УДК 517.946

Вейцбліт О. Й.[1], Шепель М. С.[1], Вигоднер І. В.[2]
[1]Херсонський державний університет, Херсон, Україна
[2]Херсонський національний технічний університет, Херсон, Україна

ПРОГРАМНИЙ ЗАСІБ ВІДОКРЕМЛЕННЯ КОРЕНІВ НА ВІДРІЗКУ

DOI: 10.14308/ite000657

Усі поширені, відомі алгоритми чисельного розв'язання рівняння на відрізку прямої передбачають, що заздалегідь відомими є відрізки ізоляції коренів. Це такі відрізки, в кожному з яких рівняння має один і тільки один розв'язок. У цій роботі пропонується метод, що дозволяє знайти всі корені рівняння f(x) = 0 для довільної неперервно диференційованої функції f(x) на заданому відрізку прямої з заданою точністю. Збіжність методу є експоненціальною. Отже метод автоматично відділяє корені. У курсі чисельних методів він потрапляє в його ідейний центр, примушує замислитись над структурою теорії, прояснити та поглибити її розуміння. Метод роботи реалізований у десктопі на мові Java.

Ключові слова: *чисельні методи; оператор стиску; експоненціальна збіжність; відрізки ізоляції; десктоп; діаграма класів; віконний інтерфейс; графічні об'єкти.*

Вступ. Ми живемо у час технологій, перш за все – у добу стрімкого розвитку інформаційних технологій. В результаті постійно триває швидка зміна поглядів на весь комплекс проблем, пов'язаних з їх вивченням та викладанням. Проте, незмінним і незаперечним є те, що розвиток інформаційних технологій є можливим лише за умови застосування чисельних методів математики (методів обчислень).

Методи обчислень – це алгоритми знаходження чисельних розв'язків основних математичних задач, тому навчальний курс обчислювальних методів базується на всіх основних математичних курсах. Це створило досить вільний стиль викладання у поширених підручниках чисельних методів, де необхідною передумовою їх вивчення вважається ґрунтовне знання всієї математики в цілому. Такі вимоги і такий стиль є серйозною проблемою навіть для найкращих студентів. Проте, будь-який підручник не може бути збірником рецептів розв'язання всіх реальних задач, бо різноманітність їх є практично необмеженою. Його реальна мета – вивчення основних понять, задач та ідей чисельних методів. Ці ідеї, поняття та задачі чисельних методів виявляються загальними насправді для всіх розділів математики. Здоровий глузд і традиційний для математики стиль викладання також вимагають обмежитись у викладанні в основному найбільш простими і відомими обчислювальними методами. Методи розв'язання рівнянь на відрізку – важлива складова чисельних методів і завдяки відносній простоті це мабуть ідейний центр при їх вивченні [1].

Водночас чисельні методи щільно зв'язані з курсами алгоритмізації та програмування. Уявлення, що при вивченні методів обчислень програми є лише застосуванням та ілюстрацією теоретичних надбань, створює дуже скривлену панораму. Насправді вони мотивують подальші поняття та методи, вони є базою вправ, пов'язаних з модифікацією алгоритмів. Відкритість таких програм прислуговує легкості їх модифікації, потужний графічний інтерфейс – аналізу результатів, який іноді здатен перетворюватися у "математику в малюнках".

Метод цієї роботи реалізований у десктопі на мові Java. Програми на Java транслюються в байт-код, що виконується віртуальною машиною Java (JVM) – програмою, що обробляє байтовий код і передавальні інструкції обладнанню як інтерпретатор.

Перевагою такого способу виконання програм є повна незалежність байт-коду від операційної системи й устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина. Іншою важливою особливістю технології Java є гнучка система безпеки, забезпечена тим, що виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання.

До недоліків концепції віртуальної машини часто відносять зниження продуктивності. Програми, написані на Java, мають репутацію більш повільних і займають більше оперативної пам’яті, ніж написані на мові C [2]. Проте, ряд удосконалень значно збільшив швидкість виконання програм на Java:

- застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми з можливістю збереження версій класу в машинному коді (JIT-технологія),
- широке використання переносного орієнтованого коду (native-код) в стандартних бібліотеках,
- апаратні засоби, що забезпечують прискорену обробку байт-коду (наприклад, технологія Jazelle, підтримувана деякими процесорами фірми ARM),
- нові особливості мови для підтримки кращого аналізу коду (такі, як внутрішні класи, клас StringBuffer, спрощені логічні обчислення тощо).

Швидкість виконання програм, написаних на мові Java, була істотно поліпшена з випуском в 1997-1998 роках так званого JIT-компілятора в версії 1.1. Згодом проведена оптимізація віртуальної машини Java – з 2000 року для цього використовується віртуальна машина HotSpot. Станом на лютий 2012 року код Java 7 приблизно вже лише в 1.8 рази повільніше коду, написаного на мові C.

Ідеї, закладені в концепцію і різні реалізації середовища віртуальної машини Java, надихнули безліч ентузіастів на розширення переліку мов, які могли б бути використані для створення програм, що виконуються на віртуальній машині. Ці ідеї знайшли також вираз в специфікації загальномовної інфраструктури CLI, закладеної в основу платформи .NET компанією Microsoft. Сьогодні багато платформ пропонують апаратну підтримку виконання програм на мові Java.

Прямуючи тепер безпосередньо до мети даної роботи, слід зауважити, що всі поширені, загальновідомі алгоритми знаходження коренів рівняння (методи Ньютона, дихотомії, простої ітерації, …) передбачають заздалегідь відомими відрізки ізоляції коренів – це такі, в кожному з яких рівняння має один і тільки один розв‘язок [2]. Отже, чисельне розв’язання рівняння на відрізку складається з двох етапів:

1) відокремлення коренів, тобто знаходження для кожного з них відрізку ізоляції та
2) застосування на кожному відрізку ізоляції одного з методів для обчислення кореня з заданою похибкою.

Причому відокремлення коренів не розглядається, як алгоритм у звичайному сенсі цього слова: графічний метод відокремлення не є математично обґрунтованим і не гарантує отримання відрізків ізоляції; аналітичний метод може бути застосований лише до вузьких класів функцій [3]. Все це не випадково, не є недоробком, наявність відрізку ізоляції є необхідною умовою застосування теорії стискуючого відображення [4], на якій ґрунтуються всі методи знаходження коренів [5]. А насправді ця теорія є підґрунтям всіх коректних чисельних методів взагалі [6]. Ось чому метод та програма цієї роботи потрапляють в ідейний центр освітнього курсу чисельних методів: вони примушують замислитись над структурою теорії, прояснити та поглибити її розуміння.

Мета статті. У цій роботі пропонується програмний засіб реалізації методу, що дозволяє знайти всі корені рівняння $f(x) = 0$ для довільної неперервно диференційованої функції $f(x)$ на відрізку прямої $[a;b]$ з заданою точністю, причому його збіжність є експоненціальною. Отож, він автоматично відділяє корені. На кожному отриманому таким чином відрізку ізоляції можна застосувати і будь-який інший алгоритм: отже, цей метод природно вважати перш за все алгоритмом відокремлення коренів.

Виклад основного матеріалу дослідження. Основна конструкція методу: розглянемо ітераційний процес $x_{k+1} = \varphi(x_k)$, де $\varphi(x) = x - \lambda \cdot f(x)$ із сталим λ. Оберемо таке λ, щоби функція $\varphi(x)$ була монотонно зростаючою: $\varphi'(x) = 1 - \lambda \cdot f'(x) \geq 0$, звідки $|\lambda| \leq 1/M$, де $M = \max\limits_{[a;b]} |f'(x)|$. З таким λ монотонним є ітераційний процес $x_{k+1} = \varphi(x_k)$: $x_{k+1} \geq x_k$ або $x_{k+1} \leq x_k$ для всіх k. Отже існує границя $\lim\limits_{k\to\infty} x_k = x^*$. Звідси $\varphi(x^*) = x^*$: справді, $\varphi(x^*) = \varphi(\lim\limits_{k\to\infty} x_k) = \lim\limits_{k\to\infty} \varphi(x_k) = \lim\limits_{k\to\infty} x_{k+1} = x^*$. Отже $f(x^*) = 0$, а ітераційний процес $x_{k+1} = \varphi(x_k)$ збігається до кореня $f(x)$.

На рис.1 корені – це точки перетину графіка функції $y = \varphi(x)$ з прямою $y = x$.

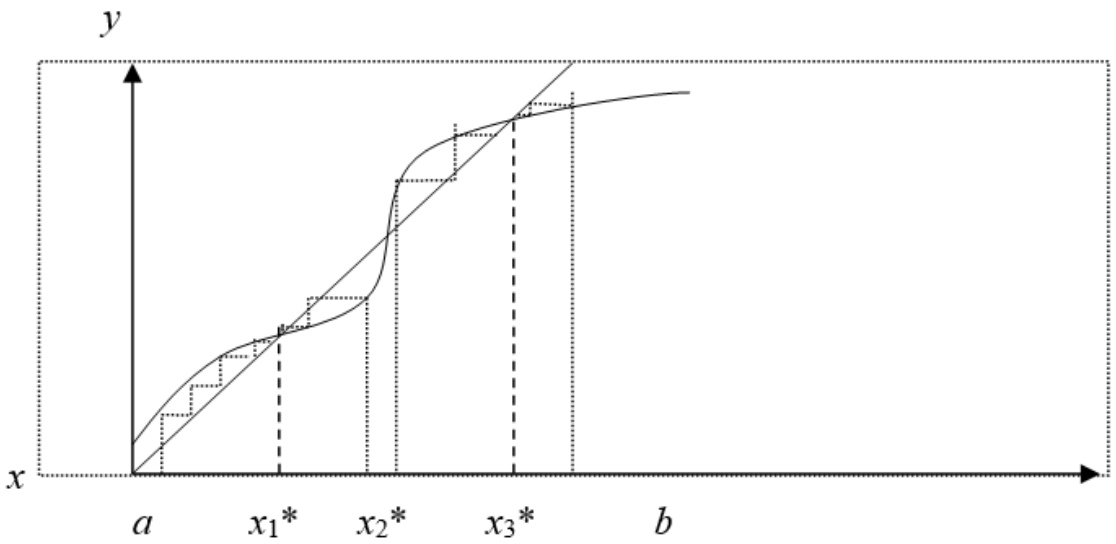

Рис.1. Основна конструкція методу.

З рисунку видно, що до першого кореня x_1^* збігається процес $x_{k+1} = \varphi(x_k)$, якщо він починається в його околі між a та другим коренем x_2^*. Тобто x_1^* – стійка нерухома точка процесу; так само і третій корінь x_3^*. Другий корінь x_2^* – нестійка нерухома точка, до неї не збігається жоден процес $x_{k+1} = \varphi(x_k)$, якщо не починається безпосередньо в точці x_2^*. У загальному випадку нерухома точка x_i^* є стійкою, якщо $\varphi'(x_i^*) < 1$ та нестійкою, якщо $\varphi'(x_i^*) > 1$. Стійкі та нестійкі нерухомі точки чергуються: за стійкою нестійка і навпаки. Якщо змінити знак λ, тобто розглянути процес $x_{k+1} = \varphi_1(x_k)$, де $\varphi_1(x) = x + \lambda \cdot f(x)$, то стійкі нерухомі точки перетворяться на нестійкі і навпаки. Отже або процес $x_{k+1} = \varphi(x_k)$, або $x_{k+1} = \varphi_1(x_k)$ збігається до кожного кореня $f(x)$ на $[a;b]$.

Насправді так буде, якщо всі точки x_i^* не вироджені: $\varphi'(x_i^*) \neq 1$, звідки $f'(x_i^*) \neq 0$, тобто всі корені $f(x)$ на $[a;b]$ однократні. Проте, якщо навіть $f(x)$ має кратні корені, то згідно з теоремою Сарда [7], функція $f_1(x) = f(x) \pm \delta \cdot \sin x$ має лише однократні корені при майже всіх δ. Тоді до $f_1(x)$ можна застосувати описаний алгоритм, а корені $f_1(x)$ відрізняються від коренів $f(x)$ щонайбільше на припустиму похибку ε, якщо $0 < \delta < \varepsilon$.

Тепер сформулюємо алгоритм відокремлення коренів формально. Нехай $f(x)$ – задана неперервно диференційована функція на заданому відрізку $[a;b]$, що не має кратних коренів; ε – задана припустима похибка, $M = \max\limits_{[a;b]} |f'(x)|$.

Початкова ітерація 1) x_0 повинна задовольняти умові $f(x_0) \cdot f'(x_0) < 0$. Таке значення завжди знайдеться на відрізку від a до першого кореня x_1^* оскільки там в деякому малому околі x_1^* якщо $f(x) > 0$, то $f(x)$ спадає, тобто $f'(x) < 0$ і навпаки якщо $f(x) < 0$, то $f'(x) > 0$. Зокрема $x_0 = a$, якщо $f(a) \cdot f'(a) < 0$.

2) Оберемо λ таким, що $|\lambda| = 1/M$, $\lambda f'(x_0) > 0$.

Основний цикл. Нехай $\varphi(x) = x - \lambda \cdot f(x)$. Обчислюємо $x_1 = \varphi(x_0)$, $q_1 = \lambda \cdot f'(x_1)$, $x_2 = \varphi(x_1)$, $q_2 = \lambda \cdot f'(x_2)$, …, $x_k = \varphi(x_{k-1})$, $q_k = \lambda \cdot f'(x_k)$, … . Тут $0 < q_i = \lambda \cdot f'(x_i) < 1$.

Цикл продовжується доки $(x_i - x_{i-1})/ q_k > \varepsilon$.

Перехід до наступного циклу. Якщо $(x_k - x_{k-1})/ q_k < \varepsilon$, то 1) фіксуємо наближене значення кореня $x^* = x_k$; 2) обираємо нову початкову ітерацію $x_0 = x^* + \varepsilon$; 3) змінюємо знак λ, тобто множимо λ на -1. Повертаємось до основного циклу.

Алгоритм продовжується доки x_k не перевищить b.

Такий алгоритм дозволяє визначити всі корені рівняння $f(x) = 0$ на відрізку прямої $[a;b]$ з заданою точністю ε, причому його збіжність є експоненціальною. Це безпосередньо випливає з наступного загального твердження.

Теорема. Нехай функція $y = \varphi(x)$ відображає відрізок $[a;b]$ у себе та задовольняє на $[a;b]$ умові $|\varphi(x_2) - \varphi(x_1)| \le q |x_2 - x_1|$, де $0 < q < 1$. Тоді

1) Відображення $y = \varphi(x)$ є стискуючим відображенням на $[a;b]$.
2) $[a;b]$ – відрізок ізоляції нерухомої точки x^* відображення $y = \varphi(x)$: $x^* = \varphi(x^*)$.
3) Ітераційний процес $x_k = \varphi(x_{k-1})$ на $[a;b]$ збігається до x^* при кожному початковому x_0.
4) Швидкість цієї збіжності є експоненціальною: $|x_k - x^*| \le q^k |x_0 - x^*|$.

Справді, за умовою теореми відображення $y = \varphi(x)$ є оператором стиску на $[a;b]$. Тому з принципу стискуючих відображень [4] випливає, що на $[a;b]$ існує одна і тільки одна нерухома точка відображення $y = \varphi(x)$ (за умовою це x^*). 3) також випливає з принципу стискуючих відображень безпосередньо. 4) З умови теореми маємо: $|x_k - x^*| = |\varphi(x_{k-1}) - \varphi(x^*)| \le q |x_{k-1} - x^*| = q |\varphi(x_{k-2}) - \varphi(x^*)| \le q^2 |x_{k-2} - x^*| \le \ldots \le q^k |x_0 - x^*|$.

Інструменти реалізації. Метод цієї роботи реалізований у десктопі на мові Java версії 1.7 [7]. При створенні програмного засобу були використані такі системні бібліотеки Java:

- java.lang.Math для розрахунків функцій;
- apache.commons.math3.analysis.differentiation.DerivativeStructure для розрахунків похідних;
- jfree.chart для побудови графічних об'єктів;
- java.swing для створення віконного інтерфейсу.

1. Клас Math, пакету java.lang.Math містить методи для виконання основних числових операцій таких, як елементарна експоненція, логарифм, квадратний корінь та тригонометричні функції. На відміну від чисельних методів класу StrictMath цього ж пакету, всі реалізації функцій класу Math не вимагають суворо відтворювати результати біт-на-біт. Це послаблення дозволяє більш ефективну реалізацію. Для забезпечення більш високопродуктивних реалізацій методів математики генераторам коду рекомендується використовувати локальні бібліотеки, специфічні для платформи, або, де це можливо, інструкції мікропроцесорів. Такі реалізації більш високої продуктивності повинні також відповідати специфікації математики.

Ця специфікація стосується двох властивостей: точності повернутого результату та монотонності методу. Точність методів з плаваючою точкою згідно з методами обчислень вимірюється одиницею на останньому місці. Метод повертає число з плаваючою точкою, найближче до точного результату, якщо він завжди має помилку менше, ніж 0,5 Кбайт. Саме такий метод вважається методом з правильним округленням. Натомість, метод з правильним округленням не завжди є практичним, для деяких методів класу Math допускається більша помилка: звичайно до 1 Кбайт. Окрім точності при індивідуальних аргументах, важливим є збереження правильних відносин між різними аргументами. Тому методи, що мають помилки більші ніж 0,5 Кбайт, повинні бути напівмонотонними: коли математична функція з плаваючою точкою не зменшується, то і наближення не зменшується; коли математична функція не зростає, то так само не зростає і наближення.

2. Клас `DerivativeStructure` використовується для роботи з диференціалами і відноситься до пакету org.apache.commons.math3.analysis.differentiation компанії Apache. Цей клас являє собою реалізацію розширення на номери Ралла, описані в роботі Дана Калмана "Рекурсивне багатоваріантне автоматичне диференціювання" [8].

Номери Rall - це розширення реальних чисел, що використовуються у всьому математичному виразі; вони тримають похідну разом із значенням функції. Похідні структури Дана Калмана зберігають всі часткові похідні до будь-якого заданого порядку щодо будь-якої кількості вільних параметрів. Таким чином, числа Rall можна розглядати як похідні структури для похідної одного порядку та одного вільного параметра; а дійсні числа можна розглядати як похідні структури з похідною нульового порядку та без вільних параметрів. Екземпляри derivivativeStructure можуть бути використані до математичних функцій безпосередньо завдяки арифметичним операторам, що надаються у вигляді методів цього класу (+, -, *, /,%, sin, cos ...).

3. Пакет jfree.chart містить класи для створення графіків. Проект JFreeChart був заснований у лютому 2000 року Дейвідом Гилбертом. Сегодня библиотека JFreeChart широко використовується в Java-додатках для створення широкого спектра графіків. Використовуючи JFreeChart, можна створювати всі основні типи 2D та 3D графіків: кругові діаграми, гістограми, лінійні та часові діаграми. Бібліотека дозволяє створювати зображення декількох форматів: типу PNG, JPEG, SVG (Scalable Vector Graphics) та інші. JFreeChart забезпечена добре документованим API, поставляється з відкритим вихідним кодом і безкоштовно. Це дозволяє використовувати її в комерційних цілях без будь-яких додаткових витрат. Дистрибутив бібліотеки jfreechart поставляється у вигляді zip файлу і включає maven проект з документацією і демонстраційними прикладами. Завантажити останню версію бібліотеки jfree.chart можна з офіційного сайту [9].

Основою наборів даних для створення різних графічних зображень типу кругових діаграм і гістограм є клас AbstractDataset. Даний клас є батьком DefaultPieDataset, який використовується для формування кругових діаграм, і DefaultCategoryDataset для побудови гістограм. Для побудови тимчасової діаграми використовується набір даних TimeSeries і колекція TimeSeriesCollection, що реалізує інтерфейс XYDataset. Вузлові точки на графіку зображені певними фігурами (квадрат, трикутник тощо) в кольорі графіка. Фігури можна не зображати, використовуючи метод setShapesVisible (setSeriesShapesVisible) з параметром false. Також можна приховати лінії графіка методом setLinesVisible (setSeriesLinesVisible), залишивши тільки значення. Для визначення кольору і ширини лінії графіка використовуються методи setSeriesPaint і setSeriesStroke. Використання класу XYSplineRenderer дозволяє представити графіки не у вигляді кусочно-лінійних функцій, а у вигляді згладжених ліній.

4. Графічний інтерфейс в Java пройшов вельми тернистий шлях розвитку і становлення. Довгий час його звинувачували в повільній роботі, жадібності до ресурсів системи і обмеженої функціональності. Першою спробою Sun створити графічний інтерфейс

для Java була бібліотека AWT (Abstract Window Toolkit) - інструментарій для роботи з різними віконними середовищами. Sun зробив прошарок на Java, яка викликає методи з бібліотек, написаних на C. Бібліотечні методи AWT створюють і використовують графічні компоненти операційного середовища. З одного боку, це добре, тому що програма на Java схожа на інші програми в рамках однієї ОС. Але при запуску її на іншій платформі можуть виникнути розбіжності в розмірах компонентів і шрифтів, які будуть псувати зовнішній вигляд програми. Щоб забезпечити мультиплатформеність AWT інтерфейси викликів компонентів були уніфіковані, внаслідок чого їх функціональність вийшла трохи урізаною. Та й набір компонентів вийшов досить невеликий. Так наприклад, в AWT немає таблиць, а в кнопках не підтримує відображення іконок. Проте пакет java.awt входить в Java з самого першого випуску і його можна використовувати для створення графічних інтерфейсів. Таким чином, компоненти AWT не виконують ніякої «роботи». Це просто «Java-оболонка» для елементів управління тієї операційної системи, на якій вони працюють. Всі запити до цих компонентів перенаправляються до операційної системи, яка і виконує всю роботу. Використані ресурси AWT намагається звільняти автоматично. Це трохи ускладнює архітектуру і впливає на продуктивність. Написати щось серйозне з використанням AWT буде трохи важко. Зараз її використовують хіба що для аплетів.

Слідом за AWT Sun розробила графічну бібліотеку компонентів Swing, повністю написану на Java. Для відтворення використовується 2D, що принесло з собою відразу кілька переваг. Набір стандартних компонентів значно перевершує AWT за різноманітністю і функціональністю. Swing дозволяє легко створювати нові компоненти, наслідуючи від існуючих, і підтримує різні стилі і скінчи. Творці нової бібліотеки, призначеної для користувача інтерфейсу Swing, не стали «винаходити велосипед» і в якості основи для своєї бібліотеки вибрали AWT. Звичайно, мова не йшла про використання конкретних великовагових компонентів AWT (представлених класами Button, Label і їм подібними). Найважливішою відмінністю Swing від AWT є те, що компоненти Swing взагалі не пов'язані з операційною системою і тому набагато більш стабільні і швидкі. Такі компоненти в Java називаються легковагими (lightweight), і розуміння основних принципів їх роботи багато в чому пояснить роботу Swing. Для створення графічного інтерфейсу додатку необхідно використовувати спеціальні компоненти бібліотеки Swing, звані контейнерами вищого рівня (top level containers). Вони являють собою вікна операційної системи, в яких розміщуються компоненти користувальницького інтерфейсу. До контейнерів вищого рівня відносяться вікна JFrame і JWindow, діалогове вікно JDialog, а також аплет JApplet (який не є вікном, але теж призначений для виведення інтерфейсу в браузері). Контейнерами вищого рівня Swing є великовагові компоненти і це є винятком із загального правила. Всі інші компоненти Swing є легковажними.

Кожен раз, як тільки створюється контейнер вищого рівня, будь то звичайне вікно, діалогове вікно або аплет, в конструкторі цього контейнера створюється кореневa панель JRootPane. Контейнери вищого рівня Swing стежать за тим, щоб інші компоненти не змогли "пробратися" за межі JRootPane. Коренева панель JRootPane додає в контейнери властивість "глибини", забезпечуючи можливість не тільки розміщувати компоненти один над іншим, а й при необхідності міняти їх місцями, збільшувати або зменшувати глибину розташування компонентів. Така можливість необхідна при створенні багатодокументного додатки Swing, у якого вікна представляють легковагі компоненти, розташовані один над одним, а також випадають (контекстними) меню у спливаючі підказки. Усі складові кореневої панелі JRootPane можна отримати або змінити. Для цього у неї є набір методів get / set. Програмним способом JRootPane можна отримати з використанням методу getRootPane (). Крім контейнерів вищого рівня, коренева панель застосовується у внутрішніх вікнах JInternalFrame, створюваних в багатодокументне застосування і розташованих на "робочому

столі" JDesktopPane. Це дозволяє забути про те, що дані вікна представляють собою звичайні легковагі компоненти, і працювати з ними як зі справжніми контейнерами вищого рівня.

У підставі кореневої панелі (контейнера) лежить так звана багатошарова панель JLayeredPane, що займає весь доступний простір контейнера. Саме в цій панелі розташовуються всі інші частини кореневої панелі, в тому числі і всі компоненти користувальницького інтерфейсу. JLayeredPane використовується для додавання в контейнер властивості глибини (depth). То є, багатошарова панель дозволяє організувати в контейнері третій вимір, уздовж якого розташовуються шари (layers) компонента. У звичайному контейнері розташування компонента визначається прямокутником, який показує, яку частину контейнера займає компонент. При додаванні компонента в багатошарову панель необхідно вказати не тільки прямокутник, яку він обіймав компонентом, але і шар, в якому він буде розташовуватися. Шар в багатошарової панелі визначається цілим числом. Чим більше визначальне шар число, тим вище шар знаходиться. Перший доданий у контейнер компонент виявляється вище компонентів, доданих пізніше. Найчастіше розробник не має справу з позиціями компонентів. При додаванні компонентів їх положення змінюються автоматично. Проте багатошарова панель дозволяє міняти позиції компонентів динамічно, вже після їх додавання в контейнер. Можливості багатошарової панелі широко використовуються деякими компонентами Swing. Особливо вони важливі для багатодокументного застосування, спливаючих підказок і меню. Багатодокументна Swing додатки задіють спеціальний контейнер JDesktopPane («робочий стіл»), успадкований від JLayeredPane, в якому розташовуються внутрішні вікна Swing. Найважливіші функції багатодокументного додатки - розташування «активного» вікна над іншими, згортання вікон, їх перетягування - забезпечуються механізмами багатошарової панелі. Основна перевага від використання багатошарової панелі для спливаючих підказок і меню - це прискорення їх роботи. Замість створення для кожної підказки або меню нового великовагового вікна, розташованого над компонентом, в якому виник запит на висновок підказки або меню, Swing створює швидкий легкий компонент. Цей компонент розміщується в досить високому шарі багатошарової панелі вище всіх інших компонентів і використовується для виведення підказки або меню.

Реалізація програмного засобу. Вся програма складається з основної форми та модулю. Основну форму можна поділити на три основні частини. Перша частина представляє рамку з елементами (GroupBox), на ній знаходяться поля для вводу коефіцієнтів та відрізків пошуку коренів (TextBox), кнопки вибору функції (RadioButton) та кнопки для того, щоб виконати розрахунок (Button). Друга частина – це поле, де будуть відображатись вихідні данні. Третя частина – графік, який буде показувати нашу ітераційну функцію.

Модуль містить всі методи, функції та класи, які обробляють дії користувача. В ньому знаходяться наступний перелік класів:

I. основний клас програми, або графічний інтерфейс;
II. клас, який об’єднує всі ресурси з графічним інтерфейсом;
III. клас, який підключає стилі візуалізації, задає параметри за умовчання, та запускає форму, роблячи її видимою;
IV. клас реалізації основного методу;
V. клас, в якому представлені константи та статичні методи для тригонометричних, логарифмічних та інших математичних формул, потрібних для реалізації алгоритму.

Діаграма класів додатку наведена на наступному рисунку.

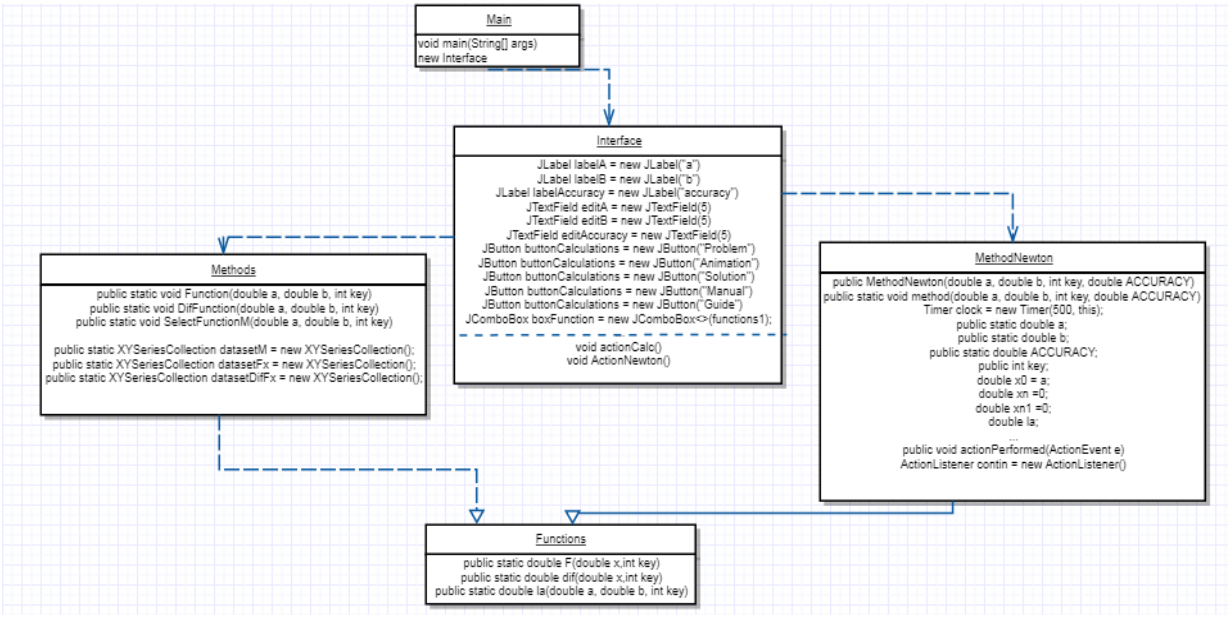

Рис. 2. Діаграма класів.

Розгляд програми буде виконано в виді опису реалізації процесів та функцій. Почнемо зі створення інтерфейсу для користувача.

```
    public static JFrame frame = new JFrame();
    public static JPanel panelShowStep = new JPanel(new
GridLayout(1,1));
    public static JPanel panelShowFx = new JPanel();
    public static JPanel panelShowDifFx = new JPanel();
    public static JPanel panelShowSelectM = new JPanel();
    public static JTextField editA = new JTextField(5);
    public static JTextField editB = new JTextField(5);
    public static JTextField editAccuracy = new JTextField(5);
    public static JTextField editTime = new JTextField(5);
    panelSelect.setBackground(new Color(73, 163, 129));
    panelSelect.setPreferredSize(new Dimension(190,0));
    frame.add(panelSelect, BorderLayout.WEST);
    JPanel panelFunction = new JPanel(new GridLayout(2,10));
    panelFunction.setForeground(new Color(73, 163, 129));
    panelFunction.setPreferredSize(new Dimension(0,150));
```

Задамо розміри вікна за допомогою методу Dimension. Відношення сторін для коректного переводу функції в координати об’являємо BorderLayout. Для установки кольору очистки екрана пишемо наступне setForeground(new Color(73, 163, 129)).

Створюємо об’єкти панелі JPanel для розміщення компонентів інтерфейсу. Також JTextField, JButton, JLabel потрібні для введеня та роботи з даними.

Розглянемо метод для вибору функції. Першим буде обробник завантаження формул.

```
public static double F(double x,int key){
      switch (key) {
      case 1: return  Math.pow(Math.sin(x), 2);
      case 2: return  Math.pow(Math.cos(x), 2);
      case 3: return Math.pow(x, 2)-1;//
```

```
        case 4: return  Math.sin(x)+Math.cos(x);
        case 5: return  Math.pow(x, 2)- 2*x;
        case 6: return  Math.pow(x, 4)-Math.pow(x, 2) -1;//
        case 7: return  Math.pow(x,2)+x/2 -5;
        case 8: return  Math.abs(Math.pow(x,3))-27;
        case 10: return  Math.sin(x)*Math.cos(x);
        case 11: return Math.pow(x,2)+ 2*Math.sin(x)-3;
        default:
             return 0;
        }
```

Умова визначення параметрів проекції у case в залежності від розмірів, які ми обрали.

```
public static double dif(double x,int key){
    int params = 1;
    int order = 2;
    double xRealValue =  x;
    DerivativeStructure myX = new DerivativeStructure(params, order,
0, xRealValue);
    DerivativeStructure f;
    switch (key) {
      case 1:
           f = (myX.sin().pow(2));
           return     f.getPartialDerivative(1);
      case 2:
           f = (myX.cos().pow(2));
      return     f.getPartialDerivative(1)…
```

Функція, яка буде здійснювати візуалізацію текстових рядків, встановлює координати виведення растрових символів відповідно до координат, знайдених в алгоритмі, а потім у циклі перебирає всі символи в параметрі рядка тексту. Кожен з символів проходить візуалізацію за допомогою функції. В ній вказується шрифт для виведення і змінна типу char. Розглянемо функції класу Methods для додавання координат: в нашому випадку вони виглядатимуть наступним чином

```
public static void Function(double a, double b, int key)
public static void DifFunction(double a, double b, int key)
public static void SelectFunctionM(double a, double b, int key)
```

Встановлюємо позицію виведення растрових символів в переданих координатах *x* та *y* Gl.glRasterPos2f (x, y). В циклі перебираємо значення з масиву, який містить значення рядка для візуалізації. Символ С візуалізуємо.

Далі буде опис функції, що обчислює координати для побудови графіка. В ній ініціалізується масив і проводиться обчислення всіх координат графіка в залежності від зазначеного діапазону значень x і кроку збільшення цих значень.

Треба звернути увагу на те, що при ініціалізації масиву для зберігання координат має бути зазначена така кількість елементів масиву, щоб в подальшому їх вистачило для розміщення всіх координат; інакше станеться виняток, так як програма в процесі роботи спробує звернутися до області пам'яті, яка їй не належить.

Далі розберемо один з прикладів пошуку точок.

```
    public static void method(double a, double b, int key, double
ACCURACY){
```

```
dataset.addSeries(seriesYx);
dataset.addSeries(seriesGGx);
dataset.addSeries(seriesRoot);

double x0 = a;
double xn =0;
double xn1 =0;
double la = Math.abs(Functions.la(a, b, key));
….
}
```

Функція public static void *method* робить обчислення координат графіка для першого прикладу та заносить їх у серії XYSeries. Вона включає в себе такі дії. Визначення локальних змінних x і x0, та la, яка повинна змінювати знак після кожного знайденого кореня. Процес ініціалізації масиву public static XYSeries *seriesGx* = new XYSeries("G(X)"); Він буде зберігати значення всіх точок для побудови графік. Цикл обчислення всіх значень x для $x0$, що належить проміжку [a;b] з кроком в x. Першим обчисленням *x* для поточного x0

```
for(x0=a;x0<=b;x0+=STEP){
            xn = x0 + la*Functions.F(x0, key);
            xn1 = xn + la*Functions.F(xn, key);

            seriesYx.add(x0,x0);
            seriesGGx.add(x0,xn);
            seriesGGx.add(xn,xn1);
            x0=xn;
            xn = x0 + la*Functions.F(x0, key);
            xn1 = xn + la*Functions.F(xn, key);
            seriesGGx.add(x0,xn);
            seriesGGx.add(xn,xn1);
            x0=xn;
            ///
            //seriesGx.add(x0,xn);
            //
            seriesYx.add(x0,x0);
            //seriesGGx.add(xn,xn);
            //seriesGGx.add(xn,xn1);
            if(Math.abs(xn1-xn)<ACCURACY){
                  roots.add(String.valueOf(Math.rint(xn*1000)/1000));
                  System.out.println("root:"+xn);
                  seriesRoot.add(xn,xn1);
                  la=-la;
                  x0= xn+50*ACCURACY;
                  //method(a, b, key);
            }
```

Змінюємо прапорець, котрий сигналізує про те, що корінь знайдено умовою if(Math.*abs*(xn1-xn)<ACCURACY).

Розглянемо, як виконується візуалізація графіка. Ми винесемо це до окремої функції. В ній спочатку буде перевірений прапор, який сигналізує про те, що обчислені координати графіка занесені в масив (змінна not_calculate). У тому випадку, якщо прапор указує, що підрахунку значень ще не було, викликається функція, яка рахуватиме значення координат графіка і заповнить ними масив. Наступним кроком реалізується прохід циклом for по масиву значень координат точок графіка та візуалізація, причому ми візуалізуємо не їх, а об'єднуємо ці точки в лінію, гуртуючись на значенні координат точок, як на вершинах.

Інтерфейс додатку. При зверненні до додатку відразу відкривається головна сторінка (рис.3).

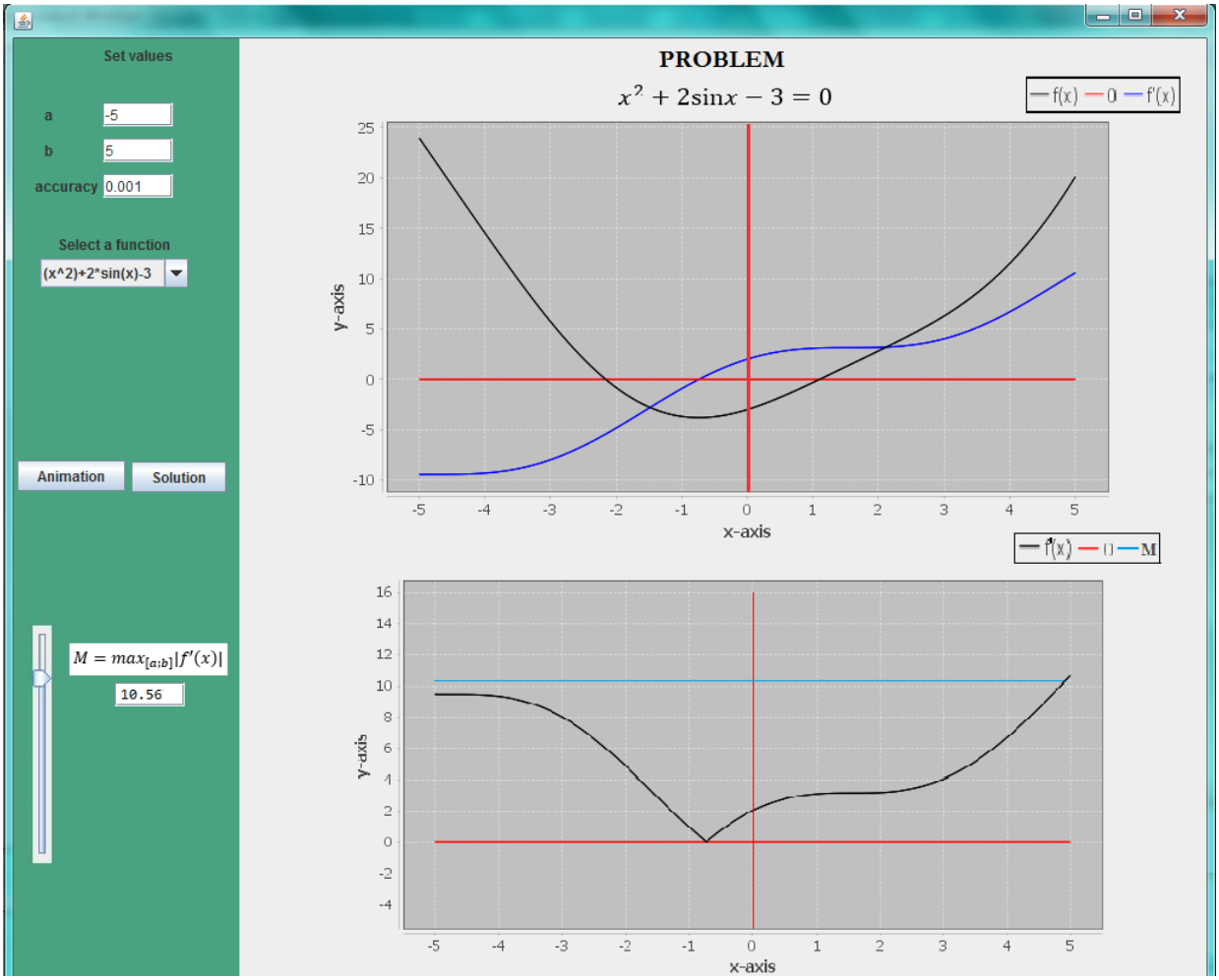

Рис. 3. Головна сторінка додатку.

Зліва зверху тут розташовані поля для завдання початкової a та кінцевої b точок відрізка, функції $f(x)$ на заданому відрізку $[a;b]$ та припустимої похибки ε. Після їх завдання праворуч з'являється графік функцій $f(x)$ та $f'(x)$. За його допомогою треба переконатись, що $f(x)$ не має кратних коренів, тобто що $f(x)$ та $f'(x)$ не мають спільних коренів. На графіку $|f'(x)|$ нижче за допомогою полоси прокрутки, що визначає синю горизонтальну лінію на графіку з ординатою M треба визначити значення M між $\max\limits_{[a;b]}|f'(x)|$ та $\max\limits_{[a;b]}|f'(x)| + \varepsilon$. Кнопки *Animation* та *Solution* відкривають наступні сторінки додатку. На сторінці **Animation** за допомогою традиційних інструментів анімації зверху ліворуч наочно демонструється алгоритм цієї статті (рис. 4).

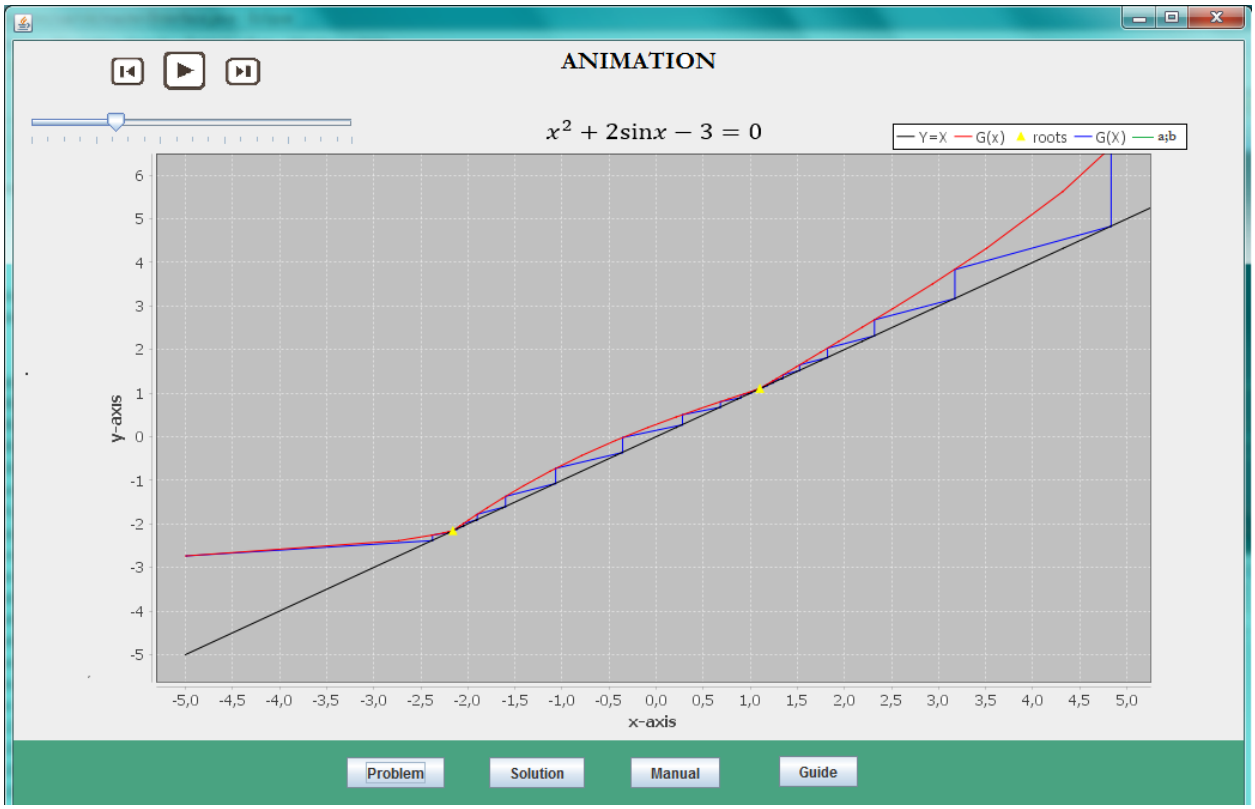

Рис. 4. Сторінка Animation.

На сторінці **Solution** наведена відповідь розглянутої задачі: всі корені на відрізку $[a;b]$ та їх зображення на графіку $f(x)$ (рисунок 5).

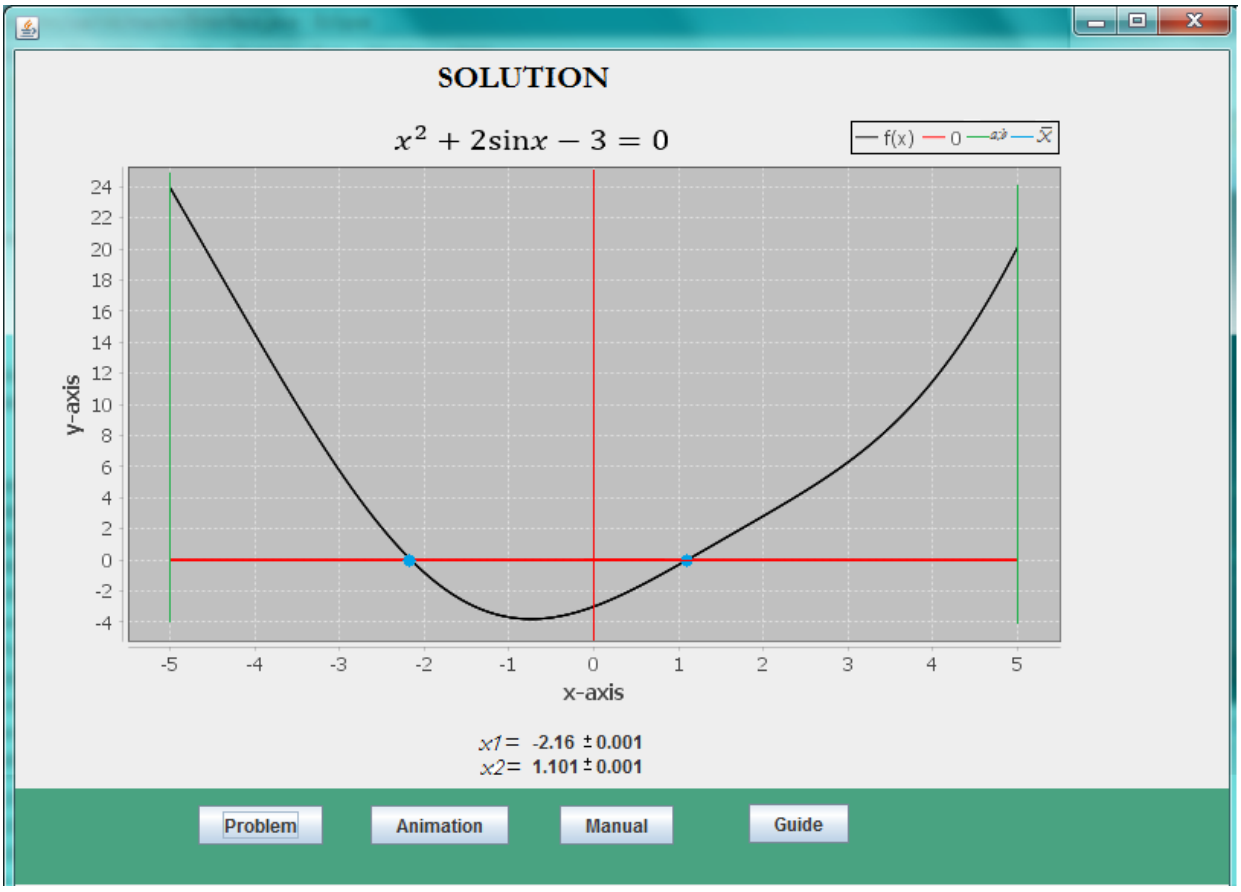

Рис. 5. Сторінка Solution.

Натиснувши *Manual*, відкриваємо навчальний посібник з методу відділення коренів, кнопкою Guide методичні вказівки для використання додатку. З кожної сторінки додатку відповідними кнопками можна перейти до інших сторінок для продовження роботи з програмою.

Висновки. Незаперечним є те, що розвиток інформаційних технологій є можливим лише за умови застосування чисельних методів математики (методів обчислень). Методи розв'язання рівнянь на відрізку – важлива складова чисельних методів і завдяки відносній простоті це мабуть ідейний центр при їх вивченні.

Всі відомі алгоритми знаходження коренів рівняння передбачають, що заздалегідь відомими є відрізки ізоляції коренів. Це не випадково, не є недоробком: наявність відрізку ізоляції є необхідною умовою застосування теорії стискуючого відображення – насправді підґрунтя всіх коректних чисельних методів. Ось чому метод цієї роботи потрапляє в ідейний центр освітнього курсу чисельних методів: він примушує замислитись над структурою цієї теорії, прояснити та поглибити її розуміння.

У роботі пропонується метод, що дозволяє знайти всі корені рівняння $f(x) = 0$ для довільної неперервно диференційованої функції $f(x)$ на даному відрізку прямої з заданою точністю, причому його збіжність є експоненціальною. Отже цей метод автоматично знаходить відрізки ізоляції коренів.

Уявлення, що при вивченні методів обчислень програми є лише застосуванням та ілюстрацією теоретичних надбань, створює дуже скривлену панораму. Насправді вони мотивують подальші поняття та методи, вони є базою вправ, пов'язаних з модифікацією алгоритмів. Метод цієї роботи реалізований у десктопі на мові Java і виконується віртуальною машиною Java, що забезпечує повну незалежність байт-коду від операційної системи та устаткування. Потужний графічний інтерфейс цього додатку прислуговує аналізу результатів обчислень, спрямованому на перетворення у "математику в малюнках".

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Березин, И. С. & Жидков, Н. П. (1966). *Методы вычислений (т.1)*. Москва: "Наука".
2. Tproger. (2018). *Все для изучения Java. Примеры разработки.* Retrieved from https://tproger.ru/tag/java.
3. Вейцбліт, О. Й. (2011). *Методи обчислень. Методичні рекомендації до виконання лабораторних робіт.* Херсон: Айлант.
4. Рид, М. & Саймон, Б. (1977). *Методы современной математической физики (т.1)*. Москва: Мир.
5. Вейцбліт, О. Й. (2011). *Методи обчислень. Навчальний посібник.* Херсон: Айлант.
6. Бахвалов, Н. С. (1973). *Численные методы (т.1)*. Москва: "Наука".
7. Арнольд, В. И. (1978). *Дополнительные главы теории обыкновенных дифференциальных уравнений*. Москва: "Наука.
8. Kalman, D. (2002). Doubly Recursive Multivariate Automatic Differentiation. *Magazine Mathematics*, 75 (3), 187-202.
9. JFreeChart. (2013). *Downloading JFreeChart.* Retrieved from http://www.jfree.org/jfreechart/download.html.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Berezyn, I. S. & Zhydkov, N. P. (1966). *Methods of calculation (t.1).* Moscow: "Nauka".
2. Tproger (2018). *Everything for learning Java. Examples of development*. Retrieved from https://tproger.ru/tag/java.
3. Veitsblit, O. Y. (2011). *Methods of calculation. Methodical recommendations for laboratory work.* Kherson: Ailant.
4. Ryd, M. & Saimon, B. (1977). *Methods of modern mathematical physics (t.1).* Moscow: Myr.

5. Veitsblit, O. Y. (2011). *Methods of calculation. Tutorial*. Kherson: Ailant.
6. Bakhvalov, N. S. (1973). *Numerical methods (t.1).* Moscow: “Nauka”.
7. Arnold, V. Y. (1978). *Additional chapters of the theory of ordinary differential equations*. Moscow: “Nauka”.
8. Kalman, D. (2002). Doubly Recursive Multivariate Automatic Differentiation. *Magazine Mathematics*, 75 (3), 187-202.
9. JFreeChart. (2013). *Downloading JFreeChart.* Retrieved from http://www.jfree.org/jfreechart/download.html.

Стаття надійшла до редакції 22.01.2018.
The article was received 22 January 2018.

Alexander Veitsblit, Nikita Shepel, Inna Vygodner
Kherson State University, Kherson, Ukraine

ALGORITHM OF SEPARATION OF ROOTS ON A SEGMENT

All known widespread algorithms of a numerical solution of an equation on a straight line segment assume that segments of isolation of roots are already known. These are such segments, on each of which a solution is one and only one. The method of this work allows to discover all solutions of an equation f(x) = 0 for arbitrary continuously differentiable function f(x) on the set segment of a straight line with the set accuracy. Convergence of the method is exponential. Thus, the method automatically separates roots. In the course of numerical methods it hits in its ideological center, forcing to consider all structure of the theory to make it clear and deepen its understanding. The method of this work is realized in a desktop in language Java.

Key words: numerical methods; contraction operator; exponential convergence; segments of isolation; desktop; the diagram of classes; the window interface; graphic object.

Вейцблит А. Й., Шепель М. С., Выгоднер И. В.
Херсонский государственный университет, Херсон, Украина

АЛГОРИТМ ОТДЕЛЕНИЯ КОРНЕЙ НА ОТРЕЗКЕ

Все известные, распространённые алгоритмы численного решения уравнения на отрезке прямой предполагают, что уже известны отрезки изоляции корней. Это такие отрезки, на каждом из которых решение одно и только одно. Метод этой работы позволяет найти все корни уравнения f(x) = 0 для произвольной непрерывно дифференцируемой функции f(x) на заданном отрезке прямой с заданной точностью. Сходимость метода экспоненциальная. Таким образом, метод автоматически отделяет корни. В курсе численных методов он попадает в его идейный центр, заставляя обдумать всю структуру теории, прояснить и углубить её понимание. Метод этой работы реализован в десктопе на языке Java.

Ключевые слова: численные методы; сжимающий оператор; отрезки изоляции; диаграмма классов; экспоненциальная сходимость; оконный интерфейс; графический объект.