF_K[1][0] + k2[i] * COEFF_K[1][1];
                        }
                        functionValues = model.func(t + step * COEFF_M[2], y);
                        for (int i = 0; i < model.getSize(); i++) {
                                k3[i] = step * functionValues[i];
```

```
                                y[i] = solutionsX[i] + k1[i] * COEFF_K[2][0] + k2[i] * COEFF_K[2][1] +
k3[i] * COEFF_K[2][2];
                        }
                        functionValues = model.func(t + step * COEFF_M[3], y);
                        for (int i = 0; i < model.getSize(); i++) {
                                k4[i] = step * functionValues[i];
                                y[i] = solutionsX[i] + k1[i] * COEFF_K[3][0] + k2[i] * COEFF_K[3][1] +
k3[i] * COEFF_K[3][2] + k4[i] * COEFF_K[3][3];
                        }
                        functionValues = model.func(t + step * COEFF_M[4], y);
                        for (int i = 0; i < model.getSize(); i++) {
                                k5[i] = step * functionValues[i];
                                y[i] = solutionsX[i] + 9017d / 3168 * k1[i] - 355d / 33 * k2[i] - 46732d
/ 5247 * k3[i] - 49d / 176 * k4[i]
                                                - 5103d / 18656 * k5[i];
                                y[i] = solutionsX[i] + k1[i] * COEFF_K[4][0] + k2[i] * COEFF_K[4][1] +
k3[i] * COEFF_K[4][2] + k4[i] * COEFF_K[4][3] + k5[i] * COEFF_K[4][4];
                        }
                        functionValues = model.func(t + step * COEFF_M[5], y);
                        for (int i = 0; i < model.getSize(); i++) {
                                k6[i] = step * functionValues[i];
                                y[i] = solutionsX[i] + 35d / 384 * k1[i] + 500d / 1113 * k3[i] + 125d /
192 * k4[i] - 2187d / 6784 * k5[i]
                                                + 11d / 84 * k6[i];
                                y[i] = solutionsX[i] + k1[i] * COEFF_K[5][0] + k2[i] * COEFF_K[5][1] +
k3[i] * COEFF_K[5][2] + k4[i] * COEFF_K[5][3] + k5[i] * COEFF_K[5][4] + k6[i] * COEFF_K[5][5];
                        }
                        for (int i = 0; i < model.getSize(); i++)
                                solutionsX[i] += k1[i] * COEFF_K[5][0] + k2[i] * COEFF_K[5][1] + k3[i] *
COEFF_K[5][2] + k4[i] * COEFF_K[5][3] + k5[i] * COEFF_K[5][4] + k6[i] * COEFF_K[5][5];
                        t += step;
                        final double localT = t;
                        if (t > nextProgressBarUpdate){
                                nextProgressBarUpdate += stepGraph / 4;
                                Platform.runLater(() -> pb.setProgress(localT/t1));
                        }
                        double errorEstimate = 0;
                        for (int i = 0; i < k5.length; i++){
                                double temp = Math.abs(k5[i] - k6[i]);
                                if (temp > errorEstimate)
                                        errorEstimate = temp;
                        }
                        if (verifyStep(errorEstimate))
                                stepChanges.add(new Pair<>(t, step));
                        if (t > stepsForGraph[point]){
                                for (int i = 0; i < model.getSize(); i++)
                                        functionValuesGraph[i][point] = solutionsX[i];
                                stepsForGraph[point + 1] = stepsForGraph[point] + stepGraph;
                                point++;
                        }
                } while (t < t1);
                if (stepChanges.size() > 25_000)
                        stepChanges.subList(25_000, stepChanges.size()).clear();
                List<List<Pair<Double, Double>>> list = new ArrayList<>();
                for (int i = 0; i < model.getSize(); i++){
                        List<Pair<Double, Double>> local = new ArrayList<>();
                        for (int j = 0; j < points; j++)
                                local.add(new Pair<>(stepsForGraph[j], functionValuesGraph[0][j]));
                        list.add(local);
                }
                return list;
        }

        /**
         * Метод динамічної зміни кроку (трьох зон).
         * @param ee локальна похибка
         */
        private boolean verifyStep(double ee){
                if (ee > epsilon)
                        return (step /= 2) > 0;
                else if (ee < epsilon / 25)
                        return (step *= 5) > 0;
                return false;
        }

}
```