МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

БІЛОУС РОСТИСЛАВ ВІТАЛІЙОВИЧ

Допускається до захисту:
в. о. завідувача кафедри,
прикладної математики
_____ Вєтров О.С.
«____» _____2022 року

ДОСЛІДЖЕННЯ ТОЧНОСТІ ОБЧИСЛЕНЬ З ПЛАВАЮЧОЮ КОМОЮ ДЛЯ СУЧАСНИХ АПАРАТНИХ ПЛАТФОРМ

Спеціальність 113 Прикладна математика

Кваліфікаційна (магістерська) робота
(відповідно до стандарту спеціальності та ОП)

Науковий керівник:
І. Г. Крикун, доцент кафедри
прикладної математики,
к. ф.-м. н., доцент
_____
(підпис)

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця 2022

АНОТАЦІЯ

Білоус Р. В. Дослідження точності обчислень з плаваючою комою для сучасних апаратних платформ.

Спеціальність 113 «Прикладна Математика», спеціалізація «Прикладна математика». Донецький національний університет імені Василя Стуса, Вінниця, 2022.

У магістерській роботі досліджена послідовність Мюллера та збіжність комп'ютерних досліджень. Продемонстровано виникнення похибок при комп'ютерних обчисленнях нескладних виразів. Побудовано математичне обґрунтування проблеми нестійких початкових значень, коли ми наперед знаємо, число до якого має збігатись значення членів послідовності, але стандартні методи комп'ютерних обчислень дають некоректний результат.

Ключові слова: комп'ютерні обчислення, послідовність Мюллера, збіжність, похибки, програмування, стійкість, числа з плаваючою комою.

39 с., 1 табл., 16 рис., 23 джерела.

ABSTRACT

Bilous R. Study of the accuracy of floating-point calculations for modern hardware platforms. Specialty 113 "Applied mathematics", programme "Applied mathematics". Vasyl` Stus Donetsk National University, Vinnytsia, 2022

In the master`s work, the sequence of Muller and the convergence of computer research are studied. Errors in computer calculations of simple expressions have been demonstrated. A mathematical substantiation of the problem of unstable initial values is constructed, when we know in advance the number to which the values of the members of the sequence should coincide, but standard methods of computer calculations give an incorrect result.

Keywords: computer calculations, Muller's sequence, convergence, errors, programming, stability, floating point numbers.

Pages 39. Tabl. 1. Fig. 16. Bibliography: 23 items.

ЗМІСТ

ВСТУП

Проблема надійності (достовірності) комп'ютерних обчислень є однією з фундаментальних проблем комп'ютерних наук [1, 2], оскільки лежить на перетині прикладної математики з одного боку, та фізико-технічних обмежень комп'ютерної техніки – з іншого. Фізичні обмеження на обсяг пам'яті, що виділяється комп'ютерною технікою для обробки числа накладає принципові обмеження на можливості та логіку організації комп'ютерних обрахунків, що також потребує залучення специфічних математичних методів дослідження.

При цьому, проблема полягає не лише у можливості некоректного результату теоретичного дослідження. Як показує практика, наслідки невірних комп'ютерних обчислень можуть бути катастрофічними. Відомий випадок: 25 лютого 1991 року установка ППО «Patriot» не змогла перехопити ворожу ракету, і снаряд потрапив в барак солдатів США, в результаті чого загинуло 28 осіб. Офіційне розслідування показало [3], що 24-бітові процесори перехоплювача при конвертації часу роблять помилку в 0,013 секунди щогодини. «Patriot» не перезапускався понад 100 годин, що стало причиною помилки обчислення положення ракети на 600 метрів.

В сучасній вітчизняній науковій літературі тематика точності комп'ютерних обчислень, на думку автора, відображена явно недостатньо, на рівні констатації проблеми. Майже будь-який академічний підручник з методів обчислень – невід'ємної дисципліни курсу підготовки фахівців як фізико-математичного профілю, так і спеціалістів з комп'ютерних наук – містить, як правило, лише математичні відомості про похибки та межі точності обчислень. Прикладна ж сторона питання, тобто саме комп'ютерна реалізація обрахунків, залишається поза увагою. Технічні ж дисципліни, на кшталт «Програмування», якщо знову ж таки судити з навчальної літератури, дають зазвичай лише базові поняття конвертації чисел до бінарного коду і назад, та наводять загальні відомості про існуючи стандарти представлення чисел с плаваючою комою.

Частково через це автор і зацікавився тематикою коректності комп'ютерних обчислень. Для прикладу, в даній роботі автор зосередився на розгляді двох відомих випадках некоректної поведінки програмних обрахунків: прикладі Рампа [4] та послідовності Мюллера [5], що не відображені у вітчизняній літературі.

Основною проблемою при проведені представленого дослідження було відсутність загальної методології аналітичного дослідження нелінійних рекурентних послідовностей. Теорія побудови розв'язку лінійних рекурентних рівнянь добре вивчена, з основними поняттями та методами можна ознайомитись в доброму підручнику з дискретної математики [5], наявна серйозна наукова література, присвячена зв'язку рекурентних (різницевих) рівнянь та обчислювальних процесам у інформатиці (наприклад, [7, 8]). Що стосується випадку нелінійності, то загальних методів аналітичного розв'язку подібного роду рівнянь не існує, і кожен окремий випадок представляє собою творчу задачу. Автор ознайомився із доступними для нього матеріалами з цього приводу [9, 10], запозичив деякі ідеї, але основна методологія дослідження аналізу збіжності послідовності Мюллера (розділ 3) була розроблена їм самостійно разом із науковим керівником.

Прикладна задача дослідження стійкості комп'ютерних обчислень була розглянута як з точки зору практичної реалізації мовами програмування, так і з математичного боку, оскільки в даному випадку лише інтуїтивний підхід приводить до некоректних результатів.

Частина результатів магістерської роботи знайшла застосування при виконанні науково-дослідної роботи «Розробка методів дослідження міцності та стійкості тонкостінних оболонок та пружних твердих тіл з рідиною при дії різного виду динамічних навантажень» під керівництвом академіка НАН України, д.ф.-м.н., професора В.П. Шевченка (№ держреєстрації 0119U100042).

Актуальність даної теми полягає в тому, що в наш час людина майже щодня має справу з комп'ютерними обчисленнями. Хоча розглянуті в роботі похибки та їм подібні трапляються вкрай рідко, але, зважаючи на кількість

ЕОМ у світі, та на ще більшу кількість операцій, що відбуваються щосекунди, виникненням таких похибок та їх впливом на навколишній світ людство не може нехтувати.

Мета дослідження: проаналізувати проблеми, що виникають при комп'ютерних обчисленнях в різних програмах, шляхи їх уникнення та підвищення точності на основі найбільш відомих прикладів – прикладу Рампа та послідовності Мюллера.

Об'єкт дослідження: числові розрахунки з використанням програмного забезпечення.

Предмет дослідження: похибки, що виникають в числових розрахунках з використанням програмного забезпечення та шляхи їх уникнення.

Для досягнення поставленої мети необхідно виконати наступні **завдання:**

1) Дослідити представлення дійсних чисел у пам'яті комп'ютера в різних програмних середовищах;
2) Проаналізувати процес виникнення помилок при комп'ютерних обчисленнях в різних програмних середовищах та програмних модулях;
3) Розглянути основні методи, які дають змогу позбутись деяких помилок комп'ютерних обчислень;
4) Вивчити точність збіжності комп'ютерних обчислень на прикладі послідовності Мюллера.

Наукова новизна: в ході роботи було встановлено існування можливості прогнозування можливих похибок та парадоксальних результатів комп'ютерних обчислень в різних програмних середовищах, а також було побудоване математичне обґрунтування проблеми нестійких початкових значень та виведена формула для перевірки початкових значень для розглянутої рекурентної послідовності відомої як послідовність Мюллера.

Практичне значення отриманих результатів.

За допомогою отриманих в дослідженні результатів можна намагатись передбачувати виникнення парадоксальних результатів при комп'ютерних

обчисленнях. Також результати дослідження можуть бути використані при вивченні інших неточностей та парадоксів, що виникають при застосуванні комп'ютерних обчислень та в цілому при побудові алгоритмів та написанні програм (різними мовами програмування), в яких використовуються логіка пов'язана з числам з плаваючою комою.

Апробація результатів дослідження.

Основні результати, отримані в магістерській роботі, були опубліковані у статті в закордонному фаховому виданні [23], в колективній монографії [14] та доповідались на декількох міжнародних конференціях, що проводилась як в Україні [11, 12, 13], так і за кордоном [22].

Структура магістерської роботи. Магістерська робота складається із вступу, трьох розділів, в першому з яких наводяться основні поняття з теорії представлення чисел з плаваючою комою у пам'яті комп'ютера та програмні приклади неточностей при комп'ютерних обчисленнях; в другому розділі детально описуються приклади несподіваних результатів в комп'ютерних обчисленнях та з'ясовуються причини їх виникнення, в третьому описується рекурентна послідовність відома як послідовність Мюллера та проводиться її подальше дослідження; загальних висновків до дослідження та списку використаних джерел.

Загальний розмір магістерської роботи – 39 сторінок, а розмір основної частини – 36 сторінок. Магістерська робота містить 16 ілюстрацій. Список використаних джерел складається з 23 джерел.

РОЗДІЛ 1. ПРОБЛЕМАТИКА КОМП'ЮТЕРНИХ ОБЧИСЛЕНЬ

В розділі коротко наведені деякі приклади некоректності комп'ютерних обчислень, що виникають у зв'язку із обмеженим обсягом пам'яті, що виділяється комп'ютером для роботи із числами. Дійсні числа, що є фундаментальним об'єктом математики, перетворюються на числа з плаваючою комою (*floating point numbers*) згідно ухвалених стандартів.

Неврахування фундаментальних основ бінарної арифметики та засад стандарту IEEE-754 може суттєво вплинути на коректність як теоретичного дослідження (якщо для моделювання була використана комп'ютерна техніка), так і просто на якість продукту при розробці прикладних програм.

Програми реалізуються мовами C++ або Python в залежності від того, коли це доречніше: в C++ ми можемо наочно експериментувати, змінюючи типи чисел з плаваючою комою *float* та *double*; в Python же за замовчуванням реалізована довга арифметика.

1.1. Простий приклад похибки комп'ютерних обчислень

Розглянемо наступний приклад особливості комп'ютерних обчислень. Перевіримо за допомогою мови програмування C++, чи виконується закон дистрибутивності у комп'ютерних обчисленнях. Проста програма реалізована на C++ матиме вигляд (Рис. 1.1 нижче)

```
#include <iostream>
using namespace std;
int main() {
    double a = 0.987;
    double b = 0.278;
    double c = 1.268;
    double expr_1 = a*c + b*c;
    double expr_2 = (a + b)*c;
    cout<<(expr_1 == expr_2);
}
```

Рис. 1.1. Лістинг програми для перевірки виконання закону дистрибутивності для формату *double*

```
#include <iostream>
using namespace std;
int main() {
    float a = 0.9871;
    float b = 0.2781;
    float c = 1.268;
    float expr_1 = a*c + b*c;
    float expr_2 = (a + b)*c;
    cout<<(expr_1 == expr_2);
}
```

Рис. 1.2. Лістинг програми для перевірки виконання закону дистрибутивності для формату *float*

Очікувано, ми отримаємо результат 1 (тобто *true*). Трохи змінимо значення чисел a та b, збільшивши їх на 0,0001 (тобто змінимо числа a та b на 0,01 % та 0,04 % відповідно). В результаті вже отримаємо результат 0 (в значенні *false*). Якщо ми виведемо на консоль відповідні значення для випадку $a = 0{,}9871$ та $b = 0{,}2781$ з точністю 20 знаків після коми то отримаємо

$$a \cdot c + b \cdot c \to 1{,}60427359999999996631,$$

$$(a + b) \cdot c \to 1{,}60427360000000018836.$$

різні значення, що несуттєво, але відрізняються від реального значення 1,6042736 (абсолютна похибка складає $3{,}4 \cdot 10^{-17}$ та $1{,}9 \cdot 10^{-16}$). Змінимо типи змінних у програмі на *float* (Рис. 1.2), і в результаті вже отримаємо результат *true*. Тобто, саме зменшення точності обчислень повертає нас до коректного з

математичної точки зору результату, що на перший погляд виглядає дещо парадоксально.

Справедливою є стара істина про те, що порівнювати числа з плаваючою комою завжди треба дуже обережно, оскільки результати з теоретичної точки зору можуть видаватись некоректними. І для того, щоб переконатись у цьому не потрібно вигадувати складні приклади. Достатньо ознайомитись з відповідною документацією стосовно класичної проблеми комп'ютерних обчислень

" $(0{,}1 + 0{,}2)? \, 0{,}3$" [15].

Дивлячись на отримані результати, може скластися думка, що принциповість проблеми коректності комп'ютерних обчислень є дещо надуманою, бо навіть в наведеному прикладі похибка складає значення співмірне із машинним епсилоном. Та й при реалізації реальних обчислень в прикладних задачах точність в 16 знаків після коми є скоріш за все надмірною. Тим не менш, аналізуючи отримані результати лише на одному прикладі обчислень, ми можемо зробити попередні висновки:

- змінюючи тип операндів у виразах (в нашому прикладі експериментуючи з типами *float* та *double*), можна отримати принципово різні результати, обраховані формально на одному і тому ж наборі даних;
- точність комп'ютерних обрахунків може якісно впливати результат (в нашому прикладі отримані результати відрізняються від математичного зовсім несуттєво, але формально перестає виконуватись закон дистрибутивності);
- результат комп'ютерних обрахунків залежить від порядку виконання арифметичних операцій;
- порівняння чисел з плаваючою комою доречніше здійснювати не на точну рівність, а за схемою $|x - y| < \varepsilon$.

1.2. Представлення чисел з плаваючою комою у різних програмних форматах

Зазначені складності комп'ютерних обчислень виникають в наслідок способу представлення дійсних чисел у пам'яті комп'ютера, що регулюється стандартами IEEE-754 [16].

Опис стандарту IEEE-754 виходить за межі даної роботи, тому обмежимось лише демонстрацією представлення чисел з описаного вище прикладу у форматах *float* та *double*.

Формат *float* – під число виділяється 32 біти: старший біт знаковий (0 означає додатне число, 1 – число є від'ємним), далі 8 біт виділяється під експоненту, і останні 23 біти під нормалізовану мантису. Схематично це можна зобразити на Рис. 1.3. як

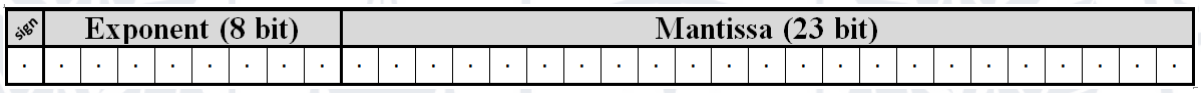

Рис. 1.3. Представлення числа типу *float* у пам'яті комп'ютера

Числа з прикладу, конвертовані у бінарний код згідно правил формату *float*, матимуть вигляд

$$(0.9871)_{10} = (00111111\ 01111100\ 10110010\ 10010110)_2,$$

$$(0.2781)_{10} = (00111110\ 10001110\ 01100011\ 00100000)_2,$$

$$(1.2680)_{10} = (00111111\ 10100010\ 01001101\ 11010011)_2.$$

Формат *double* (подвійна точність) – під число виділяється 64 біти: старший біт знаковий, 11 біт під експоненту, і останні 52 біти відповідно виділяються під нормалізовану мантису (Рис. 1.4).

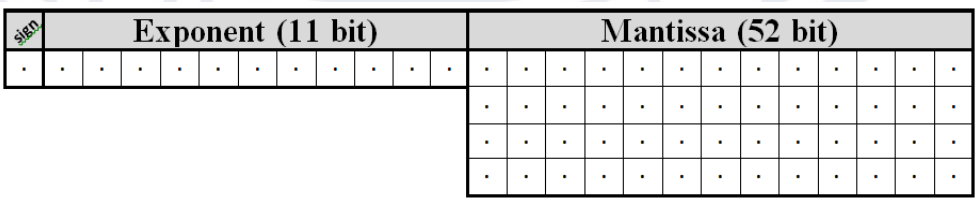

Рис. 1.4. Представлення числа типу *double* у пам'яті комп'ютера

Згадані числа у форматі *double* запишуться:

$$(0.9871)_{10} = (00111111\ 11101111\ 10010110\ 01010010\ 10111101\ 00111100\ 00110110\ 00010001)_2,$$

$$(0.2781)_{10} = (00111111\ 11010001\ 11001100\ 01100011\ 11110001\ 01000001\ 00100000\ 01011100)_2,$$

$$(1.2680)_{10} = (00111111\ 11110100\ 01001001\ 10111010\ 01011110\ 00110101\ 00111111\ 01111101)_2.$$

Наостанок продемонструємо результат зворотної конвертації з бінарного коду назад до десяткової системи числення.

Для формату *float:*

$$(0.9871)_{10} \to (...)_2 \to (0.9871000051498413085937 5)_{10}.$$

Для формату *double:*

$$(0.9871)_{10} \to (...)_2 \to (\ 0.98709999999999997744026813961 7)_{10}.$$

Цікаво відзначити, що у даному конкретному прикладі значення типу *float* виявляється більше за оригінальне, а значення типу *double* – менше.

Простий приклад порівняння двох однакових за абсолютним значенням чисел представлених у різних форматах реалізованих мовою програмування Java 8 (Рис. 1.5.).

```
public class MyClass {
    public static void main(String args[]) {
      float a = 0.3f;
      double b = 0.3;
      System.out.println("a == b >> " + (a == b));
      System.out.println("a-b >> " + (a-b));
    }
}
```

Рис. 1.5. Лістинг програми для порівняння числа 0,3 представленого у форматах float та double

У цьому прикладі порівнюється число 0,3 представлене у форматах *float* та *double.*

```
a == b >> false
a-b >> 1.1920928966180355E-8
```

Рис. 1.6. Результат роботи програми для порівняння числа 0,3 представленого у форматах float та double

В результаті(Рис. 1.6.) ми можемо бачити, що абсолютні значення двох на перший погляд однакових чисел не однакові та досить сильно відрізняються якщо ми говоримо про точність комп'ютерних обчислень.

Також ця ж проблема проявляє себе при відніманні, спробуємо отримати 0 віднімання від числа 1 число 0,1 десять разів, для цього створимо програму мовою програмування Java8 (Рис. 1.7.)

```
public class MyClass {
    public static void main(String args[]) {
        double x = 1.0;
        while(x!=0.0){
          System.out.println(x);
          x-=0.1;
        }
    }
}
```

Рис. 1.7. Лістинг програми яка демонструє проблеми при відніманні чисел з плаваючою комою

В результаті виконання програми ми отримаємо вічний цикл, тому, що *x* ніколи не дорівнюватиме 0

```
1.0
0.9
0.8
0.7000000000000001
0.6000000000000001
0.5000000000000001
0.40000000000000013
0.30000000000000016
0.20000000000000015
0.10000000000000014
1.3877787807814457E-16
-0.09999999999999987
-0.19999999999999987
-0.2999999999999999
-0.3999999999999999
-0.4999999999999999
-0.5999999999999999
-0.6999999999999998
-0.7999999999999998
-0.8999999999999998
-0.9999999999999998
-1.0999999999999999
```

Рис. 1.8. Частина результати роботи програми з Рис. 1.7.

Висновки до розділу 1

У першому розділі було розглянуто стандарт IEEE-754 та різні похибки комп'ютерних обчислень, пов'язані з представленням дійсних чисел під час операцій з ними у різних програмних середовищах.

Наведено приклади, які показують, що ми не можемо нехтувати різницею у представленні чисел з плаваючою комою у стандарті IEEE-754 та у реальному житті.

РОЗДІЛ 2. НЕСПОДІВАНІ РЕЗУЛЬТАТИ КОМП'ЮТЕРНИХ ОБЧИСЛЕНЬ.

В попередньому розділі автором були розглянуті досить прості випадки некоректних комп'ютерних обчислень, що виникають при роботі з числами з плаваючою комою. Описані приклади можна назвати лінійними, оскільки причини виникнення особливостей обчислень стає очевидною навіть при побіжному аналізі, якщо взяти до уваги стандарт IEEE-754.

2.1. Парадоксальні результати комп'ютерних обчислень. Приклади ніби спростування Теореми Ферма

В попередньому розділі автором був наведений елементарний приклад проблемних ситуацій, що можуть виникати при організації комп'ютерних обрахунків. Обмеження, що накладаються на представлення дійсних чисел у різних форматах згідно стандарту [16], призводять до некоректних по факту результатів обчислень навіть при виконанні в комп'ютерній програмі елементарних арифметичних дій. В [14] автором розглянуто ще кілька подібних прикладів, реалізованих мовами програмування C++ та Python. В принципі, всі згадані приклади не змінюють нашого розуміння результатів обчислень, а лише змушують постійно пам'ятати про невід'ємну похибку комп'ютерних обчислень, пов'язану із скінченим обсягом пам'яті, що може бути виділений для зберігання дійсного числа.

Далі наведемо приклади справді парадоксальності деяких випадків комп'ютерних обчислень, принаймні таке вони справляють перше враження.

Зображення на рисунку 2.1 – кадр зі серії «The Wizard of Evergreen Terrace» відомого анімаційного серіалу The Simpsons, а зображення на рисунку 2.2 – серія "Treehouse of Horror VI. Homer3 (Homer Cubed)".

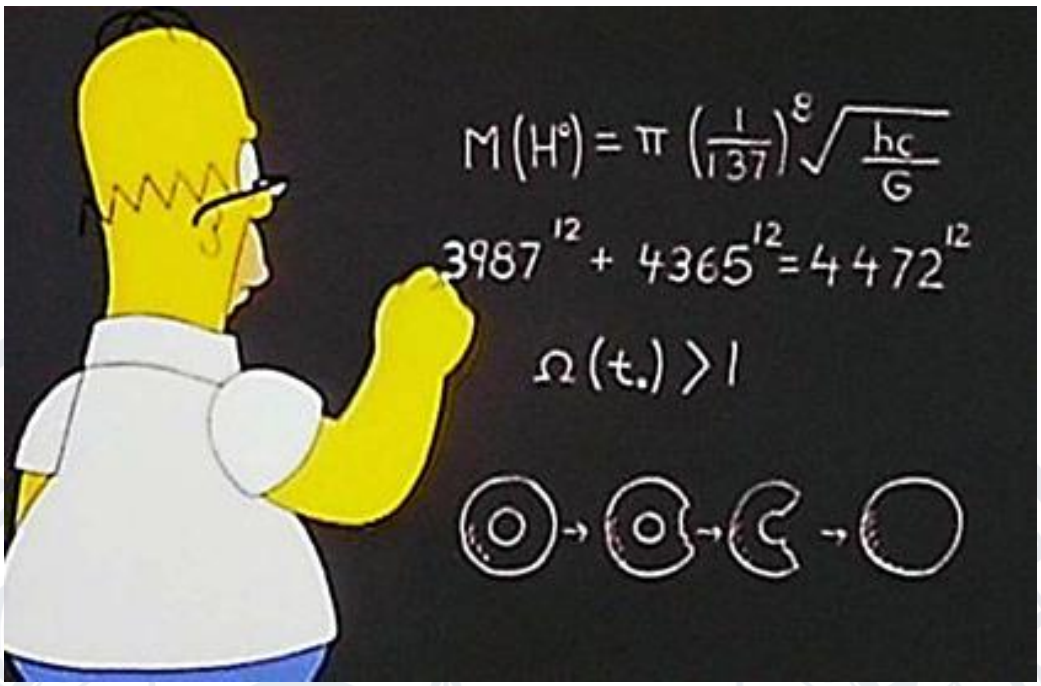

Рис. 2.1. Перший контр-приклад Гомера Сімпсона до Великої Теореми Ферма

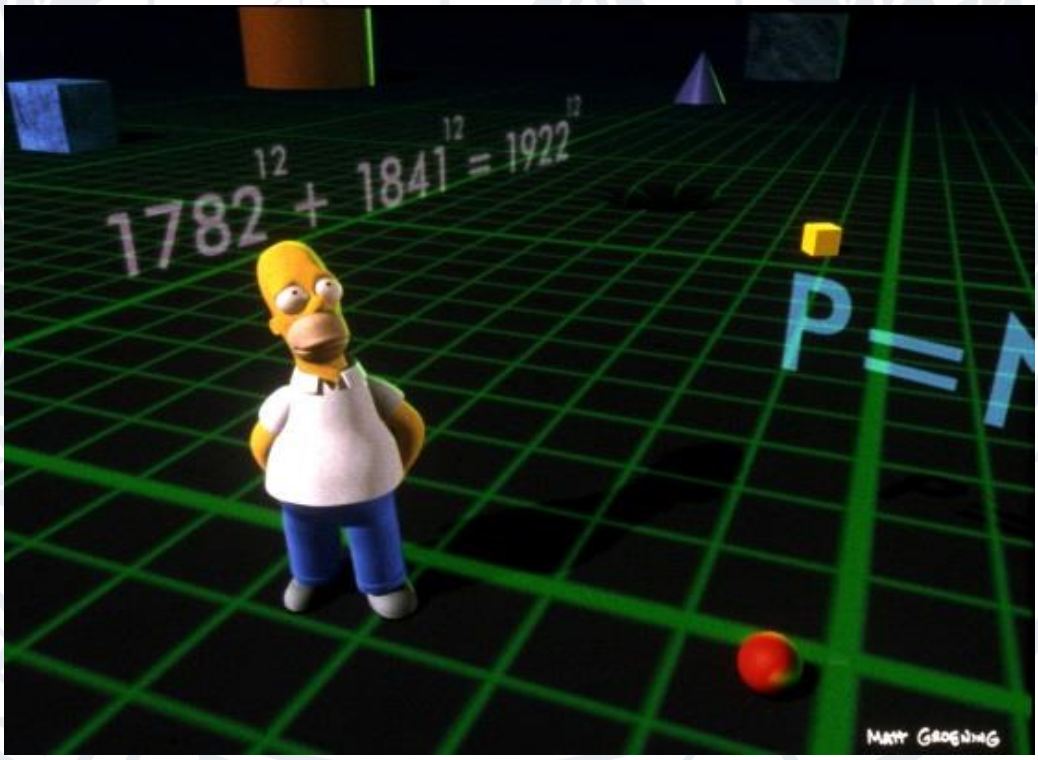

Рис. 2.2. Другий контр-приклад Гомера Сімпсона до Великої Теореми Ферма

Гомер Сімпсон знайшов щонайменше два контр-приклади, що ніби доводять нам, що Велика Теорема Ферма не вірна. Оскільки нам відомо, що Велика Теорема Ферма була остаточно доведена у 1995 році, то ми можемо

просто сприйняти наведені числа як жарт з випадковими числами, наведений лише для наукоподібності. Насправді зазначений жарт значно цікавіший, особливо з позицій проблематики, що підіймаються у даній роботі.

Саймон Сінгх [17] пропонує за допомогою калькулятора переконатись у істинності $3987^{12} + 4365^{12} = 4472^{12}$, але ми перевіримо припущення Гомера реалізувавши програму, написану на C++.

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    float an = pow(398.7,12);
    float bn = pow(436.5,12);
    float cn = pow(447.2,12);
    cout << (an + bn == cn);
}
```

Рис. 2.3. Лістинг програми що реалізую припущення Гомера з Рис. 2.1.

Результатом виконання представленої програми буде 1 (*true*). Одразу зазначимо, що ми замінили наведені на Рис. 2.1 числа їх аналогами, скороченими в 10 разів, оскільки для оригінальних значень результати обчислень дорівнюють *inf*.

Справді, значення виразу:

$$4472^{12} = 6397665634848672580686235832216857578412 4416$$

занадто велике, щоб зберігати його у змінній типу *float* (чотирьох байтів виявляється недостатньо). Тому ми і реалізуємо наступний трюк: будемо перевіряти на рівність з математичної точки зору еквівалентний початковому вираз $398.7^{12} + 436.5^{12} =^{?} 447.2^{12}$.

Ми отримали нібито підтвердження рівності (і спростування твердження теореми Ферма), але реальні математичні значення чисел 398.7^{12}, 436.5^{12}, 447.2^{12} наступні

$$398.7^{12} = 161344746097512912834964919705 15,151715346481$$

$$436.5^{12} = 478421817399473213327397389826 39,336181640625$$

$$447.2^{12} = 639766563484867258068623583221 68,575784124416$$

і можна переконатись, що

$$398{,}7^{12} + 436{,}5^{12} - 447{,}2^{12} = 12118868093738726309 85{,}912112862690.$$

Занадто велика похибка, щоб її ігнорувати.

Повернемось до представлення чисел з плаваючою комою у стандарті IEEE-754 [16]. Числа з прикладу $398{,}7^{12}$, $436{,}5^{12}$, $447{,}2^{12}$ у форматі *float* матимуть наступний вигляд

$$398{,}7^{12} \rightarrow = (01110011\ 01001011\ 10100101\ 01001101)_2$$
$$436{,}5^{12} \rightarrow = (01110100\ 00010110\ 11110110\ 10011011)_2$$
$$447{,}2^{12} \rightarrow = (01110100\ 01001001\ 11011111\ 11101110)_2$$

Якщо тепер виконати зворотну конвертацію з двійкового коду до десяткової системи числення отримаємо

$$\textit{float}(398.7^{12}) = 161344751043042927942754631680 00.000000000000$$
$$\textit{float}(436.5^{12}) = 478421802922309671738398725898 24.000000000000$$
$$\textit{float}(447.2^{12}) = 639766553965352599681153357578 24.000000000000$$

Числа вже змінені та відрізняються від оригіналу.

Можна переконатись, що

$$\textit{float}(398.7^{12}) + \textit{float}(436.5^{12}) = \textit{float}(447.2^{12})$$

Звідси і результат *true* роботи нашої програми з Рис. 2.3. Варто дуже обачливо ставитись до комп’ютерних обчислювальних експериментів у математиці.

Зазначимо, що для типу *double* відповідний результат буде *false*: більш висока точність не дає нам отримати некоректний результат. Але як ми побачимо далі, і тип *double* не є панацеєю від усіх проблем комп’ютерних обчислень.

2.2. Приклад Рампа.

Розглянемо далі приклад, запропонований Рампом [4] ще 1988 року. Необхідно обчислити значення виразу

$$f(x,y) = y^2 \cdot \left(y^2 \cdot \left(y^2 \cdot \left(5{,}5 \cdot y^2 (333{,}75 - x^2)\right) - 121 \cdot x^2\right) + 11 \cdot x^4\right) - 2 \cdot x^2$$

в точці (77617; 33096).

Задача виглядає досить простою. Якщо тимчасово не зважати на останній доданок $\frac{x}{2y}$ (який насправді і немає великого значення), та враховуючи, що $y = 33096$ парне число, то очевидним є той факт, що значення виразу $f(x,y) - \frac{x}{2y}$ в загалі має бути цілим числом.

Для реалізації прикладу Рампа скористаємось Python 3.x, оскільки в Python за замовчуванням реалізований механізм довгої арифметики, що дає змогу працювати з цілими числами довільної довжини. Також, важливо зазначити, що Python дає можливість дуже просто звертатись до великої кількості сторонніх спеціальних бібліотек для професійної роботи з комп'ютерними обчисленнями.

```
x=77617
y=33096
print (333.75*y**6+x**2*(11*x**2-y**2-y**6-121*y**4-2)+5.5*y**8+0.5*x/y)
```

Рис. 2.4. Лістинг програмної реалізації прикладу Рампа

Результатом виконання наведеної в Рис. 2.4. програми буде значення 11805916207174113034240 хоча насправді

$$f(77617{,}33096) = -\frac{54767}{66192}.$$

Особливо цікавим з даному прикладі є не те, що комп'ютерні обрахунки є некоректними (це бачили вже не раз), а те, наскільки нестійким є цей результат. Збільшимо або зменшимо будь-яке число 77617 або 33096 бодай на одиницю, і комп'ютер стане рахувати ближче до реальності, принаймні порядки

математичного та машинного результатів будуть значно більш співвідносні. Характерна проблема саме в точці (77617; 33096).

Пояснимо у чому особливість зазначеної точки. Відокремимо додатну та від'ємну частини виразу (як вже було сказано: останній доданок $0.5 \cdot x/y$ на коректність результату обчислення суттєво не впливає). Відділимо додатну та від'ємну частини виразу $f(x,y) - \frac{x}{2y}$, як це продемонстровано на Рис. 2.5.

```
x=77617
y=33096
positive = 1335*y**6//4+x**2*11*x**2*y**2+11*y**8//2
negative = x**2*y**6+121*x**2*y**4+2*x**2
print(positive, negative, sep="\n")
```

Рис. 2.5. Лістинг програмної реалізації прикладу Рампа з відокремленими додатною та від'ємною частинами

Отримаємо результат

$$positive = 79171122165662886646897613164268499 84;$$

$$negative = 79171122165662886646897613164268499 86.$$

Тобто в нашому прикладі $positive - negative = -2$, і тоді остаточний результат збігається з математичним. Діло у тому, що однією із головних проблем при роботі з числами з рухомою комою є операція віднімання, оскільки при відніманні близьких за значенням чисел значимі розряди можуть втрачатись, і результат у цьому випадку буде некоректним. Якщо ми змінимо в програмі один рядок, замінивши цілочисельне ділення (//) на звичайне (/), тобто

$$positive = \frac{1335 \cdot y^6}{4} + x^2 \cdot (11 \cdot x^2 y^2) + \frac{11 \cdot y^8}{2}$$

результат вже буде дорівнювати

$$positive = 79171122165662886297210756321299660 80$$

і різниця між значеннями *positive* та *negative* складатиме вже -34968685684296883906. Якщо ж записати вираз для змінної *positive* в більш звичному для користувача Python форматі

$$positive = \frac{1335}{4} \cdot y^6 + x^2 \cdot (11 \cdot x^2 \cdot y^2) + \frac{11}{2} \cdot y^8,$$

то результат буде вже

$$positive = 79171122165662898103126963495412695 04$$

оскільки даються в знаки різні похибки округлень для різних арифметичних операцій (як ми вже зазначали у попередньому розділі: порядок операцій також має значення для точності обчислень). Різниця в цьому випадку між отриманим значенням *positive* і попереднім складатиме 1180591620717411303424, а значення $positive - negative = 1145622935033114419518$. Коректно змінюємо послідовність виконання арифметичних операцій і одразу отримаємо похибки порядку 10^{21}.

Зазначимо, що в Python є можливість певною мірою нівелювати деякі проблеми, що виникають про обрахунках з числами з плаваючою комою – наприклад, використовуючи функціонал модуля *decimal* для контролю точності обчислень. З офіційною документацією щодо модуля *decimal* можна ознайомитись за посиланням [18]. Не вдаючись в зайві подробиці, зазначимо, що модуль *decimal* (зокрема клас *Decimal*) дає можливість працювати з числами з плаваючою комою як із звичайними дробами.

При всьому вище сказаному, для прикладу Рампа лише використання можливостей класу *Decimal* видається недостатнім. Пропускаючи декілька проміжних етапів дослідження, автор знайшов спосіб отримати коректний результат програмних обрахунків. Спосіб цей полягає у спільному використанні модулю *decimal* разом із застосуванням схеми скорочення множення (схеми Горнера), з метою скорочення кількості арифметичних операцій, так і збільшення точності обчислень. У цьому випадку ми маємо переписати вираз $f(x, y) - \frac{x}{2y}$ у вигляді

$$f(x,y) = y^2 \cdot \left(y^2 \cdot \left(y^2 \cdot \left(5{,}5 \cdot y^2(333{,}75 - x^2)\right) - 121 \cdot x^2\right) + 11 \cdot x^4\right) - 2 \cdot x^2$$

Програмно реалізуємо описаний підхід (Рис. 2.6.).

```
x=77617
y=33096
x2=Decimal(x)**2
y2=Decimal(y)**2
y4=y2**2
x4=x2**2
print(y2*(y2*(y2*(Decimal(5.5)*y2+(Decimal(333.75)-x2))-121*x2)+11*x4)-2*x2)
```

Рис. 2.6. Лістинг програмної реалізації прикладу Рампа з використанням бібліотеки *Decimal*

В результаті отримаємо очікуване значення -2.

Висновки до розділу 2

У другому розділі були представлені конкретні приклади парадоксальних результатів начебто простих обчислень, що пов'язані з обмеженістю пам'яті комп'ютера.

В результаті проведених експериментів та досліджень, можна зробити висновок, що в рідкісних випадках, навіть прості операції можуть призвести до великих похибок та непередбачуваних результатів. Що в свою чергу змушує звернути увагу на методи, що дозволяють зменшити частоту виникання таких помилок та спрогнозувати потенційне їх виникнення.

РОЗДІЛ 3. ПРОБЛЕМА СТІЙКОСТІ КОМП'ЮТЕРНИХ ОБЧИСЛЕНЬ (НА ПРИКЛАДІ ПОСЛІДОВНОСТІ МЮЛЛЕРА).

В даному розділі ми розглянемо досить складний випадок парадоксальності деяких висновків, що в нас виникнуть при дослідженні числової збіжності певної нелінійної рекурентної послідовності, відомої в літературі як послідовність Мюллера.

Для аналізу особливості поведінки програмних обчислень зазначеного об'єкту, нам вже недостатньо просто знання про стандарт представлення тих чи інших чисел в пам'яті комп'ютера. Також лише загальних відомостей про точки стійкості-нестійкості (як в теорії диференціальних рівнянь) виявиться з теоретичної точки зору замало. Ми будемо вимушені залучити потужний математичний апарат для розуміння тих нетривіальних проблем, що можуть виникати при програмній реалізації досить нескладної на перший погляд математичної задачі.

3.1. Постановка задачі.

Розглянемо математичний об'єкт, що був наведений назвемо послідовністю Ж.-М. Мюллером у фундаментальній роботі [5], присвяченій проблемі прикладних комп'ютерних обчислень. Це нелінійне неоднорідне рекурентне рівняння другого порядку, яку будемо називати просто послідовністю Мюллера

$$\begin{cases} u(n) = 111 - \frac{1130}{u(n-1)} + \frac{3000}{u(n-1)\cdot u(n-2)}, \\ u(0) = 2,\ u(1) = -4. \end{cases} \tag{3.1}$$

Реалізуємо просту програму на Python (Рис. 3.1.).

```
u0 = 2
u1 = -4
for n  in range(2, 31):
    un = 111 - 1130 / u1 + 3000 / u0 / u1
    u0 = u1
    u1 = un
    print("%.10f" % un)
```

Рис. 3.1. Лістинг програмної реалізації послідовності Мюллера

Результати роботи програми для перших 30 значень послідовності відображені в Таблиці 3.1. в стовпці «Python».

Таблиця 3.1. Результати роботи програми з Рис. 3.1. для перших 30 значень послідовності Мюллера

n	Математичний результат		Python
0	2	2.0000000000	2.0000000000
1	-4	-4.0000000000	-4.0000000000
2	37/2	18.5000000000	18.5000000000
3	347 / 37	9.3783783784	9.3783783784
4	2707 / 347	7.8011527378	7.8011527378
5	19367 / 2707	7.1544144810	7.1544144810
6	131827 / 19367	6.8067847369	6.8067847369
7	869087 / 131827	6.5926327687	6.5926327687
8	5605147 / 869087	6.4494659338	6.4494659339
9	35584007 / 5605147	6.3484520567	6.3484520587
10	223269667 / 35584007	6.2744385982	6.2744386301
11	1388446127 / 223269667	6.2186957398	6.2186962479
12	8574817387 / 1388446127	6.1758373049	6.1758454797
13	52669607447 / 8574817387	6.1423590812	6.1424914958
14	322121160307 / 52669607447	6.1158830666	6.1180393880
15	1963244539967 / 322121160307	6.0947394393	6.1299926795
16	11932055130427 / 1963244539967	6.0777223048	6.6529208479
17	72355270235687 / 11932055130427	6.0639403225	14.7110136879
18	437946318679747 / 72355270235687	6.0527217610	64.8393305454
19	2646751398406607 / 437946318679747	6.0435521102	96.7174472768
20	15975875822080267 / 2646751398406607	6.0360318811	99.7948677633
21	96332092090684727 / 15975875822080267	6.0298473250	99.9875918848
22	580376738335123987 / 96332092090684727	6.0247496524	99.9992516762

23	3494181358965822047 / 580376738335123987	6.0205399841	99.9999549130
24	21024692798570322907 / 3494181358965822047	6.0170582573	99.9999972854
25	126446180015298890567 / 21024692798570322907	6.0141749146	99.9999998367
26	7601671962111781090 27 / 126446180015298890567	6.0117845879	99.9999999902
27	4568453757863992482287 / 76016719621117810902 7	6.0098012393	99.9999999994
28	27447975450168574034347 / 4568453757863992482287	6.0081543789	100.0000000000
29	164874117215934539909207 / 27447975450168574034347	6.0067860930	100.0000000000
30	9901760258702227179708 67 / 164874117215934539909207	6.0056486888	100.0000000000

Другий та третій стовпці Таблиці 3.1. демонструють контрольні результати математичних обчислень у вигляді звичайних дробів (третій стовпчик – їх раціональне наближення з прийнятою в програмі точністю в 10 знаків після коми).

Аналізуючи дані Таблиці 3.1, можна зробити висновок, що до 13 члену послідовності математичні та програмні результати добре співвідносяться, а починаючи з 14 члену поступово розбігаються в різні сторони: аналітичні результати при збільшені *n* будуть прямувати до значення 6, тоді як програмні обрахунки так само при збільшені *n* застигнуть на значенні 100.

Некоректність комп’ютерних обчислень вже не є для нас дивиною: перший розділ представленої роботи був цілком присвячений демонстрації різних класичних аспектів даної проблеми. Послідовність Мюллера являє собою своєрідний унікальний об’єкт, оскільки мають місце наступні оригінальні аспекти програмних обрахунків:

1) Зазначена послідовність (3.1) в залежності від початкових значень може як розбігатись, так і збігатись до одного з трьох значень: 5, 6 та 100. Тому некоректна збіжність (3.1) при заданих початкових значеннях не викличе підозру, і буде здаватись, що ми отримали цілком прийнятний результат.

2) Некоректна збіжність до числа 100 спостерігається не для будь-яких початкових значень, а лише за наявності конкретної залежності $u(1) = 11 - \frac{30}{u(0)}$ ($u(0) = 2,\ u(1) = -4$ якраз такий випадок).
3) Якби б ми зупинили наші програмні обчислення на 7-16 кроці, ми б отримали більш-менш коректний результат з відповідною відносною похибкою в найгіршому випадку біля 10 %. Це забагато, але значно краще, ніж ситуація, що розгортається при $n > 16$. Має місце цікавий парадокс обчислень: чим більше членів послідовності ми враховуємо, тим менш точний ми отримуємо результат. З одного боку – це класична проблема стійкої-нестійкою точки в процесі збіжності обчислень. Але з іншого боку природа цієї нестійкості поки що залишається невідомою.

Подібний випадок нестійкості комп'ютерних обчислень заслуговує додаткового математичного дослідження.

Щодо програмної реалізації, то Python дає нам можливість отримати коректний результат обчислення, навіть поки ще не розуміючи причино-наслідкових зав'язків означеної проблеми.

Аналогічно тому, як в розділі 1 для контролю точності ми підключали модуль *decimal*, так для задачі коректного обчислення значень рекурентної послідовності (3.1) нам потрібний функціонал модуля *fraction* [19] для роботи із раціональними числами.

```
from fractions import Fraction
u0 = Fraction(2)
u1 = Fraction(-4)
for n in range(2, 31):
    un = 111 - 1130 / u1 + 3000 / u0 / u1
    u0 = u1
    u1 = un
    print("%.10f" % un)
```

Рис. 3.2. Лістинг програмної реалізації послідовності Мюллера з використанням бібліотеки *Fractions*

В даному випадку отримані результати будуть співпадати із даними третього стовпця Таблиці 3.1.

На перспективу можна мати на увазі можливості сторонньої бібліотеки *mpmath* [20], що містить зокрема механізми роботи з інтервальною арифметикою. Але в конкретному випадку використання можливостей *mpmath* не дасть суттєвих результатів.

3.2. Математичне дослідження проблеми.

Проблемі парадоксу Мюллера присвячена досить обмежена кількість наукових джерел. Автор [21] зачіпає різні аспекти питання, але вичерпного дослідження причин обчислювальної нестійкості (3.1), та й подібних рекурентних послідовностей більш загального вигляду, автор не зустрічав. Спробуємо побудувати наочну обчислювальну схему, а не лише обмежуватись констатацією, що точка 100 для (3.1) є стійкою, а 6 – нестійкою.

В роботі пропонується алгебраїчний підхід до побудови замкнутого вигляду розв'язку послідовності (3.1).

Розглянемо вираз

$$\begin{cases} u(n+2) = A + \dfrac{B}{u(n+1)} + \dfrac{C}{u(n+1)\cdot u(n)}, \\ u(0) = x,\ u(1) = y. \end{cases} \tag{3.2}$$

Для зручності введемо до розгляду числа z_1, z_2, z_3, що задовольняють співвідношенням

$$\begin{aligned} A &= z_1 + z_2 + z_3 \\ B &= -z_1 z_2 - z_1 z_3 - z_2 z_3 \\ C &= z_1 z_2 z_3 \end{aligned} \tag{3.3}$$

Таким чином z_1, z_2, z_3 є коренями характеристичного рівняння

$$z^3 - Az^2 - Bz - C = 0.$$

Приклад (3.1) відповідає набору умов $z_1, z_2, z_3 \in \text{Ў}$ та $|z_1| < |z_2| < |z_3|$. В подальшому в роботі ми обмежимось розглядом лише вказаного випадку.

Пропускаючи деякі проміжні кроки, позначимо

$$Q_n = \frac{z_3^{n+2}}{(z_3-z_2)(z_3-z_1)(z_2-z_1)}\cdot$$

$$\cdot\left[x\cdot\left(y\cdot\left(z_2-z_1-\left(\frac{z_2}{z_3}\right)^{n+2}\cdot(z_3-z_1)+\left(\frac{z_1}{z_3}\right)^{n+2}\cdot(z_3-z_2)\right)-\right.\right.$$

$$\left.\left.-\left(z_2^2-z_1^2-\left(\frac{z_2}{z_3}\right)^{n+2}\cdot(z_3^2-z_1^2)+\left(\frac{z_1}{z_3}\right)^{n+2}\cdot(z_3^2-z_2^2)\right)\right)+\right.$$

$$\left.+z_1z_2\cdot(z_2-z_1)-\left(\frac{z_2}{z_3}\right)^{n+2}\cdot z_1z_3\cdot(z_3-z_1)+\left(\frac{z_1}{z_3}\right)^{n+2}\cdot z_2z_3\cdot(z_3-z_2)\right]$$

Припустимо далі, що розв'язок (3.2) у замкнутому вигляді має вид

$$\begin{cases} u(0)=x,\ u(1)=y, \\ u(k+2)=\dfrac{Q_{k+1}}{Q_k}, \qquad k=\overline{0,\infty} \end{cases} \tag{3.4}$$

Для зручності подальших перетворень, введемо наступні векторні позначення

$$\begin{cases} \overrightarrow{a^{(n)}}=\left(z_2-z_1;-(z_3-z_1)\cdot\left(\dfrac{z_2}{z_3}\right)^{n+2};(z_3-z_2)\cdot\left(\dfrac{z_1}{z_3}\right)^{n+2}\right), \\ \vec{\lambda}_1=(1;1;1),\ \overrightarrow{\lambda_2}=(z_2+z_1;z_3+z_1;z_3+z_2),\ \vec{\lambda}_3=(z_1z_2;z_1z_3;z_2z_3). \end{cases}$$

Тоді можемо записати

$$Q_n = \frac{z_3^{n+2}}{(z_3-z_2)(z_3-z_1)(z_2-z_1)}\cdot\left\langle \overrightarrow{a^{(n)}},\ x\cdot y\cdot\vec{\lambda}_1-x\cdot\overrightarrow{\lambda_2}+\vec{\lambda}_3\right\rangle, \tag{3.5}$$

де $\langle g\rangle$ розуміємо у сенсі скалярного добутку векторів.

Доведемо, що система (3.4) у позначеннях (3.5) є замкнутим розв'язкам (3.2) при вказаних вище обмеженнях. Доведення здійснімо методом математичної індукції.

Доведення.

Перевіримо, чи є справедливим (3.4) при $k=0$. Маємо наступний вираз

$$x_2=\frac{Q_1}{Q_0}=\frac{\dfrac{z_3^3}{(z_3-z_2)(z_3-z_1)(z_2-z_1)}\cdot\left(\overrightarrow{a^{(1)}},x\cdot y\cdot\vec{\lambda}_1-x\cdot\overrightarrow{\lambda_2}+\vec{\lambda}_3\right)}{\dfrac{z_3^2}{(z_3-z_2)(z_3-z_1)(z_2-z_1)}\cdot\left(\overrightarrow{a^{(0)}},x\cdot y\cdot\vec{\lambda}_1-x\cdot\overrightarrow{\lambda_2}+\vec{\lambda}_3\right)},$$

або після всіх необхідних перетворень можна записати

$$x_2 = z_3 \frac{\frac{x\cdot y\cdot\left(z_1^3z_3+z_1z_2^3+z_2z_3^3-z_1^3z_2-z_1z_3^3-z_2^3z_3\right)+x\cdot\left(z_1^3z_3^2+z_1^2z_2^3+z_2^2z_3^3-z_1^3z_2^2-z_1^2z_3^3-z_2^3z_3^2\right)+z_1z_2z_3\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)}{z_3^3}}{\frac{x\cdot y\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)}{z_3^2}}.$$

Розглянемо окремо вирази в душках в чисельнику і знаменнику.

$$\underline{z_1^3z_3+z_1z_2^3+z_2z_3^3-z_1^3z_2-z_1z_3^3-z_2^3z_3} = z_1\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)-$$

$$-z_1^2z_2^2-z_1z_2z_3^2+z_1^3z_2+z_1^2z_3^2+z_1z_2^2z_3+z_1z_2^3+z_2z_3^3-z_1^3z_2-z_1z_3^3-z_2^3z_3=$$

$$=z_1\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)+z_2\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)-$$

$$-z_1^2z_2z_3-z_2^2z_3^2+z_1^2z_2^2+z_1z_2z_3^2+z_2^3z_3-z_1^2z_2^2-z_1z_2z_3^2+z_1^2z_3^2+z_1z_2^2z_3+z_2z_3^3-z_1z_3^3-z_2^3z_3=$$

$$=z_1\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)+z_2\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)+$$

$$+z_3\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)=\underline{\left(z_1+z_2+z_3\right)\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)}.$$

Так само,

$$\underline{z_1^3z_3^2+z_1^2z_2^3+z_2^2z_3^3-z_1^3z_2^2-z_1^2z_3^3-z_2^3z_3^2}=$$

$$=z_1z_2\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)-$$

$$-z_1^3z_2z_3-z_1z_2^2z_3^2+z_1^2z_2z_3^2+z_1z_2^2z_3+z_1^3z_3^2+z_2^2z_3^3-z_1^2z_3^3-z_2^3z_3^2=$$

$$=z_1z_2\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)+z_1z_3\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)-$$

$$-z_1^2z_2^2z_3-z_1z_2z_3^3-z_1z_2^2z_3^2+z_1^2z_2z_3^2+z_1z_2^2z_3+z_2^2z_3^3-z_2^3z_3^2=$$

$$=z_1z_2\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)+z_1z_3\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)+$$

$$+z_2z_3\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)=$$

$$=\underline{\left(z_1z_2+z_1z_3+z_2z_3\right)\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)}$$

В кінці кінців

$$x_2 = z_3 \frac{\frac{\left(x\cdot y\cdot\left(z_1+z_2+z_3\right)+x\cdot\left(z_1z_2+z_1z_3+z_2z_3\right)+z_1z_2z_3\right)\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)}{z_3^3}}{\frac{x\cdot y\cdot\left(z_1^2z_3+z_1z_2^2+z_2z_3^2-z_1^2z_2-z_1z_3^2-z_2^2z_3\right)}{z_3^2}},$$

тобто

$$x_2 = \frac{x\cdot y\cdot\left(z_1+z_2+z_3\right)+x\cdot\left(z_1z_2+z_1z_3+z_2z_3\right)+z_1z_2z_3}{x\cdot y},$$

що відповідає дійсності.

Припустимо, для $k=n$ має місце $u(n+2)=\dfrac{Q_{n+1}}{Q_n}$.

Розглянемо випадок $k=n+1$, тобто вираз

$$u(n+3)=A+\frac{B}{u(n+2)}+\frac{C}{u(n+1)\cdot u(n+2)}=\frac{A\cdot u(n+1)\cdot u(n+2)+B\cdot u(n+1)+C}{u(n+1)\cdot u(n+2)}=$$

$$=\frac{A\cdot\dfrac{Q_n}{Q_{n-1}}\dfrac{Q_{n+1}}{Q_n}+B\cdot\dfrac{Q_n}{Q_{n-1}}+C}{\dfrac{Q_n}{Q_{n-1}}\dfrac{Q_{n+1}}{Q_n}}=\frac{A\cdot Q_{n+1}+B\cdot Q_n+C\cdot Q_{n-1}}{Q_{n+1}}.$$

Враховуючи позначення (3.3), можемо записати

$$A\cdot Q_{k+1}+B\cdot Q_k+C\cdot Q_{k-1}=(z_1+z_2+z_3)\cdot Q_{k+1}-(z_1z_2+z_1z_3+z_2z_3)\cdot Q_k+z_1z_2z_3\cdot Q_{k-1}=$$

$$=\frac{z_3^{k+2}}{(z_3-z_2)(z_3-z_1)(z_2-z_1)}\times\Big[z_3\cdot(z_1+z_2+z_3)\cdot\Big(\overrightarrow{a^{(k+1)}},x\cdot y\cdot\vec{\lambda}_1-x\cdot\vec{\lambda}_2+\vec{\lambda}_3\Big)-$$

$$-(z_1z_2+z_1z_3+z_2z_3)\cdot\Big(\overrightarrow{a^{(k)}},x\cdot y\cdot\vec{\lambda}_1-x\cdot\vec{\lambda}_2+\vec{\lambda}_3\Big)+z_1z_2\cdot\Big(\overrightarrow{a^{(k-1)}},x\cdot y\cdot\vec{\lambda}_1-x\cdot\vec{\lambda}_2+\vec{\lambda}_3\Big)\Big]=$$

$$=\frac{z_3^{k+2}}{(z_3-z_2)(z_3-z_1)(z_2-z_1)}\times$$

$$\times\Big(z_3\cdot(z_1+z_2+z_3)\cdot\overrightarrow{a^{(k+1)}}-(z_1z_2+z_1z_3+z_2z_3)\cdot\overrightarrow{a^{(k)}}+z_1z_2\cdot\overrightarrow{a^{(k-1)}},x\cdot y\cdot\vec{\lambda}_1-x\cdot\vec{\lambda}_2+\vec{\lambda}_3\Big)$$

Використавши тотожність

$$z_3\cdot(z_1+z_2+z_3)\cdot\left(\frac{z_i}{z_3}\right)^2-(z_1z_2+z_1z_3+z_2z_3)\cdot\left(\frac{z_i}{z_3}\right)+z_1z_2=\frac{z_i^3}{z_3},\ i=\overline{1,3}$$

отримаємо

$$Q_{k+2}=(z_1+z_2+z_3)\cdot Q_{k+1}-(z_1z_2+z_1z_3+z_2z_3)\cdot Q_k+z_1z_2z_3\cdot Q_{k-1}.$$

Таким чином, робимо висновок, що $u(n+3)=\dfrac{Q_{n+2}}{Q_{n+1}}$. Доведення завершено.

Ми підтвердили справедливість запропонованої нами формули (3.4)-(3.5). Тоді, оскільки вірним є твердження, що

$$\lim_{n\to\infty}\frac{Q_n}{z_3^{n+2}}=\frac{x\cdot y-x\cdot(z_2+z_1)+z_1z_2}{(z_3-z_2)(z_3-z_1)},$$

можемо записати, що

$$\lim_{n\to\infty}\frac{Q_{n+1}}{Q_n}=\lim_{n\to\infty}u\left(n+2\right)=z_3$$

за умови, що $y\neq\left(z_2+z_1\right)-\frac{z_1z_2}{x}$.

Ось ми і отримали шукане значення початкового значення $y=\left(z_2+z_1\right)-\frac{z_1z_2}{x}$, що порушувало коректність обчислень. Прямою підстановкою зазначеного значення y в Q_n отримаємо

$$u\left(n+2\right)=z_2\cdot\left(\frac{z_1}{z_2}+\frac{\left(\frac{z_1}{z_2}-1\right)\cdot\left(x-z_1\right)}{\left(\frac{z_1}{z_2}\right)^{n+1}\cdot\left(x-z_2\right)-x+z_1}\right),$$

тобто в свою чергу для випадку $y=\left(z_2+z_1\right)-\frac{z_1z_2}{x}$ матимемо $\lim_{n\to\infty}u\left(n+2\right)=z_2$.

Узагальнюючи, у випадку $z_1,z_2,z_3\in\text{Ў}$ та $\left|z_1\right|<\left|z_2\right|<\left|z_3\right|$ для (3.2) у разі збіжності можемо записати аналітичний розв'язок

$$\begin{cases}u\left(0\right)=u(1)=z_i, & u\left(n\right)\underset{n\to\infty}{\to}z_i\\ u(1)=z_1+z_2-\frac{z_1\cdot z_2}{u\left(0\right)}, & u\left(n\right)\underset{n\to\infty}{\to}z_2\\ u(0)=x,\ u(1)=y, & u\left(n\right)\underset{n\to\infty}{\to}z_3\end{cases}\qquad(3.6)$$

Повертаючись до конкретного прикладу (2.1), ми бачимо, що $z_1=5,\ z_2=6,\ z_3=100$ є коренями рівняння $z^3-111\cdot z^2+1130\cdot z-3000=0$ і початкові значення $u(0)=2,$ $u(1)=-4$ відповідає другому випадку (3.6), і послідовність $u(n)$ має збігатись до значення $z_2=6$. Але по факту, виходячи вже з відомих нам особливостей організації процесу комп'ютерних обчислень, ми розуміємо, що значення виразу $x\cdot y-x\cdot\left(z_2+z_1\right)+z_1z_2$ в силу принципової похибки обчислень не буде строго дорівнювати 0 (якщо обраховувати в форматі *float* для Python або ж *float* чи *double* для C++), і відповідно вироджується до третього випадку збіжності з

(3.6). Тому то ми і отримуємо в останньому стовпці таблиці 1 починаючи з двадцятого члена послідовності значення близькі до100.

Залишилось останнє математичне питання – аналіз умов розбіжності рекурентної послідовності (3.2) за прийнятих нами умов умов $z_1, z_2, z_3 \in \breve{У}$ та $|z_1| < |z_2| < |z_3|$.

3.3. Збіжність рекурентної послідовності Мюллера.

Розглянемо рівняння $x(n+2) = 0$, тобто

$$A + \frac{B}{x(n+1)} + \frac{C}{x(n+1) \cdot x(n)} = 0. \tag{3.7}$$

З (3.7) слідує, що якщо ввести позначення $B_1 = B/A$ та $C_1 = C/A$, отримаємо нелінійну рекурентну послідовність першого порядку

$$\begin{cases} x(n+1) = -B_1 - \dfrac{C_1}{x(n)}, \\ x(0) = x_0 \ (x_0 \neq 0). \end{cases} \tag{3.8}$$

Надалі будемо проводити дослідження (3.8) вже за відомою схемою. Розглянемо відповідне характеристичне рівняння – квадратне рівняння з коренями λ_1 та λ_2

$$\begin{cases} x^2 + B_1 \cdot x + C_1 = 0, \\ B_1 = -\lambda_1 - \lambda_2, \\ C_1 = \lambda_1 \cdot \lambda_2. \end{cases}$$

Розглянемо декілька перших членів рекурентної послідовності (3.8), групуючи в чисельнику та знаменнику відповідних дробів подібні доданки

$$x(1) = \frac{x_0 \cdot (\lambda_1 + \lambda_2) - \lambda_1 \cdot \lambda_2}{x_0},$$

$$x(2) = \frac{x_0 \cdot (\lambda_1^2 + \lambda_1 \cdot \lambda_2 + \lambda_2^2) - (\lambda_1^2 \cdot \lambda_2 + \lambda_1 \cdot \lambda_2^2)}{x_0 \cdot (\lambda_1 + \lambda_2) - \lambda_1 \cdot \lambda_2},$$

$$x(3) = \frac{x_0 \cdot (\lambda_1^3 + \lambda_1^2 \cdot \lambda_2 + \lambda_1 \cdot \lambda_2^2 + \lambda_2^3) - (\lambda_1^3 \cdot \lambda_2 + \lambda_1^2 \cdot \lambda_2^2 + \lambda_1 \cdot \lambda_2^3)}{x_0 \cdot (\lambda_1^2 + \lambda_1 \cdot \lambda_2 + \lambda_2^2) - (\lambda_1^2 \cdot \lambda_2 + \lambda_1 \cdot \lambda_2^2)},$$

$$x(4)=\frac{x_0\cdot\left(\lambda_1^4+\lambda_1^3\cdot\lambda_2+\lambda_1^2\cdot\lambda_2^2+\lambda_1\cdot\lambda_2^3+\lambda_2^4\right)-\left(\lambda_1^4\cdot\lambda_2+\lambda_1^3\cdot\lambda_2^2+\lambda_1^2\cdot\lambda_2^3+\lambda_1\cdot\lambda_2^4\right)}{x_0\cdot\left(\lambda_1^3+\lambda_1^2\cdot\lambda_2+\lambda_1\cdot\lambda_2^2+\lambda_2^3\right)-\left(\lambda_1^3\cdot\lambda_2+\lambda_1^2\cdot\lambda_2^2+\lambda_1\cdot\lambda_2^3\right)},$$

…

Винесемо в елементі послідовності $x(k)$ за душки в першому доданку в чисельнику член λ_2^k (і так само у знаменнику елемент λ_2^{k-1}), а відповідно у другому доданку винесемо за душки член $\lambda_1\cdot\lambda_2^k$. Тоді, наприклад, для $x(4)$ будемо мати

$$x(4)=\frac{x_0\cdot\lambda_2^4\cdot\left(\frac{\lambda_1^4}{\lambda_2^4}+\frac{\lambda_1^3}{\lambda_2^3}+\frac{\lambda_1^2}{\lambda_2^2}+\frac{\lambda_1}{\lambda_2}+1\right)-\lambda_1\cdot\lambda_2^4\cdot\left(\frac{\lambda_1^3}{\lambda_2^3}+\frac{\lambda_1^2}{\lambda_2^2}+\frac{\lambda_1}{\lambda_2}+1\right)}{x_0\cdot\lambda_2^3\cdot\left(\frac{\lambda_1^3}{\lambda_2^3}+\frac{\lambda_1^2}{\lambda_2^2}+\frac{\lambda_1}{\lambda_2}+1\right)-\lambda_1\cdot\lambda_2^3\cdot\left(\frac{\lambda_1^2}{\lambda_2^2}+\frac{\lambda_1}{\lambda_2}+1\right)}=$$

$$=\lambda_2\cdot\frac{x_0\cdot\left(\frac{\lambda_1^4}{\lambda_2^4}+\frac{\lambda_1^3}{\lambda_2^3}+\frac{\lambda_1^2}{\lambda_2^2}+\frac{\lambda_1}{\lambda_2}+1\right)-\lambda_1\cdot\left(\frac{\lambda_1^3}{\lambda_2^3}+\frac{\lambda_1^2}{\lambda_2^2}+\frac{\lambda_1}{\lambda_2}+1\right)}{x_0\cdot\left(\frac{\lambda_1^3}{\lambda_2^3}+\frac{\lambda_1^2}{\lambda_2^2}+\frac{\lambda_1}{\lambda_2}+1\right)-\lambda_1\cdot\left(\frac{\lambda_1^2}{\lambda_2^2}+\frac{\lambda_1}{\lambda_2}+1\right)}.$$

Поки що ми не накладали жодних обмежень на значення коренів характеристичного рівняння λ_1 та λ_2, але очевидно можливі два варіанти: $\lambda_1\neq\lambda_2$ або $\lambda_1=\lambda_2$.

Спочатку розглянемо випадок $\lambda_1\neq\lambda_2$. Також будемо додатково вважати, що $|\lambda_1|<|\lambda_2|$.

Позначимо

$$P_k(\lambda_1,\lambda_2)=\frac{\lambda_2}{\lambda_2-\lambda_1}\cdot\left(-x_0\cdot\left(\left(\frac{\lambda_1}{\lambda_2}\right)^{k+1}-1\right)+\lambda_1\cdot\left(\left(\frac{\lambda_1}{\lambda_2}\right)^{k}-1\right)\right). \qquad (3.9)$$

Припустимо, що враховуючи (3.9), *k* член рекурентної послідовності (3.8) можна записати

$$x(k)=\frac{\lambda_2\cdot P_k(\lambda_1,\lambda_2)}{P_{k-1}(\lambda_1,\lambda_2)}, \qquad (3.10)$$

Або в явному вигляді

$$x(k)=\lambda_2\cdot\frac{-x_0\cdot\left(\left(\frac{\lambda_1}{\lambda_2}\right)^{k+1}-1\right)+\lambda_1\cdot\left(\left(\frac{\lambda_1}{\lambda_2}\right)^{k}-1\right)}{-x_0\cdot\left(\left(\frac{\lambda_1}{\lambda_2}\right)^{k}-1\right)+\lambda_1\cdot\left(\left(\frac{\lambda_1}{\lambda_2}\right)^{k-1}-1\right)}.$$

Після спрощення

$$x(k)=\lambda_1+\frac{\lambda_2-\lambda_1}{1-\left(\frac{\lambda_1}{\lambda_2}\right)^{k}\cdot\frac{x_0-\lambda_2}{x_0-\lambda_1}}. \qquad (3.11)$$

З виразу (3.11) слідує, що ми маємо ще окремо розглянути два випадки, початкового значення x_0, коли $x_0=\lambda_1$ або

$$x_0=\lambda_2\cdot\frac{1-\left(\frac{\lambda_2}{\lambda_1}\right)^{k-1}}{1-\left(\frac{\lambda_2}{\lambda_1}\right)^{k}}..$$

Пряма підстановка в (3.8) показує, що при $x_0=\lambda_1$ значення $x(k)$ стає дорівнювати λ_1 для будь-якого *k*. А за початкового значення

$$x_0=\lambda_2\cdot\frac{1-\left(\frac{\lambda_2}{\lambda_1}\right)^{k-1}}{1-\left(\frac{\lambda_2}{\lambda_1}\right)^{k}}$$

при $k\in\mathrm{N}$ виникне проблема ділення на нуль на *k* кроці, оскільки $x(k-1)=0$.

Вираз (3.11) є розв'язком рекурентного рівняння (3.8) за умови $\lambda_1\neq\lambda_2$. Доведення цього факту здійснюється методом математичної індукції, як це було продемонстровано у розділі 3.2.

У випадку $\lambda_1=\lambda_2$ розв'язком (2.11) очевидно буде вираз

$$x(k)=\lambda_2\cdot\left(1+\frac{x_0-\lambda_2}{x_0\cdot k-\lambda_2\cdot(k-1)}\right). \qquad (3.12)$$

Доведення коректності формули (3.12) здійснюється аналогічно розглянутим раніше випадкам. Якщо при цьому $x_0 = \lambda_2 \cdot \left(1 - \frac{1}{k}\right)$ також при натуральному $k > 1$ виникне проблема ділення на нуль на k кроці, оскільки $x(k-1) = 0$.

Залишається лише зазначити, що λ_1 та λ_2 обраховуються відносно значень $B_1 = B/A$ та $C_1 = C/A$, при цьому A, B, C визначаються відповідно до формул (3.3). Для подальших досліджень може виявитись доречним виразити значення λ_1 та λ_2 через z_1, z_2, z_3.

Висновки до розділу 3

У цьому розділі була досліджена рекурентна послідовність Мюллера другого порядку. За розглянутим вище, можна бути впевненим, що хоч досліджувана послідовність збігається при комп'ютерних обчисленнях членів послідовності виникає суттєва похибка.

В результаті досліджень, було побудоване математичне обґрунтування проблеми нестійких початкових значень та виведена формула для перевірки початкових значень для розглянутої послідовності.

Також було доведено збіжність рекурентної послідовності Мюллера за допомогою методі математичної індукції.

ВИСНОВКИ

В роботі були представлені результати дослідження автора, пов’язані із аналізом проблематики достовірності комп’ютерних обчислень, їх стійкості та збіжності. Були продемонстровані відповідні приклади некоректної роботи програм: як зовсім прості, так і значно менш очевидні, як то приклад Рампа та послідовність Мюллера.

Для послідовності Мюллера автором вперше було побудоване математичне обґрунтування проблеми нестійких початкових значень, коли ми наперед знаємо, до якого числа має збігатись значення членів послідовності, але стандартними методами комп’ютерних обчислень ми завжди в цьому випадку отримуємо некоректний результат.

Окрім математичних викладок, автор акцентував увагу на суто програмних можливостях контролю за точністю складних обчислень. З цією метою були наведені приклади ефективного використання функціоналу модулів *decimal* та *fraction* в Python 3.x, що допомагають певною мірою зменшити проблему коректності програмних обрахунків. Зазначимо, що обмежений обсяг роботи не дав змоги розглянути додатково спеціальні модулі для проведення складних обрахунків в Python *numpy* та *mpmath*, функціонал яких також заслуговує на увагу в контексті означеної проблеми.

В перспективі, автор вважає доречним продовжити дослідження особливостей стійкості обчислень послідовності Мюллера у випадку однакових та комплексних коренів характеристичного рівняння, більше уваги приділивши проблемі розбіжності рекурентної послідовності.

Також в майбутньому актуальним було б узагальнення результатів дослідження на нелінійні рекурентної послідовності вищих порядків. Також цілком ймовірним є те, що автор зможе віднайти власний приклад парадоксальної розбіжності результатів отриманих аналітично та за допомогою комп’ютерних обчислень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kopec D. Classic Computer Science in Python. Shelter Island: Manning Publications, 2019. 224 p.
2. Kopec D. Classic Computer Science in Java. Shelter Island: Manning Publications, 2020. 264 p.
3. Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia: IMTEC-92-26, 1992. URL: https://www.gao.gov/assets/220/215614.pdf
4. Rump S. M. Algorithms for Verified Inclusions: Theory and Practice. In: *Reliability in Computing – The Role of Interval Methods in Scientific Computing.* New York, 1988. P. 109-126.
5. Muller J.-M., Brisebarre N., de Dinechin F., Jeannerod C.-P. et al. Handbook of Floating-Point Arithmetic. Basel: Birkhäuser, 2018. 627 p.
6. Андерсон Дж. А., Дискретная математика и комбінаторика. Москау: Вильямс, 2004. 960 с.
7. Andrica D., Bagdasar O. Recurrent Sequences: Key Results, Applications, and Problems. New York: Springer, 2020. 410 p.
8. Greene, D. H., Knuth D. E. Mathematics for the Analysis of Algorithms. 3rd edition. Boston – Basel – Berlin: Birkhauser, 1990. 133 p.
9. Hengkrawita C., Laohakosolb V., Pimsert W. Solutions of a class of nonlinear recursive equations and applications. *Science Asia*. 2011. № 37. P. 136-144.
10. Rabinovich S., Berkolaiko G., Havlin S. Solving nonlinear recursions. *Journal of Mathematical Physics*.1996. Vol. 37, №. 11. P. 5828-5836.
11. Білоус Р. В. Дослідження стійкості комп'ютерних обчислень деяких рекурентних послідовностей другого порядку. *Конференція молодих учених «Підстригачівські читання – 2020»* : тези конференції, м. Львів, 26–28 травня 2020 р. Львів. 2020. URL: http://www.iapmm.lviv.ua/chyt2020/abstracts/Bilous2.pdf

12. Bilous R.V., Varer B.Y. Some problems of difference equations computer calculations. *6th Ya. B. Lopatynsky International School-Workshop on Differential Equations and Applications* : Book of Abstracts, Vinnytsia, 18-20 June 2019. Vinnytsia, 2019. P. 24-25.
13. Білоус Р. В., Варер Б. Ю. Точність та принципові обмеження комп'ютерних обчислень засобами мов програмування. *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення*: тези Міжнародної наукової інтернет-конференції, 2 квітня 2019 року. Вип. 37. 2019. *URL*: http://www.konferenciaonline.org.ua/arhiv-konferenciy/arhiv-konferenciy02-04-2019
14. Білоус Р.В., Вєтров О.С. Проблема коректності комп'ютерних обчислень при підготовці майбутніх фахівців галузі ІТ. *Heritage of european science: engineering and technology, informatics, security, transport, architecture.* Monographic series «European Science». Book 2. Part 3. Karlsruhe: SWorld-NetAkhatAV, 2020. С. 81-96.
15. Floating Point Math. URL: https://0.30000000000000004.com/
16. IEEE Standard for Floating-Point Arithmetic https://irem.univ-reunion.fr/IMG/pdf/ieee-754-2008.pdf
17. Singh S. The Simpsons and Their Mathematical Secrets. Bloomsbury, 2013. 272 p.
18. Бібліотека *Decimal* для мови програмування *Phyton3*. URL: https://docs.python.org/3/library/decimal.html
19. Бібліотека *Fractions* для мови програмування *Phyton3*. URL: https://docs.python.org/3/library/fractions.html
20. Бібліотека *MpMath* для мови програмування *Phyton3*. URL: https://mpmath.org/
21. Kahan W. How Futile are Mindless Assessments of Roundoff in Floating-Point Computation? 2006. 56 p. URL: http://http.cs.berkeley.edu/~wkahan/Mindless.pdf
22. Bilous R. V., Krykun I. H. Study of the convergence of computer calculations. *The 29th Conference on Applied and Industrial Mathematics dedicated to the*

memory of Academician Mitrofan M. Choban: abstracts of conference, 25-28 August 2022, Chisinau, Moldova. P. 169-170.

23. Bilous R. V., Krykun I. H. A Review of Problems of Accuracy of Computer Calculations. *International Journal of Mathematical, Engineering, Biological and Applied Computing,* 2022, Vol. 1, № 2. P. 150-158.

ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Білоус Ростислав Віталійович
Прізвище, ім'я, по-батькові

Інформаційних та прикладних технологій
Факультет

113 Прикладна математика
Шифр та назва спеціальності

Прикладна математика
Освітня програма

ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (магістерська) робота на тему:

«ДОСЛІДЖЕННЯ ТОЧНОСТІ ОБЧИСЛЕНЬ З ПЛАВАЮЧОЮ КОМОЮ ДЛЯ СУЧАСНИХ АПАРАТНИХ ПЛАТФОРМ»

є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних прав, закріплених статтями 21–25 Закону України «Про авторське право та суміжні права»;
- не отримувалась іншими особами, а також дані та інформація не отримувались у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна робота буде відхилена без права захисту або під час захисту за неї буде поставлена оцінка «незадовільно».

_____ (дата)

_____ (підпис здобувача освіти)