Зміст

[Вступ 6](#_Toc450508214)

[1 Постановка задачі 7](#_Toc450508215)

[2 Теоретичні відомості 8](#_Toc450508216)

[2.1. Метод Якобі 8](#_Toc450508217)

[2.2. Метод Гауса-Зейделя 9](#_Toc450508218)

[2.3. Метод градієнтного спуску (приєднаних градієнтів) 10](#_Toc450508219)

[3 Опис алгоритмів 13](#_Toc450508220)

[3.1. Загальний алгоритм 13](#_Toc450508221)

[3.2. Алгоритм методу Якобі 13](#_Toc450508222)

[3.3. Алгоритм методу Гауса-Зейделя 15](#_Toc450508223)

[3.4. Алгоритм методу градієнтного спуску 16](#_Toc450508224)

[4 Опис програмного забезпечення 19](#_Toc450508225)

[4.1. Функціональна структура програмного забезпечення 19](#_Toc450508226)

[4.2. Опис функцій частин програмного забезпечення 19](#_Toc450508227)

[4.2.1. Користувацькі функції 20](#_Toc450508228)

[4.2.2. Стандартні функції 27](#_Toc450508229)

[5 Тестування програмного забезпечення 29](#_Toc450508230)

[5.1. План тестування 29](#_Toc450508231)

[5.2. Приклади тестування 29](#_Toc450508232)

[5.2.1. Виконання програми, коли у матрицю системи невірно введено число 29](#_Toc450508233)

[5.2.2. Виконання програми, коли у стовпець вільних членів невірно введено число 30](#_Toc450508234)

[5.2.3. Виконання програми, коли у матрицю системи введено число з точністю більшою, ніж 3 знаки після коми 31](#_Toc450508235)

[5.2.4. Перевірка вірності роботи програми, коли користувачем вірно введені всі дані та для обраного методу рішення сходиться 32](#_Toc450508236)

[5.2.4.1. Перевірка коректності виконання програми при використані методу Якобі 32](#_Toc450508237)

[5.2.4.2. Перевірка коректності виконання програми при використані методу Гауса-Зейделя 33](#_Toc450508238)

[5.2.4.3. Перевірка коректності виконання програми при використані методу градієнтного спуску 34](#_Toc450508239)

[6 Інструкція користувача 35](#_Toc450508240)

[6.1. Робота з програмою 35](#_Toc450508241)

[6.2. Формат вхідних та вихідних даних 40](#_Toc450508242)

[6.3. Системні вимоги 40](#_Toc450508243)

[7 Аналіз і узагальнення результатів 41](#_Toc450508244)

[Висновки 42](#_Toc450508245)

[Перелік посилань 43](#_Toc450508246)

Вступ

Чисельне рішення СЛАР — одна з найбільш часто зустрічаються завдань у науково-технічних дослідженнях. Така задача виникає в математичній фізиці (чисельне рішення диференціальних та інтегральних рівнянь), економіці, статистиці. При цьому прикладні задачі часто потребують вирішення великих і надвеликих СЛАР з числом невідомих більше 1000. До таких СЛАР, наприклад, призводить чисельне рішення двовимірних і особливо тривимірних задач математичної фізики.

Існуючі бібліотеки програм розроблені на основі, так званих, прямих методів розв'язку СЛАР (наприклад метод Гауса та його модифікації). Число арифметичних операцій таких методів для знаходження чисельного розв'язку СЛАР розмірністю   
складає — . Кубічна залежність числа арифметичних операцій від розміру матриці СЛАР призводить до нереально великого часу рішення при навіть на найсучасніших ЕОМ.

Ітераційні методи розв'язку СЛАР набагато економніші, як по машинному часу рішення, так і по використанню оперативної пам'яті. Так, якщо ітераційний метод швидко сходяться з числом ітерацій , то час виконання, пропорційний вже квадрату розміру матриці — . При використанні ітераційних методів, що швидко сходяться, стає можливим рішення СЛАР розмірністю .

В даний час відсутні бібліотеки підпрограм широкого призначення для чисельного рішення великих і надвеликих СЛАР. Таким чином, розробка ефективних ітераційних алгоритмів для рішення СЛАР великої розмірності є актуальним завданням.

Тому метою даної роботи є вивчення та аналіз роботи найбільш відомих та швидкісних методів рішення СЛАР, таких як метод простих ітерацій (або так званий метод Якобі), метод Гауса-Зейделя, метод градієнтного спуску (або приєднаних градієнтів).

# Постановка задачі

Розробити програмне забезпечення, що буде знаходити рішення для заданої СЛАР наступними методами:

* Метод Якобі
* Метод Гауса-Зейделя
* Метод градієнтного спуску

Вхідними даними для даної роботи є матриця коефіцієнтів та стовпець вільних членів для СЛАР розмірність яких знаходиться в межах від 2 до 5.

Вихідними даними для даної роботи являється сукупність дійсних чисел, що є розв’язками даної системи.

# Теоретичні відомості

Квадратичну систему з *n* лінійних рівнянь можна задати наступним чином:

|  |  |
| --- | --- |
|  | (2.1) |

де:

, ,

Тоді якщо , то система (2.1) має розв’язок та він єдиний. Якщо система має єдиний розв’язок, то його можна знайти одним із наступних методів.

## Метод Якобі

Сутність методу Якобі полягає в тому, що матриця розбивається на дві матриці: діагональну матрицю та матрицю . Тобто:

,

Тоді систему (2.1) можна переписати у наступному вигляді:

|  |  |
| --- | --- |
|  | (2.2) |

Помноживши систему (2.2) зліва на отримаємо:

|  |  |
| --- | --- |
|  | (2.3) |

Якщо відоме певне наближення , тоді від виразу (2.3) можна перейти до наступної ітераційної форми:

або

Метод Якобі для системи (2.1) є збіжним для довільного початкового наближення тоді і тільки тоді, коли матриця має домінантну головну діагональ:

Умова завершення ітераційного процесу Якобі при досягнені точності у спрощеній формі має вигляд:

## Метод Гауса-Зейделя

Сутність методу Гауса-Зейделя полягає в тому, що матриця розбивається на три матриці: діагональну матрицю , верхню трикутною та нижню трикутною. Тобто:

, ,

Тоді систему (2.1) можна переписати у наступному вигляді:

|  |  |
| --- | --- |
|  | (2.4) |

Якщо відоме певне наближення , тоді від виразу (2.4) можна перейти до наступної ітераційної форми, яка задана системою:

,

де , , ,

або

Метод Гауса-Зейделя для системи (2.1) є збіжним для довільного початкового наближення тоді і тільки тоді, коли матриця має домінантну головну діагональ. Також даний метод сходиться, якщо матриця симетрична і додатньо визначена.

Умова завершення ітераційного процесу Гауса-Зейделя при досягнені точності у спрощеній формі має вигляд:

## Метод градієнтного спуску (приєднаних градієнтів)

Якщо матриця симетрична та додатньо визначена, то розв’язок системи (2.1) можна знайти методом приєднаних градієнтів.

Процес пошуку розв’язку полягає в мінімізації наступного функціоналу:

За позначено скалярний добуток.

Якщо являє собою n взаємно приєднаних векторів, то Z утворює базис для простору \ Mathbb {R} ^ п і можна виразити рішення системи в даному базисі:

Нескладними математичними перетвореннями з виразу (2.5) отримаємо:

Вираз (2.6) дає нам наступний метод рішення системи (2.1): найти послідовність приєднаних векторів-напрямків, а потім обчислити їх коефіцієнт .

Якщо відоме певне наближення . Тоді для (k+1) кроку алгоритму маємо:

де – розв’язок системи (3.1) на k-тому кроці, – вектор напрямку на   
k-тому кроці, – нев’язка на k-тому кроці.

Для довільного початкового наближення вищеописаний алгоритм буде працювати за умови, що матриця симетрична, додатньо визначена та:

# Опис алгоритмів

## Загальний алгоритм

1. ПОЧАТОК
2. КІНЕЦЬ

|  |  |
| --- | --- |
| Змінна | Призначення |
|  | Точність з якою необхідно знайти корені розв’язків  () |
| Matrix | Матриця системи |
| b | Стовпець вільних членів |
| norm | Норма поточного розв’язку системи |
| flag | Ознака сходимості методу (false – метод не сходиться,  true – метод сходиться) |

## Алгоритм методу Якобі

1. ПОЧАТОК
2. Обчислити визначник матриці (*Matrix*) (**det**):
   1. ЯКЩО матриця має одиничну розмірність, ТО позначити як визначник даної матриці єдиний елемент цієї матриці. ІНАКШЕ обчислити визначник розкладом за першим стовпцем:
      1. Ініціалізація визначника поточної матриці (*Matrix*) *detRes* = 0.
      2. ЦИКЛ проходу по всіх елементах першого стовпця матриці (*Matrix*) :
         1. Обчислити алгебраїчне доповнення поточного елементу :
            1. Утворити доповнювальний мінор для поточного елементу (**marixCut**):

Ініціалізація нової матриці (*newMatrix*) розмірності (n-1).

ЦИКЛ проходу по всіх елементах матриці (*Matrix*):

ЯКЩО поточний елемент не знаходиться на першому стовпці АБО   
(*i-тому*) рядку, ТО добавити його в нову матрицю (*newMatrix*).

* + - 1. Знайти визначник доповнювального мінору (тобто знайти визначник матриці *newMatrix*) (**det**) (аналогічно пункту 2).
      2. Добавити до *detRes* добуток поточного елементу та визначника його доповнювального мінору.

1. Присвоїти *flag* = true.
2. ЯКЩО визначник не дорівнює нулю, ТО перевірити умови сходимості методу (**isSolution**):
   1. Приведення матриці до канонічного (діагонального) вигляду (**stableMatrix**):
      1. Ініціалізація масиву *element* (*element*[*i*] = *false*, *i* =1, 2, …, n).
      2. ЦИКЛ проходу по всіх рядках матриці (*Matrix*):
         1. Знайти номер індексу найбільшого за абсолютним значенням елементу поточного рядка (*index*).
         2. ЯКЩО такий індекс (*index*), вже був знайдений (*element[index]* = true), ТО для даної матриці (*Matrix*) метод не сходиться (присвоїти *flag* = false), ІНАКШЕ присвоїти *element[index]* = true.
      3. Ініціалізація нової матриці (*newMatrix*) такої ж розмірності, що й поточна матриця (*Matrix*).
      4. ЦИКЛ проходу по всіх рядках матриці (*Matrix*):
         1. Знайти номер індексу найбільшого за абсолютним значенням елементу поточного рядка (*index*) матриці (*Matrix*).
         2. Присвоїти поточний рядок матриці (*Matrix*) *index*-рядку матриці (*newMatrix*) (тобто рядку, який має номер *index*).
      5. Присвоїти матриці (*Matrix*) матрицю (*newMatrix*).
   2. ЦИКЛ проходу по всіх рядках матриці (*Matrix*):
      1. Знайти суму абсолютних значень всіх елементів поточного рядка (*sum*).
      2. Знайти найбільший за абсолютним значенням елемент поточного рядка (*max*).
      3. ЯКЩО *max(sum – max)*, ТО для даної матриці (*Matrix*) метод не сходиться (присвоїти *flag* = false).

ІНАКШЕ перейти до пункту 6.

1. ЯКЩО метод сходиться (), ТО розв’язати СЛАР (**Jacobi**):
   1. Задати попереднє наближення розв’язку (.
   2. Присвоїти *norm* = .
   3. ПОКИ :
      1. Обчислити наступне наближення (:
         1. Присвоїти .
         2. Помножити на .
      2. Обчислити поточне значення норми розв’язку (*norm*) (**normCalc**):
         1. Присвоїти *norm =* .
         2. ЦИКЛ проходу по всіх елементах поточного наближення :

ЯКЩО , ТО *norm =.*

* + 1. Присвоїти .

1. КІНЕЦЬ

## Алгоритм методу Гауса-Зейделя

1. ПОЧАТОК
2. Обчислити визначник матриці (*Matrix*) (**det**) (аналогічно пункту 2 методу Якобі).
3. Присвоїти *flag* = true.
4. ЯКЩО визначник не дорівнює нулю, ТО перевірити умови сходимості методу (**isSolution**) (аналогічно пункту 4 методу Якобі), ІНАКШЕ перейти до пункту 6.
5. ЯКЩО метод сходиться (), ТО розв’язати СЛАР (**Seidel**):
   1. Задати попереднє наближення розв’язку (.
   2. Присвоїти *norm* = .
   3. ПОКИ :
      1. Обчислити наступне наближення (:
         1. ЦИКЛ проходу по всіх елементах поточного розв’язку (:
            1. Знайти суму:
            2. Знайти суму:
            3. Обчислити .
         2. Помножити на .
      2. Обчислити поточне значення норми розв’язку (*norm*) (**normCalc**) (аналогічно пункту 5.3.2 методу Якобі).
      3. Присвоїти .
6. КІНЕЦЬ

## Алгоритм методу градієнтного спуску

1. ПОЧАТОК
2. Обчислити визначник матриці (*Matrix*) (**det**) (аналогічно пункту 2 методу Якобі).
3. Присвоїти *flag* = true.
4. ЯКЩО визначник не дорівнює нулеві, ТО перевірити умови сходимості методу (**isSolution**):
   1. Перевірити матрицю на симетричність (**isSemetric**):
      1. ЦИКЛ проходу по всіх строках матриці (*Matrix*) (:
         1. ЯКЩО, ТО для даної матриці (*Matrix*) метод не сходиться (присвоїти *flag* = false).
   2. Перевірити матрицю на додатньовизначеність за критерієм Сильвестра (**isPositive**):
      1. ЦИКЛ перебору всіх кутових мінорів (:
         1. Ініціалізація нової матриці (*newMatrix*) розмірності .
         2. ЦИКЛ проходу по всіх елементах матриці (*Matrix*)   
            ():

ЯКЩО для поточного елементу , ТО додати поточний елемент до нової матриці (*newMatrix*).

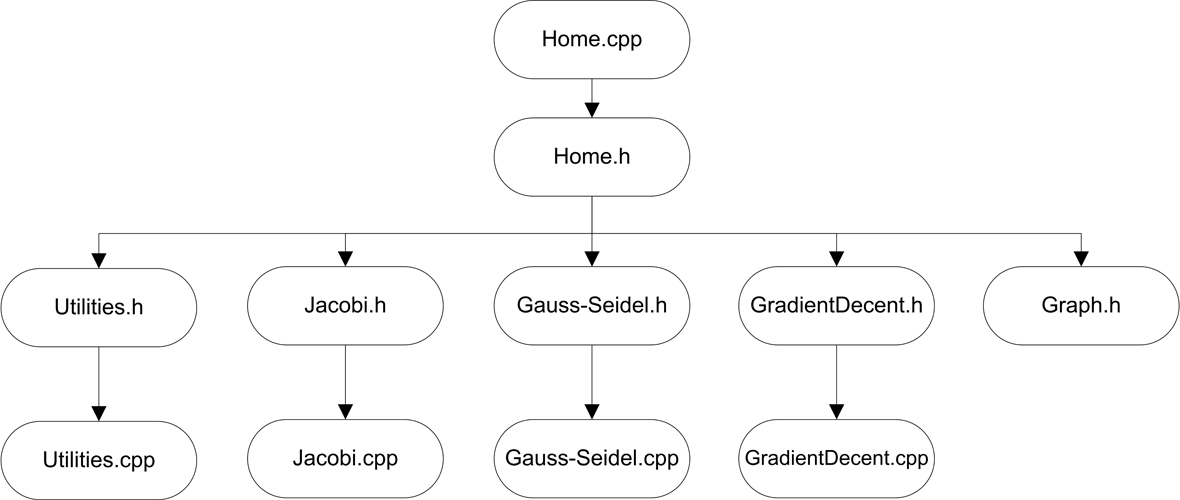
* + - 1. Обчислити визначник нової матриці (*newMatrix*) (**det**) (аналогічно пункту 2 методу Якобі).
      2. ЯКЩО визначник нової матриці (*newMatrix*) від’ємний АБО дорівнює нулеві, ТО для даної матриці (*Matrix*) метод не сходиться (присвоїти *flag* = false).

ІНАКШЕ перейти до пункту 6.

1. ЯКЩО метод сходиться (), ТО розв’язати СЛАР (**GradientDecent**):
   1. Задати попереднє наближення розв’язку (.
   2. Задати попередню незв’язку як .
   3. Задати попередній вектор напрямку як .
   4. Присвоїти *norm* = .
   5. ПОКИ :
      1. Обчислити довжину для вектора напрямку
      2. Обчислити наступне наближення .
      3. Обчислити наступне значення незв’язки
      4. Обчислити коефіцієнт
      5. Обчислити наступний вектор напрямку   
         .
      6. Обчислити поточне значення норми розв’язку (*norm*) (**normCalc**) (аналогічно пункту 5.3.2 методу Якобі).
      7. Присвоїти .
      8. Присвоїти
      9. Присвоїти
2. КІНЕЦЬ

# Опис програмного забезпечення

## Функціональна структура програмного забезпечення



## Опис функцій частин програмного забезпечення

В ході виконання поставленого завдання було створено наступні модулі та бібліотеки:

* Home.h — реалізує інтерфейс користувача
* Graph.h — реалізує побудову та виведення графіка
* Jacobi.h — реалізує знаходження розв’язку СЛАР методом Якобі
* Gauss-Seidel.h — реалізує знаходження розв’язку СЛАР методом Гауса-Зейделя
* GradientDecent.h — реалізує знаходження розв’язку СЛАР методом градієнтного спуску.
* Utilities.h — реалізує допоміжні функції (знаходження визначника матриці системи, перевірки матриці системи на симетричність і т.д.), які необхідні для розв’язання поставленої задачі.

### Користувацькі функції

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | det | Обчислення визначника квадратної матриці | Матриця для якої необхідно знайти визначник та її розмірність | Визначник матриці | Utilities.h |
| 2 | matrixCut | Утворення мінору заданого елементу | Матриця, в якій знаходиться елемент, її розмірність та індекс рядка елемента | Матриця, яка є мінором для заданого елементу | Utilities.h |
| 3 | AlgebralAdditionCalc | Обчислення алгебраїчного доповнення заданого елементу | Матриця, в якій знаходиться елемент, її розмірність та індекс рядка елемента | Алгебраїчне доповнення до матриці | Utilities.h |
| 4 | normCalc | Обчислення норми поточного наближення | Поточне наближення, попереднє наближення та кількість елементів в наближеннях | Норма поточного рішення | Utilities.h |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | isSemetric | Перевірка матриці системи на симетричність | Система лінійних алгебраїчних рівнянь, для якої необхідно перевірити матрицю на симетричність | TRUE — матриця системи симетрична,  FASLE — матриця системи не симетрична | Utilities.h |
| 6 | isPositive | Перевірка матриці системи на додатньо визначеність | Система лінійних алгебраїчних рівнянь, для якої необхідно перевірити матрицю додатньо визначеність | TRUE — матриця системи додтньо визначена,  FASLE — матриця системи не додтньо визначена | Utilities.h |
| 7 | cornerMinor | Утворення кутового мінору | Матриця системи для якої необхідно утворити кутовий мінор та його розмірність | Кутовий мінор заданої розмірності | Utilities.h |
| 8 | matrixMult | Множення матриці на вектор | Матриця системи та вектор, на який необхідно помножити дану систему | Вектор – результат виконання множення | Utilities.h |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 13 | isDiagDominate | Перевірка матриці на діагональний перевіс | Система лінійних алгебраїчних рівнянь, матрицю якої необхідно перевірити на діагональний перевіс | TRUE – матриця системи має діагональний перевіс  FALSE - матриця системи не має діагонального перевісу | Utilities.h |
| 14 | CreateMatrix | Створення матриці заданої розмірності | Розмірність матриці | Матриця заданої розмірності | Utilities.h |
| 15 | freeMatrix | Видалення матриці | Адреса матриці та її розмірність | - | Utilities.h |
| 16 | CreateMas | Створення масиву заданого розміру | Розмір масиву | Масив заданого розміру | Utilities.h |
| 17 | freeMas | Видалення масиву | Адреса масиву | - | Utilities.h |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 18 | masCopy | Копіювання масиву | Масив, який необхідно скопіювати. Масив, в який необхідно скопіювати. Розмір масиву | - | Utilities.h |
| 19 | Jacobi | Знаходження розв’язки СЛАР методом Якобі | Система лінійних алгебраїчних рівнянь, для якої необхідно знайти рішення | Рішення системи | Jacobi.h |
| 20 | isSolved | Перевірка на сходимість методу Якобі | Система лінійних алгебраїчних рівнянь, яку не обхідно перевірити на сходимість методу Якобі | TRUE – метод сходиться для даної системи  FALSE – метод не сходиться для даної системи | Jacobi.h |
| 21 | nextSolution | Знаходження наступного наближення для методу Якобі | СЛАР для якої знаходиться розв’язок, попереднє наближення, масив в який потрібно записати наступне наближення | Наступне наближення | Jacobi.h |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 22 | Seidel | Знаходження розв’язки СЛАР методом Гауса-Зейделя | Система лінійних алгебраїчних рівнянь, для якої необхідно знайти рішення | Рішення системи | Gauss–Seidel.h |
| 23 | isSolved | Перевірка на сходимість методу Гауса-Зейделя | Система лінійних алгебраїчних рівнянь, яку не обхідно перевірити на сходимість методу Гауса-Зейделя | TRUE – метод сходиться для даної системи  FALSE – метод не сходиться для даної системи | Gauss–Seidel.h |
| 24 | nextSolution | Знаходження наступного наближення для методу Гауса-Зейделя | СЛАР для якої знаходиться розв’язок, попереднє наближення, масив в який потрібно записати наступне наближення | Наступне наближення | Gauss–Seidel.h |
| 25 | GradientDescent | Знаходження розв’язки СЛАР методом градієнтного спуску | Система лінійних алгебраїчних рівнянь, для якої необхідно знайти рішення | Рішення системи | GradientDescent.h |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 26 | isSolved | Перевірка на сходимість методу градієнтного спуску | Система лінійних алгебраїчних рівнянь, яку не обхідно перевірити на сходимість методу градієнтного спуску | TRUE – метод сходиться для даної системи  FALSE – метод не сходиться для даної системи | GradientDescent.h |
| 27 | BSolve\_Click | Обробка події натискання кнопки «Знайти рішення» | Посилання на відправника, додаткові аргументи виклику | - | Home.h |
| 28 | dataFree\_CellEndEdit | Обробка події зміни вмісту стовпця вільних членів | Посилання на відправника, додаткові аргументи виклику | - | Home.h |
| 29 | dataMatrix\_CellEndEdit | Обробка події зміни вмісту матриці системи | Посилання на відправника, додаткові аргументи виклику | - | Home.h |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 30 | BClear\_Click | Обробка події натискання кнопки «Очистити вміст» | Посилання на відправника, додаткові аргументи виклику | - | Home.h |
| 31 | solsSize\_ValueChanged | Обробка події змінення розміру системи | Посилання на відправника, додаткові аргументи виклику | - | Home.h |
| 32 | Home\_Load | Обробка події загрузки вікна програми | Посилання на відправника, додаткові аргументи виклику | - | Home.h |
| 33 | print | Виведення рішення системи на форму | Рішення СЛАР | - | Home.h |
| 34 | DataRead | Перевірка вхідних даних на валідність | - | TRUE – вхідні дані коректні  FALSE – вхідні дані не коректні | Home.h |
| 35 | DataRead | Читання вхідних даних | СЛАР, в яку необхідно прочитати дані | - | Home.h |
| 36 | Resize | Зміна розмірів матриці системи, стовпця вільних членів та вектора відповіді | - | - | Home.h |
| 37 | Clear | Очищає вміст матриці системи, стовпця вільних членів та вектора відповіді | - | - | Home.h |

### Стандартні функції

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | round | Округлення числа до найближчого цілого | Число, що необхідно округлити | Округлене число | cmath |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | abs | Знаходження абсолютного значення числа | Число для якого необхідно знайти абсолютне його значення | Абсолютне значення числа | cmath |
| 3 | min | Знаходження мінімуму двох чисел | Два числа з яких необхідно знайти мінімум | Мінімум з двох чисел | Algorithm |
| 4 | ToDouble | Приводить до типу double певну змінну | Змінна простого типу | Число з плаваючою крапкою | - |

# Тестування програмного забезпечення

## План тестування

Усі можливі випадки виникнення помилок у програмі залежать від вхідних даних, тобто тої інформації, що отримується від користувача. Тому тестування програми полягає у виявленні правильності та коректності обробки програмою різних вхідних даних.

Задля виявлення усіх похибок роботи програми потрібно запустити виконання алгоритмів в таких умовах:

* У матрицю системи або стовпець вільних членів невірно введено число (або введена інформація не являється числом)
* У матрицю системи або стовпець вільних членів введено число з точністю більшою, ніж 3 знаки після коми

Для вирішення відповідних помилкових ситуацій слід вдосконалити алгоритми роботи програми та обробити всі виключні ситуації.

Далі слід упевнитись, що усі методи програми коректно працюють на усіх наборах вхідних значень.

## Приклади тестування

### Виконання програми, коли у матрицю системи невірно введено число

Якщо під час введення інформації в матрицю системи користувач введе недозволений символ, то програма виведе у спеціально відведене місце на графічному інтерфейсі (далі: системне поле) відповідне повідомлення, зафарбує в червоний колір комірку таблиці в якій було зроблено помилку та заблокує можливість натискання кнопки «Знайти рішення» (рисунок 5.1) до тих пір, поки користувач не виправить вхідні дані.

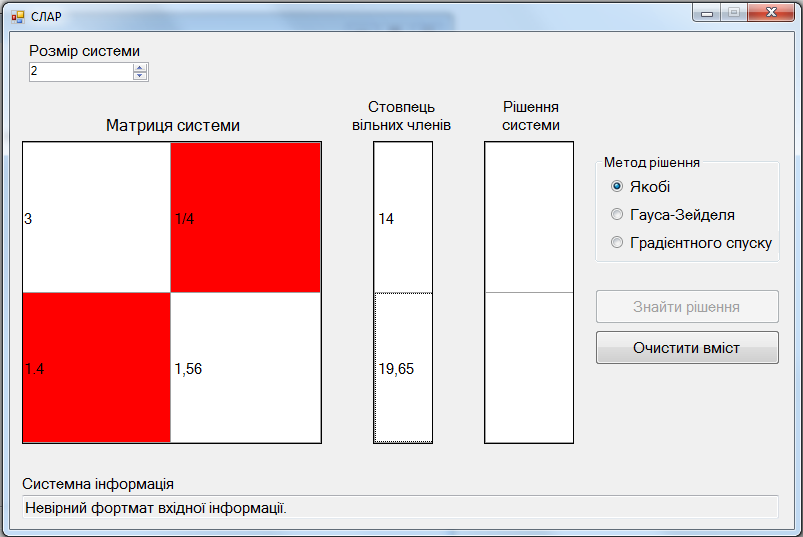


Рисунок 5.1 — Вікно програми, після введення некоректної інформації в матрицю системи

### Виконання програми, коли у стовпець вільних членів невірно введено число

Якщо під час введення інформації в стовпець вільних членів користувач введе недозволений символ, то програма виведе у системне поле відповідне повідомлення, зафарбує в червоний колір комірку таблиці в якій було зроблено помилку та заблокує можливість натискання кнопки «Знайти рішення»   
(рисунок 5.2) до тих пір, поки користувач не виправить вхідні дані.

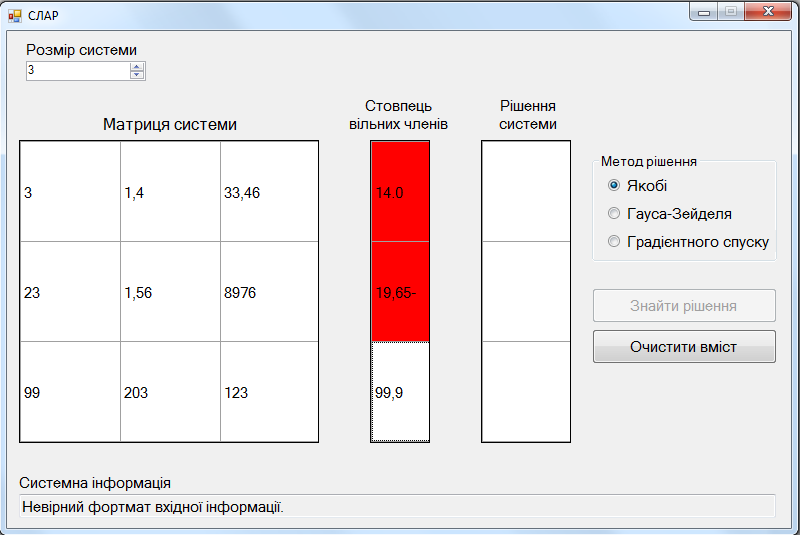


Рисунок 5.2 — Вікно програми, після введення некоректної інформації в стовпець вільних членів

### Виконання програми, коли у матрицю системи введено число з точністю більшою, ніж 3 знаки після коми

Якщо під час введення інформації в матрицю системи користувач введе число з точністю більшою, ніж 3 знаки після коми (рисунок 5.3), то програма автоматично округлить його до 3-х знаків за математичними правилами (рисунок 5.4).

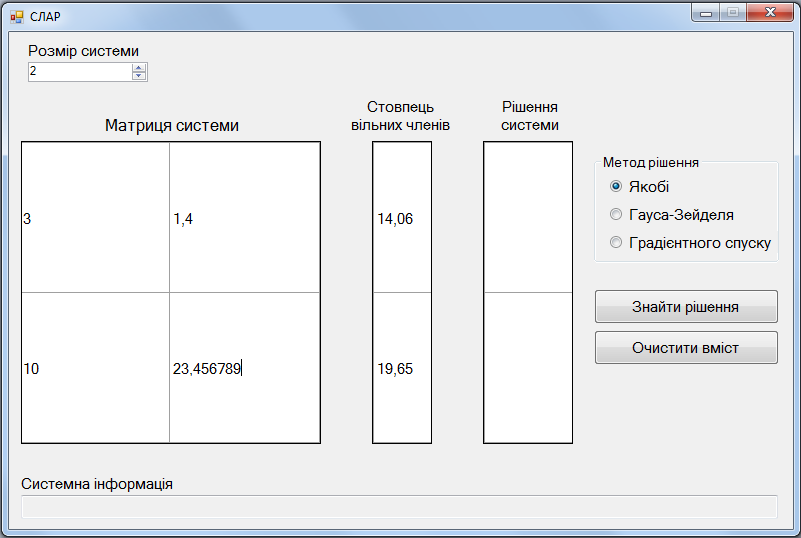


Рисунок 5.3 — Вікно програми під час введення числа з точністю більшою,   
ніж 3 знаки після коми

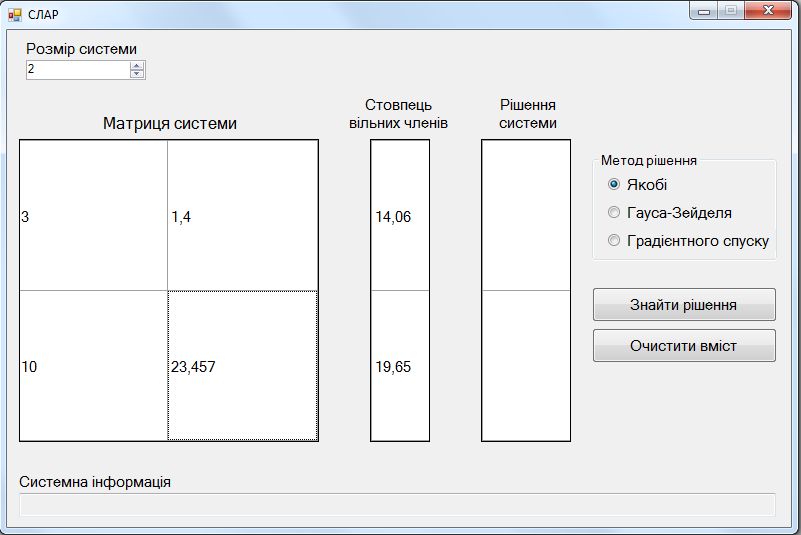


Рисунок 5.4 — Вікно програми після завершення введення числа з точністю більшою, ніж 3 знаки після коми

### Перевірка вірності роботи програми, коли користувачем вірно введені всі дані та для обраного методу рішення сходиться

#### Перевірка коректності виконання програми при використані методу Якобі

Метод даного тесту є перевірка коректності роботи програми при використані методу Якобі. Якщо обрати довільну систему для якої метод Якобі сходиться та натиснути кнопку «Знайти рішення», то програма виведе наступне рішення:

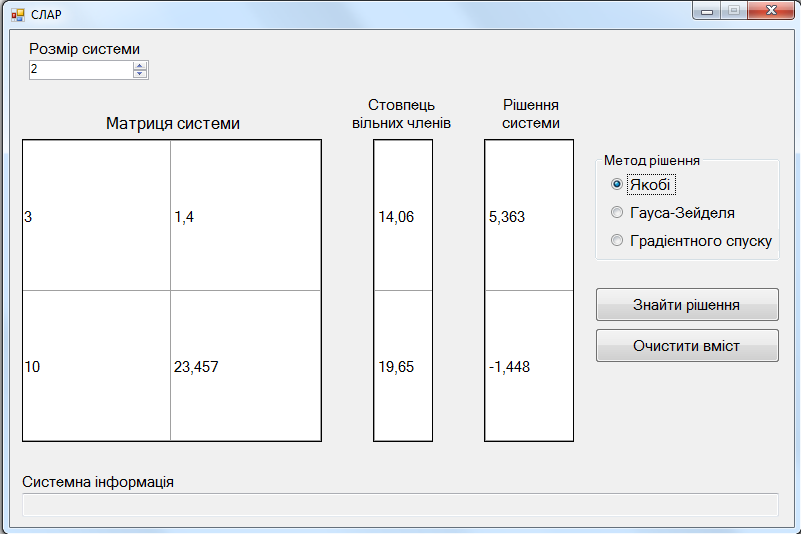


Рисунок 5.5 — Вікно програми після коректного виконання методу Якобі

Для перевірки коректності виконання скористаюся стороннім програмним забезпеченням — MS Excel 2010:

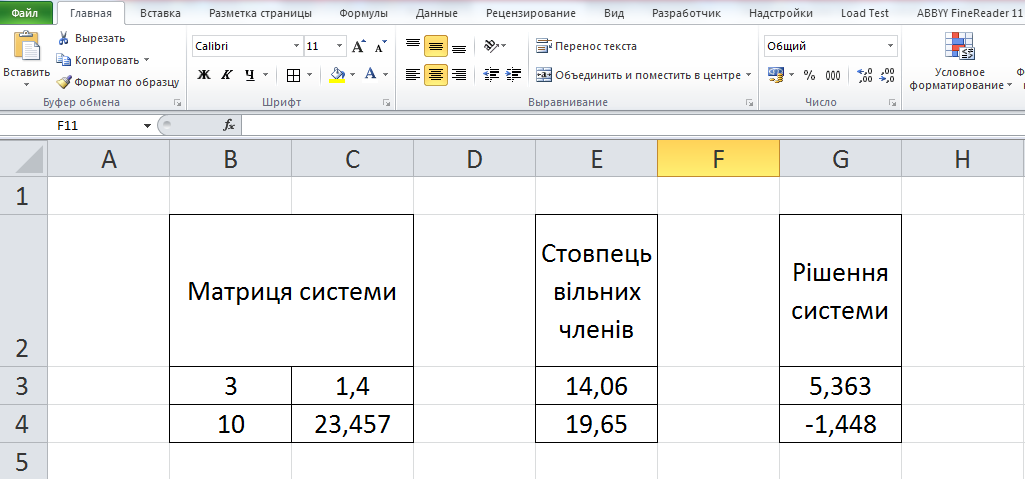


Рисунок 5.6 — Результат обчислення відповіді в MS Excel 2010

З рисунку 5.5 та рисунку 5.6 стає очевидним, що програма працює коректно при використані методу Якобі.

#### Перевірка коректності виконання програми при використані методу Гауса-Зейделя

Метод даного тесту є перевірка коректності роботи програми при використані методу Гауса-Зейделя. Якщо обрати довільну систему для якої метод Гауса-Зейделя сходиться та натиснути кнопку «Знайти рішення», то програма виведе наступне рішення:

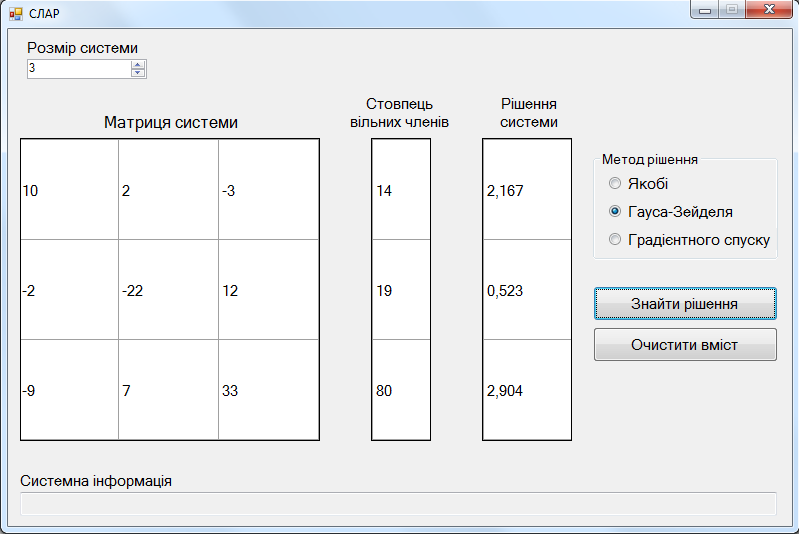


Рисунок 5.7 — Вікно програми після коректного виконання методу   
Гауса-Зейделя

Для перевірки коректності виконання скористаюся стороннім програмним забезпеченням — MS Excel 2010:

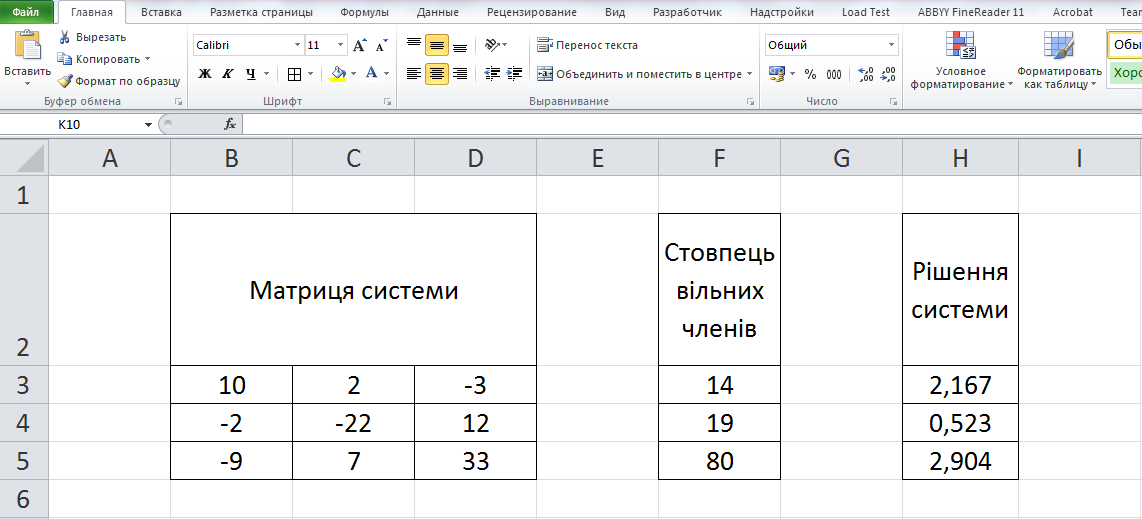


Рисунок 5.8 — Результат обчислення відповіді в MS Excel 2010

З рисунку 5.7 та рисунку 5.8 стає очевидним, що програма працює коректно при використані методу Гауса-Зейделя.

#### Перевірка коректності виконання програми при використані методу градієнтного спуску

Метод даного тесту є перевірка коректності роботи програми при використані методу градієнтного спуску. Якщо обрати довільну систему для якої метод градієнтного спуску сходиться та натиснути кнопку «Знайти рішення», то програма виведе наступне рішення:

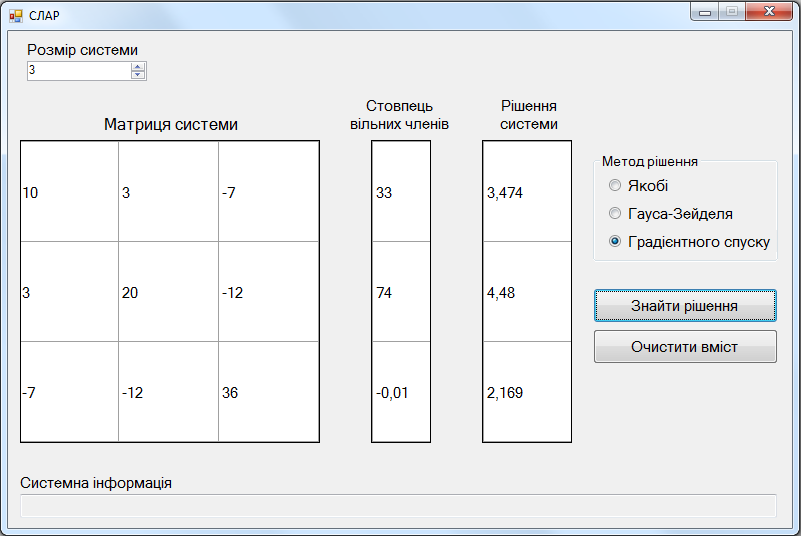


Рисунок 5.9 — Вікно програми після коректного виконання методу   
градієнтного спуску

Для перевірки коректності виконання скористаюся стороннім програмним забезпеченням — MS Excel 2010:

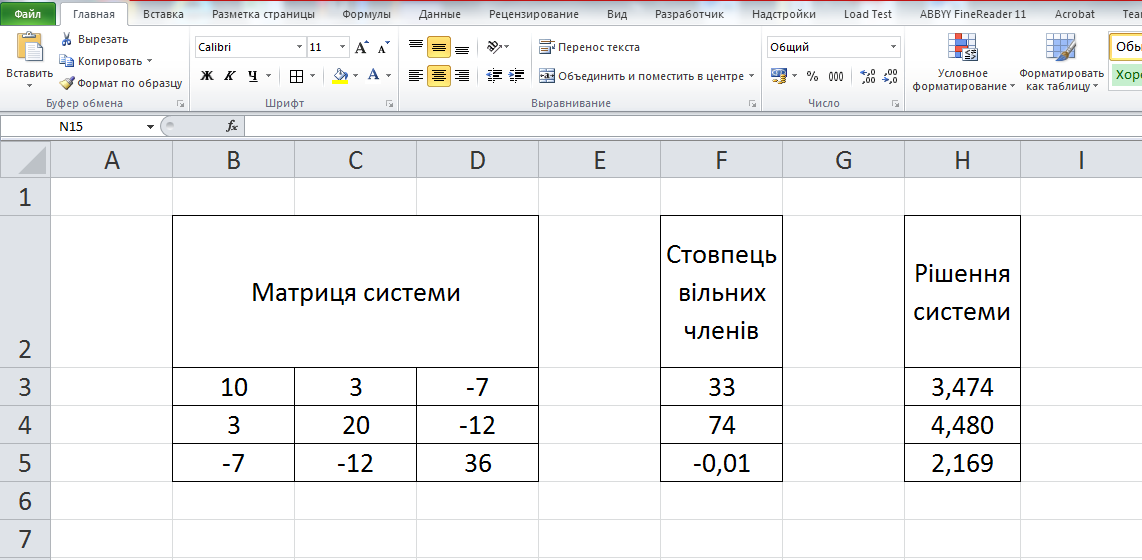


Рисунок 5.10 — Результат обчислення відповіді в MS Excel 2010

З рисунку 5.9 та рисунку 5.10 стає очевидним, що програма працює коректно при використані методу градієнтного спуску.

# Інструкція користувача

## Робота з програмою

Після запуску виконавчого файлу з розширенням \*.exe, відкривається головне вікно програми (Рисунок 6.1).

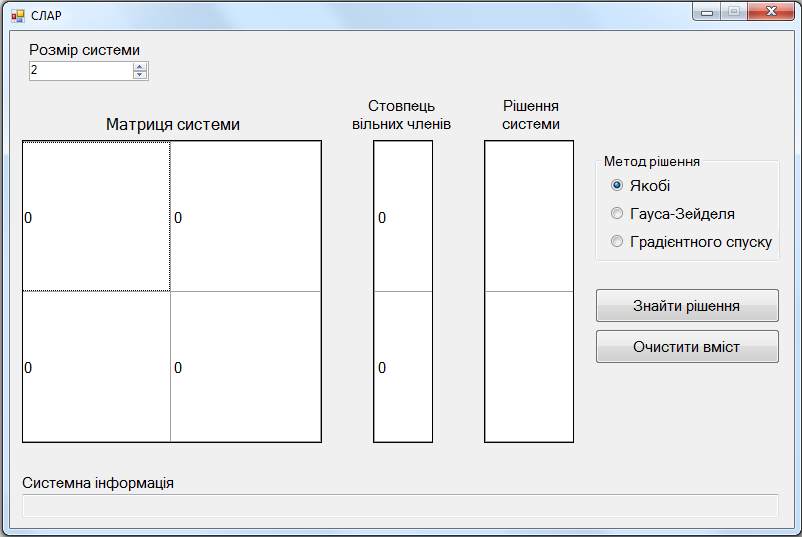


Рисунок 6.1 — Головне вікно програми

Далі за допомогою лічильника з назвою «Розмір системи» шляхом натиску на стрілки або введенням числа з клавіатури необхідно виставити розмір системи, що буде оброблятися програмою (Рисунок 6.2):

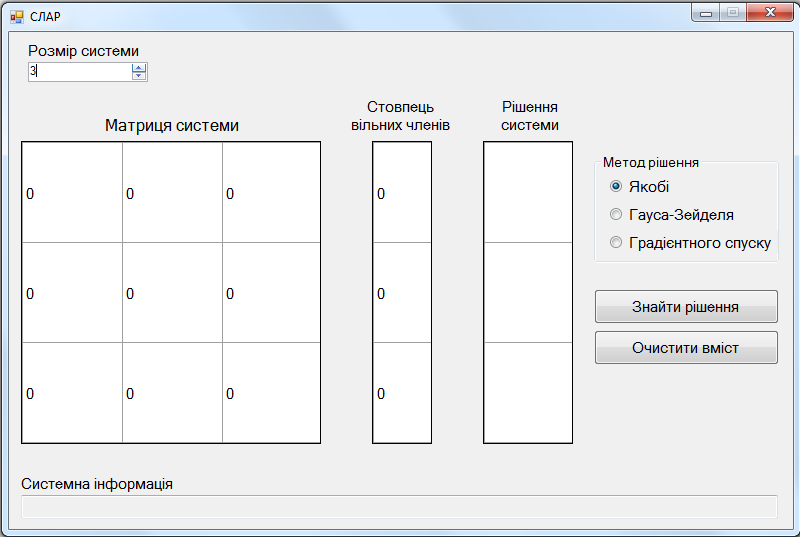


Рисунок 6.2 — Вибір необхідного розміру системи

Далі необхідно записати матрицю системи та стовпець вільних членів в масиви з відповідними назвами (Рисунок 6.3):

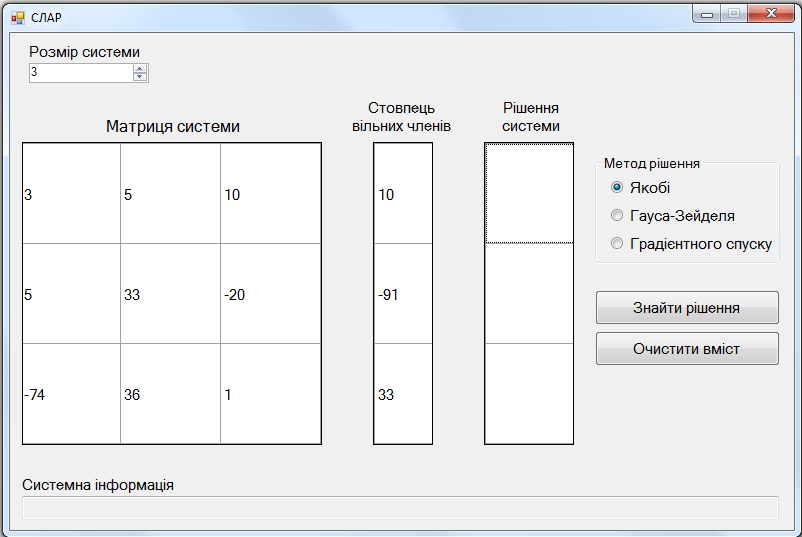


Рисунок 6.3 — Введення вхідних даних

Якщо якісь вхідні дані не є числами, то програма видасть відповідне повідомлення у полі «Системна інформація» та зафарбує комірки з невалідними даними в червоний колір (Рисунок 6.4). Для того щоб можна було продовжити виконання роботи з програмою користувач повинен виправити вхідні дані.

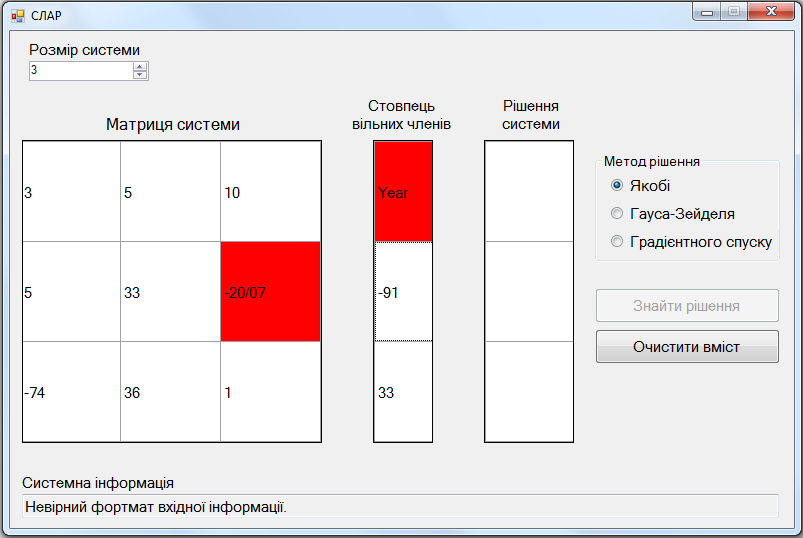


Рисунок 6.4 — Приклад невірного введення даних користувачем

Після того як дані введені коректно в полі «Метод рішення» необхідно обрати метод, яким користувач бажає знайти розв’язки введеної системи та натиснути кнопку «Знайти рішення». Якщо для даної системи існує розв’язок та обраний метод сходить, то для неї буде виведено рішення системи (Рисунок 6.5):

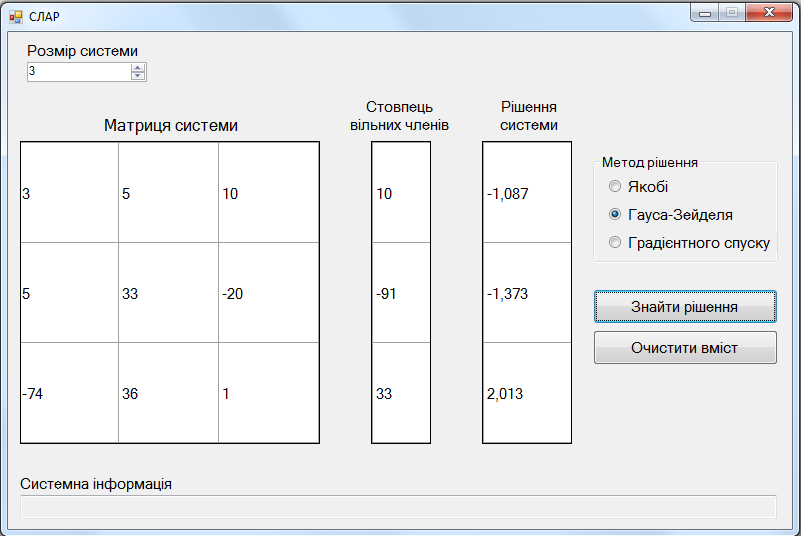


Рисунок 6.5 — Приклад результату роботи програми

Якщо для даної системи не існує розв’язку або їх безліч, то програмою в поле системної інформації буде виведене відповідне повідомлення (Рисунок 6.6):

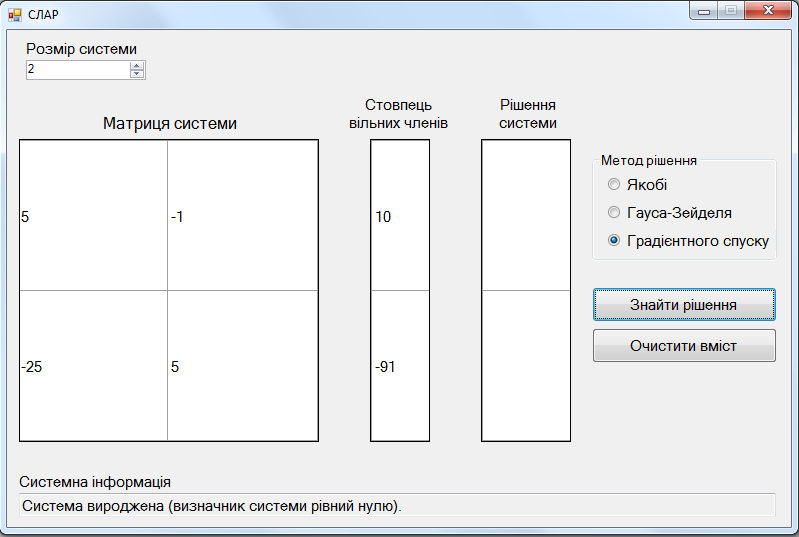


Рисунок 6.6 — Приклад введення системи для якої не існує розв’язку

Якщо для даної системи обраний метод не сходиться, то програмою в поле системної інформації буде виведене відповідне повідомлення (Рисунок 6.7):

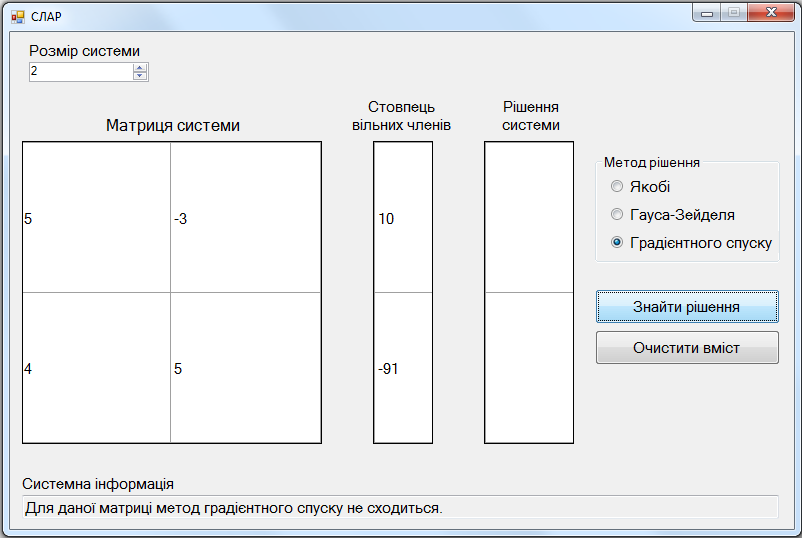


Рисунок 6.7 — Приклад введення системи для якої обраний метод   
не сходиться

Якщо під час виконання програми було обрано розмірність матриці «2» та записано систему для якої обраний метод сходиться, то програмою задля візуалізації рішення буде виведено графік (Рисунок 6.8):

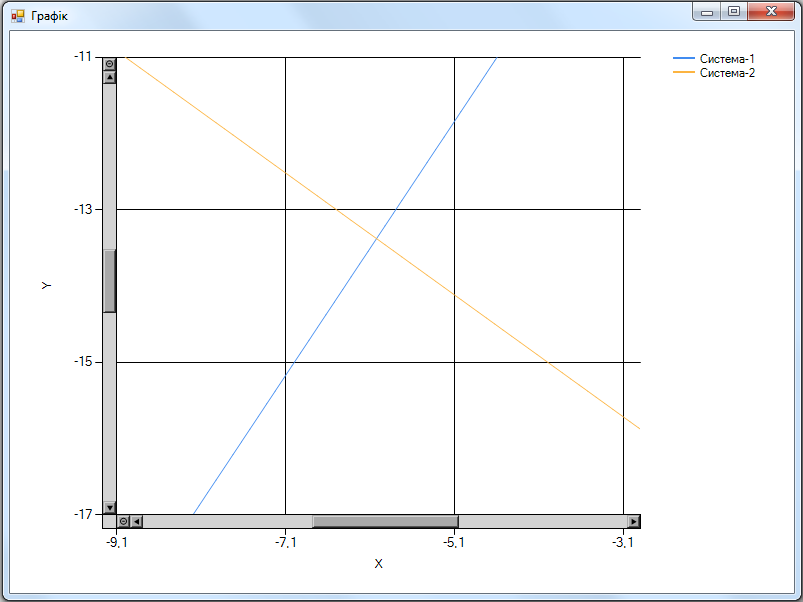


Рисунок 6.8 — Приклад введення графіку системи для якої обраний метод сходиться

Даний графік можна масштабувати виділивши необхідну область в прямокутник шляхом клацання лівою кнопкою та протягування миші   
(Рисунок 6.9 та Рисунок 6.10):

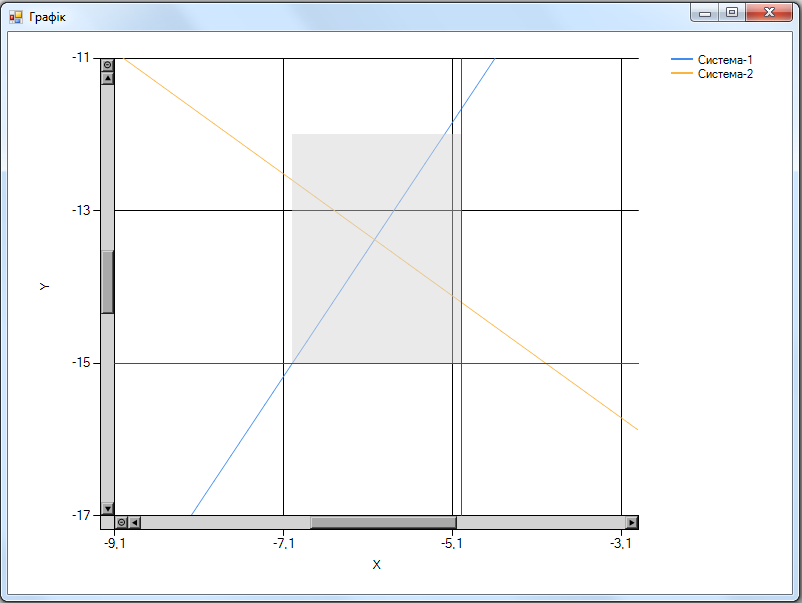


Рисунок 6.9 — Приклад виділення області графіку для масштабування

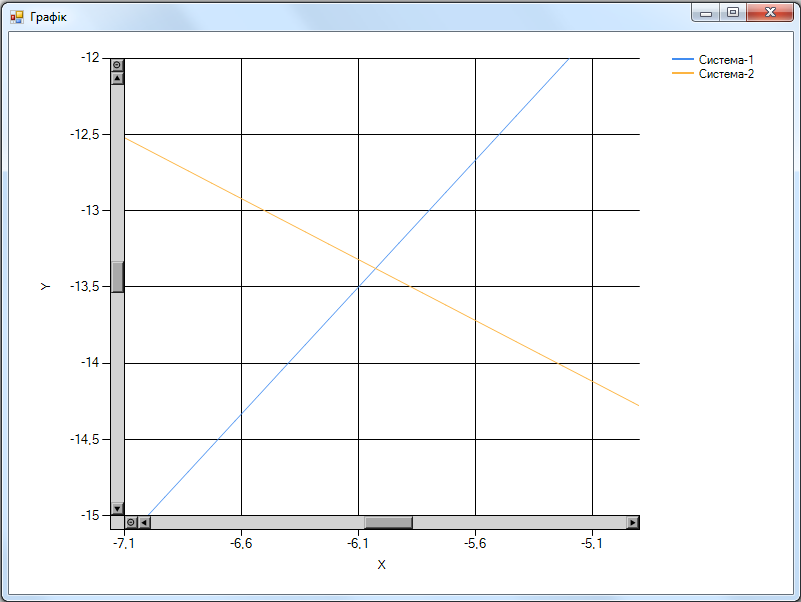


Рисунок 6.10 — Рисунок (6.8) після масштабування

Також можливо рухати графік системи за допомогою полос прокручування, що знаходяться знизу та зліва графіка. Якщо користувачеві необхідно зменшити масштаб по якійсь осі то, необхідно натиснути на кнопку на відповідні полосі прокручування.

Якщо користувачеві необхідно очисти всі введені вхідні дані, то це можна зробити за допомогою кнопки «Очистити вміст». Після її натискання матриця системи та стовпець вільних члені заповняться нулями, а вся інформація в таблиці «Рішення системи» буде стерта.

## Формат вхідних та вихідних даних

Користувачем на вхід програми подається СЛАР у матричному вигляді, тобто задається за допомогою матриці системи та стовпця вільних членів, числа яких дійсні з точністю не більше, ніж 3 числа після коми (якщо точність більша, то програма автоматично округлить їх за математичними правилами до 3-х знаків після коми).

Результатом виконання програми є розв’язок зданої СЛАР, який видається у вигляді таблиці кожне число якої записане з точністю до 3-х знаків після коми.

## Системні вимоги

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows® XP/Windows Vista/Windows 7/ Windows 8/Windows 10 (з останніми обновленнями) | Windows 7/ Windows 8/Windows 10  (з останніми обновленнями) |
| Процесор | Intel® Pentium® ІІІ  1.0 GHz або  AMD Athlon™ 1.0 GHz | Intel® Pentium® D или AMD Athlon™ 64 X2 |
| Оперативна пам'ять | 256 MB RAM (для Windows® XP) / 1 GB RAM (для Windows Vista/Windows 7/  Windows 8/Windows 10) | 1 GB RAM (для Windows® XP) / 2 GB RAM (для Windows Vista/Windows 7/  Windows 8/Windows 10) |
| Відеоадаптер | Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог) | |
| Дисплей | 800х600 | 1024х768 або краще |
| Прилади введення | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | Microsoft .Net Framework 4.5.2 або вище | |

# Аналіз і узагальнення результатів

Головною задачею курсової роботи була реалізація програми для розв’язання СЛАР наступними методами: Якобі, Гауса-Зейделя, градієнтного спуску.

Критичні ситуації у роботі програми виявлені не були. Дана програма працює коректно для будь-яких вхідних даних, про що свідчать численні тести на яких провірялася програма. Під час тестування було виявлено, що більшість помилок виникало тоді, коли користувачем вводилися не числові вхідні дані. Тому всі вхідні дані, які вводить користувач, ретельно провіряються на валідність і лише потім подаються на обробку програмі.

Під час дослідження представлених ітераційних алгоритмів знаходження розв’язки СЛАР, були виявлені наступні особливості алгоритмів:

* Для всіх розглянутих алгоритмів існує певна умова збіжності (діагональний перевіс елементів, симетричність, додатньо визначеність), що значено звужує коло задач, що може вирішувати той чи інший метод.
* Не дивлячись на те, що ітераційні методи Якобі та Гауса-Зейделя досить схожі, але вони сильно різняться швидкодією. Під час тестування програми було виявлено, що метод Гауса-Зейделя виконує в 1,5 - 2 рази менше ітерацій, ніж метод Якобі на аналогічних вхідних даних. Це можна пояснити тим, що в методі Гауса-Зейделя на черговій ітерації використовуються не лише значення невідомих знайдених на минулій ітерації, а й значення, що були отримані на поточній ітерації.
* Принциповою перевагою методу Якобі над всіма іншими методами є те, що операція обчислення наступного наближення для кожної невідомої є незалежною і її можна виконувати окремо для кожної невідомої. Тому можна значно збільшити швидкість виконання програми шляхом розпаралелення процедури знаходження наступного наближення.

Висновки

На етапі ознайомлення з теоретичними аспектами завдання даної курсової роботи було досліджено алгоритми ітераційних методів розв’язання СЛАР, що використовуються для знаходження розв’язків великих та надвеликих СЛАР, проаналізовано їх особливості та доцільність використання в межах реальних проектів, що було доведено реалізацією програмного забезпечення, яке розглядається в данні курсовій роботі.

На етапі проектування програмного забезпечення реалізовано наступні ітераційні методи: Якобі, Гауса-Зейделя, градієнтного спуску. Результатом розробки програми є набуття досвіду роботи з великими та надвеликими СЛАР, закріплення, узагальнення та поглиблення знань і навичок програмування на мовах С\С++, набуття практичних навичок з алгоритмізації прикладних задач з лінійної алгебри та аналітичної геометрії.

Для спрощення процесу взаємодії з користувачем та розширення кола можливих користувачі програмного забезпечення був розроблений інтерфейс користувача. Також був розроблена інструкція користувача, що дозволяє ознайомитися зі всіма можливостями та ключовим моментами використання програмного забезпечення.

На етапі тестування було визначено, що програма вірно обробляє всі вхідні дані та видає очікуваний результат.

Результатом курсової роботи є створена програма, яку можна використовувати у навчальних та/або інженерних цілях з метою розв’язання задач, які зводяться до знаходження рішення СЛАР або містять процес знаходження рішення СЛАР як незалежну підзадачу.

Перелік посилань

**Додаток А Технічне завдання**

МІНІСТЕРСТВО НАУКИ І ОСВІТИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра

автоматизованих систем обробки інформації та управління

Затвердив

Керівник *Муха І. П.*

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016р.

Виконавець:

Студент *Зарічковий О. А.*

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Розв'язання СЛАР

наближеними методами»

з дисципліни:

«Основи програмування»

Київ 2016

1.1 Мета: Метою курсової роботи є розробка комплексу програм для розв’язання систем лінійних алгебраїчних рівнянь.

1.2 Найменування та галузь застосування об'єкта розробки:Дана робота присвячена розробці програмного забезпечення для розв’язання систем лінійних алгебраїчних рівнянь методами простої ітерації, Гаусса-Зейделя та методом найшвидшого спуску.

1.3 Підстава для проведення робіт:Підставою для розробки програмного забезпечення є навчальний план спеціальності 6.050301 «Програмна інженерія», робоча програма дисципліни „Основи програмування”, індивідуальне завдання.

1.4 Дата початку роботи: «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2016 р.

1.5 Дата закінчення роботи: «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2016 р.

1.6 Призначення розробки: Функціональним призначенням програми є автоматизація розв’язання СЛАР, для підвищення точності та швидкості обчислень. Програма може експлуатуватися користувачем, який стикається з науковими та інженерними розрахунками для обчислення коренів розв’язання СЛАР, які могли б з високою точністю збігатися з реальними значеннями.

1.7 Вимоги до програми та програмної документації: Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:

ИСО 5807 - 85 ГОСТ на розробку програмних документів, схем алгоритмів програм, даних та систем.

ГОСТ 19.781 - 74 - Вимоги до розробки програмного забезпечення.

ГОСТ 19.101-77 (СТ СЭВ 1626 - 79) - Держстандарт на розробку програмної документації, видів програм та програмних документів.

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 95 - Розробка технічної документації.

1.8 Стадії та етапи розробки:

1. Аналіз методів вирішення поставленої задачі (до 17.04.2016 р.)

2. Розробка алгоритмічного забезпечення (до 24.04.2016 р.)

3. Розробка сценарію роботи програми (до 01.05.2016 р.)

4. Розробка програмного забезпечення (до 22.05.2016 р.)

5. Розробка інтерфейсу, планування, тестування розробленої програми (до 05.06.2016 р.)

6. Розробка пояснювальної записки (до 12.06.2016 р.).

7. Захист курсової роботи (до \_\_.\_\_.2016 р.).

1.9 Порядок контролю та приймання. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв її оцінювання.

Додаток Б Тексти програмного коду

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення*

Розв’язання СЛАР наближеними методами

(Вид носія даних)

*CD-RW*

(Обсяг програми (документа), арк., Кб)

*арк, Кб*

*студента групи ІП-51 І курсу*

Зарічкового Олександра Анатолійовича

Файл «Utilities.h»

#pragma once

namespace utilities {

const double eps = 1e-5; // Точність обчислення розвяку

// Структура, що представляє систему

struct system {

int size; // Розмір системи

double\*\* matrix; // Матриця системи

double\* free; // Стовпець вільних членів

};

double det(double\*\*, const int&); // Прототип функції обчислення визначника

double\*\* matrixCut(double\*\*, const int&, const int&); // Прототип функції уворення матриці(N - 1) - го порядку

double AlgebralAdditionCalc(double\*\*, const int&, const int&); // Прототип функції обчислення алгебраїного доповнення заданого елементу

double normCalc(double\*, double\*, const int&); // Прототип функції обчислення норми поточного рішення

bool isSemetric(const system&); // Прототип функції превірки матриці системи на симетричність

bool isPositive(const system&); // Прототип функції превірки матриці системи на додатньовизначеність

double\*\* cornerMinor(const system&, int); // Прототип функції утворення кутового мінору

double\* matrixMult(const system&, double\*); // Прототип функції множення матриці системи на вектор

double scalarMult(double\*, double\*, int&); // Прототип функції скалярного добутку векторів

void stableSystem(system&); // Прототип функції приведння матриці системи до канонічного вигляду

int indexOfMaxElement(double\*, const int&); // Прототип функції знаходження індекса максимального елементу в рядку

double lineSumElement(double\*, const int&); // Прототип функції знаходження суми елементів в рядку

bool isDiagDominate(const system&); // Прототип функції перевірки матриці системи на діагональний перевіс

double\*\* CreateMatrix(int); // Прототип функції виділення памяті під матрицю

void freeMatrix(double\*\* &, int); // Прототип функції видалення матриці з памяті

double\* CreateMas(int); // Прототип функції виділення памяті під масив

void freeMas(double\* &); // Прототип функції видалення масиву з памяті

void masCopy(double\*, double\*, int&); // Прототип функції копіювання масиву

};

Файл «Utilities.сpp»

#include "Utilities.h"

#include <cmath>

using namespace utilities;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція обчислення визначника \*

\* матриці \*

\* Параметри: \*

\* matrix - матриця для якої потірбно \*

\* порахувати визначник \*

\* size - розмірність матриці \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double utilities::det(double\*\* matrix, const int& size) {

//ЯКЩО розмірність матриці одинична, ТО повернути визначник матриці 1х1

if (size == 1) return \*\*matrix;

else {

// Обчислення визначника матриці розкладом за першим стовпцем

double detRes = 0; // Визначник поточної матриці

for (int i = 0; i < size; i++) detRes += AlgebralAdditionCalc(matrix, size, i) \* matrix[i][0];

return detRes;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція уворення матриці \*

\* (N-1)-го порядку \*

\* Параметри: \*

\* matrix - матриця для якої потірбно \*

\* знайт мінор \*

\* size - розмірність матриці \*

\* y - номер рядка елемента, для \*

\* якого потрібно порахувати \*

\* мінор \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double\*\* utilities::matrixCut(double\*\* matrix, const int& size, const int& y) {

int elementCount = 0; // Кількість елементів в утвореній матриці

// Cтворюємо нову матрицю

double\*\* newMatrix = CreateMatrix(size - 1);

// Заповнюємо матрицю

for (int i = 0; i < size; i++)

for (int z = 1; z < size; z++)

if (i != y) {

newMatrix[elementCount / (size - 1)][elementCount % (size - 1)] = matrix[i][z];

elementCount++;

}

return newMatrix;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція обчислення алгебраїчного \*

\* доповнення заданого елементу \*

\* Параметри: \*

\* matrix - матриця для якої потірбно \*

\* знайти алгебраїчне доповнення \*

\* size - розмірність матриці \*

\* y - номер рядка елемента, для \*

\* якого потрібно порахувати \*

\* алгебраїний мінор \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double utilities::AlgebralAdditionCalc(double\*\* matrix, const int& size, const int& y) {

double\*\* newMatrix = matrixCut(matrix, size, y);

double result = det(newMatrix, size-1) \* (y & 1 ? -1 : 1); // Алгебраїчне доповнення

freeMatrix(newMatrix, size - 1);

return result;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція обчислення норми \*

\* поточного наближення \*

\* Параметри: \*

\* х - поперднє наближення \*

\* хk - поточне наближення \*

\* size - розмірність масивів \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double utilities::normCalc(double\* x, double\* xk, const int& size) {

double norm = abs(\*x - \*xk);

for (int i = 1; i < size; i++)

if (abs(x[i] - xk[i]) > norm) norm = abs(x[i] - xk[i]);

return norm;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція перевірки матриці \*

\* системи на симетричність \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool utilities::isSemetric(const system& sols) {

for (int i = 0; i < sols.size; i++)

for (int z = i; z < sols.size; z++)

if (sols.matrix[i][z] != sols.matrix[z][i]) return false;

return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція перевірки матриці \*

\* системи на додатньовизначеність \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool utilities::isPositive(const system& sols) {

// Користуємося критерієм Сильвестра

for (int i = 1; i <= sols.size; i++) {

double\*\* newMatrix = cornerMinor(sols, i);

// Якщо визначник кутового мінору не є додатньою, то матриця не є додатньовизначеною

if (det(newMatrix, i) <= 0) return false;

freeMatrix(newMatrix, i);

}

return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція утворення кутового \*

\* мінору матриці \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\* size - розмірність кутового мінору \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double\*\* utilities::cornerMinor(const system& sols, int size) {

double\*\* newMatrix = CreateMatrix(size);

for (int i = 0; i < size; i++)

for (int z = 0; z < size; z++)

newMatrix[i][z] = sols.matrix[i][z];

return newMatrix;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція множення матриці на \*

\* вектор \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\* mas - вектор, на який множиться СЛАР \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double\* utilities::matrixMult(const system& sols, double\* mas) {

double\* newVector = CreateMas(sols.size);

for (int i = 0; i < sols.size; i++) {

newVector[i] = 0;

for (int z = 0; z < sols.size; z++) newVector[i] += mas[z] \* sols.matrix[i][z];

}

return newVector;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція знаходження скалярного \*

\* добутку векторів \*

\* Параметри: \*

\* vector1 - перший векртор \*

\* vector2 - другий вектор \*

\* size - розмірність векторі \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double utilities::scalarMult(double\* vector1, double\* vector2, int& size) {

double scalar = 0;

for (int i = 0; i < size; i++) scalar += vector1[i] \* vector2[i];

return scalar;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція стабілізації матриці \*

\* відносно головної діагоналі \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void utilities::stableSystem(system& sols) {

// Масив який вказує на якому рядку було замічено діагональний перевіс

int\* mas = new int[sols.size];

for (int i = 0; i < sols.size; i++) mas[i] = -1;

for (int i = 0; i < sols.size; i++) {

int index = indexOfMaxElement(sols.matrix[i], sols.size);

double sum = lineSumElement(sols.matrix[i], sols.size);

if (sum >= abs(2 \* sols.matrix[i][index])) {

delete[] mas;

return;

}

if(mas[index] != -1){

delete[] mas;

return;

}

mas[index] = i;

}

system newSols;

newSols.size = sols.size;

newSols.matrix = CreateMatrix(sols.size);

newSols.free = CreateMas(sols.size);

for (int i = 0; i < sols.size; i++) {

newSols.free[i] = sols.free[mas[i]];

for (int z = 0; z < sols.size; z++) newSols.matrix[i][z] = sols.matrix[mas[i]][z];

}

freeMatrix(sols.matrix, sols.size);

freeMas(sols.free);

sols = newSols;

delete[] mas;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція знаходження \*

\* максимального за абсолютним \*

\* значення елемента в поточному \*

\* рядку \*

\* Параметри: \*

\* mas - поточний рядок \*

\* size - кількість елементів в рядку \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int utilities::indexOfMaxElement(double\* mas, const int& size) {

int index = 0; // Індекс максимального елементу

for (int i = 1; i < size; i++) if (abs(mas[i]) > abs(mas[index])) index = i;

return index;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція знаходження \*

\* суми абсолютних значеннь \*

\* елементів в поточному \*

\* рядку \*

\* Параметри: \*

\* mas - поточний рядок \*

\* size - кількість елементів в рядку \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double utilities::lineSumElement(double\* mas, const int& size) {

double sum = 0; // Сума елементів

for (int i = 0; i < size; i++) sum += abs(mas[i]);

return sum;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція перевірки матриці системи \*

\* на діагональний перевіс \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool utilities::isDiagDominate(const system& sols) {

for (int i = 0; i < sols.size; i++) {

double sum = lineSumElement(sols.matrix[i], sols.size);

if (sum >= abs(2 \* sols.matrix[i][i])) return false;

}

return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція виділення динамічної \*

\* памяті під матрицю \*

\* Параметри: \*

\* size - розмірність матриці \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double\*\* utilities::CreateMatrix(int size) {

double\*\* matrix = new double\*[size];

for (int i = 0; i < size; i++)

matrix[i] = new double[size];

return matrix;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція видалення матриці \*

\* Параметри: \*

\* matrix - матриця, яку необхідно \*

\* видалити \*

\* size - розмірність матриці \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void utilities::freeMatrix(double\*\* & matrix, int size) {

for (int i = 0; i < size; i++) delete[] matrix[i];

delete[] matrix;

matrix = nullptr;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція виділення динамічної \*

\* памяті під масив \*

\* Параметри: \*

\* size - розмірність масиву \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double\* utilities::CreateMas(int size) {

double\* mas = new double [size];

return mas;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція видалення масиву \*

\* Параметри: \*

\* mas - масив, який необхідно \*

\* видалити \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void utilities::freeMas(double\* & mas) {

delete[] mas;

mas = nullptr;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція копіювання масиву \*

\* Параметри: \*

\* scr - масив, який потрібно копіювати \*

\* dist - масив, в який потірбно \*

\* копіювати \*

\* size - розмірність масиву \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void utilities::masCopy(double\* src, double\* dist, int& size) {

for (int i = 0; i < size; i++) dist[i] = src[i];

}

Файл «Jacobi.h»

#pragma once

#include "Utilities.h"

using utilities::system;

namespace Jacobi {

double\* Jacobi(utilities::system&); // Прототип функції рішення системи методом Якобі

bool isSolved(utilities::system&); // Прототип функції перевірки на сходимість метода Якобі

void nextSolution(utilities::system&, double\*, double \*); // Прототип функції знаходження наступного наближення

};

Файл «Jacobi.cpp»

#include "Jacobi.h"

using namespace Jacobi;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція рішення СЛАР методом \*

\* Якобі \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double\* Jacobi::Jacobi(system& sols) {

double\* x = utilities::CreateMas(sols.size);

double\* xk = utilities::CreateMas(sols.size);

for (int i = 0; i < sols.size; i++) xk[i] = 0;

double norm = 2 \* utilities::eps;

while (norm >= utilities::eps) {

utilities::masCopy(xk, x, sols.size);

nextSolution(sols, x, xk);

norm = utilities::normCalc(x, xk, sols.size);

}

utilities::freeMas(x);

return xk;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція перевірки на сходимість \*

\* методу Якобі \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Jacobi::isSolved(system& sols) {

utilities::stableSystem(sols);

if (utilities::isDiagDominate(sols)) return true;

else return false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція знаходження наступного \*

\* наближення для методу Якобі \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\* x - поперднє наближення \*

\* xk - поточне наближення \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Jacobi::nextSolution(system& sols, double \*x, double\* xk) {

for (int i = 0; i < sols.size; i++) {

xk[i] = sols.free[i];

for (int z = 0; z < sols.size; z++)

if (i != z) xk[i] -= sols.matrix[i][z] \* x[z];

xk[i] /= sols.matrix[i][i];

}

}

Файл «Gauss-Seidel.h»

#pragma once

#include "Utilities.h"

namespace Seidel {

double\* Seidel(utilities::system&); // Прототип функції рішення системи методом Гауса-Зейделя

bool isSolved(utilities::system&); // Прототип функції перевірки на сходимість метода Гауса-Зейделя

void nextSolution(utilities::system&, double\*, double \*); // Прототип функції знаходження наступного наближення

};

Файл «Gauss-Seidel.cpp»

#include "Gauss–Seidel.h"

using namespace Seidel;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція рішення СЛАР методом \*

\* Гауса-Зейделя \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double\* Seidel::Seidel(utilities::system& sols) {

double\* x = utilities::CreateMas(sols.size);

double\* xk = utilities::CreateMas(sols.size);

for (int i = 0; i < sols.size; i++) xk[i] = 0;

double norm = 2 \* utilities::eps;

while (norm >= utilities::eps) {

utilities::masCopy(xk, x, sols.size);

nextSolution(sols, x, xk);

norm = utilities::normCalc(x, xk, sols.size);

}

delete[] x;

return xk;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція перевірки на сходимість \*

\* методу Гауса-Зейделя \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Seidel::isSolved(utilities::system& sols) {

utilities::stableSystem(sols);

if (utilities::isDiagDominate(sols) || (utilities::isPositive(sols) && utilities::isSemetric(sols))) return true;

else return false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція знаходження наступного \*

\* наближення для методу \*

\* Гауса-Зейделя \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\* x - поперднє наближення \*

\* xk - поточне наближення \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Seidel::nextSolution(utilities::system& sols, double\* x, double \* xk) {

for (int i = 0; i < sols.size; i++) {

xk[i] = sols.free[i];

for (int z = 0; z < i; z++) xk[i] -= sols.matrix[i][z] \* xk[z];

for (int z = i + 1; z < sols.size; z++) xk[i] -= sols.matrix[i][z] \* xk[z];

xk[i] /= sols.matrix[i][i];

}

}

Файл «GradientDescent.h»

#pragma once

#include "Utilities.h"

namespace GradientDescent {

bool isSolved(utilities::system&); // Прототип функції перевірки на сходимість метода градієнтоного спуску

double\* GradientDescent(utilities::system&); // Прототип функції рішення системи методом градієнтоного спуску

};

Файл «GradientDescent.cpp»

#include "GradientDescent.h"

using utilities::scalarMult;

using utilities::matrixMult;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція рішення СЛАР методом \*

\* градієнтного спуску \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double\* GradientDescent::GradientDescent(utilities::system& sols) {

double\* x = utilities::CreateMas(sols.size); // Поточне рішення системи

double\* xk = utilities::CreateMas(sols.size); // Наступне наближення рішення

double\* r = utilities::CreateMas(sols.size); // Поточне значення незвязка

double\* rk = utilities::CreateMas(sols.size); // Наступне значення незвязки

double\* z = utilities::CreateMas(sols.size); // Поточне значення вектору напряку

double\* zk = utilities::CreateMas(sols.size);// Наступне значення вектору напрямку

// Задання початкових уточнень

for (int i = 0; i < sols.size; i++) {

xk[i] = 0;

rk[i] = sols.free[i];

zk[i] = sols.free[i];

}

double norm = 2 \* utilities::eps;

while (norm >= utilities::eps) {

utilities::masCopy(xk, x, sols.size);

utilities::masCopy(rk, r, sols.size);

utilities::masCopy(zk, z, sols.size);

double\* Az = matrixMult(sols, z);

int j;

for (j = 0; j < sols.size; j++) if (Az[j]) break;

if (j == sols.size) break;

double alpha = scalarMult(r, r, sols.size) / scalarMult(Az, z, sols.size) ; // Скалярний крок градієнту

for (int i = 0; i < sols.size; i++) xk[i] += alpha \* z[i];

for (int i = 0; i < sols.size; i++) rk[i] -= alpha \* Az[i];

double beta = scalarMult(rk, rk, sols.size) / scalarMult(r, r, sols.size); // Скалярна корекція напрямку

utilities::masCopy(rk, zk, sols.size);

for (int i = 0; i < sols.size; i++) zk[i] += beta \* z[i];

utilities::freeMas(Az);

norm = utilities::normCalc(x, xk, sols.size);

}

utilities::freeMas(x);

utilities::freeMas(r);

utilities::freeMas(rk);

utilities::freeMas(z);

utilities::freeMas(zk);

return xk;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Функція перевірки на сходимість \*

\* методу градієнтного спуску \*

\* Параметри: \*

\* sols - система лінійних алгебраїчних \*

\* рівнянь \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool GradientDescent::isSolved(utilities::system& sols) {

return utilities::isPositive(sols) && utilities::isSemetric(sols);

}

Файл «Graph.h»

#pragma once

#include "Utilities.h"

#include <algorithm>

namespace CourseWork {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Summary for Graph

/// </summary>

public ref class Graph : public System::Windows::Forms::Form

{

public:

Graph(utilities::system& sols, double\* x)

{

InitializeComponent(sols, x);

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~Graph()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::DataVisualization::Charting::Chart^ CGraph;

protected:

private:

/// <summary>

/// Required designer variable.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(utilities::system& sols, double\* x)

{

System::Windows::Forms::DataVisualization::Charting::ChartArea^ chartArea1 = (gcnew System::Windows::Forms::DataVisualization::Charting::ChartArea());

System::Windows::Forms::DataVisualization::Charting::Legend^ legend1 = (gcnew System::Windows::Forms::DataVisualization::Charting::Legend());

System::Windows::Forms::DataVisualization::Charting::Series^ series1 = (gcnew System::Windows::Forms::DataVisualization::Charting::Series());

System::Windows::Forms::DataVisualization::Charting::Series^ series2 = (gcnew System::Windows::Forms::DataVisualization::Charting::Series());

this->CGraph = (gcnew System::Windows::Forms::DataVisualization::Charting::Chart());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->CGraph))->BeginInit();

this->SuspendLayout();

//

// CGraph

//

chartArea1->Name = L"ChartArea1";

this->CGraph->ChartAreas->Add(chartArea1);

legend1->Name = L"Legend1";

this->CGraph->Legends->Add(legend1);

this->CGraph->Location = System::Drawing::Point(0, 0);

this->CGraph->Name = L"CGraph";

series1->ChartArea = L"ChartArea1";

series1->ChartType = System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Spline;

series1->Legend = L"Legend1";

series1->LegendText = L"Система-1";

series1->Name = L"Series1";

series2->ChartArea = L"ChartArea1";

series2->ChartType = System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Spline;

series2->Legend = L"Legend1";

series2->LegendText = L"Система-2";

series2->Name = L"Series2";

this->CGraph->Series->Add(series1);

this->CGraph->Series->Add(series2);

this->CGraph->Size = System::Drawing::Size(784, 562);

this->CGraph->TabIndex = 0;

this->CGraph->Text = L"chart1";

//

// Graph

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(784, 562);

this->Controls->Add(this->CGraph);

this->Name = L"Graph";

this->Text = L"Графік";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->CGraph))->EndInit();

this->ResumeLayout(false);

double xl = x[0] - 10;

double xr = x[0] + 10;

double yl = x[1] - 10;

double yr = x[1] + 10;

if (sols.matrix[0][1] != 0) {

yl = std::min(yl, (sols.free[0] - sols.matrix[0][0] \* xl) / sols.matrix[0][1]);

yr = std::max(yr, (sols.free[0] - sols.matrix[0][0] \* xr) / sols.matrix[0][1]);

}

if (sols.matrix[1][1] != 0) {

yl = std::min(yl, (sols.free[1] - sols.matrix[1][0] \* xl) / sols.matrix[1][1]);

yr = std::max(yr, (sols.free[1] - sols.matrix[1][0] \* xr) / sols.matrix[1][1]);

}

chartArea1->AxisX->ScaleView->Zoom(xl, xr);

chartArea1->CursorX->IsUserEnabled = true;

chartArea1->CursorX->IsUserSelectionEnabled = true;

chartArea1->AxisX->ScaleView->Zoomable = true;

chartArea1->AxisX->Title = "X";

chartArea1->AxisY->ScaleView->Zoom(yl, yr);

chartArea1->CursorY->IsUserEnabled = true;

chartArea1->CursorY->IsUserSelectionEnabled = true;

chartArea1->AxisY->ScaleView->Zoomable = true;

chartArea1->AxisY->Title = "Y";

if(sols.matrix[0][1] != 0)

for (double i = xl; i <= xr; i += 0.1) series1->Points->AddXY(i, (sols.free[0] - sols.matrix[0][0] \* i) / sols.matrix[0][1]);

else for (double i = yl; i <= yr; i += 0.1) series1->Points->AddXY(sols.free[0] / sols.matrix[0][0], i);

if (sols.matrix[1][1] != 0)

for (double i = xl; i <= xr; i += 0.1) series2->Points->AddXY(i, (sols.free[1] - sols.matrix[1][0] \* i) / sols.matrix[1][1]);

else for (double i = yl; i <= yr; i += 0.1) series2->Points->AddXY(sols.free[1] / sols.matrix[1][0], i);

}

#pragma endregion

};

}

Файл «Home.h»

Файл «Home.сpp»

#include "Home.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void main(array<String^>^ arg) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

CourseWork::Home form;

Application::Run(%form);

}