

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кваліфікаційна наукова
праця на правах рукопису

ГОРЯЧИЙ ОЛЕГ ЯРОСЛАВОВИЧ

УДК 004.051:004.021:004.222.3:004.382.7+004.056.5

ДИСЕРТАЦІЯ

**РОЗРОБЛЕННЯ ГЕНЕРАТОРІВ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ
НА ОСНОВІ ПОКРАЩЕНИХ МЕТОДІВ ОБЧИСЛЕННЯ ЕЛЕМЕНТАРНИХ
ФУНКЦІЙ ДЛЯ ЗАДАЧ КІБЕРБЕЗПЕКИ**

125 – Кібербезпека та захист інформації

12 – Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

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

_____ О. Я. Горячий

Науковий керівник:

Журавель Ігор Михайлович
д.т.н., ст.наук.співробітник

Львів – 2025

АНОТАЦІЯ

Горячий О. Я. Розроблення генераторів псевдовипадкових чисел на основі покращених методів обчислення елементарних функцій для задач кібербезпеки. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 125 «Кібербезпека та захист інформації» (12 – Інформаційні технології). – Національний університет «Львівська політехніка» МОН України, Львів, 2025.

Дисертаційна робота присвячена розв’язанню важливого наукового завдання – розробці та дослідженню генераторів псевдовипадкових послідовностей (ГПВП) для кібербезпеки (КБ) та захисту інформації (ЗІ), побудованих на основі вдосконалених методів наближеного обчислення елементарних функцій в арифметиці з рухомою комою (FP) і таких, що задовольняють вимогам статистичних тестів NIST.

У дисертації розглядаються універсальні програмні та програмно-апаратні способи обчислення ділення (DIV), оберненої до аргументу функції (RCP), квадратного (SQRT) та зворотного квадратного кореня (RSQRT), кубічного (CBRT) та зворотного кубічного кореня (RCBRT) тощо, а також їх модифікацій, що базуються на комбінації наступних методів: алгоритму швидкого зворотного квадратного кореня (FISR); ітераційних методах другого та вищих порядків; методу мінімаксної поліноміальної апроксимації; швидких наближених апаратних інструкцій, доступних на процесорах Intel та ARM. Алгоритм FISR набув популярності завдяки використанню в комп’ютерній грі Quake III Arena; застосування спеціальної магічної константи (MC) у ньому забезпечує швидку апроксимацію RSQRT для FP чисел. Основна ідея роботи полягає в проектуванні покращених методів, зокрема на основі алгоритму FISR, та дослідженні їх використання для генерації псевдовипадкових чисел (ПВЧ) та послідовностей (ПВП), придатних для різноманітних задач у сфері інформаційної безпеки (ІБ). Вони можуть бути реалізовані на будь-якій платформі, що підтримує FP обчислення, оскільки є досить гнучкими та простими. Проте, будуть особливо

корисними та ефективними під час проєктування новітніх інформаційних систем, технологій та комплексів створення, обробки, передачі, зберігання, захисту інформації, а також автоматизованих систем управління інформаційною та кібербезпекою для платформ, що не мають відповідних апаратних інструкцій, зокрема деяких мікроконтролерів (МК).

Існує безліч прикладних задач, в тому числі в галузі інформаційної безпеки, вирішення яких потребує швидких і/або точних методів DIV, RCP, SQRT чи RSQRT, наприклад в стеганографії, криптографії, нейронних мережах, статистиці, системах виявлення вторгнень та аномалій тощо. Більшість елементарних функцій з математичних бібліотек не забезпечують повної точності, крім того вони можуть бути погано оптимізованими та мати низьку швидкодію, а тому актуальною задачею є розроблення нових та вдосконалення існуючих способів обчислення елементарних функцій, що дозволяють отримати як грубе наближення функції, так і результат необхідної точності. Генератори ПВЧ (ГПВЧ) мають безліч як криптографічних, так і некриптографічних застосувань, зокрема для потокового шифрування, стеганографії, протоколів автентифікації та захищеного багатокористувацького обчислення, моделювання та симуляції, комп'ютерних ігор тощо. Відомі алгоритми ГПВЧ можуть мати статистичні недоліки, криптографічні вразливості, а також, як показано на прикладі *Dual_EC_DRBG*, криптографічні бекдори. Крім того, часто зустрічається поєднання інтенсивних обчислень розглянутих елементарних функцій та генерації ПВЧ. Це особливо важливо в контексті функціонування та безпеки обмежених пристроїв, наприклад МК, що опрацьовують або передають великі масиви даних через безпроводний незахищений канал зв'язку. Якщо такі пристрої не мають надійних джерел ентропії та механізмів генерації та оновлення ключів, вони будуть вразливими до різноманітних атак.

Існує тривала дискусія про випадковість, незалежність та розподіл цифр відомих ірраціональних математичних констант, квадратних коренів простих чисел, наприклад $\sqrt{2}$, а також деяких раціональних чисел із великим простим знаменником. На основі цієї властивості можна будувати прості ГПВЧ,

використовуючи обчислення з довільною точністю. Проте, такі методи є досить повільними, мають низьку збіжність та обмежену апаратну підтримку. Запропонований в цій дисертації ГПВЧ використовує альтернативний підхід – швидку апроксимацію нелінійних елементарних функцій для рекурентної послідовності аргументів, що містить прості FP операції подвійної точності.

Об'єктом дослідження є процеси обчислення елементарних функцій та генерації ПВП в FP арифметиці. **Предметом дослідження** є методи підвищення точності та швидкодії алгоритмів апроксимації таких функцій як RCP, DIV, RSQRT та SQRT в FP арифметиці, методи генерації ПВЧ на їх основі, а також оцінювання їх якості для застосувань у сфері інформаційної та кібербезпеки.

В процесі дослідження використано методи огляду літератури, аналізу та синтезу, порівняння, імітаційного моделювання, елементи теорій чисельних методів, програмування, ймовірностей і математичної статистики, оптимізації, оцінки збіжності, похибок, вимірювання, експерименту та автоматизації.

У першому розділі «**Огляд відомих методів обчислення елементарних функцій та генерації псевдовипадкових послідовностей**» проведено огляд вітчизняної та зарубіжної наукової літератури за тематикою дослідження; здійснено аналіз, класифікацію та порівняння відомих методів, визначено способи підвищення точності та швидкодії алгоритмів; а також здійснено аналіз методів оцінки якості генераторів ПВП; сформульовано основні вимоги до проектування нових та розробки вдосконалених алгоритмів для задач інформаційної та кібербезпеки. Досліджено теоретичні основи класичного алгоритму FISIR, його модифікації та узагальнення, а також модифікації методу Хаусхолдера, що дозволяє зокрема для функції RSQRT уникнути операції ділення. Здійснено порівняння обчислювальної складності та практичних похибок відомих методів обчислення експоненційних, логарифмічних та степеневих функцій на основі методу MC, зокрема RSQRT, RCP, RCBRT, SQRT та CBRT для типів *float* і *double*. Проведено також аналіз відомих способів генерації та оцінювання якості ПВП, зокрема для криптографічних та некриптографічних цілей. Досліджено застосування чисельних методів та FP арифметики для задач кібербезпеки,

зокрема для генерації ПВП. Встановлено, що основними недоліками відомих методів є їхні статистичні та криптографічні вразливості, а також низька швидкодія. Для подальших досліджень обрано статистичні тести NIST STS, оскільки вони призначені для перевірки випадковості генераторів ПВЧ та істинно випадкових послідовностей (ІВП) для криптографічних систем.

У другому розділі **«Вибір методів і розроблення методики дослідження»** розроблено основну методику проєктування вдосконалених алгоритмів обчислення елементарних функцій в FP арифметиці, зокрема використання імітаційного моделювання, модифікація алгоритмів для обчислення інших функцій, застосування спеціальної форми запису з використанням FMA та модифікація параметрів з метою мінімізації максимальних відносних похибок; сформульовано методи оцінки похибок та точності, вимірювання швидкодії та аналізу збіжності; розглянуто методи проєктування та дослідження ГПВЧ на основі FP арифметики для застосувань у сфері кібербезпеки та захисту інформації, зокрема методику дослідження статистичних характеристик, швидкодії та періоду повторення сформованих ПВП. Імітаційне моделювання включає аналітичну модель методу MC, полінома мінімаксної апроксимації методом Ремеза, запропонованої математичної моделі спеціальних апаратних інструкцій для ARM та Intel, а також комбінацій модифікованих ітераційних методів. Для визначення оптимальних значень коефіцієнтів розроблено набір алгоритмів рандомізованої чисельної багатовимірної оптимізації. Запропоновано новий підхід до генерації ПВП, що використовує рекурентне рівняння та чисельні методи в FP арифметиці. Розроблено методи автоматизації їх статистичного тестування для задач кібербезпеки на основі тестів NIST та визначення періоду повторення для різних початкових значень та параметрів генерації ПВП.

У третьому розділі **«Проектування генераторів псевдовипадкових чисел на основі вдосконалених алгоритмів обчислення елементарних функцій»** представлено результати розробки модифікованих алгоритмів обчислення елементарних функцій та проєктування ГПВЧ на їх основі, актуальних для застосувань у сфері кібербезпеки та захисту інформації. Запропоновані ефективні

алгоритми забезпечують підвищену точність і швидкодію завдяки використанню методу MC, ітераційних методів Ньютона-Рафсона (NR) та Хаусхолдера (H) вищих порядків в спеціальних формах, а також адаптивному вибору параметрів залежно від вхідних даних з метою мінімізації відносної похибки. Вперше запропоновано метод швидкого обчислення функції RCP на основі тотожності з поліномами найкращого рівномірного наближення 3-го порядку, методи на основі модифікованої ітерації Хаусхолдера 2-го порядку (Ho2) для RCP та RSQRT, методи додаткових магічних констант (2MC та 3MC) для зменшення кількості FP множень для функцій RSQRT та RCP, методи переключення магічних констант (DC та 8DC) для обчислення RSQRT та SQRT на основі розбиття інтервалу визначення функції та оптимізації параметрів. Ці методи не використовують складної операції ділення та громіздких LUT. На їх основі реалізовано набір покращених алгоритмів обчислення функцій RCP, DIV, RSQRT та SQRT для застосувань у сфері КБ та ЗІ із підвищеною точністю (на 2-14 бітів вища точність) і/або швидкодією (до 5 множень менше) для FP чисел одинарної та подвійної точності. Запропоновані алгоритми досягають похибки на рівні 1 ULP, а для функції RCP забезпечується повна точність. Реалізовано також набір покращених простих алгоритмів обчислення степеневих, логарифмічних та показникових функцій. Запропонований у цьому розділі новий спосіб генерації ПБЧ базується на використанні швидких чисельних методів апроксимації функцій, зокрема RCP та RSQRT, для FP чисел типу *double*. В залежності від початкового значення та обраних параметрів ГПБЧ формується рекурентна послідовність 64-бітових FP аргументів та генерується псевдовипадкова байтова послідовність.

У четвертому розділі «**Практичне тестування та аналіз ефективності запропонованих алгоритмів та генераторів псевдовипадкових чисел на ПК, міні-комп'ютері та МК для задач у сфері кібербезпеки та захисту інформації**» здійснено практичне тестування та дослідження ефективності для задач інформаційної та кібербезпеки запропонованих та відомих алгоритмів на різноманітних пристроях, що підтримують FP обчислення, зокрема на Intel, Raspberry Pi та ESP-32. Окрім програмних методів, на прикладі функцій RCP та

RSQRT для чисел типу *float* та *double*, на персональних комп'ютерах (ПК), міні-комп'ютері та МК було проаналізовано також апаратні та програмно-апаратні реалізації із використанням процесорів Intel Core i5/i7 та ARM Cortex-A53. Наведено результати дослідження та тестування запропонованих ГПВЧ для задач кібербезпеки та захисту інформації, а саме: статистичних характеристик з використанням тестів NIST, графічних тестів та ентропії; періоду повторення; обчислювальної складності; швидкодії; параметрів генерації та діапазонів початкових значень. Дослідження виконувалося у три етапи; порівнювались найкращі запропоновані ГПВЧ з стандартними методами генерації ПВЧ, зокрема з бібліотеки *random*, та апаратними генераторами ІВП на Intel та ESP-32. Результати тестування показали, що деякі варіанти запропонованих алгоритмів, зокрема *rsqrt2dc_everyBit* (3/0), *rcp3_everyBit* (3/0), *rsqrt1dc8_everyBit* (3/0) та *x_xored* (2/1) є трохи швидшими за найшвидші ГПВЧ *random_device* та *mt19937* на Intel, а також апаратні генератори ІВП RDSEED та RDRAND; при цьому не вимагають значних обсягів пам'яті та тривалих термінів реініціалізації. Було виявлено, що ГПВЧ *rsqrt2dc_everyBit* (3/0) не проходить графічні тести, а швидкий генератор *x_xored* (2/1) має суттєві статистичні недоліки. Найкращі характеристики серед досліджених мають ГПВЧ *rsqrt2dc_everyBit* (1/0) та *rcp3_everyBit* (3/0), доля проходження тестів NIST яких складає відповідно 79% та 75%. Зазначено сфери застосування та перспективні напрямки подальших досліджень, зокрема для задач кібербезпеки та захисту інформації. Детально проаналізовано два приклади практичного застосування розроблених ГПВЧ. В тому числі, ГПВЧ *rsqrt2dc_everyBit* (1/0) був використаний під час проектування вдосконаленої моделі стеганосистеми для цифрових зображень, що здійснює аналіз ділянок вбудовування з використанням псевдовипадкової бітової маски, мінімізує довжину стеганографічного ключа, а також виконує шифрування даних.

У **висновках** дисертаційної роботи узагальнено основні наукові та практичні результати, наведено порівняння ефективності методів, окреслено сфери їх застосування для кібербезпеки та захисту інформації, а також сформульовано рекомендації щодо впровадження та подальших досліджень.

У додатках до дисертації долучено акти впровадження; програмні коди основних розроблених алгоритмів та інструментів дослідження, зокрема запропонованих методів рандомізованої чисельної багатовимірної оптимізації та скриптів для автоматизації процесу дослідження ГПВЧ на основі статистичних тестів NIST, що можуть безпосередньо застосовуватись іншими дослідниками; результати порівняння відомих та розроблених алгоритмів із зазначенням їх параметрів; програмні коди алгоритмів, використаних при проєктуванні запропонованих ГПВЧ; результати їх тестування, дослідження та порівняння на різних платформах; а також подано список наукових публікацій автора.

Таким чином, в роботі вперше запропоновано набір покращених методів для швидкої апроксимації функцій RCP, DIV, RSQRT, SQRT, RCBRT, CBRT та ін., в тому числі без операції ділення, на основі методів MC, кусково-поліноміальної апроксимації та наближених апаратних інструкцій Intel і ARM з підвищеною точністю, що завдяки оптимізації дозволяють отримати якісний компроміс між точністю, швидкодією та використаними апаратними ресурсами для задач кібербезпеки та захисту інформації. Вперше запропоновано новий клас генераторів ПВЧ на основі рекурентного рівняння та досліджуваних чисельних методів обчислення елементарних функцій в FP арифметиці, що мають задовільні статистичні характеристики для сфери кібербезпеки та захисту інформації за результатами тестів NIST STS. На основі запропонованих методів реалізовано набір покращених алгоритмів для чисел типу *float* та *double*, на основі яких можна розробити спеціалізовану кросплатформенну бібліотеку, що підтримує ефективні обчислення із різною точністю (для грубої апроксимації, майже з повною точністю, наприклад 1 ULP, та з коректним заокругленням) для визначених форматів з рухомою комою, а також бібліотеку легких та швидких генераторів псевдовипадкових чисел для задач кібербезпеки, придатних для МК та ПЛІС.

Ключові слова: генератор псевдовипадкових чисел, елементарні функції, чисельні методи, числа з рухомою комою, кібербезпека, інформаційна безпека, алгоритм FISR, програмна імплементація, стеганографія, криптографія, точність, швидкодія, мінімізація відносних похибок, шифрування, статистичні тести NIST.

ANNOTATION

Horyachyy O. Y. Design of pseudorandom number generators based on improved methods for elementary function computation for cybersecurity applications. – Qualification scientific work on the rights of the manuscript.

A dissertation submitted in fulfilment of the Ph.D. degree in speciality 125 «Cybersecurity and Information Protection» (12 – Information Technologies). – Lviv Polytechnic National University of Ministry of Education and Science of Ukraine, Lviv, 2025.

The dissertation addresses a significant scientific problem – the design and analysis of Pseudorandom Sequence Generators (PRSGs) for cybersecurity and information protection applications, based on enhanced methods for approximate evaluation of elementary functions in floating-point (FP) arithmetic, which satisfy the requirements of NIST statistical tests.

The dissertation explores universal software and hardware-oriented approaches for computing division (DIV), reciprocal (RCP), square root (SQRT) and reciprocal square root (RSQRT), cube root (CBRT) and reciprocal cube root (RCBRT), among others, as well as their modifications based on a combination of the following approaches: the Fast Inverse Square Root (FISR) algorithm; second and higher order iterative methods; minimax polynomial approximation; and fast approximate hardware instructions available on Intel and ARM processors. The FISR algorithm gained popularity due to its use in the Quake III Arena video game, where the application of a special magic constant (MC) enables fast RSQRT approximation for FP numbers. The work aims to develop enhanced methods, e.g., based on the FISR algorithm, and the investigation of their use in pseudorandom number (PRN) and pseudorandom sequence (PRS) generation suitable for various tasks in the field of information security. These methods are flexible and simple to implement on any FP-capable platform, and are particularly effective for designing next-generation information systems and technologies for data creation, processing, transmission, storage, and protection, as well as automated information and cybersecurity management systems, especially on platforms without dedicated hardware support, such as some microcontrollers (MCUs).

Numerous applied problems, including those in the field of information security, demand fast and/or accurate methods for DIV, RCP, SQRT, or RSQRT computations – e.g., in steganography, cryptography, neural networks, statistics, intrusion and anomaly detection systems, and others. Most elementary functions in mathematical libraries lack full accuracy; in addition, such implementations may be poorly optimized and have low performance. Therefore, developing new and enhancing existing methods for computing elementary functions that support both rough approximations and results of required accuracy remains a relevant task. Pseudorandom number generators (PRNGs) have numerous cryptographic and non-cryptographic applications, including stream ciphers, steganography, authentication protocols, secure multiparty computation, modeling and simulation, computer games, and more. Well-known PRNG algorithms may exhibit statistical weaknesses, cryptographic vulnerabilities, and, as evidenced by *Dual_EC_DRBG*, kleptographic backdoors. Moreover, intensive computations involving the aforementioned elementary functions are often combined with PRN generation. This is particularly relevant for the operation and security of resource-constrained devices, such as MCUs, that process or transmit large volumes of data over unsecured wireless channels. Without reliable sources of entropy and mechanisms for key generation and renewal, such devices become vulnerable to various attacks.

There has been an ongoing debate about the randomness, independence, and distribution of the digits of well-known irrational mathematical constants, square roots of prime numbers e.g., $\sqrt{2}$, and certain rational numbers with large prime denominators. Based on this property, simple PRNGs can be constructed using arbitrary precision computations. However, such methods are quite slow, have low convergence, and limited hardware support. The PRNG proposed in this dissertation uses an alternative approach – fast approximation of nonlinear elementary functions for a recurrent sequence of arguments using double-precision FP operations.

The **object of the research** is the processes of computing elementary functions and generating PRN in FP arithmetic. The **subject of the research** is methods for improving the accuracy and performance of approximation algorithms for functions such as RCP, DIV, RSQRT, and SQRT in FP arithmetic, methods for generating PRNs

based on these functions, as well as methods for evaluating their quality for applications in information and cybersecurity.

The research employed methods of literature review, analysis and synthesis, comparison, simulation modeling, elements of numerical methods, programming, probability theory, mathematical statistics, optimization, convergence and error analysis, measurement, experimentation, and automation.

In the first chapter, **“Review of Known Methods for Elementary Function Computation and Pseudorandom Sequence Generation,”** a review of domestic and foreign scientific literature on the research topic is conducted; an analysis, classification, and comparison of known methods are performed, methods for improving the accuracy and performance of algorithms are determined; PRSG assessment methods were analyzed; the main requirements for the design of new and enhanced algorithms for cybersecurity and information protection tasks are formulated. Theoretical foundations of the classical FISR algorithm, its modifications and generalizations, as well as adaptations of the Householder method – particularly those that eliminate division in RSQRT computation – have been investigated. A comparison was made of the computational complexity and empirical errors of the methods for calculating exponential, logarithmic, and power functions using the MC, including RSQRT, RCP, RCBRT, SQRT, and CBRT for both *float* and *double* types. Known methods for PRSG construction and quality evaluation – both cryptographic and non-cryptographic – were also analyzed. Numerical methods and FP arithmetic were also investigated for their application in cybersecurity, particularly for PRSG implementations. The known methods have been found to suffer from statistical and cryptographic vulnerabilities, as well as low performance. For further analysis, the NIST statistical tests were selected, as they are designed to assess the randomness of generators of PRNs and true random sequences (TRS) used in cryptographic systems.

In the second chapter, **“Selection of Methods and Development of Research Methodology,”** the core methodology for designing improved algorithms for computing elementary functions in FP arithmetic is developed. This includes the use of simulation modeling, modification of algorithms for other functions, application of

FMA-based forms of calculation, and parameter tuning to minimize maximum relative errors. Methods for error and accuracy estimation, performance measurement, and convergence analysis are defined. The chapter also addresses the design and analysis of PRNGs based on FP arithmetic for tasks in cybersecurity and information protection, including the methodology for evaluating statistical properties, performance, and periods of the generated PRS. The simulation modeling includes an analytical model of the MC method, the minimax polynomial approximation via the Remez algorithm, the proposed mathematical models of special ARM and Intel hardware instructions, as well as combinations of modified iterative methods. A set of randomized numerical multidimensional optimization techniques has been developed to determine the optimal coefficient values. A novel approach to PRS generation is proposed, utilizing a recurrence relation and numerical methods in FP arithmetic. Methods for automating NIST statistical testing for cybersecurity applications and determining the period of PRNGs for various initial values and generation parameters have been developed.

In the third chapter, **“Design of Pseudorandom Number Generators Based on Enhanced Algorithms for Computing Elementary Functions,”** presents the development of modified algorithms for computing elementary functions and the design of PRNGs based on these algorithms, relevant for applications in cybersecurity and information protection. The proposed efficient algorithms achieve improved accuracy and performance through the use of the MC method, Newton-Raphson (NR) and higher-order Householder (H) iterative methods in specialized forms, as well as adaptive parameter selection based on the input argument to minimize relative error. For the first time, a method for fast computation of the RCP function based on an identity with third-order best minimax polynomials is proposed; as well as methods based on a modified second-order Householder iteration (Ho2) for RCP and RSQRT; methods of additional magic constants (2MC and 3MC) to reduce the number of FP multiplications for the RSQRT and RCP functions; and methods of dynamic constants (DC and 8DC) for computing RSQRT and SQRT based on splitting the function’s domain and optimizing parameters. These methods do not use complex division operations or bulky LUTs. Based on them, a set of improved algorithms for computing the RCP, DIV, RSQRT,

and SQRT functions for cybersecurity applications has been implemented, offering enhanced accuracy (by 2-14 bits more) and/or performance (up to 5 multiplications less) for single- and double-precision FP numbers. The proposed algorithms ensure an error within 1 ULP; full accuracy is achieved for the RCP function. A set of improved simple algorithms for computing power, logarithmic, and exponential functions has also been implemented. The novel method for PRN generation proposed in this section is based on fast numerical approximations of RCP and RSQRT functions for double-precision FP numbers. Depending on the initial value and generator parameters, a recurrent sequence of 64-bit FP arguments is formed, producing a pseudorandom byte sequence.

In the fourth chapter, **“Experimental Evaluation and Efficiency Analysis of the Proposed Algorithms and Pseudorandom Number Generators on PC, Single-Board Computer, and MCU for Cybersecurity and Information Protection Tasks,”** practical testing and evaluation of the efficiency of the proposed and known algorithms for cybersecurity applications were carried out on various FP-capable devices, including Intel, Raspberry Pi, and ESP-32. In addition to software methods on personal computers (PCs), a mini-computer, and a MCU, hardware and software-hardware implementations of the RCP and RSQRT functions for *float* and *double* types were analyzed using Intel Core i5/i7 and ARM Cortex-A53 processors. The results of the analysis and testing of the proposed PRNGs for cybersecurity and information protection tasks, including statistical properties evaluated with NIST STS, graphical tests, and entropy analysis; period length; computational complexity; performance; generation parameters; and ranges of initial values. The study was conducted in three stages. The comparison of the best-proposed PRNGs with standard methods of generating PRN, including those from the random library, and hardware generators of TRS on Intel and ESP-32 was carried out. Some proposed algorithms, including *rsqrt2dc_everyBit* (3/0), *rcp3_everyBit* (3/0), *rsqrt1dc8_everyBit* (3/0), and *x_xored* (2/1), are slightly faster than the fastest PRNGs on Intel – *random_device* and *mt19937* – as well as the hardware TRS generators RDSEED and RDRAND, while requiring neither significant memory usage or lengthy reinitialization. It was found that the PRNG *rsqrt2dc_everyBit* (3/0) fails graphical tests, and the fast generator *x_xored* (2/1) has significant statistical flaws. The PRNGs with

the highest statistical quality among those studied are *rsqrt2dc_everyBit* (1/0) and *rcp3_everyBit* (3/0), with NIST test pass rates of 79% and 75%, respectively. Potential application areas and promising directions for further research are outlined, particularly for applications in cybersecurity and information protection. Two practical use cases of the developed fast PRNGs were analyzed in detail. In particular, the PRNG *rsqrt2dc_everyBit* (1/0) was used in the design of an advanced image-based steganographic system that analyzes embedding regions using a pseudorandom bitmask, minimizes the length of the steganographic key, and performs data encryption.

The **conclusions** summarize the main scientific and practical results, present quantitative evaluation and comparison of methods, outline areas of application in cybersecurity and information protection, and provide recommendations for implementation and directions for further research.

The **appendices** include acts of implementation of the dissertation results; source codes of the key developed research algorithms and tools, including the proposed methods of randomized numerical multidimensional optimization and scripts for automating the evaluation of PRNGs using NIST STS, which other researchers can directly use; comparison results of known and proposed algorithms with their parameters; source codes of the algorithms used in the design of the proposed PRNGs; results of their testing, evaluation, and comparison on different platforms; as well as a list of the author's scientific publications.

Thus, the dissertation introduces, for the first time, a set of enhanced methods for fast approximation of functions such as RCP, DIV, RSQRT, SQRT, RCBRT, and CBRT, including division-free approaches, based on the MC method, piecewise polynomial approximation, and approximate hardware instructions of Intel and ARM architectures. Improved precision is achieved through optimization, providing an effective trade-off between accuracy, performance, and hardware resource usage for cybersecurity and information protection tasks. Additionally, a novel class of PRNGs is proposed, based on recurrence relations and numerical techniques for computing elementary functions in FP arithmetic, which demonstrate satisfactory statistical properties for cybersecurity and information protection according to the results of NIST

STS testing. Based on the proposed methods, a set of enhanced algorithms has been implemented for *float* and *double*, which can serve as the foundation for the development of a specialized cross-platform library. This library would support efficient computation with varying precision (for rough approximations, near-full accuracy, e.g., 1 ULP, or correctly rounded results) for specific FP formats, and a library of lightweight and fast PRNGs for cybersecurity applications suitable for MCUs and FPGAs.

Keywords: pseudorandom number generator, elementary functions, numerical methods, floating-point numbers, cybersecurity, information security, FISR algorithm, software implementation, steganography, cryptography, accuracy, speed, minimization of relative errors, encryption, statistical testing NIST STS.

Список публікацій здобувача:

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Horyachyy O. Simple effective fast inverse square root algorithm with two magic constants / O. Horyachyy, L. Moroz, V. Otenko // International Journal of Computing. – 2019. – Vol. 18(4). – P. 461–470. (*Scopus*, Q3)
2. Максимович В. Генератори псевдовипадкових бітових послідовностей на основі чисельних методів в арифметиці з рухомою комою / В. Максимович, М. Шабатура, О. Горячий, Н. Лужецька // Сучасна спеціальна техніка. – Київ : Державний науково-дослідний інститут МВС України, 2021. – № 1(64). – С. 81–92.
3. Hrynchyshyn A. An efficient algorithm for fast inverse square root / A. Hrynchyshyn, O. Horyachyy, O. Tymoshenko, L. Moroz // Processing, transmission and security of information: monograph – Bielsko-Biała : Wydawnictwo Naukowe ATH w Bielsku-Białej, 2018. – Vol. 2. – P. 105–114.
4. Moroz L. Modified fast inverse square root and square root approximation algorithms: the method of switching magic constants / L. V. Moroz, V. V. Samotyy, O. Y. Horyachyy // Computation. – 2021. – Vol. 9(2) : 21. (*Scopus*, *Web of Science*, Q2)
5. Горячий О. Дослідження множини початкових значень генераторів псевдовипадкових чисел на основі арифметики з рухомою комою /

О. Горячий, В. Максимович, М. Шабатура // Сучасний захист інформації. – Київ : Державний університет інформаційно-комунікаційних технологій, 2024. – № 2(58). – С. 91–102.

Наукові праці, які засвідчують апробацію матеріалів дисертації:

6. Moroz L. An effective floating-point reciprocal / L. Moroz, V. Samoty, O. Horyachyy // Proceedings of the 2018 IEEE 4th International symposium on wireless systems within the International conferences on intelligent data acquisition and advanced computing systems (IDAACS-SWS 2018), Lviv : IEEE, September 20-21, 2018. – P. 137–141. (*Scopus, Web of Science*)
7. Moroz L. Algorithms for calculating the square root and inverse square root based on the second-order Householder's method / L. Moroz, V. Samoty, O. Horyachyy, U. Dzelendzyak // Proceedings of the 2019 10th IEEE International conference on intelligent data acquisition and advanced computing systems: technology and applications (IDAACS), Vol. 1, Metz : IEEE, September 18-21, 2019. – P. 436–442. (*Scopus, Web of Science*)
8. Horyachyy O. Improving the efficiency of methods for calculating elementary functions in floating-point arithmetic / O. Horyachyy // The International doctoral consortium on informatics education and educational software engineering research, Druskininkai : Vilnius University, December 2–6, 2019. – P. 2–7.
9. Горячий О. Дослідження ефективності простих стеганографічних методів обробки цифрових зображень із використанням генераторів псевдовипадкових послідовностей / О. Горячий, І. Журавель // Збірник тез доповідей V Міжнародної науково-практичної конференції «Інформаційна безпека та інформаційні технології» (ІБІТ 2024), Львів : ЛДУ БЖД, 27 листопада, 2024. – С. 225–230.
10. Горячий О. Я. Порівняння простих методів стеганографічного захисту інформації на основі аналізу цифрових зображень та використання псевдовипадкових послідовностей / О. Я. Горячий, І. М. Журавель // Матеріали V Міжнародної науково-технічної конференції

«Автоматизація, електроніка, інформаційно-вимірювальні технології: освіта, наука, практика», Харків : НТУ «ХПІ», 28–29 листопада, 2024. – С. 27–28.

Наукові праці, які додатково відображають наукові результати дисертації:

- 11.Horyachyy O. Improving the methods of protection and hacking of modern Wi-Fi networks / O. Horyachyy, T. Boretsky, I. Horiachyi // Materials of 7th International scientific and technical conference on information protection and information systems security, Lviv : Lviv Polytechnic Publishing House, May 30-31, 2019. – P. 60–61.
- 12.Мороз Л. Проста модифікація алгоритму швидкого обчислення зворотного квадратного кореня для чисел з рухомою комою одинарної точності / Л. В. Мороз, А. Гринчишин, О. Я. Горячий // Автоматика, вимірювання та керування. – Львів : Видавництво Національного університету «Львівська політехніка», 2019. – Вип. 1(1). – С. 39–45.
- 13.Horyachyy I. Information security management of a banking institution based on outsourcing / I. Horyachyy, O. Horyachyy, T. Boretsky // Materials of 8th International scientific and technical conference on information protection and information systems security, Lviv : Lviv Polytechnic Publishing House, November 11-12, 2021. – P. 11–12.
- 14.Shyshko D. Overview of main cybersecurity challenges of smart cities and methods of their protection / D. Shyshko, O. Horyachyy // Materials of 8th International scientific and technical conference on information protection and information systems security, Lviv : Lviv Polytechnic Publishing House, November 11-12, 2021. – P. 31–32.
- 15.Горячий О. Аналіз ефективності генераторів псевдовипадкових чисел на основі квадратного кореня простого числа / О. Горячий, З. Яремчук // Збірник тез доповідей V Міжнародної науково-практичної конференції «Інформаційна безпека та інформаційні технології» (ІБІТ 2024), Львів : ЛДУ БЖД, 27 листопада, 2024. – С. 234–237.

ЗМІСТ

ВСТУП.....	24
РОЗДІЛ 1. ОГЛЯД ВІДОМИХ МЕТОДІВ ОБЧИСЛЕННЯ ЕЛЕМЕНТАРНИХ ФУНКЦІЙ ТА ГЕНЕРАЦІЇ ПСЕВДОВИПАДКОВИХ ПОСЛІДОВНОСТЕЙ	37
1.1 Особливості представлення чисел з рухомою комою у форматі IEEE 754...	37
1.2 Аналіз відомих методів обчислення елементарних функцій у FP арифметиці для сфери кібербезпеки та захисту інформації.....	41
1.3 Дослідження алгоритму FISR та його модифікацій для обчислення функції RSQRT.....	49
1.3.1 Магічна константа.....	51
1.3.2 Метод Ньютона-Рафсона.....	55
1.3.3 Метод Хаусхолдера.....	59
1.4 Обчислення інших логарифмічних, експоненціальних та степеневих елементарних функцій на основі методу MC	61
1.5 Методи генерації випадкових і псевдовипадкових чисел та бітових послідовностей для криптографічних та некриптографічних застосувань	65
1.6 Методи оцінювання якості ГПВБП для задач у сфері кібербезпеки та захисту інформації.....	68
1.6.1 Статистичне тестування ГПВЧ.....	69
1.6.2 Набір статистичних тестів NIST STS для криптографічних застосувань	70
1.7 Формулювання вимог і вибір способів удосконалення алгоритмів для інформаційної та кібербезпеки.....	72
Висновки до розділу 1	73
РОЗДІЛ 2. ВИБІР МЕТОДІВ І РОЗРОБЛЕННЯ МЕТОДИКИ ДОСЛІДЖЕННЯ .	75
2.1 Методи проектування та дослідження вдосконалених алгоритмів обчислення елементарних функцій.....	75
2.1.1 Побудова математичної моделі алгоритмів.....	75
2.1.2 Методи аналізу алгоритмів.....	82
2.1.3 Розробка методики покращення алгоритмів щодо відносної похибки ...	84
2.2 Методи проектування та дослідження ГПВЧ на основі арифметики з рухомою комою для застосувань у сфері кібербезпеки та захисту інформації ..	96
2.2.1 Особливості проектування ГПВЧ на основі FP арифметики	96
2.2.2 Методика дослідження статистичних характеристик ПБП для задач у сфері кібербезпеки та захисту інформації на основі тестів NIST	97
2.2.3 Визначення періоду повторення ГПВЧ	101
2.2.4 Аналіз швидкодії генераторів ПБП.....	101

Висновки до розділу 2	102
РОЗДІЛ 3. ПРОЄКТУВАННЯ ГЕНЕРАТОРІВ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ НА ОСНОВІ ВДОСКОНАЛЕНИХ АЛГОРИТМІВ ОБЧИСЛЕННЯ ЕЛЕМЕНТАРНИХ ФУНКЦІЙ	104
3.1 Проєктування ефективних модифікованих алгоритмів обчислення елементарних функцій на основі методу МС для задач у сфері кібербезпеки та захисту інформації	104
3.1.1 Покращені алгоритми обчислення функцій RCP та DIV підвищеної точності та швидкодії	106
3.1.2 Прості модифікації алгоритму FISR	110
3.1.3 Покращені алгоритми обчислення функцій RSQRT та SQRT підвищеної точності	112
3.1.4 Покращені алгоритми обчислення функцій RSQRT та SQRT підвищеної швидкодії	121
3.1.5 Покращені алгоритми обчислення інших степеневих, показникових та логарифмічних функцій	126
3.2 Проєктування ефективних модифікованих алгоритмів обчислення елементарних функцій на основі методу поліноміальної апроксимації	127
3.3 Проєктування ефективних модифікованих алгоритмів RCP, DIV, RSQRT та SQRT на основі швидких наближених апаратних інструкцій Intel та ARM	129
3.4 Проєктування ГПВП для задач у сфері кібербезпеки та захисту інформації, побудованих на основі чисельних методів в FP арифметиці	130
3.4.1 Загальна структура запропонованих ГПВЧ	131
3.4.2 Опис можливих варіантів реалізації запропонованих ГПВЧ	133
Висновки до розділу 3	136
РОЗДІЛ 4. ПРАКТИЧНЕ ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ ЗАПРОПОНОВАНИХ АЛГОРИТМІВ ТА ГЕНЕРАТОРІВ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ НА ПК, МІНІ-КОМП'ЮТЕРІ ТА МК ДЛЯ ЗАДАЧ У СФЕРІ КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ	137
4.1 Аналіз та тестування покращених алгоритмів обчислення елементарних функцій для задач інформаційної та кібербезпеки	138
4.1.1 Порівняння запропонованих покращених алгоритмів обчислення функцій RCP та RSQRT	139
4.1.2 Тестування алгоритмів обчислення елементарних функцій на ПК, міні-комп'ютері та МК	143
4.2 Дослідження основних характеристик запропонованих ГПВЧ для задач кібербезпеки та захисту інформації, побудованих на основі FP арифметики та покращених алгоритмів обчислення елементарних функцій	147

4.2.1 Дослідження статистичних характеристик сформованих ПВП на основі тестів NIST STS для криптографічних застосувань	147
4.2.2 Визначення періоду повторення сформованих ПВП	155
4.2.3 Дослідження швидкодії запропонованих ГПВЧ	156
4.2.4 Детальний аналіз результатів статистичного дослідження запропонованих ГПВЧ для сформованих ПВП максимальної довжини	158
4.3 Аналіз і порівняння отриманих результатів тестування запропонованих ГПВЧ на основі чисельних методів у FP арифметиці.....	161
4.3.1 Порівняння з відомими методами за статистичними показниками.....	161
4.3.2 Порівняння основних характеристик та оцінка ефективності запропонованих ГПВЧ для задач у сфері кібербезпеки та захисту інформації.....	163
4.4 Сфери застосування та перспективні напрямки подальших досліджень	163
4.5 Практичне використання ГПВЧ для задач інформаційної та кібербезпеки	166
4.5.1 Приклад 1. Використання розробленого ГПВЧ для формування індивідуальних варіантів завдань для студентів	166
4.5.2 Приклад 2. Використання розробленого ГПВЧ для вбудовування та шифрування даних у стеганографічних системах	168
Висновки до розділу 4	171
ВИСНОВКИ.....	174
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	180
ДОДАТКИ.....	203
Додаток А. Акти впровадження	203
А.1 У ПП НВП «Спаринг-Віст Центр» – під час розроблення захищеного програмного забезпечення для моделювання та симуляції роботи вимірювальних пристроїв.....	204
А.2 На кафедрі безпеки інформаційних технологій Національного університету «Львівська політехніка» – при виконанні науково-дослідних робіт	205
А.3 На кафедрі безпеки інформаційних технологій Національного університету «Львівська політехніка» – у навчальному процесі для підготовки студентів спеціальності 125 «Кібербезпека та захист інформації»	207
Додаток Б. Програмний код розроблених алгоритмів та методів дослідження мовою C++	209
Б.1 Обчислення похибок алгоритмів.....	214
Б.2 Вимірювання швидкодії алгоритмів	217
Б.3 Методи оптимізації параметрів алгоритмів	220

Б.4 Визначення періоду повторення послідовності ПВЧ.....	233
Додаток В. Програмний код розроблених bat-файлів для автоматизації дослідження ГПВБП.....	237
В.1 Статистичне тестування файлу із ПВБП з використанням набору тестів NIST STS	237
В.2 Аналіз результатів статистичного тестування NIST STS.....	237
В.3 Статистичне тестування NIST STS набору файлів	238
В.4 Статистичне тестування NIST STS набору файлів із автоматичною зміною параметра m	240
В.5 Виклик процедури для визначення періоду повторення послідовності ПВЧ та опрацювання результатів	240
Додаток Г. Порівняння відомих та запропонованих алгоритмів обчислення елементарних функцій на основі методу МС	242
Додаток Д. Алгоритми наближеного обчислення функцій RCP та RSQRT використані при проектуванні ГПВЧ	262
Додаток Е. Порівняння швидкодії відомих та запропонованих апаратних, програмних та програмно-апаратних методів обчислення елементарних функцій та методів генерації ІВП та ПВП на різних обчислювальних платформах.....	267
Додаток Ж. Результати статистичного тестування ГПВЧ з використанням тестів NIST STS та графічних тестів.....	273
Додаток И. Список публікацій здобувача за темою дисертації та відомості про апробацію результатів дисертації	286

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

2MC	– Two Magic Constants, використання двох магічних констант
8DC	– Eight Dynamic Constants, переключення восьми магічних констант
CBRT	– Cube Root, кубічний корінь
CPU	– Central Processing Unit, центральний процесор
DC	– Dynamic Constants, переключення магічних констант
DIV	– Division, ділення
FISR	– Fast Inverse Square Root, швидкий зворотний квадратний корінь
FMA	– Fused Multiply-Add, поєднане множення та додавання
FP	– Floating-Point, формат чисел з рухомою комою
FPU	– Floating-Point Unit, математичний співпроцесор
H	– Householder, Хаусхолдер
Ho2	– Second-order Householder, Хаусхолдер другого порядку
IoT	– Internet of Things, Інтернет речей
LSB	– Least Significant Bit, найменший значущий біт
LUT	– Lookup Table, таблиця пошуку
MC	– Magic Constant, магічна константа
MCU	– Microcontroller Unit, мікроконтролер
NIST	– National Institute of Standards and Technology, Національний інститут стандартів і технології
NR	– Newton-Raphson, Ньютон-Рафсон
PRN	– Pseudorandom Number, псевдовипадкове число
PRNG	– Pseudorandom Number Generator, генератор псевдовипадкових чисел
PRS	– Pseudorandom Sequence, псевдовипадкова послідовність
PRSG	– Pseudorandom Sequence Generator, генератор псевдовипадкових послідовностей
RCBRT	– Reciprocal Cube Root, зворотний кубічний корінь
RCP	– Reciprocal, обернена до аргументу функція
RSQRT	– Reciprocal Square Root, зворотний квадратний корінь

SIMD	– Single Instruction, Multiple Data, одиночний потік команд, множинний потік даних
SQRT	– Square Root, квадратний корінь
STS	– Statistical Test Suite, набір статистичних тестів
TRS	– True Random Sequence, істинно випадкова послідовність
ULP	– Unit of Least Precision, одиниця найменшої точності
XOR	– Exclusive OR, взаємовиключне АБО
ГПВБП	– Генератор псевдовипадкових бітових послідовностей
ГПВП	– Генератор псевдовипадкових послідовностей
ГПВЧ	– Генератор псевдовипадкових чисел
ЗІ	– Захист інформації
ІБ	– Інформаційна безпека
ІВП	– Істинно випадкова послідовність
КБ	– Кібербезпека
МК	– Мікроконтролер
ОС	– Операційна система
ПВБП	– Псевдовипадкова бітова послідовність
ПВП	– Псевдовипадкова послідовність
ПВЧ	– Псевдовипадкові числа
ПК	– Персональний комп'ютер
ПЛІС	– Програмована логічна інтегральна схема

ВСТУП

Актуальність теми.

Обчислення елементарних функцій – оберненого до аргументу значення (RCP), ділення (DIV), зворотного квадратного кореня (RSQRT), квадратного кореня (SQRT), зворотного кубічного кореня (RCBRT) та кубічного кореня (CBRT) – є необхідним для розв’язання широкого кола прикладних задач, зокрема у сфері кібербезпеки (КБ) та захисту інформації (ЗІ), коли вхідні аргументи подано у форматі з рухомою комою (FP). Такі функції використовуються, зокрема, в сучасних засобах інформаційної та кібербезпеки, що ґрунтуються на статистичних, криптографічних, стеганографічних, біометричних та чисельних методах, моделюванні, нейронних мережах, комп’ютерній графіці, цифровій обробці сигналів і поведінковому аналізі. [1-15]. Зокрема, обчислення елементарних функцій SQRT і DIV часто використовується для статистичного аналізу та генерації псевдовипадкових послідовностей (ПВП), які є ключовими компонентами багатьох систем кібербезпеки та захисту інформації [2,15,16], наприклад захищеного багатокористувацького обчислення [13]. Залежно від сфери застосування, іноді необхідно забезпечити максимально можливу точність результату із коректним заокругленням [17-22] (наприклад, у криптографії та чисельних методах), інколи – високу швидкодію за прийняттого рівня точності [4,6,11,23-25] (зокрема для задач кібербезпеки, статистики, комп’ютерного бачення та 3D графіки). Основна ідея дисертації, що зумовлює її актуальність і наукову новизну, полягає у дослідженні можливості використання покращених методів швидкої апроксимації згаданих вище нелінійних елементарних функцій у FP арифметиці для проектування генераторів ПВП (ГПВП), придатних для задач кібербезпеки та захисту інформації, що задовольняють вимогам статистичних тестів NIST STS.

Швидкий розвиток інформаційно-комунікаційних технологій, широке розповсюдження Інтернету речей (IoT) та накопичення значних обсягів інформації, в тому числі особистого та конфіденційного характеру, дають можливість використання новітніх систем обробки, передачі та захисту

інформації, а також аналізу та прийняття рішень в режимі реального часу. Оскільки на практиці більшість задач містить неточні вхідні дані, отримання точного розв'язку може бути недоцільним [25]. При цьому швидкість опрацювання, аутентифікації, виявлення загроз та реагування на інциденти інформаційної безпеки (ІБ) має ключове значення для систем управління ІБ як на рівні держави, так і на рівні окремих організацій [9,26-30]. Для більшості таких застосувань достатньо швидких наближених обчислень на мікроконтролерах (МК) та IoT-пристроях з обмеженими ресурсами [8,9,31-33]. З іншого боку, точність роботи автоматизованої системи має бути достатньою для забезпечення прийнятно малого рівня хибних спрацювань. Не менш важливим з економічного погляду та з погляду безпеки є адекватний до наявних загроз рівень захисту як самих пристроїв, так і даних, що в них циркулюють [31,34-38].

На сучасних платформах арифметичні FP операції та типові елементарні функції можуть бути реалізовані у вигляді апаратних інструкцій математичного співпроцесора (FPU). В іншому випадку, операції реалізуються програмно використовуючи бібліотечні функції, що зазвичай є досить повільними. При цьому, діючий стандарт IEEE 754-2019 [39] вимагає, щоб базові операції, включаючи поєднане множення і додавання (FMA), та функція SQRT забезпечували коректне заокруглення результату. Інші поширені бібліотечні функції (тригонометричні, логарифмічні, експоненціальні, степеневі та ін.) можуть давати результат, що відрізняється від точного на декілька біт [40,41].

Функції DIV та SQRT, що є поширеними як в обчисленнях загального, так і спеціального призначення, зокрема у сфері кібербезпеки та захисту інформації, є досить ресурсозатратними порівняно з іншими базовими FP операціями. Згідно з А. Фогом [42], вони часто реалізуються апаратно та на більшості платформ мають високу латентність та низьку пропускну здатність. Зазвичай інструкція SQRT використовує від 3 до 10 разів більше тактів процесора (CPU), ніж $\times$ та $+$, аналогічно для інструкції DIV цей показник складає від 2 до 12 разів [1,8,42-45]. Для прикладу, незважаючи на нечасте використання операції DIV (близько 3% серед усіх операцій в CPU загального призначення), саме ця інструкція може

бути причиною суттєвого зниження продуктивності обчислень (до 40%) [43,46]. FPU багатьох платформ містить також швидкі наближені інструкції для RCP та RSQRT, що в залежності від архітектури можуть мати різну точність. Зазвичай такі інструкції реалізуються як таблиці пошуку (LUT), проте розмір таких таблиць зростає експоненціально із збільшенням необхідної точності. Тому на практиці LUT комбінують з іншими методами, наприклад ітераційними. Незважаючи на це, багато платформ не підтримують апаратних інструкцій DIV, SQRT, RCP та RSQRT для необхідних типів даних. Як приклад можемо навести 32-бітний МК ESP-WROOM-32, що широко використовується в IoT [47].

Ще однією важливою проблемою, особливо для задач, що пов'язані із функціонуванням інформаційних систем, захистом конфіденційності та цілісності інформації є генерація випадкових або псевдовипадкових послідовностей високої якості [35,36,48-55] як для криптографічних [36,54-59], так і для некриптографічних застосувань [55,60,61]. Це пов'язано з тим, що багато обчислювальних пристроїв, зокрема дешевих мікроконтролерів у IoT, не містять необхідних ресурсів, зокрема надійних та достатньо швидких джерел ентропії, що можуть бути використані для генерації послідовностей істинно випадкових чисел [36,54,62,63]. Крім того, такі генератори, як програмні, так і апаратні, можуть мати низьку продуктивність, серйозні статистичні недоліки або катастрофічні вразливості [36,53-55,63-65].

Сучасні ГПВП, серед яких генератори псевдовипадкових чисел (ГПВЧ) та генератори псевдовипадкових бітових послідовностей (ГПВБП), можуть формувати псевдовипадкові дані різних типів – від окремих булевих значень до послідовностей бітів, цілих чисел, чисел із фіксованою та рухомою комою тощо. Вони використовуються в стеганографії та криптографії, наприклад, для генерації ключів та інших випадкових даних для алгоритмів блокового та потокового шифрування, стеганографії, цифрового підпису, автентифікації та різноманітних криптографічних протоколів [51,54,56,57,59,60,62,66-68]. Зокрема, набувають популярності протоколи захищеного багатокористувацького обчислення [68-72], що оперують розподіленими конфіденційними даними, в

тому числі в FP форматі [68,72-74]. Для таких застосувань може додатково вимагатися криптографічна стійкість відповідних ГПВП [36,49,54,62,75]. Псевдовипадкові числа (ПВЧ) застосовуються в кібербезпеці, наукових дослідженнях, виробництві, статистиці, комп'ютерній графіці, чисельних методах, безпроводних технологіях, наприклад, для надійної передачі даних, пристроїв зашумлення, систем прийняття рішень, навчання нейронних мереж, методу Монте-Карло, симуляції випадкових процесів, оптимізації та рандомізованого пошуку [32,49,52,61,76,77].

Багато науковців, переважно закордонних, займалися питаннями дослідження, розробки та покращення методів обчислення елементарних функцій, призначених як для програмної, так і для апаратної їх реалізації. Серед яких: Мороз Л. В., Попов В. В., Гринчишин А., Борецький Т. Р., Вакал Л. П., Грінченко Л. Г., Muller J.-M., Walczyk C. J., Schulte M. J., Cieśliński J. L., Lefèvre V., de Dinechin F., Defour D., Stine J. E., Lomont C., Detrey J., Blinn J., Eberly D. H., Nikmehr H., Robertson M., Parhami B., Chen Y., Kahan W., Kadlec J., Ziv A., Khuong P. та ін. Зокрема, в науковій літературі основна увага приділяється реалізації даних методів на програмованих логічних інтегральних схемах (ПЛІС) із фіксованою чи з рухомою комою. При цьому використовуються зазвичай наступні методи: цифрові рекурентні, ітераційні, поліноміальні, на основі LUT або на основі методу FISR. Питанням аналізу, тестування, порівняння відомих чи розробки нових ГПВП займалися, зокрема, наступні вітчизняні та закордонні науковці: Максимович В. М., Потий А., Поперешняк С. В., Гарасимчук О. І., Королев Р. В., Шабатура М. М., Горпенюк А. Я., Костів Ю. М., Фауре Е. В., Совин Я. Р., Гриненко Т., Височіненко М. С., Горбенко Ю. І., Гапак О. М., Долотій М. Г., Matsumoto M., Marsaglia G., Bakiri M., Wang L., Shrestha B., Shen H., Cheng H., Harase S., Nishimura T., Tsang W. W., Blum L., Kietzmann P., Guyeux C., Bailey D., Kordov K., Saito M. та ін. Популярні в літературі алгоритми генерації ПВБП та ПВЧ будуються, переважно, на основі конгруентних генераторів, рекурентних генераторів в кінцевому полі, генераторів на основі регістра зсуву зі зворотним

зв'язком тощо. Вони потребують таких елементарних бітових і цілочислових операцій як додавання за модулем 2 (XOR), цілочисельне ділення, отримання залишку від ділення, бітового зсуву тощо. Деякі з цих генераторів можуть мати достатньо великий період повторення та хороші статистичні характеристики, наприклад вихор Мерсена (Mersenne twister, MT19937) [36,53], проте зазвичай не є криптостійкими. Класичні криптостійкі ГПВЧ є досить повільними, оскільки будуються на основі блокових і потокових шифрів або хеш-функцій. Альтернативні методи ГПВП базуються на обчисленні елементарних функцій [2,10,16,78] або хаотичних систем [56-58,79]. Зокрема, останнім часом у науковій літературі набувають популярності генератори на основі теорії хаосу, зокрема через їхню придатність для застосування в стеганографії та криптографії. Проте, недостатньо уваги приділено дослідженню можливості генерації ПВЧ з використанням методів апроксимації функцій в $\mathbb{F}_p$ арифметиці.

Отже, актуальним науковим завданням є розроблення та дослідження генераторів псевдовипадкових чисел на основі покращених методів обчислення елементарних функцій, придатних для практичного використання у задачах кібербезпеки та захисту інформації на різних комп'ютерних платформах і системах, зокрема на персональних комп'ютерах (ПК), МК та ПЛІС.

Зв'язок роботи з науковими програмами, планами, темами. Роботу виконано на кафедрі безпеки інформаційних технологій Інституту комп'ютерних технологій, автоматики та метрології Національного університету «Львівська політехніка». Тема дисертаційної роботи відповідає науковому напрямку кафедри. Дисертацію виконано в межах науково-дослідних робіт кафедри безпеки інформаційних технологій за темами «Розробка та покращення ефективності алгоритмів обчислення елементарних функцій для задач захисту інформації» (№ державної реєстрації 0120U103215), «Розробка та дослідження генераторів псевдовипадкових чисел і послідовностей» (№ державної реєстрації 0113U005267) та «Розроблення стеганографічних методів приховування даних у цифрових зображеннях» (№ державної реєстрації 0120U100024).

Мета і завдання дослідження. Метою роботи є розроблення нових і

вдосконалення наявних простих та ефективних методів обчислення елементарних функцій у FP арифметиці, що дозволяють отримати якісний компроміс між необхідною точністю та швидкістю обчислень, а також проєктування на їх основі генераторів псевдовипадкових чисел із задовільними статистичними характеристиками, придатних для задач інформаційної та кібербезпеки.

Для досягнення цієї мети необхідно вирішити такі основні завдання:

а) провести огляд та класифікацію методів обчислення елементарних функцій, дослідити теоретичні основи алгоритму FISR, його модифікації та узагальнення; визначити сфери застосування, сформулювати основні вимоги для сфери кібербезпеки та захисту інформації, способи підвищення точності й швидкодії, зокрема для обчислення RCP, DIV, RSQRT та SQRT, RCBRT та CBRT; провести огляд та класифікацію методів генерації ПБП та ПБЧ; визначити сфери застосування та основні вимоги до них; дослідити використання чисельних методів та FP арифметики для генерації ПБП; провести аналіз методів оцінювання якості ГПБП на основі графічних та оціночних тестів;

б) розробити методикау дослідження для аналізу, тестування та вдосконалення алгоритмів, придатних для задач у сфері інформаційної та кібербезпеки;

в) спроектувати нові та вдосконалити відомі методи швидкої апроксимації степеневих, логарифмічних та показникових елементарних функцій в FP арифметиці на основі методу MC; на їх основі розробити покращені алгоритми обчислення функцій RCP, RSQRT та RCBRT без операції $\div$ із підвищеною точністю і/або швидкодією для чисел одинарної та подвійної точності, модифікації алгоритмів для обчислення DIV, SQRT та CBRT, а також прості модифікації алгоритмів на основі MC, кусково-поліноміальної апроксимації та швидких апаратних інструкцій;

г) спроектувати нові прості та швидкі генератори ПБЧ, зокрема для сфери кібербезпеки та захисту інформації, на основі алгоритмів наближеного обчислення елементарних функцій RCP, RSQRT та SQRT в FP арифметиці, що відповідають вимогам тестів NIST;

д) здійснити практичну реалізацію та тестування ефективності запропонованих та відомих алгоритмів обчислення елементарних функцій, а також ГПВЧ та генераторів істинно випадкових послідовностей (ІВП), на різноманітних пристроях, що підтримують FP обчислення, зокрема на сучасних ПК (Intel i5 та i7), міні-комп'ютерах (ARM Cortex A-53) та МК (Xtensa), а також дослідити можливість апаратної реалізації алгоритмів на ПЛІС;

е) дослідити статистичні характеристики запропонованих ГПВЧ з використанням тестів NIST, графічних тестів, обчислення ентропії; провести тестування періоду повторення, визначення простору допустимих початкових значень та швидкодії ГПВЧ; визначити генератори з найкращими характеристиками для задач у сфері кібербезпеки та захисту інформації, а також запропонувати методи їх подальшого вдосконалення.

Об'єкт дослідження – процеси обчислення елементарних функцій та генерації псевдовипадкових послідовностей в FP арифметиці.

Предмет дослідження – методи підвищення точності та швидкодії алгоритмів апроксимації таких елементарних функцій як RCP, DIV, RSQRT та SQRT, RCBRT та CBRT в FP арифметиці, методи генерації псевдовипадкових чисел на їх основі, а також методи їх оцінювання для застосувань у сфері інформаційної та кібербезпеки.

Методи дослідження. Для вирішення поставлених завдань було використано такі методи: огляду літератури, аналізу та синтезу, порівняння, основні положення алгебраїчної теорії чисел та полів, теорії чисельних методів, чисельної оптимізації, експерименту, елементів теорії програмування, вимірювання, автоматизації, оцінки збіжності, комп'ютерного моделювання, елементи теорії похибок, теорії ймовірностей і математичної статистики, теорії часової складності алгоритмів та імітаційного моделювання.

Наукова новизна отриманих результатів. Найважливішими науковими результатами є такі:

1. Вперше розроблено новий клас методів ГПВЧ на основі рекурентного рівняння та чисельних методів в FP арифметиці, що мають максимальний період

повторення $L = 6,66 \cdot 10^8$ 8-бітових значень (байт), множину початкових значень близько 2^{60} елементів, високу швидкодію (близько 40 Мб/с на Intel i5) та задовільні статистичні характеристики тестів NIST у порівнянні з найкращими відомими ГПВЧ з бібліотеки random (до 80% згенерованих ПВП проходять всі 188 тестів при зміні початкових значень; значення ентропії близьке до 8 біт) для стеганографії, криптографії, комп'ютерних ігор, моделювання та симуляції. Їх використання підвищило рівень безпеки вдосконаленої стеганографічної системи (зокрема конфіденційності, цілісності та стійкості до виявлення) на основі аналізу цифрових зображень та дозволило зменшити розмір стеганоключа на 94%.

2. Вперше розроблено набір методів швидкої апроксимації функцій RCP, DIV, RSQRT, SQRT, RCBRT, CBRT та ін., які базуються на модифікованих методах МС, Ньютона-Рафсона (NR) та Хаусхолдера (H) різних порядків. Методи записані у спеціальній формі з оптимізованими параметрами, що мінімізують максимальну відносну похибку $\delta_{\max}$, забезпечуючи високу точність та швидкодію обчислень для задач у сфері кібербезпеки та захисту інформації. Зокрема:

- покращений метод швидкого обчислення функції RCP, що ґрунтується на тотожності з поліномами найкращого рівномірного наближення 3-го порядку, що забезпечує точність після першої ітерації модифікованого методу NR рівну 13 бітам, що дає зниження $\delta_{\max}$ в 2,2 раза порівняно з найкращим відомим методом Мороза та ін. з однаковою кількістю арифметичних операцій; після другої та третьої ітерації NR з оптимізованими параметрами забезпечується похибка на рівні 1 ULP для чисел одинарної та подвійної точності, дозволяючи зменшити $\delta_{\max}$ в 2,8-3 рази при меншій кількості операцій $\times$ порівняно з відомими методами;
- покращений метод швидкого обчислення RCP на основі модифікованого методу Хаусхолдера 2-го порядку (Ho2) з оптимізованими параметрами, що на першій ітерації забезпечує точність 14 біт, зменшуючи $\delta_{\max}$ в 7,5 раз порівняно з методом Мороза та ін. при використанні додаткового FP +; після другої та третьої ітерації NR з оптимізованими параметрами

досягається повна точність з коректним заокругленням для чисел типу *float* (24 біти) та *double* (53 біти) і використовує на два $\times$ менше і/або має на два біти вищу точність;

- покращений метод швидкого обчислення RSQRT на основі модифікованого методу No2 з оптимізованими параметрами, що на першій ітерації забезпечує 15 бітів точності, зменшуючи $\delta_{\max}$ в 32,9 раз порівняно з найкращими модифікаціями алгоритму FISR при використанні додаткового FMA; після другої та третьої ітерації NR з оптимізованими параметрами досягається точність на рівні 1 ULP для чисел одинарної та подвійної точності, що дозволяє в 2,2-3,5 рази зменшити $\delta_{\max}$ та для чисел типу *float* зменшити також на 1 кількість FMA порівняно з методами Леметра та ін. на основі двох ітерацій No2 та No3 відповідно;
- методи додаткової магічної константи (2MC) та трьох магічних констант (3MC) на основі цілочислових операцій для зменшення на 1-2 кількості $\times$ у покращених алгоритмах на основі FISR, що забезпечує спрощення апаратної реалізації та підвищення ефективності обчислень, зокрема для МК, IoT пристроїв та високопродуктивних систем; використання методу 2MC для функцій RSQRT та SQRT дозволяє зменшити $\delta_{\max}$ в 11,5-11,9 рази порівняно з базовим алгоритмом FISR для двох ітерацій NR, а використання методу 3MC для RCP дозволяє зменшити $\delta_{\max}$ в 21,2 рази порівняно з алгоритмом Хуана і Чена для двох ітерацій Гольдшмідта;
- методи переключення магічних констант (DC та 8DC) для RSQRT та SQRT одинарної та подвійної точності, що передбачають розбиття інтервалу визначення функції на дві, чотири або більше нерівних частин та оптимізують коефіцієнти алгоритмів на кожній з них, забезпечуючи якісний компроміс між точністю, швидкодією та розміром LUT і зменшення $\delta_{\max}$ в 11,8 раз та 101,2 раз у порівнянні з методом Вальчика та ін. при використанні відповідно двох та восьми магічних констант.

3. Удосконалено методи обчислення RCP та RSQRT на основі апаратних інструкцій Intel і ARM та модифікованих методів NR та H, що на відміну від класичних алгоритмів забезпечують повну або майже повну точність результату.

4. Удосконалено методи чисельної багатовимірної оптимізації з елементами випадковості, що використовуються для визначення таких практичних значень параметрів, що мінімізують $\delta_{\max}$ для визначених типів, та, зокрема, дозволили отримати для функції RCP оптимізовані варіанти алгоритмів з повною точністю.

5. Набули подальшого розвитку методи автоматизації статистичного тестування та визначення періоду повторення ПБП з використанням тестів NIST STS, вимірювання швидкодії алгоритмів обчислення елементарних функцій в FP арифметиці та продуктивності ГПВЧ і генераторів ІВП на різних платформах.

Практичне значення отриманих результатів.

1. Розроблено новий клас ефективних алгоритмів генерації ПБЧ, що використовують запропоновані алгоритми апроксимації функцій RCP та RSQRT в FP арифметиці для чисел типу *double*. Проведено статистичне тестування генераторів пакетом тестів NIST, отримані залежності періоду повторення сформованих ПБП від вибору початкового значення ГПВЧ, визначено прийнятні діапазони початкових значень і встановлено генератори з найкращими характеристиками для застосування у сфері кібербезпеки та захисту інформації. Вони призначені насамперед для некриптографічних задач на 64-бітних МК, CPU, ПЛІС та деяких IoT пристроях, що підтримують основні FP операції.

2. Сформульовано основні вимоги та запропоновано методику проєктування вдосконалених алгоритмів обчислення елементарних функцій в FP арифметиці щодо підвищення їх точності та швидкодії для використання у сфері інформаційної та кібербезпеки. Розроблено математичну модель модифікованих алгоритмів та набір інструментів для їх вдосконалення. Вони можуть бути застосовані науковцями та комп'ютерними інженерами на практиці для розробки нових модифікацій алгоритмів, зокрема для обчислення степеневих, логарифмічних, експоненціальних та інших функцій на основі методу МС, кусково-поліноміальних, дробово-раціональних методів або спеціалізованих

апаратних інструкцій різних платформ, а також модифікованих ітераційних методів різного порядку збіжності з або без $\div$. Запропоновані в дисертаційній роботі методи автоматизації статистичного тестування на основі тестів NIST STS, визначення періоду повторення ПВП, вимірювання швидкодії алгоритмів та продуктивності ГПВЧ є універсальними та ефективними інструментами, що можуть бути безпосередньо застосовані іншими дослідниками при розробці та аналізі чисельних методів, криптографічних алгоритмів, генераторів ПВП та ІВП.

3. Наведені в дисертації та в додатках методи, оптимізовані коефіцієнти й програмні коди мовою C++ алгоритмів обчислення елементарних функцій RCP, DIV, RSQRT, SQRT, RCBRT, CBRT, $y = 2^x$, $y = e^x$, $y = \log_2 x$ та $y = \ln x$ для нормалізованих чисел одинарної та подвійної точності, дають змогу використовувати їх у сфері інформаційної та кібербезпеки як у програмному, так і в апаратному вигляді на різних платформах – зокрема на МК, CPU, ПЛІС та GPU – із можливістю забезпечення необхідного компромісу між точністю, швидкістю та використанням апаратних ресурсів. Інші переваги запропонованих алгоритмів: більшість з них не використовують складної операції $\div$ та громіздких LUT; запропоновані алгоритми добре піддаються векторизації, а відтак можуть бути реалізовані в режимі SIMD; розглянуті методи можуть бути узагальнені для інших типів даних стандарту IEEE 754.

4. Запропоновані легкі та швидкі ГПВЧ на основі чисельних методів в FP арифметиці, що мають задовільні статистичні характеристики та придатні для застосувань у сфері кібербезпеки та захисту інформації, є основою для створення спеціалізованої кросплатформенної бібліотеки для МК та CPU. Також, відповідні покращені алгоритми є основою спеціалізованої математичної бібліотеки, що підтримує ефективні обчислення елементарних функцій із різною точністю: для швидкої апроксимації (наприклад, RCP з точністю 13 або 14 біт; функції RSQRT – з точністю 10, 13, 15 або 16 біт тощо), грубого обчислення (похибка на рівні 1 ULP) та повної точності (з коректним заокругленням результату).

5. Отримані теоретичні та практичні результати досліджень впроваджені в ПП НВП «Спаринг-Віст Центр», а також використовувались при виконанні

науково-дослідних робіт та у навчальному процесі кафедри безпеки інформаційних технологій Національного університету «Львівська політехніка», що підтверджено відповідними актами впровадження.

Особистий внесок здобувача. Усі наукові результати дисертаційної роботи автор отримав самостійно. У працях, опублікованих у співавторстві, здобувачеві належить: [7] – аналіз літератури, дослідження відомих алгоритмів, теоретичне обґрунтування, програмна реалізація, експерименти, модифікація та оптимізація параметрів алгоритмів, аналіз результатів; [8] – огляд літератури, теоретичне опрацювання, вибір методики досліджень, проєктування та чисельна оптимізація алгоритмів, програмна реалізація, експерименти, аналіз ефективності та оформлення результатів; [15,85] – аналіз літератури, опрацювання теоретичних основ, розробка методики, проєктування та реалізація ГПВЧ, експерименти та оформлення результатів; [30,34,38,83,84] – огляд літератури, теоретичне обґрунтування, формулювання ідей, оформлення результатів; [80] – аналіз літератури, опрацювання теоретичних основ, формування рекомендацій щодо покращення алгоритмів, вибір методики, розробка чисельного методу оптимізації параметрів алгоритмів, реалізація, експерименти та аналіз ефективності алгоритмів; [81] – формулювання ідей, практична реалізація, дослідження похибок, аналіз та представлення результатів; [82] – теоретичне опрацювання, формулювання ідей та постановка задачі, покращення алгоритмів, мінімаксна апроксимація, реалізація, дослідження точності та швидкодії на МК, оформлення результатів; [86,87] – аналіз літератури, теоретичне обґрунтування, розробка методики досліджень, реалізація стеганографічних методів, їх тестування та оформлення результатів.

Апробація результатів дисертації. Основні положення і результати дисертації доповідались, обговорювались та опубліковано в матеріалах міжнародних конференцій, симпозіумів та консорціумів: IV міжнародному IEEE симпозіумі «Безпроводні системи» в рамках міжнародних конференцій «Інтелектуальний збір даних і сучасні обчислювальні системи» (IDAACS-SWS 2018). – (м. Львів, 20-21 вересня 2018 р.); VIII міжуніверситетській конференції

студентів, Ph.D. студентів та молодих вчених «Інженер XXI століття». – (м. Бельсько-Бяла, Польща, 7 грудня 2018 р.); X міжнародній IEEE конференції «Інтелектуальний збір даних і сучасні обчислювальні системи: технології та застосування» (IDAACS 2019). – (м. Мец, Франція, 18-21 вересня 2019 р.); X міжнародному докторському консорціумі «Дослідження освіти в галузі інформатики та комп'ютерної інженерії» спільно з семінаром Nordplus «Культурно різноманітні підходи до вивчення математики та комп'ютерного мислення». – (м. Друскінінкай, Литва, 2-6 грудня 2019 р.); V міжнародній науково-практичній конференції «Інформаційна безпека та інформаційні технології» (ІБІТ-2024). – (м. Львів, 27 листопада, 2024 р.); V міжнародній науково-технічній конференції «Автоматизація, електроніка, інформаційно-вимірювальні технології: освіта, наука, практика». – (м. Харків, 28-29 листопада, 2024 р.). Також, матеріали дисертації неодноразово обговорювались на наукових семінарах кафедри «Безпека інформаційних технологій» Національного університету «Львівська політехніка».

Публікації. Основні наукові результати за темою дисертації опубліковано у 15 наукових працях: 1 стаття у науковому періодичному виданні іншої держави, яке включено до міжнародних наукометричних баз Scopus та Web of Science, 3 статті у періодичних фахових виданнях України з технічних наук, одна з яких входить до міжнародної наукометричної бази Scopus, 1 стаття в науковому нефармовому виданні України, 1 розділ колективної монографії та 9 публікацій в збірниках матеріалів та тез наукових конференцій, дві з яких включені до міжнародних наукометричних баз даних Scopus та Web of Science.

Структура та обсяг дисертації. Дисертаційна робота викладена на 288 сторінках, включаючи 168 сторінок основного тексту, та містить анотацію, зміст, перелік умовних позначень, вступ, чотири основних розділи (34 рисунки та 9 таблиць), висновки, список використаних джерел (225 найменувань), а також 8 додатків (16 рисунків та 17 таблиць). За структурою, мовою та стилем викладення дисертація відповідає вимогам МОН України.

РОЗДІЛ 1. ОГЛЯД ВІДОМИХ МЕТОДІВ ОБЧИСЛЕННЯ ЕЛЕМЕНТАРНИХ ФУНКЦІЙ ТА ГЕНЕРАЦІЇ ПСЕВДОВИПАДКОВИХ ПОСЛІДОВНОСТЕЙ

1.1 Особливості представлення чисел з рухомою комою у форматі IEEE 754

Арифметика з рухомою комою (FP) дозволяє наближено представляти дійсні числа в широкому діапазоні використовуючи типи даних визначеного розміру. Такий формат даних є оптимальним для більшості застосувань зважаючи на швидкодію, обчислювальну складність та використання пам'яті.

Оскільки дійсні числа можуть мати нескінченну кількість точних знаків, наприклад, ірраціональні $\pi = 3,14159265\dots$, $\sqrt{2} = 1,41421356\dots$ та періодичні $2/3 = 0,6(6)$, $3/11 = 0,27(27)$, $119/36 = 3,305(5)$, числа, для представлення дійсних чисел на комп'ютерах зазвичай використовується або арифметика з фіксованою або з рухомою комою. Таке представлення не дає змогу відобразити в обмеженій пам'яті всі числа, а тому застосовується заокруглення після певного знаку.

У першому випадку кількість знаків, що виділяється на збереження цілої та дробової частини числа є фіксованою. Це дозволяє забезпечити сталі значення абсолютної похибки такого відображення та чіткі межі мінімального та максимального значення. Тому арифметика з фіксованою комою зручна, наприклад, для відображення фінансових даних, де незалежно від розміру транзакцій точність операцій має бути однаковою. Операції із фіксованою комою швидкі та легко реалізуються на практиці, зокрема використовуючи цілі числа. Крім того, вони більш енергоефективні, що важливо для IoT пристроїв. У випадку FP арифметики із числа виділяється масштабувальний коефіцієнт (порядок числа), що відображається у вигляді показника степеня. Порядок числа та дробова частина (мантиса) зберігаються окремо. Ця форма представлення ще називається експоненціальним (науковим) записом. Завдяки цьому можна представляти числа із значно ширшого діапазону можливих значень. FP арифметика гарантує фіксоване значення відносної похибки та високу точність обчислень. Це дуже важливо для таких областей застосування як наукові обчислення, математика, фізика, астрономія тощо. При цьому абсолютна похибка такого відображення

змінюється в залежності від порядку числа. В загальному випадку, чим більше за модулем число, тим вища буде його абсолютна похибка.

У стандарті IEEE 754-2019 [39] FP числа поділяються на нормалізовані та денормалізовані. Нормалізовані двійкові FP числа x представляються у вигляді:

$$x = (-1)^{S_x} \cdot M_x \cdot 2^{E_x}, \quad (1.1)$$

де S_x – знаковий біт, причому $S_x \in \{0; 1\}$, 0 – для додатніх, 1 – для від’ємних чисел; M_x – мантия числа x , що знаходиться в діапазоні $M_x \in [1; 2)$; E_x – цілочислове значення, що відображає порядок (експоненту) числа x за основою 2.

Порядок, в загальному випадку, визначається за формулою:

$$E_x = \lfloor \log_2 |x| \rfloor, \quad (1.2)$$

де операція $\lfloor x \rfloor$ повертає найбільше ціле число, що є меншим або рівним значенню аргументу. При цьому, відповідно до стандарту IEEE 754, у b бітах пам’яті, що виділені для збереження порядку, E_x представляється із зміщенням:

$$E = E_x + bias, \quad (1.3)$$

$$bias = 2^{b-1} - 1. \quad (1.4)$$

Наприклад, для чисел із одинарною точністю (32 біти) із 8-мибітною експонентою $bias = 127$. Тоді діапазон можливих значень експоненти визначається нерівністю:

$$1 - bias = E_{\min} \leq E_x \leq E_{\max} = bias. \quad (1.5)$$

Значення манти M_x можна отримати із формули:

$$M_x = \frac{|x|}{2^{E_x}}, \quad (1.6)$$

причому для нормалізованих чисел мантия має вигляд:

$$M_x = m_0.m_1m_2...m_{p-1}m_p, \quad (1.7)$$

де $m_0 = 1$ – ціла частина манти, що задана неявно (точність числа – $p + 1$ біт). А тому в пам’яті зберігається лише дробова частина манти $m_x \in [0; 1)$:

$$m_x = M_x - 1 \quad (1.8)$$

у вигляді цілого числа M , що містить p бітів:

$$M = m_x \cdot 2^p. \quad (1.9)$$

Таким чином, загальна визначена стандартом IEEE 754 структура числа з рухомою комою показана на рис. 1.1.

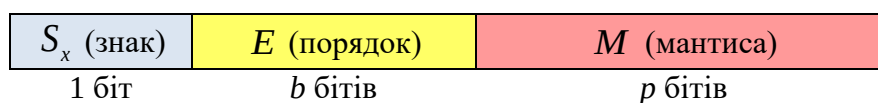


Рисунок 1.1 – Структура числа із рухомою комою

У табл. 1.1 показані основні формати FP чисел, що визначені стандартом IEEE 754, та їх параметри. Крім основних двійкових форматів стандарт визначає також десяткові формати та формати розширеної точності.

Таблиця 1.1 – Основні двійкові формати даних стандарту IEEE 754

Назва формату	Розмір типу даних n / порядку b / мантиси p , біти	Зміщення порядку $bias$ (діапазон значень E_{min} та E_{max})	Точність типу, біти (десяткові знаки)	Діапазон допустимих значень (нормалізованих додатних)	Одиниця молодшого розряду (ULP) при $E_x = 0$
Одинарна точність (binary32/float)	32 / 8 / 23	127 (від -126 до +127)	24 (~7,22)	від $1,2 \cdot 10^{-38}$ до $3,4 \cdot 10^{+38}$	$1,2 \cdot 10^{-7}$
Подвійна точність (binary64/double)	64 / 11 / 52	1023 (від -1022 до +1023)	53 (~15,95)	від $2,2 \cdot 10^{-308}$ до $1,8 \cdot 10^{+308}$	$2,2 \cdot 10^{-16}$
Чотириразова точність (binary128)	128 / 15 / 112	16383 (від -16382 до +16383)	113 (~34,02)	від $3,4 \cdot 10^{-4932}$ до $1,2 \cdot 10^{+4932}$	$1,9 \cdot 10^{-34}$

Розмір ULP залежить від порядку чисел та визначається формулою:

$$ULP = 2^{E_x - p}. \quad (1.10)$$

ULP – це відстань між двома сусідніми дискретними значеннями в FP арифметиці, тобто одиниця в молодшому розряді (LSB) мантиси. У багатьох мовах програмування це значення при $E_x = 0$ ще називається машинним нулем (епсилоном). Режим округлення чисел до найближчого (із прив'язкою до парного) визначено стандартом як режим за замовчуванням. Як показано у [88], відносну похибку такого представлення числа x можна оцінити наступною нерівністю:

$$\delta_x \leq 0,5 \cdot 2^{-p}. \quad (1.11)$$

Він забезпечує похибку 0,5 ULP, а тому найчастіше використовується на практиці. Інші режими заокруглення, зокрема округлення до $\pm\infty$ та округлення до 0, можуть мати похибку вже 1 ULP.

Число вважається денормалізованим (subnormal), якщо зміщений порядок числа $E = 0$. Тоді мантиса не містить прихованої одиниці у цілій частині:

$$M_x = 0.m_1m_2\dots m_{p-1}m_p, \quad (1.12)$$

а в якості порядку береться найменше дозволене значення $E_x = E_{\min}$. Денормалізовані числа дозволяють представляти близькі до 0 значення, тобто FP числа, що є меншими за модулем за найменше нормалізоване число. Наприклад, для чисел одинарної точності найменше денормалізоване число приблизно рівне $1,4 \cdot 10^{-45}$. Це дозволяє зменшити ефект антипереповнення (underflow) в FP арифметиці, але при цьому втрачається точність представлення чисел. Крім того, стандарт визначає, що деякі комбінації мантиси M та порядку E використовуються для спеціальних значень ($+0$ та -0 , $+\infty$ та $-\infty$, NaN).

Хоча цей стандарт є досить складним та його важко реалізувати ефективно, він дозволяє забезпечити сумісність різних програмних та апаратних засобів та в більшості випадків однаковий результат FP обчислень. Проте, бінарна FP арифметика має певні недоліки, які потрібно враховувати, особливо в галузі кібербезпеки (КБ) та захисту інформації (ЗІ): накопичення похибок округлень; результат може залежати від послідовності виконання операцій; можливі катастрофічна втрата точності та серйозні помилки обчислень (наприклад, віднімання досить близьких значень або операції із числами, що мають велику різницю порядків) [89-92]. Крім того, FP арифметика не відповідає властивостям поля (наприклад сума FP чисел $a+b$ може не бути FP числом, FP числа не володіють асоціативністю, не забезпечують дистрибутивність, не для всіх значень $a \neq 0$ існує обернене) [93].

Зважаючи на це, досить складно гарантувати, що на практиці одна й та сама програма на різних комп'ютерних пристроях із різними операційними системами (ОС) та компіляторами завжди буде повертати однаковий результат [94]. Проте,

завдяки використанню стандарту IEEE 754 можна оцінити похибки заокруглень та похибки наближення функцій, математично довести правильність результату та забезпечити, щоб алгоритм обчислень був портативним. Деякі методи підвищення портативності алгоритму, наприклад відключення розширених функцій FP арифметики, реалізація довгої арифметики чи арифметики довільної точності, обмежують продуктивність [21,94]. Серед інших методів забезпечення точності можна виділити: керування послідовністю виконання операцій, детальне налаштування компілятора, застосування перетворень без похибок, сортування аргументів за величиною, застосування проміжних типів даних вищої точності, зміна основи системи числення, використання математичних бібліотек для обчислення елементарних функцій із повною точністю тощо [93-95].

1.2 Аналіз відомих методів обчислення елементарних функцій у FP арифметиці для сфери кібербезпеки та захисту інформації

Під елементарними функціями будемо розуміти такі трансцендентні та алгебраїчні математичні функції, що найчастіше використовуються в науці та інженерії. До трансцендентних функцій відносяться тригонометричні (наприклад, $\sin x$, $\cos x$, $\tan x$, їх обернені та гіперболічні варіанти), показникові (e^x , 2^x тощо) та логарифмічні функції ($\log_2 x$, $\ln x$, $\lg x$ тощо), які неможливо представити у поліноміальному вигляді, тобто як послідовність операцій $+$, $-$, $\times$, $\div$ та піднесення до цілого чи дробового степеню. Прикладами алгебраїчних функцій є $1/x$ (RCP), $\sqrt{x}$ (SQRT), $1/\sqrt{x}$ (RSQRT), $\sqrt[3]{x}$ (CBRT), $1/\sqrt[3]{x}$ (RCBRT) та $\sqrt{a^2 + b^2}$. У даній дисертації ми зосередимось переважно на ефективній програмній реалізації таких простих алгебраїчних (ступеневих) функцій, а також операції ділення (DIV), в FP арифметиці для чисел одинарної (тип *float*) та подвійної (тип *double*) точності.

Програмна та апаратна підтримка елементарних функцій

На більшості комп'ютерних систем загального призначення (CPU) є можливість обчислення основних елементарних функцій використовуючи програмні засоби математичних бібліотек. Найпопулярнішим прикладом такої

бібліотеки є `cmath` (`math.h`) для мов програмування C/C++. Зокрема, серед функцій, що доступні в цьому заголовному файлі можна виділити тригонометричні та гіперболічні функції, показникові та логарифмічні, піднесення до степеню $\text{pow}(x, a)$ та отримання квадратного кореня $\text{sqrt}(x)$. Також після стандарту C++11 появилась підтримка кубічного кореня $\text{cbrt}(x)$, обчислення гіпотенузи $\text{hypot}(a, b)$ та поєднаного множення-додавання $\text{fma}(a, x, b)$. Усі ці функції мають версії для трьох основних FP типів даних у C/C++ – *float*, *double* та *long double* (тип *long double* в залежності від платформи може займати 8, 10, 12 чи 16 байт). На даний момент досить мало платформ апаратно підтримують 128-бітні FP числа відповідно до стандарту IEEE 754. Найчастіше, пристрої із FPU підтримують лише операції над FP числами типу *float*, що оперують 32-бітними значеннями. Точність та швидкодія більшості бібліотечних функцій залежить від компілятора, архітектури та використовуваних типів. Важливо, що відповідно до стандарту IEEE 754 функції $\text{sqrt}(x)$ та $\text{fma}(a, x, b)$ забезпечують повну точність результату [40]. Це означає, що округлений результат обчислення функції для вказаного типу даних стандарту IEEE 754 при вибраному режимі округлення буде найближчим до математично точного значення Y , що містить необмежену кількість точних знаків (похибка в 0,5 ULP у режимі округлення до найближчого).

Стандартну бібліотеку C, що підтримує більшість ОС та архітектур, досить часто критикують за повільну швидкодію та низьку оптимізацію функцій. Тому активно розробляються альтернативні математичні бібліотеки. На цей час існує багато різних бібліотек обчислення математичних функцій, комерційних та безоплатних, доступних для різних ОС, платформ, мов програмування та компіляторів чи розроблених виробниками конкретної платформи та оптимізованих для роботи на ній. Серед таких бібліотек можна виділити GNU бібліотеку мови C (`glibc`) [40], `Mathlib` [96] від ARM, `LibM` [60] від AMD, `MKL` [98] від Intel, `GNU MPFR` [22,99] від INRIA, `CR-libm` [21,100,101] від ENS Lyon, `AMathLib (libultim)` [19,102] від IBM, `Libmcr` [103] від Sun Microsystems тощо. Лише деякі із вказаних бібліотек забезпечують коректно округлений результат обчислення елементарних функцій, зокрема `MPFR`, `CR-libm`, `Libmcr` та

AMathLib. Проте, бібліотеки CR-libm та AMathLib забезпечують правильне округлення лише для чисел типу *double*. MPFR та Libmcr застосовують арифметику довільної точності, а тому їх продуктивність в рази повільніша за інші методи. Крім досить потужної та оптимізованої бібліотеки Intel Math Kernel Library (MKL), для обчислення математичних функцій можна також використовувати бібліотеки Intel Math Library (LIBM) та Short Vector Math Library (SVML) [104] для компілятора Intel C++. Аналогічно, власні реалізації основних математичних функцій пропонує компілятор Microsoft Visual C++ [105]. Слід зазначити, що деякі із згаданих бібліотек мають досить обмежений набір функцій, проте досить часто зустрічаються функції $\sqrt{x}$ чи $\text{pow}(x, a)$, трохи рідше $\sqrt[3]{x}$, $\text{inv_sqrt}(x)$ та $\text{hypot}(a, b)$. Розробляються також математичні бібліотеки для FP чисел, спеціально призначені для мікроконтролерів (МК) та вбудованих пристроїв, наприклад GoFast [106] компанії Micro Digital, Newlib (libm) [107] від Red Hat, FLIP [108] від ENS Lyon, SEGGER [109] від SEGGER Microcontroller GmbH та Qfplib [110]. Такі пристрої зазвичай мають певні обмеження на ресурси (обчислювальна здатність, пам'ять, енергозабезпечення, апаратна підтримка FP операцій тощо). Якщо деякі FP операції не підтримуються CPU, тоді вони емулюються програмно, наприклад на базі цілочислових операцій за правилами арифметики із фіксованою комою. Як зазначає О. Купріянова у своїй дисертації [5], наразі наявні бібліотеки не дають можливості користувачу вибирати варіанти реалізації математичних функцій для чисел певного типу, що мають декілька рівнів точності та швидкодії. Особливо актуальним це може бути для обмежених пристроїв. Інколи, в залежності від застосування та наявних ресурсів потрібно отримати або максимально точний результат, наприклад для криптографії, або грубе наближення функції, наприклад для поведінкового аналізу та стеганографії.

Математичні співпроцесори (FPU) багатьох CPU, наприклад Intel Core i7 та ARM Cortex A-53, мають швидкі апаратні інструкції SQRT, DIV та FMA, які відповідно до вимог стандарту IEEE 754 забезпечують повну точність результату та коректне опрацювання виняткових ситуацій [8]. Такі інструкції є переважно для FP чисел одинарної та подвійної точності. Також, можуть бути наявні

наближені апаратні інструкції RCP та RSQRT. Зокрема, подібні інструкції містяться у наборі системи команд (ISA) AMD 3DNow! [111], Intel SSE/AVX [112], ARM NEON [46] та Freescale AltiVec [113]. Такі наближені інструкції є корисними для багатьох задач статистики, комп'ютерної графіки, а також кібербезпеки, де важлива саме швидкодія та не вимагається висока точність обчислень, наприклад, для систем аналізу та прийняття рішень, виявлення вторгнень та аномалій, поведінкового аналізу та стеганографії, нормалізації векторів, побудови тіней та освітлення, проекцій та геометричного перетворення об'єктів, біометричної ідентифікації тощо. Недоліком програмних методів є їх відносно низька швидкодія, а недоліком спеціальних апаратних інструкцій є їх залежність від платформи. Хоча їх точність та швидкодія може сильно відрізнятися, проте далеко не всі платформи, такі як МК, підтримують подібні інструкції. Інструкції RCP та RSQRT від Intel мають приблизно 12 біт точності [112], а в процесорах архітектури ARM – 8 біт [8,44]. Проте, є також інструкції різних технологій, що дають 14, 23 або 28 біт точності [1,8,112]. Слід зазначити, що для Intel немає інструкцій RCP та RSQRT для чисел подвійної точності. Навіть якщо апаратні інструкції є реалізовані, вони можуть бути недоступні через відсутність підтримки ОС. Сучасні платформи також досить часто підтримують векторні (SIMD) варіанти таких наближених інструкцій.

Проблема коректного заокруглення результату

Отримання правильно заокругленого результату для всіх можливих вхідних значень функції через похибки FP обчислень та заокруглень може бути досить складною практичною задачею, що пов'язана із значними накладними витратами. Для розв'язання цієї проблеми можна використовувати проміжний тип даних більшої розрядності (це стосується як бітів мантиси, так і порядку). Даний підхід зазвичай використовується при розробці апаратних пристроїв, зокрема на ПЛІС. Він вимагає також реалізації усіх необхідних операцій для цього розширеного типу (відповідні суматори, помножувачі, регістри тощо). Зокрема, процесори Intel із часу появи FPU 8087 використовують 80-бітний формат розширеної точності. Такий підхід можна застосовувати також і при програмній реалізації на основі

типів, які або реалізовані апаратно, або емулюються за прийнятний час. Альтернативний підхід, досить поширений в літературі, полягає в ігноруванні проблеми коректного заокруглення.

Проблема отримання коректно округленого результату обчислення трансцендентних функцій у закордонній літературі має назву дилеми творця таблиці (table-maker's dilemma). Вона полягає в тому, що в загальному випадку наперед не відомо з якою точністю m потрібно обчислити наближення функції на всій області визначення, щоб гарантувати правильне округлення результату до n точних біт. Для цього, або шукають такі проблемні точки (worst cases) x , де значення функції буде дуже близьким до певного FP числа чи буде лежати на границі двох таких чисел (залежно від режиму округлення), або вибирають значення m значно більшим за n ($m \gg n$) [17]. Наприклад, обчислення проблемних точок для функції e^x методом повного перебору показало, що для чисел подвійної точності ($n=53$) достатньо $m=157$ біт для правильного округлення [100]. Математична бібліотека IBM AMathLib [102], що базується на методі Зіва [19], в найгіршому випадку використовує точність наближення $m=768$ біт [17]. Для основних операцій потрібно $m=2n$ біт точності. Щодо елементарних алгебраїчних функцій, то Х. Йордаш та Д. Матула у [18] показали, що для коректно округленого результату RCP, DIV та SQRT в FP арифметиці необхідно обчислити $2n + O(1)$ точних біти, для функції RSQRT – $3n + O(1)$.

Як бачимо, без використання проміжних типів даних вищої точності вирішити цю задачу досить складно. Тому, актуальною задачею є пошук простих та ефективних універсальних методів обчислення елементарних функцій, що дозволять отримати коректне заокруглення або якнайвищу точність результату, зокрема для застосувань у сфері КБ та ЗІ, на основі наявних апаратних ресурсів.

Класифікація відомих методів обчислення елементарних функцій

Серед найпоширеніших методів, що використовуються для обчислення елементарних функцій, можна виділити наступні класи [3,8,11,114,115]:

рекурентні (digit recurrence), цифра-за-цифрою (digit-by-digit), ітераційні, поліноміальні, дробово-раціональні, на основі таблиць (LUT) та їх комбінації.

1) *Рекурентні методи та метод цифра-за-цифрою.*

Рекурентні методи та метод «цифра-за-цифрою» дозволяють отримати за ітерацію один біт або одну цифру результату. Обчислення починається із старших розрядів та використовує зазвичай найпростіші арифметичні операції (такі як $+$, $-$, бітовий зсув тощо). А тому ці методи добре підходять для апаратної реалізації на пристроях із обмеженими обчислювальними можливостями. Ще однією перевагою рекурентних методів є гарантована можливість одержання правильного результату будь-якої точності за наперед відому кількість ітерацій. Проте, зважаючи на їх лінійну збіжність, такі методи будуть досить повільними, особливо при їх програмній реалізації. Для підвищення збіжності можуть застосовуватись методи із високою основою системи числення (high radix). Для прикладу, у сучасних процесорах для DIV зазвичай застосовується алгоритм SRT (Sweeney, Robertson, Tocher), що базується на рекурентному методі повільного ділення без відновлення та використовує LUT, завдяки чому дає найкращий компроміс між затримкою та використаними ресурсами [43]. Іншим прикладом досить популярного простого та універсального ітераційного методу типу цифра-за-цифрою, що може використовуватись для обчислення багатьох елементарних функцій (тригонометричних, експоненціальних, SQRT, логарифму, гіпотенузи, DIV тощо), є метод CORDIC (COordinate Rotation DIgital Computer). Перевагою CORDIC є те, що він не вимагає операцій $\times$, а тому є незамінним для таких пристроїв, що не мають апаратно реалізованих помножувачів.

2) *Ітераційні методи.*

Значно кращу збіжність мають ітераційні методи. Вони широко використовують операцію $\times$ та потребують початкового наближення. Найчастіше в літературі для уточнення функцій RCP та RSQRT використовуються методи Ньютона-Рафсона (NR) та Гольдшмідта, що мають квадратичну збіжність. Деякі ітераційні методи можуть також вимагати $\div$ на кожній ітерації, наприклад під час обчислення SQRT. А тому ці методи будуть неефективними на платформах, де ця

операція є ресурсозатратною, наприклад МК та ПЛІС. Перевагою методу NR є властивість самокорекції похибок [116], а перевагою методу Гольдшмідта є можливість паралельного обчислення SQRT та RSQRT [117]. Популярні також методи вищих порядків, наприклад метод Хаусхолдера (H).

Як правило, чим точніше підібрано початкове значення, тим меншу кількість ітерацій потрібно для результату необхідної точності. Тому для початкового наближення зазвичай використовуються LUT або спеціалізовані апаратні інструкції. Якщо застосовувати “грубі” методи наближення [118], виникає питання ефективності та збіжності ітераційного методу. В літературі також набуває популярності простий та дієвий спосіб отримання початкового наближення, що базується на особливостях представлення в пам’яті FP чисел стандарту IEEE 754 та використовує так звану “магічну константу” (MC) [4,6,119,120]. Детальніше цей метод розглянемо в розділах 1.3 та 1.4.

3) Поліноміальні методи.

Поліноміальні методи полягають в апроксимації функцій поліномами певного степеня (лінійним, квадратичним, кубічним наближенням тощо). Таким чином для реалізації цих методів потрібні лише операції $+$, $-$, $\times$ та, можливо, піднесення до цілого степеня. Поліноміальні методи зазвичай застосовуються для наближення складних функцій на визначеному проміжку $x \in [a; b]$. Оскільки багато трансцендентних функцій є періодичними, можливе приведення вхідного аргументу на цей проміжок. Після застосування методу, відбувається зворотне перетворення. Степінь полінома сильно залежить від особливостей функції та розміру інтервалу апроксимації $[a; b]$, а тому цей проміжок може розбиватись на декілька рівних або спеціальних піддіапазонів. Прикладом такого методу є кусково-лінійна апроксимація. Крім того, степені відповідних поліномів можуть відрізнятись (цей підхід використовує багато математичних бібліотек). Проте, з огляду на значні витрати реалізації поліноміальних методів, дані методи є не дуже привабливими для обчислення елементарних функцій на ПЛІС [121]. Велика кількість послідовних FP $\times$ впливає на затримку (латентність) алгоритмів. Для підвищення ефективності методу в літературі [122,123] пропонуються наступні

підходи: для зменшення кількості $\times$ та $+/-$ поліноми формуються за схемою Горнера; для розпаралелювання обчислень застосовують схему Естріна та, по можливості, поліноми записують у вигляді добутку поліномів низьких порядків.

Розв'язок задачі апроксимації та його точність залежить від критеріїв визначення коефіцієнтів апроксимаційного полінома. Розрізняють інтерполяційне (точкове), середньоквадратичне та рівномірне наближення. Поліноміальні апроксимації можуть бути отримані різними методами, наприклад через розклад у ряд Тейлора або використання інтерполяційного полінома Чебишева. Проте, найефективніші результати дає поліном найкращого рівномірного (мінімаксного) наближення, що базується на основі теореми Чебишева про альтернанс [124]. Для знаходження коефіцієнтів даного найкращого наближення застосовують чисельні методи, зокрема ітераційний метод Ремеза [125-128].

4) Дробово-раціональні методи.

Підняти точність наближень можна використовуючи замість поліноміальної дробово-раціональну апроксимацію. При цьому застосовується ділення двох поліномів різних порядків (раціональний дріб). Хоча для такого представлення функцій також можна застосовувати найкраще рівномірне наближення із меншим еквівалентним порядком поліномів [10,125,129], воно вимагає виконання $\div$.

5) Табличні методи.

LUT використовуються для того, щоб полегшити чи пришвидшити певні кроки алгоритму [3]. Найпростіший, але неефективний метод побудови LUT – це табуляція, що характеризується значним використанням пам'яті. Розмір LUT росте експоненціально залежно від необхідної точності, що суттєво ускладнює практичне застосування методу у чистому вигляді, а тому варто мінімізувати кількість збережених значень та їх розмір [122]. Якщо точність вхідного аргументу x та результату обчислення функції $y(x)$ відносно невелика, наприклад по 8 біт, тоді використання LUT буде швидким та ефективним [130]. Особливо гостро ця проблема виникає на простих МК, оскільки на сучасних ПЛІС є досить багато базових елементів для реалізації пам'яті (розподіленої чи блокової RAM), а кількість помножувачів навпаки є обмеженою. Тому найсучасніші методи для

отримання початкового наближення у CPU та ПЛІС використовують апаратні алгоритми на основі LUT. Як приклад можна назвати швидкі наближені інструкції RCP і RSQRT від Intel і AMD [20,131]. Також LUT широко застосовуються у трансцендентних інструкціях [132]. Ітераційні методи вносять істотну затримку, тому, залежно від платформи та типу реалізації, потрібно знайти хороший компроміс між розміром LUT і кількістю ітерацій. Методи двосторонніх (bipartite) [3,133] і багатосторонніх (multipartite) [134,135] LUT використовують попереднє опрацювання та постобробку даних, застосовуючи дві або більше паралельних таблиць і операцію $+$. Двосторонні таблиці ділять вихідний проміжок на інтервали, які додатково поділяються на менші підінтервали, де функція апроксимується лінійною інтерполяцією зі спільним нахилом [3]. Ці методи є значно ефективнішими за звичайні LUT [122,135]. Також, існують методи на основі поліноміальної апроксимації [122,136] та на основі каскадів LUT [130].

Таким чином, оптимальними універсальними методами для обчислення функцій RCP, DIV, RSQRT та SQRT, RCBRT та CBRT в FP арифметиці для сфери КБ та ЗІ будуть ітераційні методи NR квадратичної збіжності та методи вищих порядків без використання $\div$. Залежно від платформи та конкретних вимог щодо рівня точності та швидкодії, для початкового наближення доцільно застосовувати: метод магічної константи (MC), поліноміальну мінімаксну апроксимацію чи апаратні наближені інструкції RCP та RSQRT. Зокрема, класичний алгоритм швидкого зворотного квадратного кореня (FISR), що поєднує в собі простий та швидкий метод MC та ітерації NR для обчислення RSQRT, може бути вдосконалений та узагальнений для інших елементарних функцій. Мають бути розроблені нові покращені методи, що будуть придатними як для програмної, так і для ефективної апаратної реалізації на різноманітних платформах, зокрема МК.

1.3 Дослідження алгоритму FISR та його модифікацій для обчислення функції RSQRT

Варто зазначити, що в англomовній літературі усталене поняття “inverse square root” вживається у значенні “reciprocal” (або ж “multiplicative inverse”) до

функції SQRT. Це стосується, зокрема, назви методу FISR – fast inverse square root. Тому, в літературі алгоритми обчислення функції RSQRT часто позначаються як InvSqrt. Будемо також використовувати дане позначення.

Основи методу FISR для обчислення $y = 1/\sqrt{x}$ були закладені Д. Бліном у [6] для 32-бітних FP чисел. Цей метод базується на особливостях представлення чисел у FP арифметиці стандарту IEEE 754 та на лінійній залежності між двійковими логарифмами аргументу x та очікуваного результату [119]. При цьому використовується певна маніпуляція із бітами вихідного числа, т. з. “бітовий трюк”, на основі операцій бітового зсуву та цілочислового –.

Найвідоміша версія алгоритму була реалізована на C в комп’ютерній грі Quake III Arena [137], випущеній у 1999 році, та отримала широке обговорення в Інтернеті серед фахівців із комп’ютерної графіки і науковців після публікації вихідного коду гри. Обчислення RSQRT інтенсивно використовувалось у грі для нормалізації векторів під час побудови освітлення та тіней згідно з формулою:

$$\hat{u} = \frac{\vec{u}}{|\vec{u}|} = \frac{(a, b, c)}{\sqrt{a^2 + b^2 + c^2}}. \quad (1.13)$$

В оригінальній версії алгоритму FISR друга ітерація NR закоментована. Це означає, що алгоритм дозволив отримати результат необхідної точності – 9 біт – вже після першої ітерації. Цей метод широко використовується як для програмної [1,11,14,81,137-141], так і для апаратної реалізації [24,142-144] в багатьох прикладних задачах. Зокрема, багато робіт присвячено дослідженню та вдосконаленню даного алгоритму, а також спробам виведення оптимальних MC.

Оригінальний алгоритм використовував MC $R = 0x5f3759df$ для початкового наближення та ітераційний метод NR (одну або дві ітерації) для його уточнення. Використаний метод MC дозволив отримати досить швидко та точне кусково-лінійне значення початкового наближення для функції RSQRT мінімізуючи при цьому значення максимальної відносної похибки ($\delta_{\max}$). При цьому використовуються лише операції у цілочисловій та FP арифметиці. Проте, як показав К. Ломонт [120] на основі числових досліджень, вказана MC не буде оптимальною. Для початкового наближення кращою буде дещо модифікована

константа $R = 0x5f37642f$, а константа $R = 0x5f375a86$ буде кращою після першої та після другої класичної ітерації NR. Пізніше ці результати було підтверджено аналітично, зокрема в роботах [23,138,145]. Тому в лістингу 1.1 наведемо код класичного алгоритму FISR із модифікованою константою Ломонта.

Лістинг 1.1 – Алгоритм FISR із магічною константою Ломонта [120]

```

1: float InvSqrt1(float x)
2: {
3:     float xhalf = 0.5f*x;
4:     int i = *(int*)&x;          // приведення float до integer  $I_x$ 
5:     i = 0x5f375a86 - (i>>1);    // обчислення  $I_{y_0}$  на основі константи  $R$ ,
                                   // цілочислове ділення  $I_x$  на 2 та зміна знаку
6:     float y = *(float*)&i;      // приведення integer  $I_{y_0}$  до float,
                                   // отримання початкового наближення  $y_0$ 
7:     y = y*(1.5f - xhalf*y*y);   // 1-ша ітерація NR
8:     y = y*(1.5f - xhalf*y*y);   // 2-га ітерація NR
9:     return y;
10: }
```

Розглянемо далі теоретичні основи алгоритму FISR.

1.3.1 Магічна константа

Нехай маємо x у вигляді (1.1), $x > 0$. Позначимо через $\iota(x)$ перетворення, що приймає FP число x та повертає ціле i , що має таке ж бітове представлення. Аналогічно, функція $\varphi(i)$ визначає зворотнє перетворення, що приймає ціле число i та повертає FP число. Тоді з x отримаємо I_x (див. рядок 4 у лістингу 1.1):

$$I_x = \iota(x) = E \cdot N_m + M = (bias + E_x + m_x)N_m, \quad (1.14)$$

де $N_m = 2^P$, зокрема із табл. 1.1 для чисел *float* $N_m = 2^{23}$, $bias = 127$.

Згідно з формулою (1.2), експоненту E_x числа x можна розглядати як грубе наближення двійкового логарифму $\log_2 x$. Ця властивість часто використовується для наближення експоненціальних та логарифмічних функцій, зокрема у галузі

інформаційної та кібербезпеки для нейронних мереж та цифрової обробки сигналів [4]. Точніші результати дає формула, відома як апроксимація Мітчела:

$$\log_2 x = E_x + \log_2(1 + m_x) \approx E_x + m_x, \quad (1.15)$$

оскільки $m_x \in [0; 1)$. Крім того, Ч. Мак-Енірі [119] пропонує внести у (1.15) параметр k , що дозволить регулювати похибку наближення в алгоритмі FISR:

$$\log_2 x \approx E_x + m_x + k. \quad (1.16)$$

Як бачимо з рис. 1.2, функція $y = \log_2 x$ наближається прямою на проміжку $x \in [1; 2)$. В результаті отримаємо кусково-лінійну апроксимацію Мак-Енірі. Тут параметр k визначено на основі МС $R = 0x5f375a86$ у вигляді $k = 0,04503329$.

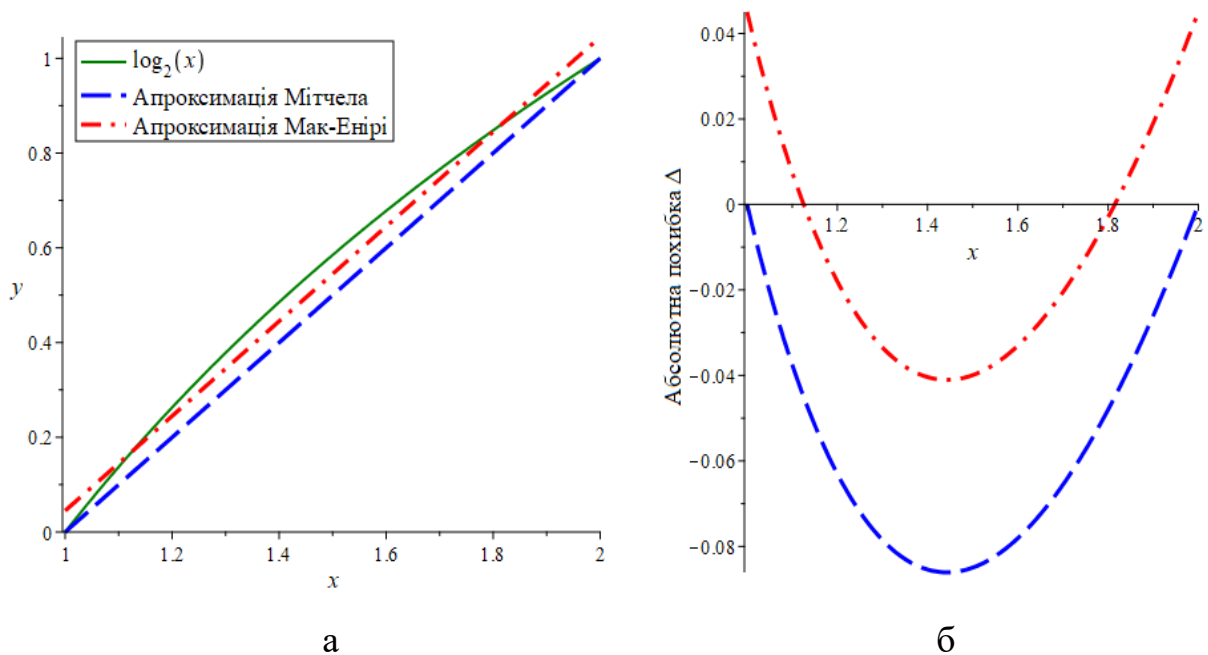


Рисунок 1.2 – (а) Наближення $\log_2 x$ методами Мітчела і Мак-Енірі на проміжку $x \in [1; 2)$ та (б) відповідні абсолютні похибки алгоритмів

Із формул (1.14) та (1.16) випливає

$$I_x \approx (bias + \log_2 x - k)N_m. \quad (1.17)$$

Як бачимо, $I_x = \iota(x)$ можна розглядати як зміщений та масштабований $\log_2 x$ (“квазілогарифм”) [6]. Аналогічно до (1.17) позначимо через $I_{y_0} = \iota(y_0)$

цілочислове представлення результату $y_0 \approx 1/\sqrt{x}$, помножимо ліву і праву частину виразу (1.17) на $-\frac{1}{2}$ та скористаємось властивістю $\log_2 y = -\frac{1}{2}\log_2 x$ [119]:

$$-\frac{1}{2}I_x \approx (-\frac{1}{2}bias - \frac{1}{2}\log_2 x + \frac{1}{2}k)N_m = (-\frac{1}{2}bias + \log_2 y + \frac{1}{2}k)N_m. \quad (1.18)$$

Компенсувати утворене зміщення між I_{y_0} можна константою R

$$R = Q \cdot N_m + T = (\frac{3}{2}bias - \frac{3}{2}k)N_m = (bias + \lfloor bias/2 \rfloor)N_m + \lfloor t \cdot N_m \rfloor, \quad (1.19)$$

де Q – порядок МС, T – цілочислова мантиса МС, а параметри t і k пов'язані співвідношенням $t = 0,5 - 1,5k$. В результаті отримаємо вираз для знаходження I_{y_0} (див. рядок 5 у лістингу 1.1):

$$I_{y_0} = R - \lfloor I_x / 2 \rfloor. \quad (1.20)$$

Наприклад, для МС $R = 0x5f375a86$ із алгоритму InvSqrt1 $Q = 190, T = 3627654, t = 0,43245006$. При цьому слід враховувати особливості виконання операції бітового зсуву $I_x \gg 1$, що є аналогом цілочислового ділення на 2. Зокрема, біт знаку $S_x = 0$ зміщується на позицію старшого біту експоненти E_{b-1} , молодший біт експоненти E_0 переміщається на позицію старшого біту мантиси M_{p-1} , а молодший біт мантиси M_0 відкидається:

$$I_x \gg 1 = \lfloor I_x / 2 \rfloor = \lfloor E / 2 \rfloor \cdot N_m + E_0 \cdot 2^{p-1} + \lfloor M / 2 \rfloor. \quad (1.21)$$

Тоді із (1.19)-(1.21) отримаємо

$$I_{y_0} = (bias + \lfloor bias/2 \rfloor - \lfloor E / 2 \rfloor) \cdot N_m + T - E_0 \cdot 2^{p-1} - \lfloor M / 2 \rfloor, \quad (1.22)$$

що відповідає значенню початкового наближення y_0 згідно з методом МС

$$y_0 = \varphi(I_{y_0}) = M_{y_0} \cdot 2^{E_{y_0}} = (1 + \frac{1}{N_m}(T - E_0 \cdot 2^{p-1} - \lfloor M / 2 \rfloor)) \cdot 2^{\lfloor bias/2 \rfloor - \lfloor E/2 \rfloor}. \quad (1.23)$$

З іншого боку, точне значення функції $y = 1/\sqrt{x}$ може бути отримане [23] з

$$\begin{aligned} y &= M_y \cdot 2^{E_y} = 1/\sqrt{M_x \cdot 2^{E_x}} = (1/\sqrt{M_x}) \cdot 2^{-E_x/2} = (1/\sqrt{M_x}) \cdot 2^{bias/2 - E/2} = \\ &= (1/\sqrt{M_x}) \cdot 2^{\lfloor bias/2 \rfloor + 0,5 - \lfloor E/2 \rfloor - E_0/2} = \frac{(\sqrt{2})^{1-E_0}}{\sqrt{M_x}} \cdot 2^{\lfloor bias/2 \rfloor - \lfloor E/2 \rfloor} = \frac{(\sqrt{2})^{1-E_0}}{\sqrt{1+M/N_m}} \cdot 2^{\lfloor bias/2 \rfloor - \lfloor E/2 \rfloor}. \end{aligned} \quad (1.24)$$

Як бачимо із формул (1.23) та (1.24), порядок однаковий у двох виразах

$$E_{y_0} = E_y = \lfloor bias/2 \rfloor - \lfloor E/2 \rfloor, \quad (1.25)$$

а отже алгоритм FISR коректно визначає експоненту результату, а мантиса залежить від значення E_0 , причому M_{y_0} використовується для наближення M_y :

$$M_{y_0} = \begin{cases} 1 + \frac{1}{N_m}(T - \lfloor M/2 \rfloor) \approx \frac{\sqrt{2}}{\sqrt{M_x}}, & \text{якщо } E_0 = 0; \\ 1 + \frac{1}{N_m}(T - 2^{p-1} - \lfloor M/2 \rfloor) \approx \frac{1}{\sqrt{M_x}}, & \text{якщо } E_0 = 1. \end{cases} \quad (1.26)$$

Треба також розглядати варіант коли $\lfloor M/2 \rfloor > T$. Загалом при аналізі (1.23) виділяють 3 можливі випадки. Як було показано у [6,138,145], для того щоб знайти відносну похибку y_0 на всьому діапазоні нормалізованих FP чисел достатньо описати її поведінку в діапазоні $x \in [1; 4)$. Зокрема, загальну форму аналітичного представлення початкового наближення y_0 методу МС, що складається із трьох кусково-лінійних частин, було виведено в роботах [138,145]. Максимальна відносна похибка такого представлення не перевищує значення $1/(2N_m)$. Запишемо для зручності y_0 у вигляді наступної системи (див. рис. 1.3):

$$y_{01} = -\frac{1}{4}x + 1 + \frac{1}{2}t + \frac{1}{4N_m}, \quad x \in [1; 2), \quad (1.27)$$

$$y_{02} = -\frac{1}{8}x + \frac{3}{4} + \frac{1}{2}t + \frac{1}{4N_m}, \quad x \in [2; x'), \quad (1.28)$$

$$y_{03} = -\frac{1}{16}x + \frac{5}{8} + \frac{1}{4}t + \frac{1}{8N_m}, \quad x \in [x'; 4). \quad (1.29)$$

Тут точка з'єднання y_{02} та y_{03} , позначена як x' , визначається формулою

$$x' = 2 + 4t + 2/N_m. \quad (1.30)$$

Аналогічні результати із відносною похибкою $\delta_{0 \max} = 3,4365 \cdot 10^{-2}$ (більше 4 точних бітів) матимемо для будь-якого діапазону нормалізованих FP чисел вигляду $x \in [2^{2k}; 2^{2(k+1)})$, $k \in \mathbb{Z}$.

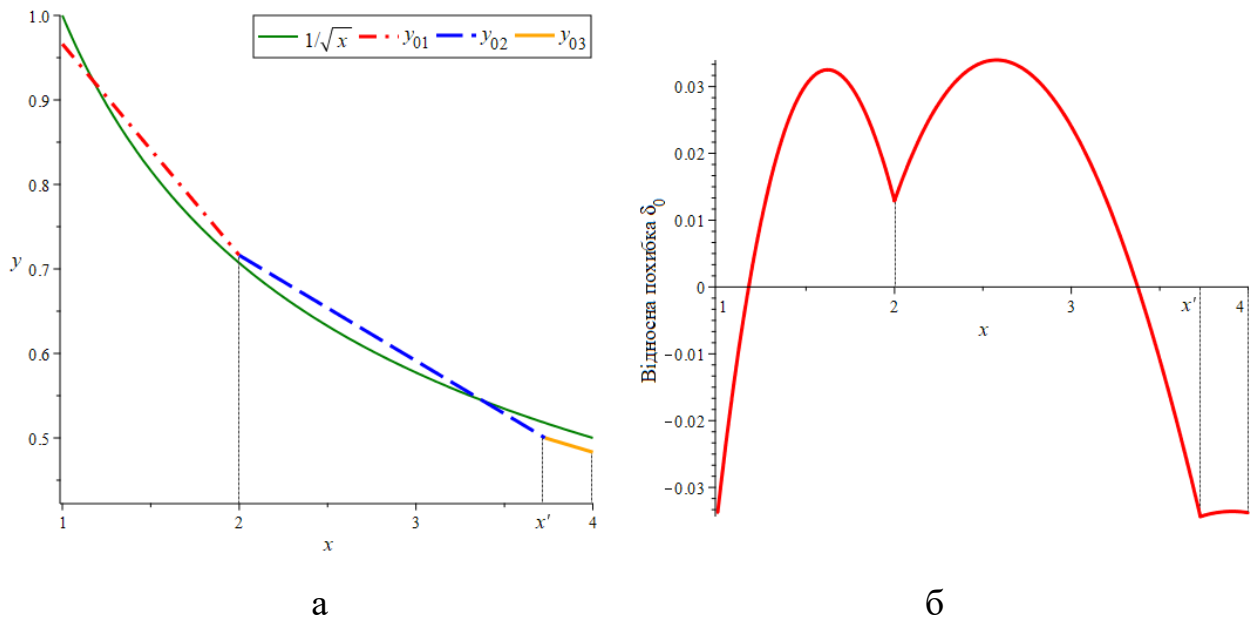


Рисунок 1.3 – (а) Початкове наближення y_0 алгоритму FISIR з МС Ломонта $R = 0x5f375a86$ на проміжку $x \in [1; 4]$. (б) Графік відносної похибки y_0

Для порівняння, в МС Бліна $R = 0x5f400000$ [6], що базується на апроксимації Мітчела, $k = 0$ та $t = 0,5$ у формулі (1.19). У цьому випадку y_0 містить лише дві кусково-лінійні ділянки y_{01} та y_{02} , а отримане початкове наближення y_0 повністю перебуває над графіком функції $y = 1/\sqrt{x}$. Значення максимальної відносної похибки y_0 складає $\delta_{0 \max} = 8,8662 \cdot 10^{-2}$.

Слід зазначити, що метод МС працює незалежно від типу даних. Потрібно лише знайти R відповідної точності, яке визначається параметром t . Значення $t = 0,432450084790142642$ отримано М. Робертсоном у [23]. Наприклад, для типу *double* маємо $Q = 1534$, $T = 1947582040717225$, а відповідне значення МС $R = 0x5fe6eb50c7b537a9$ є аналогом МС Ломонта для чисел подвійної точності.

1.3.2 Метод Ньютона-Рафсона

Далі в алгоритмі FISIR початкове наближення y_0 покращується завдяки використанню ітерацій NR для функції RSQRT (рядки 7 та 8 у лістингу 1.1):

$$y_{n+1} = y_n(1,5 - 0,5 \cdot x \cdot y_n^2), n = 0, 1, \dots \quad (1.31)$$

Ітераційний метод (1.31) отримується із загальної формули NR

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)}, n = 0, 1, \dots \quad (1.32)$$

для знаходження кореня рівняння $f(y) = 1/y^2 - x = 0$. Припустимо, що ми шукаємо лише додатний розв'язок, а початкове наближення y_0 є досить близьким до $y = 1/\sqrt{x}$. Тоді ітераційний процес має квадратичний порядок збіжності.

На рис. 1.4б,в показані аналітичні відносні похибки δ_1 та δ_2 алгоритму FISR після першої та другої ітерацій NR (y_1 та y_2) за формулою (1.31). У табл. Г.1 в додатках наведено результати порівняння реальних відносних похибок алгоритмів для типу *float*. Тут $\delta_{1\max} = 1,7513 \cdot 10^{-3}$, а $\delta_{2\max} = 4,7348 \cdot 10^{-6}$. Таким чином, алгоритм InvSqrt1 із двома ітераціями має 17 точних бітів результату. Для *double* (див. табл. Г.2) точність теж не перевищує 17 бітів ($\delta_{2\max} = 4,5973 \cdot 10^{-6}$) та 34 бітів після трьох ітерацій ($\delta_{3\max} = 3,1703 \cdot 10^{-11}$). Можна побачити, що на відміну від y_0 (див. рис. 1.3б), графіки похибок після ітерацій NR зміщені у від'ємну частину осі ординат. Це пояснюється тим, що класичний метод NR з константами $k_1 = 1,5$ та $k_2 = 0,5$ є оптимальним лише асимптотично при $y_n \rightarrow 1/\sqrt{x}$.

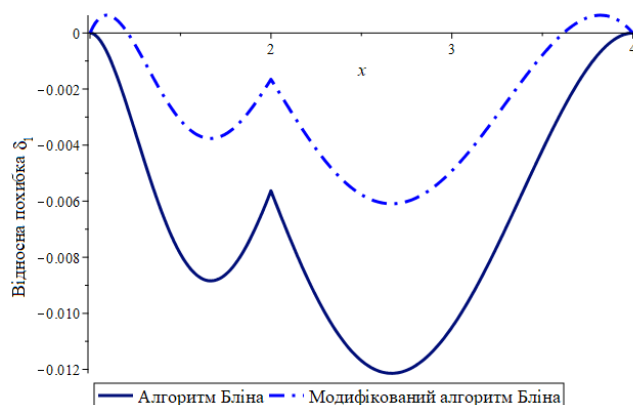
Використовуючи подібні міркування, Д. Блін [6] пропонує для початкового наближення метод MC із $R = 0x5f400000$ та модифіковану ітерацію NR

$$y_1 = y_0(k_1 - k_2 \cdot x \cdot y_0^2) \quad (1.33)$$

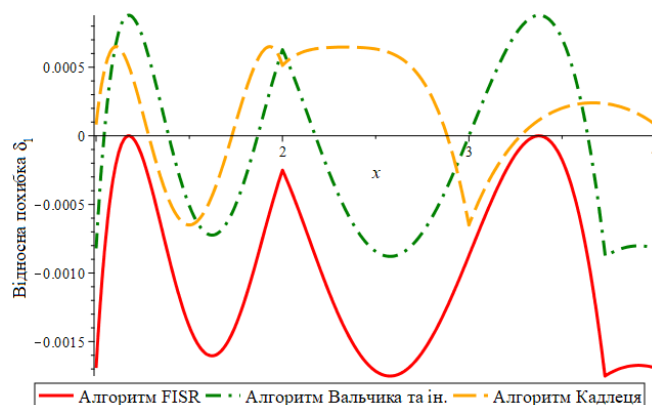
з наступними константами: $k_1 = 1,47$ та $k_2 = 0,47$. Даний підхід дозволяє зменшити $\delta_{1\max}$ приблизно вдвічі (див. рис. 1.4а). Метод адитивної корекції похибок ітерацій NR вперше був запропонований Л. Морозом у [146]. Цей підхід був використаний Ц. Вальчиком та ін. у роботах [140,147,148], зокрема, було запропоновано кілька простих модифікацій алгоритму FISR. У першому алгоритмі [147], використовується MC Ломонта $R = 0x5f375a86$ та модифікована перша ітерація NR (1.33), а друга модифікована ітерація має вигляд

$$y_2 = y_1(k_3 - k_4(k_2 \cdot x)y_1^2). \quad (1.34)$$

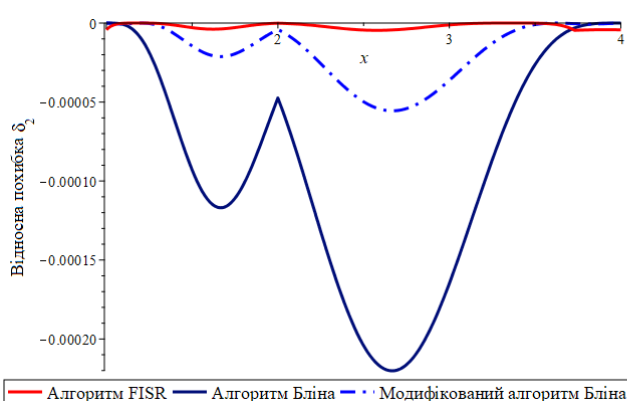
Другий алгоритм [140] використовує іншу константу $R = 0x5f376908$ та дві ітерації



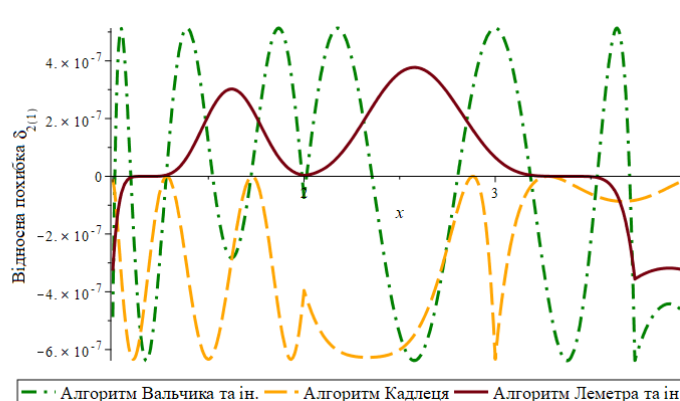
а



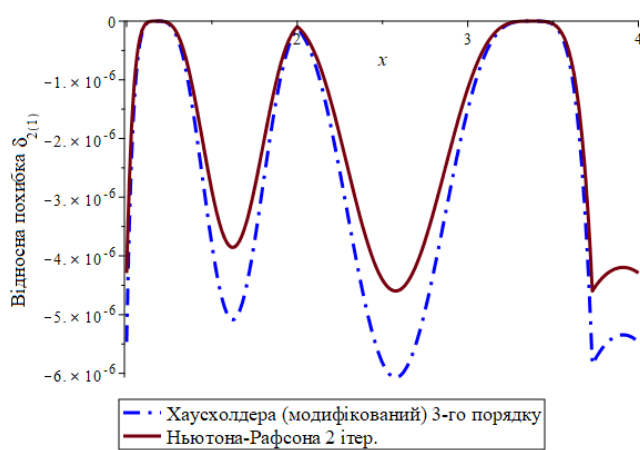
б



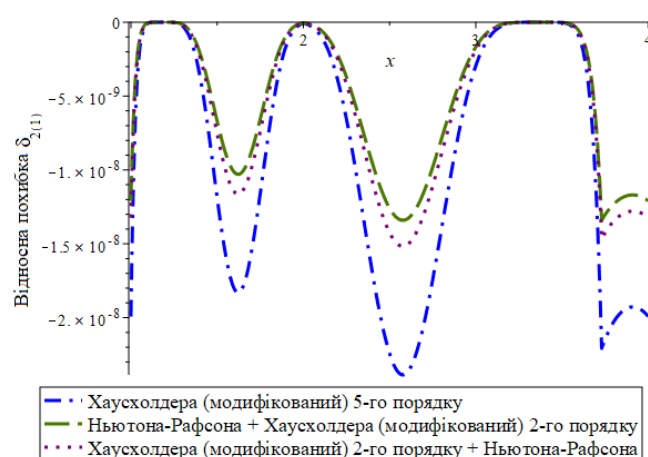
в



г



д



е

Рисунок 1.4 – Порівняння відносних похибок відомих модифікацій алгоритму FISR на основі модифікованих ітераційних методів NR та Н вищих порядків, а також їх комбінацій, після однієї чи двох ітерацій

$$y_{n+1} = y_n(k_{2n+1} - k_{2n+2} \cdot x \cdot y_n^2), \quad n = 0, 1, \dots, \quad (1.35)$$

де $k_2 = k_4 = 0,5$, завдяки чому він містить меншу кількість $\times$ після другої ітерації (див. табл. Г.1), але має трохи нижчу очікувану точність за [147]. Похибки обох алгоритмів вдвічі нижчі за похибки алгоритму *InvSqrt1* після першої ітерації (див. рис. 1.4б) та більше ніж в 6 раз нижчі після другої ітерації (див. рис. 1.4г). Це забезпечує 20 коректних бітів результату. У роботі [148] Вальчик та ін. наводять версії цих алгоритмів для чисел подвійної точності. Точність цих алгоритмів для типу *double* (табл. Г.2) теж не перевищує 20 коректних бітів для двох ітерацій ($\delta_{2\max} = 5,7617 \cdot 10^{-7}$ для першого алгоритму та $\delta_{2\max} = 5,7968 \cdot 10^{-7}$ для другого) та 41 коректного біту для трьох ітерацій (відповідно, $\delta_{3\max} = 2,4962 \cdot 10^{-13}$ та $\delta_{3\max} = 2,5229 \cdot 10^{-13}$). Зокрема, в першому алгоритмі застосовується МС Робертсона $R = 0x5fe6eb50c7b537a9$, а в другому – константа $R = 0x5fe6ed2102dcbfda$. Варто зазначити, що Ян Кадлець у [149] використав метод практичної оптимізації для пошуку таких параметрів алгоритму FISR, що мінімізують $\delta_{1\max}$ для чисел одинарної точності. Автор використовував в якості додаткового критерію пошуку також метод найменших квадратів, недоліком якого є відсутність гарантованої верхньої межі δ_1 . Тому розглянемо лише алгоритм за критерієм $\delta_{1\max}$. Тут використовується МС $R = 0x5f1ffff9$ та модифікована ітерація NR в альтернативному вигляді:

$$y_1 = k_1 \cdot y_0(k_2 - x \cdot y_0^2), \quad (1.36)$$

де $k_1 = 0,703952253$, а $k_2 = 2,38924456$. Як видно із рис. 1.4б, даний алгоритм є навіть точнішим за алгоритми Вальчика та ін. [147,148] після першої ітерації: $\delta_{1\max} = 6,5020 \cdot 10^{-4}$ (див. табл. Г.1). Третій алгоритм Вальчика та ін. [148] використовує подібну МС та дві модифіковані ітерації (1.35). Після двох ітерацій, похибка складає $\delta_{2\max} = 4,8605 \cdot 10^{-7}$, що відповідає майже 21 біту точності (див. табл. Г.1). МС для чисел типу *double* має вигляд $R = 0x5fe4000000000000c$, а точність рівна 21 біту ($\delta_{2\max} = 3,1694 \cdot 10^{-7}$).

Усі розглянуті версії алгоритму FISR використовують метод NR (1.31) у класичному чи модифікованому вигляді, що містить чотири операції $\times$ та одне – на ітерацію. Інколи, кількість $\times$ може бути зменшена до трьох, наприклад після першої ітерації в алгоритмі FISR [23,137] і Вальчика та ін. [140,148] (див. табл. Г.1-Г.2), де повторно використовується результат множення $0,5 \cdot x$. Операцію – можна об'єднати із одним $\times$, використавши FMA, що дозволить також зменшити похибки заокруглень. Це може бути особливо актуально, якщо доступна швидка реалізація операції FMA. На сучасних CPU вона виконується за таку ж кількість тактів, як і $\times$. Крім меншої кількості заокруглень, операція FMA для внутрішніх обчислень також використовує регістри вищої точності [8].

1.3.3 Метод Хаусхолдера

Ітераційний метод Н порядку $p+1$ визначається формулою [150]:

$$y_{n+1} = y_n + (p+1) \left(\frac{(1/f(y_n))^{(p)}}{(1/f(y_n))^{(p+1)}} \right), n = 0, 1, \dots, \quad (1.37)$$

де $(1/f(y_n))^{(p)}$ позначає значення похідної p -го порядку зворотної функції до $f(y_n)$. При $f(y) = 1/y^2 - x$ та $p = 0$ із формули (1.37) отримуємо метод NR (1.31). Хоча збіжність методу рівна $p+2$, він має суттєвий недолік для другого та вищих порядків: наявність $\div$ в ітераційній формулі обчислення RSQRT (див. табл. Г.3). Наприклад, метод Галлея кубічної збіжності отримується із (1.37) при $p = 1$

$$y_{n+1} = \frac{y_n(\alpha + 3)}{3\alpha + 1}, n = 0, 1, \dots, \quad (1.38)$$

де для зручності позначимо

$$\alpha = x \cdot y_n^2. \quad (1.39)$$

Метод Галлея (1.38), (1.39) є методом Н другого порядку (Ho2). Запишемо узагальнений вигляд ітераційних формул Н вищих порядків ($p > 0$) для RSQRT:

$$y_{n+1} = \frac{y_n(k_1 + \dots + \alpha(k_{(p+1)/2-1} + \alpha(k_{(p+1)/2} + \alpha)))}{k_{(p+1)/2+1} + \dots + \alpha(k_{p+1} + \alpha(k_{p+2} + \alpha))}, n = 0, 1, \dots \text{ (парного порядку);} \quad (1.40)$$

$$y_{n+1} = \frac{k_1 + \dots + \alpha(k_{p/2} + \alpha(k_{p/2+1} + \alpha))}{x \cdot y_n(k_{p/2+2} + \dots + \alpha(k_{p+1} + k_{p+2}\alpha))}, n = 0, 1, \dots \text{ (непарного порядку).} \quad (1.41)$$

Ф. Леметр та ін. [1] пропонують використовувати у FISR модифікований метод Н [151], що визначається формулою:

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)} \left(1 + \frac{f(y_n)f''(y_n)}{2!f'(y_n)^2} + \frac{f(y_n)^2(3f''(y_n)^2 - f'(y_n)f'''(y_n))}{3!f'(y_n)^4} + \dots \right), \quad (1.42)$$

$$n = 0, 1, \dots$$

Як видно із (1.42), модифікований Но1 також еквівалентний методу NR, проте ітераційні формули відрізняються від (1.37) для вищих порядків ($p > 0$):

$$y_{n+1} = y_n(k_1 + \dots + \alpha(k_{p+1} + k_{p+2}\alpha)), \quad n = 0, 1, \dots \quad (1.43)$$

Основна перевага цього методу для обчислення RSQRT у порівнянні із методом (1.37) – відсутність операції $\div$. Зокрема, у [1] рекомендується початкове наближення МС Ломонта та для чисел одинарної точності використовувати метод (1.39), (1.43) четвертого порядку (Но4), швидкість збіжності якого рівна п'яти, а коефіцієнти ітераційного методу рівні: $k_1 = \frac{315}{128}$, $k_2 = -\frac{105}{32}$, $k_3 = \frac{189}{64}$, $k_4 = -\frac{45}{32}$, $k_5 = \frac{35}{128}$.

Формула (1.43) записана за схемою Горнера для мінімізації кількості $\times$, яких у Но4 сім. Дана форма запису зручна також тим, що дозволяє використовувати FMA при обчисленні полінома, уникнувши додаткових операцій $+/-$. Недоліком схеми Горнера є неможливість розпаралелення. Графік теоретичних відносних похибок алгоритму Леметра та ін. [1] показаний на рис. 1.4г. Цей алгоритм для типу *float* має похибку $\delta_{1\max} = 6,5782 \cdot 10^{-7}$ (див. табл. Г.3), тобто на практиці забезпечує лише 20 біт точності, незважаючи на використання FMA ($3\times$, 4FMA). Для чисел подвійної точності Леметр та ін. використовують МС Робертсона та рекомендують дві ітерації модифікованого методу Но3 (див. табл. Г.4), що використовує по 6 операцій $\times$ і FMA та має практичну точність 50 біт.

Порівняння обчислювальної складності та практичних похибок алгоритмів на основі методу Н різних порядків (стандартного та модифікованого) після однієї чи кількох ітерацій наведено у табл. Г.3 та Г.4 для типів *float* і *double*. Слід зауважити, що хоча модифікований метод Н має гіршу точність, він вимагає менше обчислювальних ресурсів, що робить його привабливішим для реалізації на МК та вбудованих пристроях. Наприклад, метод Но3 ($1\div$, $4\times$, 2FMA , $1+$)

точніший (21 біт для *float* та 22 біти для *double*) за модифікований метод Но4 (3×, 4FMA). При збільшенні порядку через накопичення похибок в типі *float* точність методів майже не зміниться. Аналогічно, для *double*. Наприклад, 4 ітерації NR (9×, 4FMA) будуть мати точність 51 біт замість 69 теоретичних.

Зважаючи на використання стандартним методом операції $\div$, яка відповідно до [42] на сучасних CPU переважно у 3-8 раз повільніша за операцію $\times$, зазвичай доцільніше застосувати модифікований метод Н, але вищого порядку. Збільшення порядку на одиницю додає до формули по одній операції $\times$ та $+$. Крім того, формули (1.43) краще підходять для застосування FMA. Можна також застосовувати комбінацію із кількох ітерацій різних порядків (див. рис. 1.4е).

1.4 Обчислення інших логарифмічних, експоненціальних та степеневих елементарних функцій на основі методу МС

Узагальнення методу магічної константи

Д. Блін у [6] запропонував також подібні методи наближення інших елементарних функцій на основі МС. Зокрема, як вже зазначалось, для $y_0 \approx \log_2 x$

$$y_0 = (I_x - \text{bias} \cdot N_m) / N_m = C_m (I_x - \text{bias} \cdot N_m) = C_m (I_x - R), \quad (1.44)$$

де $C_m = 1/N_m$ – FP константа. Формула (1.44) еквівалентна (1.15) та може бути покращена щодо абсолютної похибки шляхом модифікації МС R (див. рис. 1.2). Зворотна до логарифму експоненціальна функція $y = 2^x$ апроксимується, використовуючи наступне співвідношення:

$$y_0 = \varphi(I_{y_0}) = \varphi(\lfloor x \cdot N_m \rfloor + \text{bias} \cdot N_m) = \varphi(\lfloor x \cdot N_m \rfloor + R), \quad (1.45)$$

де зміна МС дозволить мінімізувати $\delta_{0 \max}$. Інші експоненціальні, логарифмічні та степеневі функції можна визначити через математичні співвідношення як комбінацію $y = 2^x$ та $y = \log_2 x$ [6]. Загальна формула для обчислення початкового наближення $y_0 \approx x^p$ методом МС, зокрема у випадку коли степінь p можна записати у вигляді $p = \frac{l}{m}$, де $l \in \mathbb{Z}$, $m \in \mathbb{N}$, наведена нижче [119]:

$$y_0 = \varphi(I_{y_0}) = \varphi(R + \lfloor p \cdot I_x \rfloor) = \varphi(\lfloor (1-p)(bias-k) \cdot N_m \rfloor + \lfloor p \cdot I_x \rfloor) = \\ = \varphi\left(\left\lfloor \frac{m-l}{m}(bias-k) \cdot N_m \right\rfloor + \left\lfloor \frac{l}{m} \cdot I_x \right\rfloor\right). \quad (1.46)$$

Зауважимо, що при $p = -1/2$ ми отримаємо формулу (1.19), (1.20) для функції RSQRT, а при $k = 0$ – метод Бліна [6]. Цю формулу зокрема можна використати для обчислення RCP, SQRT, CBRT та RCBRT при $p = -1$, $p = 1/2$ та $p = \pm 1/3$ відповідно (див. табл. 1.2). Слід зазначити, що оптимальне R буде відрізнятися для кожної функції, а тому актуальним завданням залишається пошук параметра k [119]. Значення R залежить також від типу даних. Обчислення згідно (1.46) доцільно реалізувати із використанням лише цілочислових операцій.

Таблиця 1.2 – Апроксимація логарифмічних, експоненціальних та степеневих функцій методом МС згідно з алгоритмом Бліна [6]

Функція	$y = \log_2 x$	$y = 2^x$	$y = x^p$				
			$y = 1/x$	$y = 1/\sqrt{x}$	$y = \sqrt{x}$	$y = 1/\sqrt[3]{x}$	$y = \sqrt[3]{x}$
Формула	(1.44)	(1.45)	(1.46)	(1.20)	(1.46)	(1.46)	(1.46)
R	0x 3F800000	0x 3F800000	0x 7F000000	0x 5F400000	0x 1FC00000	0x 54AAAAAA	0x 2A555556
Кількість FP операцій	1×	1×	–	–	–	–	–
$\Delta_{0 \max}$	$8,6071 \cdot 10^{-2}$	–	–	–	–	–	–
$\delta_{0 \max}$	–	$6,1476 \cdot 10^{-2}$	$1,2500 \cdot 10^{-1}$	$8,8662 \cdot 10^{-2}$	$6,0660 \cdot 10^{-2}$	$8,1687 \cdot 10^{-2}$	$5,8267 \cdot 10^{-2}$
Бітова точність	–	4,02	3	3,5	4,04	3,61	4,1

Загальна ж формула NR для уточнення $y = x^p$ записується у вигляді

$$y_{n+1} = y_n \left(1 - p + \frac{p \cdot x}{y_n^{1/p}} \right) = -\frac{l}{m} \cdot y_n \left(\frac{l-m}{l} - x \cdot y_n^{-\frac{m}{l}} \right), n = 0, 1, \dots \quad (1.47)$$

Як бачимо, в залежності від значення p , дана формула може використовувати $\div$ і/або корінь l -го степеня, а тому для деяких типів функцій може бути неефективною з обчислювальної точки зору. Наприклад, недоліком методу (1.47)

для SQRT та CBRT є необхідність застосування ітерацій NR, що містять $\div$ на кожній ітерації. Формула для $y = \sqrt{x}$, відома як формула Герона, має вигляд

$$y_{n+1} = 0,5 \cdot \left(y_n + \frac{x}{y_n} \right), n = 0, 1, \dots \quad (1.48)$$

Із формул (1.46) та (1.47) видно, що найбільш ефективними будуть методи для функцій вигляду $y = 1/\sqrt[m]{x}$ (див. табл. Г.5). Розглянемо детальніше модифікацію алгоритму FISR для обчислення функцій RCP та RCBRT із використанням МС.

Обернена до аргументу функція

На відміну від функцій $y = 1/\sqrt{x}$ ($x > 0$) та $y = \sqrt{x}$ ($x \geq 0$), функція $y = 1/x$ визначена і для від'ємних значень $x \neq 0$. Проте, для простоти приймемо $x > 0$. У цьому випадку, (1.46) набуде вигляду

$$y_0 = \varphi(I_{y_0}) = \varphi(R - I_x), \quad (1.49)$$

де, враховуючи (1.19),

$$R = \lfloor 2(bias - k) \cdot N_m \rfloor = Q \cdot N_m + T = 2bias \cdot N_m + \lfloor t \cdot N_m \rfloor. \quad (1.50)$$

Тут у МС Бліна $Q = 2 \cdot bias$, а $T = 0$. Звідси отримуємо, що для FP чисел одинарної точності $R = 0x7f000000$ (див. табл. 1.2). Точність такого наближення є низькою, 3 точних біти, оскільки на проміжку $x \in [1; 2)$ функція наближається прямою. Значно кращу точність мають алгоритми, що використовують МС із порядком $Q = 2 \cdot bias - 1$. Тоді, згідно із [152], аналітичне представлення початкового наближення y_0 функції RCP на проміжку $x \in [1; 2)$ можна записати у вигляді:

$$y_{01} = -\frac{1}{2}x + 1 + \frac{1}{2}t, x \in [1; x'), \quad (1.51)$$

$$y_{02} = -\frac{1}{4}x + \frac{3}{4} + \frac{1}{4}t, x \in [x'; 2), \quad (1.52)$$

де $x' = 1 + t$ – точка з'єднання двох кусково-лінійних частин y_{01} та y_{02} , а параметри t і k пов'язані співвідношенням $t = -2k$. На основі чисельних методів нами було визначено що оптимальне значення МС для чисел типу *float*, що забезпечить вирівняні похибки для y_0 , рівне $R = 0x7ef311c2$, а після першої

класичної ітерації NR – $R = 0x7ef311c3$, що доведено у роботі Мороза та ін. [152]. Тут, $Q = 253$, $T = 7541187$, а $t = -0,1010204554$ і $k = 0,0505102277$. Точність отриманого наближення складає 4 біти, а після однієї та двох класичних ітерацій NR вже 8 та 17 біт. Для типу *double* MC $R = 0x7fde623822fc16e6$, що враховує лише y_0 , була виведена у [153]. У 2015 році Хуан і Чен [154] для обчислення функції RCP запропонували використовувати ітераційний метод Гольдшмідта:

$$r_0 = 1 - x \cdot y_0 \quad (1.53)$$

$$r_{n+1} = r_n \cdot r_n, n = 0, 1, \dots \quad (1.54)$$

$$y_{n+1} = y_n + y_n \cdot r_n, n = 0, 1, \dots \quad (1.55)$$

Для y_0 у [154] пропонується використовувати модифіковану MC $R = 0x7ef4fb9d$ для FP чисел типу *float* та $R = 0x7fde9f73aabb2400$ для чисел типу *double*. Хоча метод Гольдшмідта є швидшим за метод NR, проте точність запропонованого алгоритму є гіршою. Для прикладу, після двох ітерацій точність складає лише 16 біт. Крім того, метод Гольдшмідта має тенденцію до накопичення похибок [117]. Алгоритм із модифікованими ітераціями NR із [152] на базі $R = 0x7ef311c3$ забезпечує відповідно 9 та 19 точних біт після однієї та двох ітерацій (див. табл. Г.6). Слід зазначити, що модифікований ітераційний метод Н вищих порядків (1.42) буде ефективним також для обчислення RCP та RCBRT.

Зворотний кубічний корінь

Аналогічним чином даний метод можна використати для обчислення функції $y = 1/\sqrt[3]{x}$ ($x \neq 0$) [119,155-157]. З цією метою у формулі $y_0 = \varphi(R - I_x/3)$ зазвичай використовується MC Левіна $R = 0x54a21d2a$ [156], а цілочислове $\div$ виконується в арифметиці з фіксованою комою, де заміняється на комбінацію з цілочислового 64-бітного $\times$ та бітового зсуву [156,157]. Оптимальну MC для y_0 $R = 0x54a232a3$ можна отримати із алгоритму Кахана [155] для арифметики з фіксованою комою. Після однієї класичної ітерації NR (1.47) алгоритм Левіна [156] забезпечує точність 8 біт, після двох ітерацій – 16 біт, використовуючи $9\times$ та $2-$ в FP арифметиці. У випадку використання однієї ітерації модифікованого

методу Но2 (1.42) точність зростає до 12 біт ($6\times$ та $2-$). У роботі Л. Мороза та ін. [157] розглянуті модифіковані алгоритми для обчислення RCBRT та CBRT на основі методу МС (див. табл. Г.5). Альтернативні методи обчислення CBRT на основі квадратичної поліноміальної апроксимації та методу NR, а також дробово-раціональної апроксимації 4-го порядку наведені у [158]. У роботі [155] В. Кахан розглядає ітераційні методи вищого порядку для CBRT, що містять $\div$.

Варто також застосувати розглянуті методи апроксимації елементарних функцій в FP арифметиці як ключові блоки процесу генерації ПВЧ. Алгоритми різної точності дозволяють дослідити вплив нелінійності, кількості та типу використаних арифметичних операцій, характеру та рівня похибок алгоритмів на випадковість молодших “неточних” розрядів FP чисел.

1.5. Методи генерації випадкових і псевдовипадкових чисел та бітових послідовностей для криптографічних та некриптографічних застосувань

У багатьох областях науки та техніки виникає потреба генерації великого обсягу випадкових чисел, зокрема в криптографії та стеганографії, статистиці, для чисельних методів, генетичних алгоритмів, симуляції складних фізичних та біологічних процесів тощо [15,36,49,61,85,138]. Такі послідовності можуть бути отримані з допомогою генераторів істинно випадкових послідовностей (ІВП), табличного або алгоритмічного методу генерації ПВП. Генератором ІВП називається такий генератор, як правило апаратний або програмно-апаратний, що в якості джерел ентропії використовує явища фізичної природи (наприклад, квантові процеси чи тепловий шум) та здійснює подальше опрацювання отриманих випадкових даних [36,49,50,52,62,75]. Отримані послідовності є непередбачуваними, випадковими та необмеженими щодо максимальної довжини, проте такі генератори володіють рядом недоліків, що обмежують їх застосування: швидкість генерації може бути низькою, висока ціна, великі габарити, значне споживання енергії, відсутність надійних джерел ентропії та неможливість відтворення згенерованої послідовності без попереднього запису [49,55]. З іншої сторони, ГПВЧ формують детерміновані ПВП для кожного

початкового значення, зазвичай періодичні [49,52,55,61,85]. За їх застосуванням ГПВЧ та ГПВБП поділяються на криптостійкі та некриптостійкі. Для криптографічних застосувань такі генератори мають відповідати ряду додаткових вимог, зокрема неможливість за поліноміальний час передбачити наступне значення ГПВЧ на основі попередніх, неможливість відрізнити сформовану послідовність від випадкової та стійкість до криптоаналітичних атак [36,48,54].

Вони є важливими для захисту інформації, що зберігається, опрацьовується чи передається через незахищені канали зв'язку, наприклад мобільного зв'язку, кабельного телебачення та Wi-Fi [31,34,36,38]. Для прикладу, генератори випадкових та ПВП використовуються для формування різноманітних криптографічних ключів та поточкових шифрів. У цьому випадку, від криптографічної стійкості та випадковості генератора залежить безпека всієї інформаційної системи [15,50]. У системах технічного ЗІ ГПВБП використовуються під час проєктування скремблерів та генераторів шуму [15,48,85]. Якщо ГПВЧ не є криптографічно стійкими, то їх доцільніше використовувати в стеганографічних протоколах, комп'ютерних іграх, в наукових дослідженнях, для симуляцій фізичних, біологічних, фінансових та соціальних моделей, в числових методах на основі методу Монте-Карло тощо [52,55,57,60,61,85-87]. Для таких задач важливі перш за все статистичні характеристики згенерованих ПВП, зокрема їх випадковість та рівномірний розподіл, продуктивність алгоритму генерації ПВП, великий період та можливість багаторазового проведення експерименту [36,49,55,78,159].

Історично першим відомим алгоритмічним ГПВЧ на основі арифметичних операцій був метод середніх квадратів Джона фон Неймана, що полягає в послідовному піднесенні до квадрату n -значного цілого числа та вибору з нього в якості наступного елемента середніх n цифр $2n$ -значного результату. Цей метод має серйозні недоліки, а тому на практиці не застосовується. Для генерації неперіодичних послідовностей ПВЧ також часто використовуються ірраціональні математичні константи та квадратні корені з простих чисел, такі як $\pi = 3,1415926535\dots$, $\sqrt{2} = 1,4142135623 \dots$ та $e = 2,7182818284\dots$ [2,16,78,160]. В наукових

колах прийнято вважати, що сформовані з десяткових та двійкових цифр таких констант послідовності мають незалежний і однаковий розподіл та проходять статистичні тести на випадковість [16,78,161]. Наприклад, можна обчислити значення цифр числа $\sqrt{2}$ починаючи з довільної позиції [160,161]. Ці константи не є випадковими, оскільки існує математична процедура їх визначення. Також вони зустрічаються в багатьох криптографічних алгоритмах, де використовуються для початкової ініціалізації даних. Зокрема, в роботах [2,16,160] досліджувалось питання проєктування ГПВЧ на основі обчислення функцій DIV та SQRT у відповідній системі числення. У першому випадку [16] розглядались деякі раціональні числа із простим знаменником (результат – періодична ПВП), а в другому [2,84,160] – в якості аргументу функції SQRT вибиралось просте число (результат – неперіодична ПВП). Для обчислення таких функцій з довільною точністю використовуються методи типу цифра-за-цифрою, що мають низьку збіжність. Альтернативним методом у випадку генерації ПВП великого розміру є використання довгої арифметики, що також не є ефективним.

Розглянемо інші приклади популярних методів побудови сучасних ГПВЧ, як криптистійких, так і некриптистійких.

Класифікація сучасних популярних некриптистійких ГПВЧ [36,48,55,85]:

- 1) на основі регістрів зсуву з лінійним зворотним зв'язком (LFSR) [36,60,61]: генератори регістрів зсуву (Xorshift) та WELL;
- 2) на основі лінійного конгруентного методу (LCG) [16,36,55,61]: генератор множення із переносом (MWC);
- 3) на основі перекручених узагальнених регістрів зсуву (TGFSR) [36,53,55,58,61,162]: генератор вихор Мерсена (Mersenne Twister);
- 4) на основі генератора Фібоначчі з запізненням: генератор віднімання із запозиченням (SWB).

Деякі з наведених вище генераторів мають хороші статистичні властивості та досить великий період повторення, проте їх не можна застосовувати для більшості криптографічних задач. Для прикладу, період повторення ГПВЧ вихор

Мерсена складає $L = 2^{19937} - 1$ [163]. Цей ГПВЧ використовує двійкову лінійну рекуренту над масивом із 624 цілих чисел, що оновлюється кожні 624 кроків.

Класифікація сучасних популярних криптостійких ГПВЧ:

- 1) на основі односторонніх функцій [53,58,164]: генератори Блум-Блум-Шуба (BBS) та Блум-Мікалі;
- 2) на основі криптографічних шифрів та хеш-функцій [36]: генератори Яроу, Фортуна, ChaCha20, ISAAC та AES-CTR DRBG;

Недоліком криптографічно стійких генераторів є їх низька швидкодія. Наприклад, ГПВЧ Блум-Блум-Шуба [164] використовує операцію піднесення цілого числа до квадрату та обчислення остачі від ділення на число, що є добутком двох великих простих чисел. Із результату береться один біт – біт парності або молодший біт.

Останнім часом набувають популярності також ГПВЧ на основі теорії хаосу [56-58,79]. Такі генератори зазвичай мають непогану продуктивність та криптографічну стійкість завдяки високій чутливості їх початкових значень. У роботі [56] запропоновано криптографічно стійкий ГПВЧ на основі обчислення логістичного хаотичного відображення в $\mathbb{F}_p$ арифметиці. Цей генератор для внутрішніх обчислень використовує числа подвійної точності формату IEEE 754, а на виході з них формує послідовність ПВЧ із діапазону $[0; 255]$. Для цього дробова частина мантиси розглядається як ціле число, що складається з 15 десяткових цифр, із якого обчислюється залишок від ділення на 256.

Розглянуті ГПВЧ можуть використовуватись також для генерації рівномірно розподілених випадкових чисел у форматі IEEE 754 [165,166].

1.6 Методи оцінювання якості ГПВБП для задач у сфері кібербезпеки та захисту інформації

Однією з вимог криптографії, зокрема до ГПВЧ та ГПВБП, є псевдовипадковість результату. Згідно з К. Шенноном, такі криптографічні примітиви як блокові шифри, крім цього, мають володіти дифузією та заплутуванням, а хеш-функції – стійкістю до колізій [167]. Саме тому їх

використання є одним із розповсюджених методів проектування криптографічно стійких ГПВБП [37]. У випадку порушення цих вимог, виникає загроза витоку інформації про ключ або вхідне повідомлення, а тому виникає необхідність застосування статистичних тестів на випадковість [78,159,168]. Основне їх призначення полягає в оцінці того, чи результат роботи ГПВБП відрізняється за своїми характеристиками від ІВП. Слід зауважити, що при цьому деякі згенеровані генератором істинно випадкових чисел послідовності можуть мати ознаки не випадкових, і навпаки, деякі ПВП можуть здаватися більш випадковими, ніж реальні випадкові послідовності [159,169].

Можна виділити наступні основні характеристики ГПВБП [36,49,52,55,85]: 1) випадковість та статистична безпечність; 2) відтворюваність та період повторення; 3) незалежність та непередбачуваність; 4) лінійна складність та криптографічна стійкість; 5) дифузія та заплутування; 6) простота програмної й апаратної реалізації та ефективність використання пам'яті; 7) швидкодія; 8) розмір простору початкових значень.

Таким чином, хороший ГПВЧ проходить всі статистичні тести на випадковість, має достатньо великий період повторення або є неперіодичним, має рівномірний розподіл згенерованих числових послідовностей, є швидким та непередбачуваним, що є важливим як для криптографічних, так і некриптографічних цілей [36,78,85].

1.6.1 Статистичне тестування ГПВЧ

Всі методи тестування генераторів ІВП, ГПВЧ та ГПВБП можна поділити за типом представлення результатів на два великі класи: графічні тести та оціночні тести. Графічні тести відображають результати тестування в вигляді графіків та гістограм без їх кількісної оцінки: гістограма розподілу елементів послідовності, автокореляційна функція, розподіл на площині, графічний спектральний тест тощо [48]. Оціночні методи бувають евристичними та статистичними. До евристичних методів належать: визначення періоду повторення, перевірка на криптостійкість, тестування швидкодії, тестування на основі точності результатів методу Монте-Карло тощо [170]. Серед основних пакетів оціночних статистичних

тестів можна виділити NIST Statistical Test Suite [78], TestU01 [171], FIPS 140-2 [172], Дональда Кнута [173], AIS 31 [174], DieHARD [175] та Dieharder [176].

Стандарт FIPS 140-2 визначає оновлену версією першого набору базових статистичних тестів NIST FIPS 140-1 від 1994 року, що визначає рівні безпеки. Розмір однієї послідовності для тестування складає 20000 біт. Рекомендації органу із стандартизації Німеччини AIS 31 є розширенням набору статистичних тестів NIST FIPS 140-1, що призначені для тестування безпечності генераторів ІВП. Тести NIST STS призначені для дослідження бінарних послідовностей значно більшого розміру та вищого рівня безпеки, ніж FIPS 140-2. Цей набір містить 15 різних статистичних тестів, які можуть бути використані для перевірки рівномірності розподілу та криптостійкості генераторів. Пакет із 18 статистичних тестів DieHARD реалізує багато класичних тестів, проте має певні недоліки та обмеження [177]. Статистичний пакет Dieharder є розширеною версією старшого набору тестів DieHARD. Він містить також декілька тестів із набору NIST та декілька власних тестів. TestU01 наразі є найбільш повним та строгим набором статистичних тестів для генераторів випадкових та ПВП, що містить більше 516 тестів згрупованих у декілька класів відповідно до їх призначення. Тестування може займати досить тривалий час та вимагати великого обсягу даних. Для прикладу, на сучасному ПК дослідження однієї ПВП розміру 119,2 Мб тестами NIST STS із параметрами за замовчуванням займає близько години. Найбільш повний статистичний пакет Big Crush вимагає як мінімум 60 Гб вхідних даних.

1.6.2 Набір статистичних тестів NIST STS для криптографічних застосувань

Для дослідження статистичних характеристик ПВП ми будемо використовувати пакет тестів NIST Statistical Test Suite (NIST STS) [78]. Ці тести спеціально розроблені Національним інститутом стандартів і технології (NIST) для дослідження випадковості бінарних вхідних послідовностей на основі різних статистик як для окремих бітів, так і для блоків [159]. Основним застосуванням тестів NIST STS є оцінювання якості ПВП, в тому числі для їх використання в криптографічних системах [37,78,159].

Для отримання коректних результатів тестування мінімальний розмір однієї підпослідовності для деяких тестів пакету NIST STS має складати $n = 10^6$ біт, а мінімальне значення кількості таких підпослідовностей розміру n , що тестуються, має бути обернено пропорційною до рівня значущості α , наприклад при $\alpha = 0,01$ отримуємо наступну умову: $m \geq 100$. Рівень значущості визначає ймовірність відхилення випадкової послідовності. Наприклад, при $\alpha = 0,01$ серед протестованих $m = 1000$ випадкових послідовностей тести NIST можуть відхилити десять з них, а при значенні статистики $P\text{-value} \geq 0,01$ послідовність розглядається як випадкова з імовірністю 99%.

Відповідно до методики оцінювання результатів емпіричних тестів NIST STS формулюється нульова гіпотеза H_0 щодо випадковості тестованої бітової послідовності. Задається рівень значущості α , значення якого для криптографічних задач зазвичай приймається досить малим, як правило $\alpha = 0,01$ [159]. Згідно з [78], існує два основних підходи до оцінки результатів тестування NIST STS на основі одержаних значень $P\text{-value}$: за часткою підпослідовностей, що проходять тест (стратегія 1), та за рівномірністю розподілу значень $P\text{-value}$ (стратегія 2).

Відповідно до стратегії 1, яку в даній дисертації ми будемо розглядати як основну, статистичний тест вважається пройденим, якщо частка проходження цього тесту NIST, яка позначається P (proportion), потрапляє у визначений довірчий інтервал [15,78,159]. Довірчий інтервал залежить від рівня значущості α та розміру вибірки k та визначається формулою:

$$(1 - \alpha) \pm 3 \sqrt{\frac{\alpha(1 - \alpha)}{k}}. \quad (1.56)$$

Початкова гіпотеза H_0 про випадковість досліджуваної послідовності відхиляється у випадку, якщо частка проходження знаходиться поза межами визначеного довірчого інтервалу хоча б для одного із 188 тестів NIST [159].

1.7 Формулювання вимог і вибір способів удосконалення алгоритмів для інформаційної та кібербезпеки

На основі проведених досліджень можна сформулювати наступні вимоги до проектування вдосконалених алгоритмів, придатних для задач КБ та ІБ [8,33]:

- підтримка довільних типів даних стандарту IEEE-754 (*float*, *double*, *binary128* та ін.);
- компактність та простота, ефективна програмна та апаратна реалізація (зменшення кількості $\times$, уникнення важкої операції $\div$);
- використання лише тих операцій, що мають широку апаратну підтримку;
- низька латентність та висока пропускна здатність, можливість векторизації обчислень;
- мінімальне навантаження на постійну та оперативну пам'ять, уникнення громіздких LUT;
- висока точність (мінімізація похибки), визначені межі похибок на всьому діапазоні вхідних аргументів;
- кросплатформенність, можливість використання на різноманітних типах обчислювальних пристроїв (ПК, МК, ПЛІС, GPU тощо) як альтернатива спеціалізованих апаратних інструкцій, фіксований результат обчислень;
- підтримка денормалізованих чисел та опрацювання виняткових ситуацій (± 0 , $\pm \infty$, NaN);
- можливість вибору (налаштування) точності алгоритму, забезпечення компромісу між точністю та швидкодією алгоритму.

Враховавши зазначені вище вимоги, сформулюємо наступні способи модифікації існуючих та розробки нових покращених методів обчислення функцій RCP, DIV, RSQRT та SQRT, RCBRT та CBRT [7,8,33,80,82]:

1. Для отримання досить точного початкового значення потрібно використовувати метод MC або швидкі апаратні інструкції RCP та RSQRT, якщо доступні.

2. Для уточнення результату будемо поєднувати модифіковані ітераційні методи NR та/або H, переважно невеликих порядків.
3. Для підвищення точності алгоритмів:
 - потрібно застосовувати метод NR та модифікований метод H у спеціальній формі [7,8,80,82,116,117,178] із використанням функції FMA для зменшення похибок заокруглень на останній ітерації;
 - розглянемо розбиття проміжку $x \in [a; b)$ на піддіпазони із використанням різних коефіцієнтів апроксимації на кожному з них [8];
 - потрібно визначити такі значення MC R та коефіцієнтів ітераційних методів k_i , які дозволять мінімізувати максимальну відносну похибку алгоритму використовуючи теоретичні та практичні методи оптимізації;
 - будемо враховувати порядок виконання арифметичних операцій та їх кількість для мінімізації накопичення похибок заокруглень.
4. Для підвищення швидкодії алгоритмів:
 - для обчислення функцій DIV, SQRT та CBRT без операції $\div$ будемо використовувати модифікації алгоритмів для RCP, RSQRT та RCBRT;
 - в алгоритмах необхідно мінімізувати кількість використаних FP операцій, зокрема $\times$ [7];
 - під час обчислень будемо застосовувати швидкі апаратні інструкції FMA, якщо доступні;
 - потрібно враховувати порядок виконання арифметичних операцій, а також можливість їх розпаралелення та конвеєрного опрацювання;
 - потрібно враховувати можливість векторизації алгоритмів в режимі SIMD.

Висновки до розділу 1

1. Визначено особливості FP арифметики згідно вимог стандарту IEEE 754, а також підтримки в МК. Виявлено недоліки та вразливості FP арифметики в контексті задач кібербезпеки та захисту інформації, пов'язані з

накопиченням похибок, втратою точності обчислень, портативністю алгоритмів і забезпеченням повторюваності результатів.

2. Проведено класифікацію та аналіз відомих методів обчислення елементарних функцій для сфери КБ та ЗІ, зокрема рекурентних, ітераційних, поліноміальних і табличних, визначено їх переваги та недоліки. Досліджено особливості застосування методів початкового наближення на основі магічної константи, поліноміальної апроксимації та наближених апаратних інструкцій, а також ітераційних методів NR та методів вищих порядків збіжності, для обчислення функцій RCP, DIV, RSQRT, Sqrt, RCBRT та CBRT у FP арифметиці.
3. Досліджено теоретичні основи алгоритму FISR для обчислення функції RSQRT, проведено аналіз історії його виникнення та модифікацій базового алгоритму. Виявлено, що найкращі відомі модифікації алгоритму FISR дозволяють зменшити максимальні відносні похибки після першої ітерації приблизно у 2,7 раз. Досліджено використання в алгоритмі FISR ітераційних методів вищого порядку, зокрема без операції $\div$, та визначено способи узагальнення алгоритму для випадку апроксимації інших логарифмічних, експоненціальних та степеневих функцій. Здійснено порівняння обчислювальної складності та практичних похибок відомих методів обчислення елементарних функцій, зокрема RSQRT, RCP, RCBRT, Sqrt та CBRT, для FP типів одинарної та подвійної точності.
4. Проведено огляд та класифікацію відомих методів генерації ПББП та ПБЧ, а також порівняння з генераторами ІВП. Визначено сфери їх застосування, особливо для задач інформаційної та кібербезпеки. Проаналізовано основні вимоги до генераторів, зокрема для криптографічних та некриптографічних цілей. Проведено аналіз застосування чисельних методів та FP арифметики для генерації ПБП. Проведено аналіз методів оцінювання їх якості, зокрема на основі статистичних тестів NIST STS.
5. Сформульовано основні вимоги до проєктування вдосконалених алгоритмів та визначено способи підвищення їх точності та швидкодії.

РОЗДІЛ 2. ВИБІР МЕТОДІВ І РОЗРОБЛЕННЯ МЕТОДИКИ ДОСЛІДЖЕННЯ

2.1 Методи проєктування та дослідження вдосконалених алгоритмів обчислення елементарних функцій

Основна ідея розробки покращених алгоритмів полягає в модифікації початкового наближення y_0 (для методу МС, на основі зміни R) та, наприклад, першої ітерації методу NR або методу Н (модифікація коефіцієнтів k_{1j}) таким чином, що їх комбінація дозволить отримати нову апроксимацію y_1 , що мінімізує

максимальну відносну похибку $\delta_{1\max} = \max_{x \in [a; b]} \left(\frac{|y_1 - Y|}{Y} \right)$ після першої ітерації.

Отримане початкове наближення може служити відправною точкою для побудови модифікованих і/або оптимізованих алгоритмів вищої точності.

2.1.1 Побудова математичної моделі алгоритмів

Відповідно до визначених у розділі 1 способів побудови алгоритмів обчислення елементарних функцій перш за все формулюється математична модель алгоритму.

Модельовання методу магічної константи

Як уже зазначалось, відповідно до методу МС (1.44)-(1.46), початкове наближення $y_0 = \varphi(I_{y_0})$ більшості розглянутих функцій залежить від МС $R = Q \cdot N_m + \lfloor t \cdot N_m \rfloor$ та $I_x = \iota(x)$. Таким чином, значення R визначається двома параметрами: порядком Q та значенням $t \in [0; 1)$. Як правило, значення порядку Q в різних алгоритмах не змінюється та рівне, наприклад, $Q = 2 \cdot bias - 1$ для функції RCP та $Q = bias + \lfloor bias/2 \rfloor$ для функції RSQRT, $t = T / N_m \in [0; 1)$. Така модель є універсальною та дозволяє визначати похибки початкового наближення y_0 при довільних значеннях параметрів $bias$, N_m , Q та t на всьому проміжку нормалізованих значень x у вигляді (1.1). Проте, недоліком такої імітаційної моделі є її відносно низька швидкодія [152], а тому альтернативний метод

передбачає представлення початкового кусково-лінійного наближення y_0 методу МС у вигляді набору лінійних апроксимацій вигляду $y_0 = (\alpha \cdot x + \beta) \cdot 2^{E_{y_0}}$ на фіксованому проміжку $[a; b)$. У випадку функції RCP така аналітична модель на проміжку $[1; 2)$ має вигляд (1.51)-(1.52). Аналогічно для функції RSQRT на проміжку $[1; 4)$ аналітична модель описується формулами (1.27)-(1.30). Варто зазначити, що при модифікації деяких параметрів методу, зокрема Q та t , може змінитися й форма представлення таких лінійних апроксимацій (див. рис. 1.3). Наприклад, для функції RSQRT при $Q = \text{bias} + \lfloor \text{bias}/2 \rfloor$ та у випадку $1/2 \leq t < 1$ формули (1.27)-(1.30) набудуть вигляду [7,81]:

$$y_{01} = -\frac{1}{2}x + 1 + t + \frac{1}{2N_m}, \quad x \in [1; x'), \quad (2.1)$$

$$y_{02} = -\frac{1}{4}x + 1 + \frac{1}{2}t + \frac{1}{4N_m}, \quad x \in [x'; 2), \quad (2.2)$$

$$y_{03} = -\frac{1}{8}x + \frac{3}{4} + \frac{1}{2}t + \frac{1}{4N_m}, \quad x \in [2; 4), \quad (2.3)$$

де через x' позначено

$$x' = 2t + 1/N_m. \quad (2.4)$$

Модифіковані МС можуть бути корисними у випадку подальшого використання модифікованих ітераційних методів, а також у випадку розбиття проміжку $x \in [a; b)$ на дрібніші частини [7,8].

Метод знаходження полінома найкращого рівномірного наближення

Будь-яка дійсна неперервна на інтервалі $[a; b]$ функція Y може бути наближена поліномом $y_0 = P_n$ з довільною точністю [126]. Тоді поліном P_n^* найкращої рівномірної (мінімаксної) апроксимації гладкої функції Y на інтервалі $[a; b]$ визначається із наступної формули, модифікованої для відносної похибки [125,179,180]:

$$\delta_{\max}^* = \max_{x \in [a; b]} \left(\frac{|P_n^* - Y|}{Y} \right) = \min_{P_n} \left(\max_{x \in [a; b]} \left(\frac{|P_n - Y|}{Y} \right) \right). \quad (2.5)$$

Задача знаходження найкращого полінома P_n^* має єдиний розв'язок в термінах мінімаксної апроксимації. Чебишев показав, що розв'язок такої задачі характеризується наступними властивостями [126,179,181]:

- для полінома P_n^* степеню n існує $n+2$ невідомих: $n+1$ значень коефіцієнтів та значення відповідної максимальної похибки;
- функція похибки має $n+1$ корінь та $n+2$ точки екстремумів – максимумів та мінімумів;
- усі максимуми та мінімуми похибок є однаковими за величиною, а їхні знаки чергуються.

Ефективним алгоритмом для знаходження мінімаксного полінома є ітераційний метод Ремеза [126,127,180]. Для ініціалізації алгоритму необхідно обрати $n+2$ точки $a \leq x_0, x_1, \dots, x_{n+1} \leq b$, що розташовані в порядку зростання. На основі початкових значень $\{x_i\}_{i=0}^{n+1}$ сформулюємо систему із $n+2$ лінійних рівнянь для ітераційного процесу Ремеза із ваговою функцією [180]:

$$a_0 + a_1 \cdot x_i + a_2 \cdot x_i^2 + \dots + a_n \cdot x_i^n + \lambda(-1)^i \delta_c \cdot Y(x_i) = Y(x_i) \text{ для } i = \overline{0, n+1}, \quad (2.6)$$

звідки можна визначити коефіцієнти $\{a_i\}_{i=0}^n$ полінома P_n та значення відносної похибки ($\delta_c \geq 0$ та λ – знак). Оскільки отримане значення δ_c може не бути максимальним значенням функції відносної похибки δ_m , наступним кроком є визначення точок екстремуму нового наближення. Таким чином формується новий набір точок $\{x_i\}_{i=0}^{n+1}$ для системи рівнянь (2.6). Процес продовжується доти, доки не буде знайдено усі точки екстремумів, в яких похибки будуть рівні за величиною $\delta_{\max}^*$, а їх знаки будуть чергуватися. Критерієм зупинки алгоритму є

$$\delta_{\max}^+ - \delta_{\max}^- < \varepsilon, \quad (2.7)$$

де ε – визначене досить мале значення, або збіжність цього значення до постійної величини. Алгоритм Ремеза має квадратичну збіжність [125,126].

Розглянемо приклад. Для функції RCP поліном найкращого рівномірного наближення 3-го порядку на проміжку $[1; 2]$ у формі запусу за схемою Горнера має вигляд

$$P_3^* = 2,828422877 + (-2,93934142 + (1,33102253 - 0,2218370884 \cdot x) x) x. \quad (2.8)$$

Максимальна відносна похибка такого наближення рівна $\delta_{\max}^* = 1,7331 \cdot 10^{-3}$, а точність складає приблизно 9 біт.

Результати формування поліномів найкращого рівномірного наближення функцій RCP, RSQRT та SQRT методом Ремеза для задачі (2.5) показані на рис. 2.1. Як видно з рисунку, найкраще поліномами наближаються функції $y = \sqrt{x}$ та $y = 1/x$, гірше – функція $y = 1/\sqrt{x}$. Зокрема, використовуючи мінімаксні поліноми першого степеню (лінійне наближення), отримана точність y_0 для функцій SQRT, RCP та RSQRT складає 5, 4 та 3 біти. Для порівняння, метод МС дає 4 точних біти у всіх трьох випадках. Проте, перевагою методу МС є відсутність операцій $\times$ і $+/ -$ в FP арифметиці та необхідності приведення вхідного аргументу. Збільшення степеня полінома також є менш ефективним за розглянуті в розділі 1 ітераційні методи на основі МС, починаючи після першої ітерації. Для прикладу алгоритм Мороза та Гринчишина [152] після першої ітерації NR та поліном 3-го степеню (2.8) для функції RCP мають приблизно однакову точність. Після другої ітерації теоретична точність алгоритму складає 20 біт (як було показано, на практиці – 19 біт для чисел одинарної точності), а відповідний поліном 5-го степеню забезпечує наближення з точністю лише 14 біт. Аналогічно для функції RSQRT, алгоритм Кадлеця [149] після першої ітерації та відповідний поліном 5-го степеня мають точність рівну 10 біт, а алгоритм Леметри та ін. [1] на основі модифікованого методу Н 4-го порядку має теоретичну точність, що рівна 21 біту (на практиці – 20 біт для чисел типу *float*), а відповідний мінімаксний поліном 8-го степеню – лише 15 точних біт.

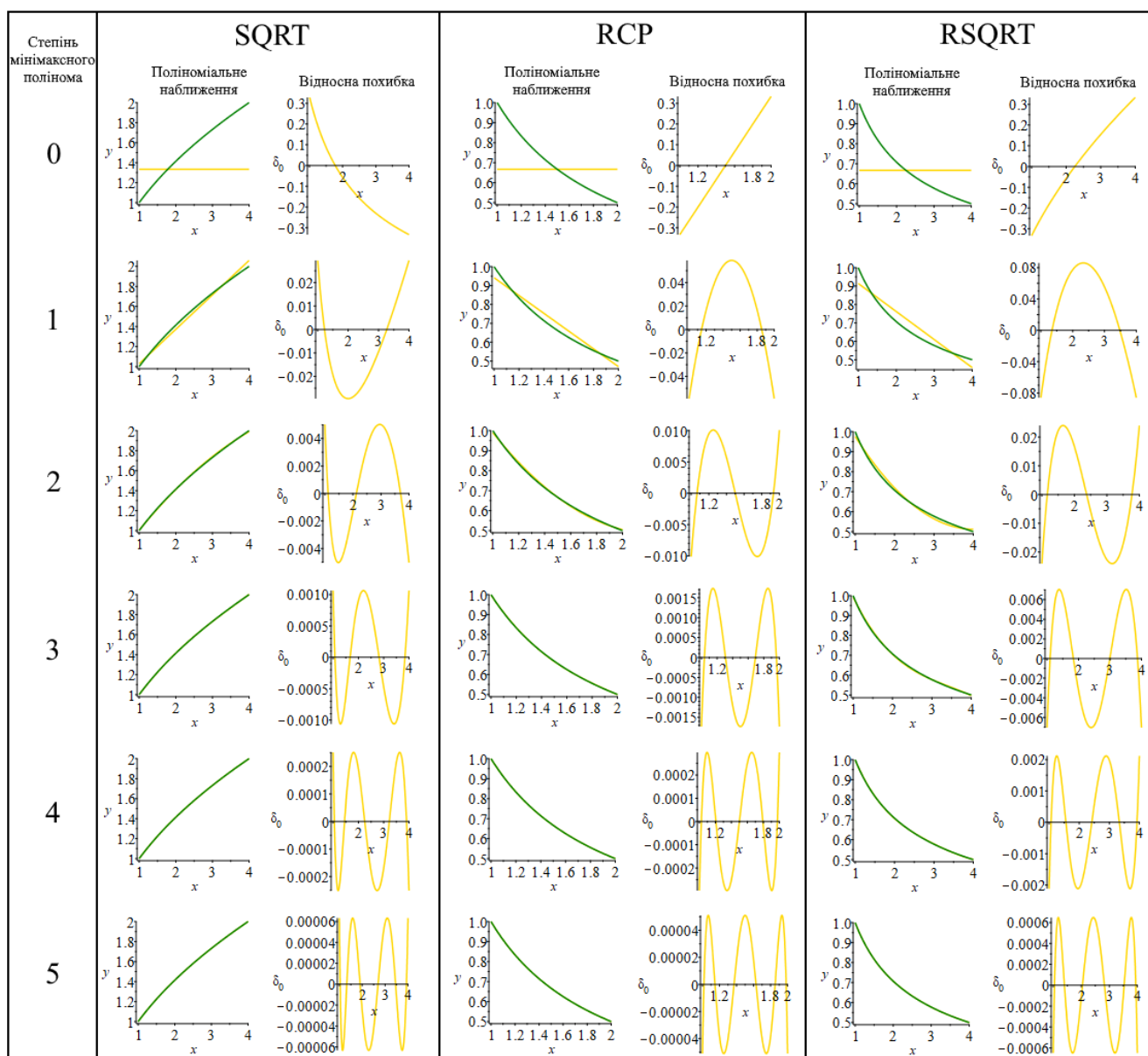
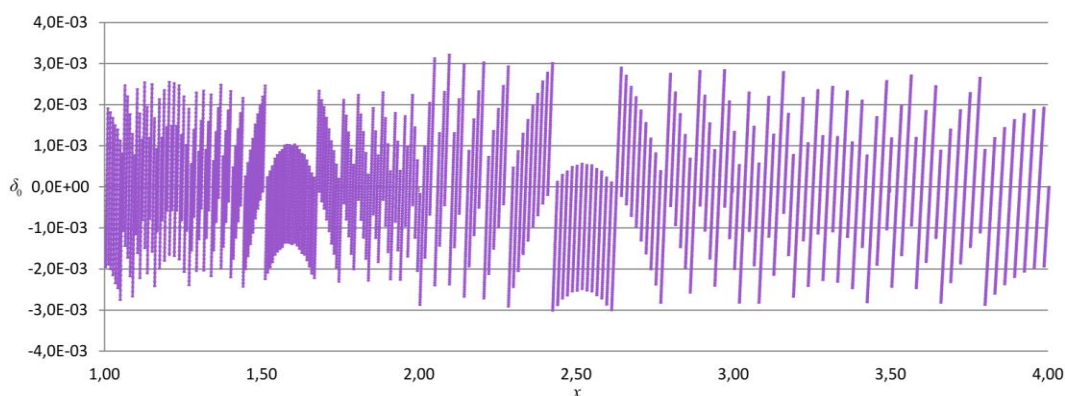


Рисунок 2.1 – Порівняння точності поліномів найкращої рівномірної апроксимації функцій $y = \sqrt{x}$, $y = 1/x$ та $y = 1/\sqrt{x}$ для відносної похибки при різних степенях поліномів ($0 \leq n \leq 5$)

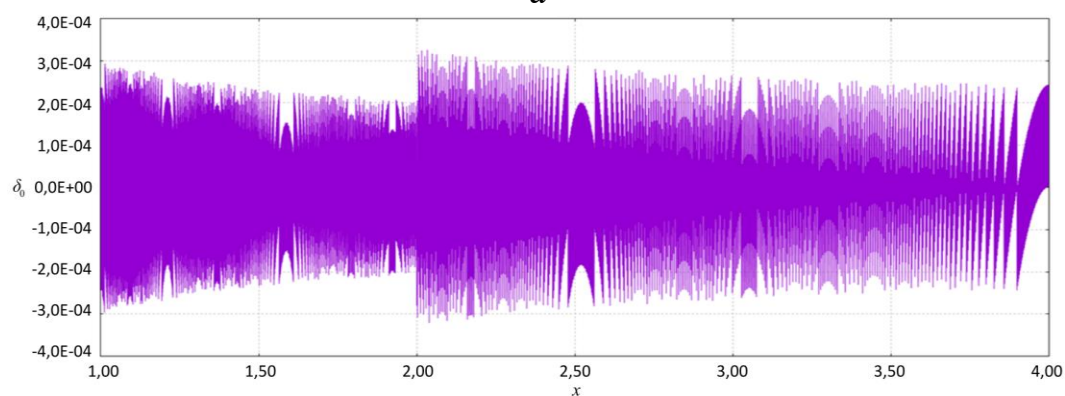
Модельовання спеціальних апаратних інструкцій наближених функцій RCP та RSQRT

Як уже зазначалось, виробники сучасних CPU загального призначення, зокрема на основі технологій Intel та ARM часто включають у FPU процесора апаратну підтримку швидкого наближення функцій RCP та RSQRT на основі LUT, що можуть використовуватись в якості початкового наближення. Тому найчастіше такі інструкції доступні лише для типу *float*. На рис. 2.2 наведено

графіки відносних похибок наближених інструкцій FRSQRTE [44] (для типів *float* та *double*) від ARM та RSQRTSS [45,112] (лише для типу *float*) від Intel для функції RSQRT. Вимірювання похибок інструкції FRSQRTE.F32 на платформі Raspberry Pi 3 Model B+ дало наступні результати: $\delta_{0\max}^+ = 3,2768 \cdot 10^{-3}$ та $\delta_{0\max}^- = -3,0354 \cdot 10^{-3}$, що дає 8 точних бітів [8]. Інструкція RSQRTSS на платформі Intel Core i7-7700HQ має значно менші максимальні відносні похибки: $\delta_{0\max}^+ = 3,2613 \cdot 10^{-4}$ та $\delta_{0\max}^- = -3,2519 \cdot 10^{-4}$, що забезпечує 11 точних бітів апроксимації. Аналогічно для функції RCP, інструкція FRECPE.F32 із набору команд ARM NEON та інструкція RCPSS від Intel мають максимальні похибки на тому ж рівні, що й інструкції RSQRT – 8 та 11 біт точності, відповідно [80].



а



б

Рисунок 2.2 – Графіки відносних похибок наближених апаратних інструкцій для обчислення $y \approx 1/\sqrt{x}$ на проміжку $x \in [1; 4]$: (а) інструкція FRSQRTE із набору команд ARM NEON та (б) інструкція RSQRTSS із набору команд Intel SSE

Крім того, архітектура ARM дозволяє також використовувати спеціальні інструкції FRSQRTS та FRECPs [44] технології NEON для реалізації методу NR:

$$y_{n+1} = y_n \cdot \text{FRSQRTS}(x, y_n^2), n = 0, 1, \dots, \quad (2.9)$$

$$y_{n+1} = y_n \cdot \text{FRECPs}(x, y_n), n = 0, 1, \dots \quad (2.10)$$

Для того, щоб побудувати імітаційну модель початкового наближення y_0 відповідної апаратної інструкції, ми використали метод інженерного аналізу (reverse engineering) даних інструкцій. Для прикладу, ми виявили, що апаратна інструкція FRSQRTE побудована на основі методу, що еквівалентний 8-бітному LUT на проміжку $x \in [1; 4)$, незалежно від типу FP чисел (*float* та *double*). При цьому даний проміжок розбивається на дві частини, $x \in [1; 2)$ та $x \in [2; 4)$, в кожній з яких функція наближається константами на інтервалах однакової довжини. Із $N_T = 2^8 = 256$ значень, половина знаходиться на першому проміжку, а половина – на другому. Деякі з сусідніх значень повторюються, а самі значення отримані на основі лінійних апроксимацій. Інструкція RSQRTSS має дещо складнішу структуру та LUT розміру 11 біт. Із $N_T = 2^{11} = 2048$ значень, поділених порівно на два проміжки, усі значення унікальні, лінійність не спостерігається.

Робота таких інструкцій описується наступною математичною моделлю:

$$y_0 = \begin{cases} y_{01}, x_0 \leq x \leq x_1 - 2^{E_x - p} \\ y_{02}, x_1 \leq x \leq x_2 - 2^{E_x - p} \\ \dots \\ y_{0N_T^*}, x_{N_T^*-1} \leq x \leq x_{N_T^*} - 2^{E_x - p} \end{cases}, \quad (2.11)$$

де $x_i = x_{i-1} + 2n_i(b_j - a_j)/N_T, i = \overline{1, N_T^*}$, n_i – кількість об'єднаних сусідніх значень LUT на проміжку $x \in [x_{i-1}; x_i - 2^{E_x - p}]$; $a_j = 2^{E_x}$ та $b_j = 2^{E_x + 1}$ – межі відповідного підінтервалу, $j = 1, 2$; N_T – розмір LUT, а N_T^* – кількість унікальних значень y_{0i} . Приклад використання цієї моделі для інструкції FRSQRTE показаний на рис. 2.3.

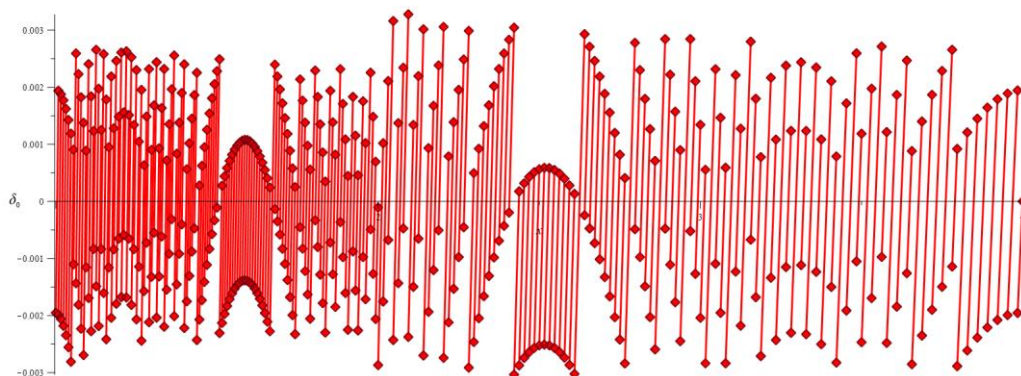


Рисунок 2.3 – Результат моделювання відносних похибок апаратної інструкції ARM NEON FRSQRTE на проміжку $x \in [1; 4)$, що побудована на основі LUT

Методи модифікації алгоритмів для обчислення інших функцій

Алгоритми обчислення різноманітних експоненціальних та логарифмічних функцій отримуються із базових $y = 2^x$ та $y = \log_2 x$ на основі звичайного $\times$ на відповідну константу, наприклад

$$e^x = 2^{x \cdot \log_2 e} = 2^{C_e \cdot x}, \quad (2.12)$$

$$\ln x = \frac{\log_2 x}{\log_2 e} = C_e \cdot \log_2 x. \quad (2.13)$$

При наявності алгоритму обчислення функції RCP, її можна використовувати як заміну операції DIV згідно із [154,182,183]:

$$\text{DIV}(a, b) = a \cdot \text{RCP}(b), \quad (2.14)$$

що потребує додаткового $\times$. Аналогічним чином можна обчислити функцію SQRT на основі розглянутих швидких алгоритмів RSQRT:

$$\text{SQRT}(x) = x \cdot \text{RSQRT}(x). \quad (2.15)$$

В даному випадку точність результату буде приблизно на тому ж рівні, що й у базовій функції. Для одержання функції CBRT можна скористатись наступною формулою, що пропонується в роботі [129] та вимагає вже $2\times$:

$$\text{CBRT}(x) = x \cdot \text{RCBRT}^2(x). \quad (2.16)$$

2.1.2 Методи аналізу алгоритмів

Для проєктування ефективних методів обчислення елементарних функцій потрібно використовувати надійні механізми їх аналізу та порівняння. Основними

кількісними показниками, які ми використовуємо для оцінки методів апроксимації функції є їх точність, швидкодія, область допустимих значень та кількість використаних арифметичних операцій.

Оцінка похибок та точності алгоритмів

Точність алгоритмів визначається через їх максимальні абсолютні $\Delta_{\max}$ та відносні $\delta_{\max}$ похибки на проміжку визначення функції $x \in [a; b]$, проте в FP арифметиці саме відносні похибки є ключовими [181], оскільки дозволяють оцінити бітову похибку [146,152,184]

$$\alpha = -\log_2(\delta_{\max}). \quad (2.17)$$

та бітову точність результату (кількість коректних бітів), що обчислюється згідно із формулою $e = \lfloor \alpha \rfloor$. Для чисельного обчислення похибок алгоритмів ми використовуємо функцію *nextafterf* (x, d) математичної бібліотеки C++ (cmath), щоб перебирати всі можливі вхідні FP значення x типу *float* з визначеного проміжку, а точне значення відповідної функції Y визначається для типів даних вищої точності. Для прикладу, якщо функція SQRT має тип *float*, тоді функція $y = \sqrt{x}$ обчислюється як мінімум із подвійною точністю (*double*). Для цього використовується функція *sqrt*(x) бібліотеки cmath. Оскільки для чисел подвійної точності (тип *double*) повний перебір усіх можливих FP значень є досить складною задачею в обчислювальному плані, навіть для сучасних ПК [116], в даній роботі ми будемо надавати перевагу методу зі сталим, проте достатньо малим кроком перебору. Результати вимірювання похибок алгоритмів, що наводяться у розділах 1 та 3 дисертації, отримані на чотирьохядерному процесорі Intel Core i7-7700HQ (2.8 ГГц, 16 Гб RAM), використовуючи компілятор GCC 4.9.2 для мови C++.

У додатку Б.1 наведено приклади використаних процедур для обчислення похибок алгоритмів у випадку чисел типу *float* та *double*.

Вимірювання швидкодії алгоритмів

При аналізі швидкодії алгоритмів основними є два показники: латентність та пропускну здатність. Латентність алгоритму – це затримка, виражена в

кількості тактів CPU або одиницях реального часу, яку обчислення даної функції вносить в ланцюжок залежностей основної програми [42]. Для оцінки латентності алгоритмів нами було розроблено метод на основі побудови ланцюжка виклику функцій, в якому існує залежність аргументу наступного алгоритму від результату обчислення попереднього [42,185]. При цьому відбувається усереднення результатів вимірювання швидкодії послідовного виклику досліджуваного алгоритму або набору таких алгоритмів при неоднократному повторенні експерименту (див. додаток Б.2). Для заміру часу виконання алгоритму на різних платформах було використано, в тому числі, крос-платформенну бібліотеку chrono для C++ та апаратний лічильник Time Stamp Counter (TSC) для платформ x86 та x86-64. У випадку вимірювання пропускну здатності визначається максимальна кількість повних викликів алгоритму обчислення функції, що можуть бути виконані за одиницю часу. При цьому виклики алгоритмів є незалежними один від одного, а аргументи можуть бути випадковими. Подібним чином визначається зворотна пропускну здатність.

Оцінка збіжності ітераційних методів

Очікуване теоретичне значення максимальної відносної похибки $\delta_{\max}$ можна оцінити на основі аналізу математичної моделі алгоритму, зокрема із врахуванням порядку збіжності використаних ітераційних методів NR і H та максимальної відносної похибки $\delta_{0\max}$ початкового наближення. Результати аналізу збіжності розглянутих методів для функції RSQRT на основі MC та однієї чи кількох ітерацій узагальнені у вигляді рис. 2.4.

2.1.3 Розробка методики покращення алгоритмів щодо відносної похибки

Основна ідея підвищення точності алгоритмів обчислення елементарних функцій полягає у визначенні таких значень магічної константи R і/або коефіцієнтів k_j ітераційних методів, а також такої форми запису алгоритмів, що дозволять на практиці мінімізувати максимальні відносні похибки результату. Для досягнення цієї мети будемо застосовувати теоретичні методи пошуку найкращого рівномірного наближення та практичні методи оптимізації щодо

відносної похибки. Для зменшення рівня похибок заокруглення та відповідно підвищення точності обчислення ітераційних методів, де це необхідно, ми використовуємо їх запис в особливій формі з використанням операції FMA.

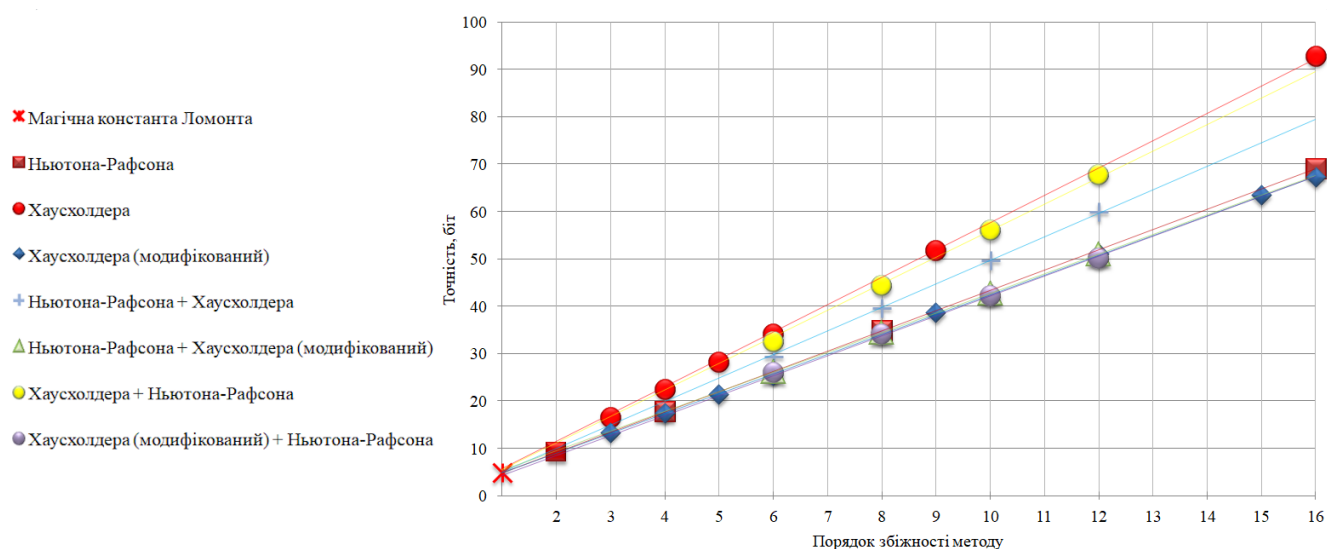


Рисунок 2.4 – Аналіз точності ітераційних методів NR, Н, модифікованого Н та їх комбінацій для функції $y = 1/\sqrt{x}$ на основі оцінки їх збіжності

Метод Ньютона-Рафсона в особливій формі

Для функції RCP ітераційний метод NR (1.47), що містить на ітерацію $2 \times$ та $1-$, відповідно до [116,117,178] може бути обчислений у наступному вигляді:

$$r_n = 1 - x \cdot y_n, \quad (2.18)$$

$$y_{n+1} = y_n + r_n \cdot y_n. \quad (2.19)$$

Тут метод NR містить додаткове $+$. Його зручно записати через функцію FMA, $fma(a, x, b) = a \cdot x + b$. Під час модифікації методу NR будемо розглядати три основні випадки:

$$y_{n+1} = y_n (k'_{n+1} - x \cdot y_n), \quad (2.20)$$

$$y_{n+1} = k''_{(n+1)1} \cdot y_n (k''_{(n+1)2} - x \cdot y_n), \quad (2.21)$$

$$y_{n+1} = y_n (k'''_{(n+1)1} - k'''_{(n+1)2} \cdot x \cdot y_n), \quad (2.22)$$

де (2.20) – частковий випадок (2.21) та (2.22), а константи двох останніх формул пов'язані співвідношеннями $k'''_{(n+1)1} = k''_{(n+1)1} \cdot k''_{(n+1)2}$ та $k'''_{(n+1)2} = k''_{(n+1)1}$.

Для функції RSQRT метод NR (1.31) зазвичай обчислюється в наступному вигляді, що містить на ітерацію 4 або $3 \times$ та одне $-$:

$$x_{half} = 0,5 \cdot x, \quad (2.23)$$

$$y_{n+1} = y_n (1,5 - x_{half} \cdot y_n^2). \quad (2.24)$$

Альтернативні форми запису на основі використання FMA включають

$$y_{n+1} = (1,5 \cdot y_n) - (x_{half} \cdot y_n) \cdot (y_n^2) \quad (2.25)$$

та

$$t_n = 0,5 - x_{half} \cdot y_n^2; \quad y_{n+1} = y_n + t_n \cdot y_n. \quad (2.26)$$

Перевагою формули (2.25) над (2.24) є менша латентність обчислень (на одне $\times$), хоча точність та пропускна здатність падає через більшу загальну кількість $\times$ (4 проти 3). Обидві формули передбачають використання одного FMA. Формула (2.26) порівняно з (2.24) містить на одне $+$ більше, передбачає використання двох операцій FMA, а тому забезпечує трохи вищу точність обчислень. Аналогічний підхід як у (2.18), (2.19) можна використати і для функції RSQRT [116]:

$$r_n = 1 - x \cdot y_n \cdot y_n, \quad (2.27)$$

$$y_{n+1} = y_n + 0,5 \cdot y_n \cdot r_n. \quad (2.28)$$

Ця формула, незалежно від номеру ітерації, містить $4 \times$ та $2+/-$, може містити 2FMA та має найвищу точність обчислення (1.31) серед наведених форм запису. Під час модифікації методу NR (1.31) будемо розглядати наступні випадки:

$$y_{n+1} = y_n (k'_{n+1} - 0,5 \cdot x \cdot y_n^2), \quad (2.29)$$

$$y_{n+1} = k''_{(n+1)1} \cdot y_n (k''_{(n+1)2} - x \cdot y_n^2), \quad (2.30)$$

$$y_{n+1} = y_n (k'''_{(n+1)1} - k'''_{(n+1)2} \cdot x \cdot y_n^2). \quad (2.31)$$

Модифікований метод Хаусхолдера в особливій формі

Надалі в даній дисертації під методом Хаусхолдера (Н) за замовчуванням будемо розуміти методи вищих порядків на основі формули (1.42). Наприклад, метод Но2 (1.39), (1.43) для RSQRT через позначення (2.27) запишеться у вигляді

$$y_{n+1} = y_n + y_n \cdot r_n (0,5 + 0,375 \cdot r_n), \quad (2.32)$$

що містить $5\times$ та $3+/-$. При цьому до формули додається $1-$, що може бути об'єднане в FMA. Аналогічно, метод Но3 запишеться як

$$y_{n+1} = y_n + y_n \cdot r_n(0,5 + r_n(0,375 + 0,3125 \cdot r_n)). \quad (2.33)$$

Методи Но2 та Но3 (1.39), (1.43) містять $z = p + 2$ константи для модифікації:

$$y_{n+1} = y_n(k'_{(n+1)1} + \alpha(k'_{(n+1)2} + k'_{(n+1)3} \cdot \alpha)), \quad (2.34)$$

$$y_{n+1} = y_n(k'_{(n+1)1} + \alpha(k'_{(n+1)2} + \alpha(k'_{(n+1)3} + k'_{(n+1)4} \cdot \alpha))). \quad (2.35)$$

Із формули (1.42), для функції RCP методи Но2 та Но3 мають вигляд

$$\alpha = x \cdot y_n, \quad (2.36)$$

$$y_{n+1} = y_n(3 + \alpha(-3 + \alpha)), \quad (2.37)$$

$$y_{n+1} = y_n(4 + \alpha(-6 - \alpha(-4 + \alpha))). \quad (2.38)$$

Ввівши позначення (2.18), ці формули представляються у вигляді

$$y_{n+1} = y_n + y_n(r_n + r_n^2), \quad (2.39)$$

$$y_{n+1} = y_n + y_n(r_n + r_n(r_n + r_n^2)). \quad (2.40)$$

що відомий також як метод Рабіновича. Особливість даного методу полягає в тому, що він може бути реалізований виключно із FMA операцій. Вибір $z = p + 1$ коефіцієнтів під час модифікації методів Но2 та Но3 для функції RCP визначається формулами

$$y_{n+1} = y_n(k'_{(n+1)1} + \alpha(k'_{(n+1)2} + \alpha)), \quad (2.41)$$

$$y_{n+1} = y_n(k'_{(n+1)1} + \alpha(k'_{(n+1)2} - \alpha(k'_{(n+1)3} + \alpha))). \quad (2.42)$$

Визначення оптимальних значень коефіцієнтів методом вирівнювання похибок

Визначення оптимальних значень коефіцієнтів полягає в теоретичному аналізі математичної моделі алгоритму, що базується на процедурі вирівнювання по модулю значень додатних та від'ємних максимальних відносних похибок $\delta_{\max}^+$ та $\delta_{\max}^-$ на кожному з підінтервалів. Для цього будується та розв'язується система рівнянь, де кількість рівнянь рівна кількості невідомих коефіцієнтів [7,81,82]. Ця процедура подібна на метод Ремеза для визначення P_n^* функції щодо відносної похибки. Проте, використання методу МС для кусково-лінійного початкового

наближення та структура ітераційного методу вносять свої особливості та обмеження на форму полінома на кожному з підінтервалів.

Для прикладу, наведемо узагальнений метод послідовних уточнень коефіцієнтів, що був використаний, зокрема, в роботі [82]. Нехай маємо початкове наближення y_0 методу МС, представлене у вигляді аналітичної моделі, визначеної на проміжку $[a; b)$. Відповідно до моделі, проміжок $[a; b)$ розбивається на декілька частини $[a_i; b_i)$, $i = \overline{1, n}$, залежно від функції, на кожній з яких застосовується лінійне наближення y_{0i} . Перша модифікована ітерація y_1 представляється відповідною математичною формулою NR або Н. Тоді метод визначення коефіцієнтів t та k_{1j} , $j = \overline{1, z}$, можна описати алгоритмом:

- 1) Вивести загальну формулу для обчислення δ_1 на проміжку $[a_i; b_i)$:

$$\delta_{1i} = \delta_{1i}(x, t, k_{1j}) = \frac{y_{1i} - Y}{Y}. \quad (2.43)$$

- 2) Записати рівняння для визначення точок екстремуму функції (2.43)

$$\frac{d\delta_{1i}}{dx} = 0 \quad (2.44)$$

на проміжках $[a_i; b_i)$, врахувавши також межі проміжків.

- 3) Задати початкові значення коефіцієнтів $t' = t_0$ та $k'_{1j} = k_{1j0}$, $j = \overline{1, z}$.

- 4) Знайти із рівняння (2.44) точки максимумів x_{ij}^+ та мінімумів x_{ij}^- δ_1 на кожному з проміжків $[a_i; b_i)$ при заданих значеннях коефіцієнтів.

- 5) Сформувати вирази для відносних похибок в цих точках δ_{1ij}^+ та δ_{1ij}^- :

$$\delta_{1ij}^+ = \delta_{1i}(x_{ij}^+, t, k_{1j}) \quad \text{та} \quad \delta_{1ij}^- = \delta_{1i}(x_{ij}^-, t, k_{1j}), \quad (2.45)$$

а також для відносної похибки на початку проміжку $[a_i; b_i)$:

$$\delta_{1i0}^+ = \delta_{1i}(a_i, t, k_{1j}) \quad \text{або} \quad \delta_{1i0}^- = \delta_{1i}(a_i, t, k_{1j}). \quad (2.46)$$

- 6) Сформувати та розв'язати систему із $z + 1$ рівнянь із $z + 1$ невідомими коефіцієнтами, t та k_{1j} , $j = \overline{1, z}$, на основі вибраних виразів із кроку 5, що забезпечують виконання умови

$$|\delta_{1ij}^+| = |\delta_{1ij}^-|, \quad (2.47)$$

попарно змінюючи знаки похибок (див. [7,81]). В результаті отримаємо покращені значення коефіцієнтів t' та k'_{1j} .

- 7) Повторити кроки 4-6, якщо отримані значення коефіцієнтів t' та k'_{1j} , $j = \overline{1, z}$, не задовольняють вимогам точності. Для зупинки алгоритму можна використовувати умову (2.7).

Слід зазначити, що в роботі [80] для функції RCP нам вдалося досягнути наближення, що після першої ітерації еквівалентне використанню двох поліномів P_3^* на проміжку $[1; 2)$. Причому, на двох частинах проміжку, $[1; x')$ та $[x'; 2)$, забезпечується однаковий рівень похибок (див. рис. 2.5а).

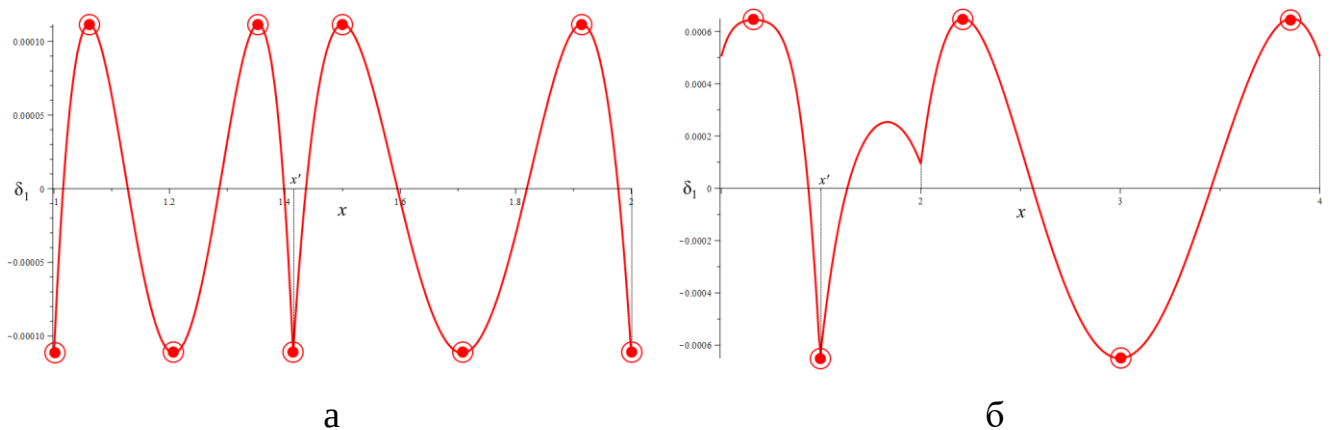


Рисунок 2.5 – Приклади результатів вирівнювання значень максимальних відносних похибок $\delta_{1\max}^+$ та $\delta_{1\max}^-$ модифікованих алгоритмів після першої ітерації:

- (а) для функції RCP із роботи [80] на основі еквівалентності поліному P_3^* та (б) для функції RSQRT із роботи [7] на основі розв'язування системи із двох рівнянь

Метод чисельної багатовимірної оптимізації параметрів алгоритму

Для визначення оптимальних значень констант R та k_j , $j = \overline{1, z}$, що забезпечують мінімальне значення максимальної відносної похибки алгоритму, можна застосовувати також методи чисельної оптимізації [7,8,80,120,149,153]. Для прикладу див. рис. 2.6 для трьох задач мінімізації одного параметра

алгоритму. У цьому випадку враховуються практичні похибки заокруглення та особливості відображення FP чисел. Зокрема, це чітко спостерігається для чисел типу *float* при використанні кількох ітерацій методу NR або ітерацій Н високих порядків, де оптимальні значення коефіцієнтів зазвичай будуть відрізнятися від теоретичних (див. рис. 2.6б,в) [7,8,80-82]. Таку спрощену задачу можна розв'язати методом повного перебору всіх можливих значень R типу *int* або k_2 типу *float*. Проте, з обчислювальної точки зору краще виконувати пошук лише в деякому околі відомого теоретичного значення.

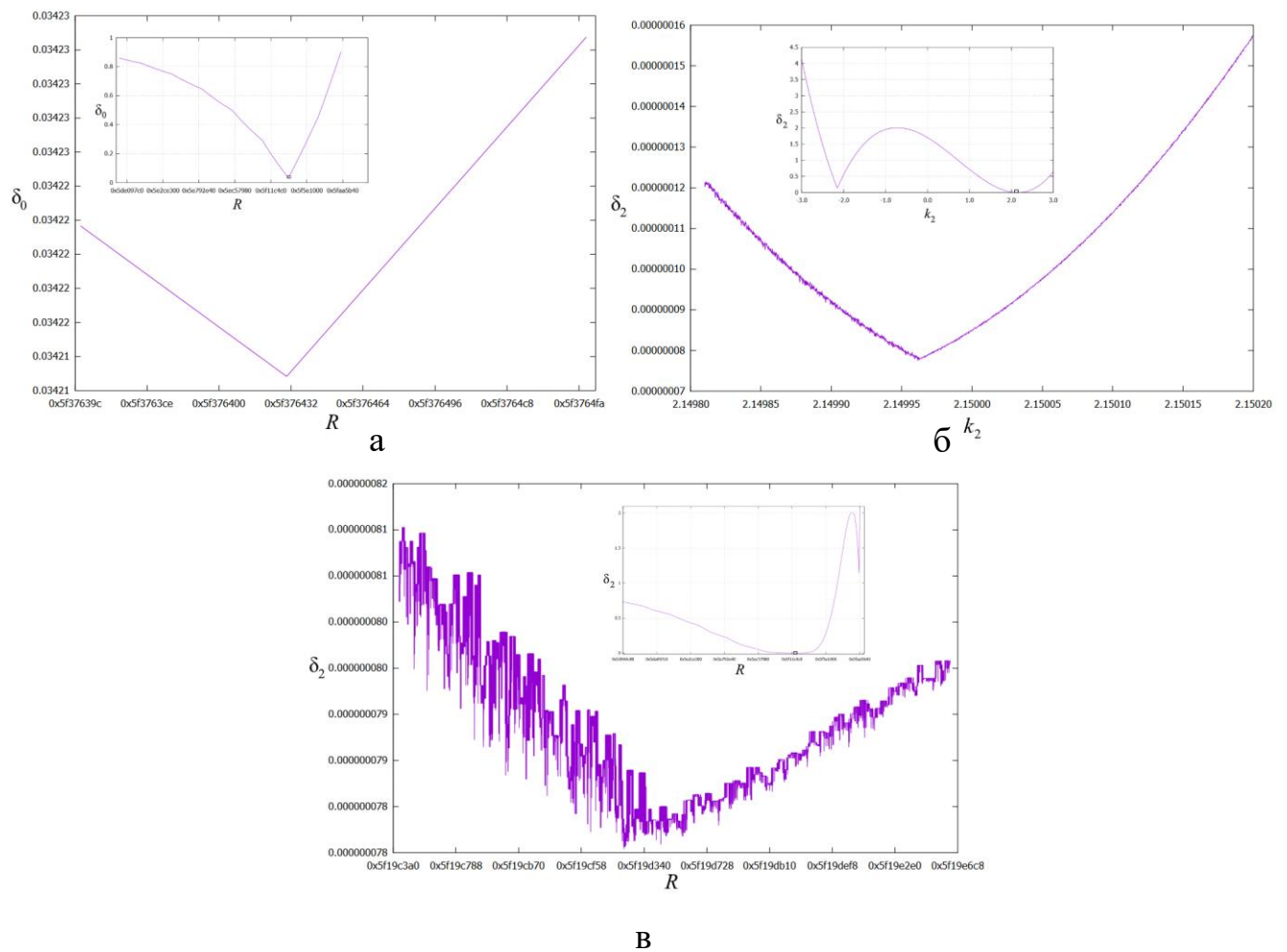


Рисунок 2.6 – Залежність $\delta_{i \max}$ алгоритмів від значення відповідних коефіцієнтів:

- (а) y_0 функції RSQRT від значення MC алгоритму FISR ($R = 0x5f37642f$ – мінімальне), (б) та (в) обчислення RSQRT методом переключення MC (DC) [8, алгоритм 4] від коефіцієнта k_2 та від MC на проміжку $x \in [2; 4)$

Якщо відомо, що на деякому проміжку функція, що мінімізується, має одну точку мінімуму, тоді для підвищення швидкодії методу оптимізації можна застосувати пошук за методом золотого перетину [186,187] (для прикладу див. рис. 2.7 для задачі на рис. 2.6в). Цей метод є простим, має логарифмічну збіжність та був використаний зокрема в роботі [153] для пошуку константи $R=0x7fde623822fc16e6$. Проте, якщо на заданому проміжку є декілька точок мінімуму, метод на основі золотого перетину збіжиться до локального мінімуму. Також, зважаючи на похибки обчислень, на практиці цей метод варто застосовувати в комбінації з методом згладжування та повним перебором значень із проміжку $[c_min; c_max]$ в околі одержаного результату $R \in [c1; c2]$ при умові $|c1 - c2| \leq R^{step}$, де R^{step} – деяке визначене значення для параметра R (аналогічно для параметра k_2 із рис. 2.6б). Альтернативою вказаним методам є перебір значень параметра R із визначеним сталим кроком R^{step} , динамічна зміна кроку та методи випадкового локального та глобального пошуку.

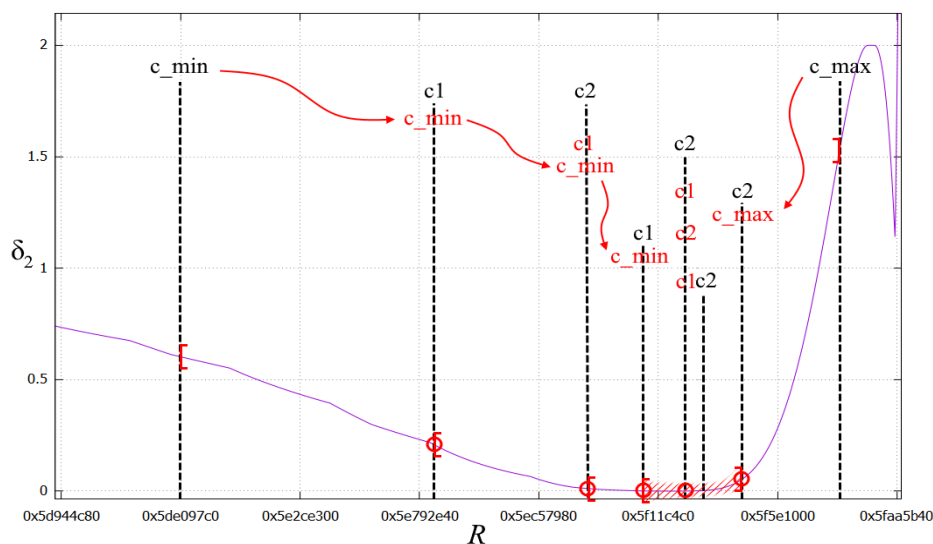


Рисунок 2.7 – Приклад застосування методу золотого перетину для пошуку мінімуму функції $\delta_{2\max}$ на проміжку $R \in [c_min; c_max]$

Тоді задачу практичної багатовимірної оптимізації параметрів алгоритму 4 із [8] на одному з проміжків $x \in [1; 2)$ або $x \in [2; 4)$ можна сформулювати в

наступному загальному вигляді (із першою модифікованою та другою класичною ітерацією NR у спеціальній формі):

Лістинг 2.1 – Загальний шаблон для практичної оптимізації алгоритму обчислення функції $y = 1/\sqrt{x}$ із двома ітераціями NR для чисел типу *float*

```

1: float InvSqrt2f (float x, int* iCoefs, int iN, float* fCoefs, int fN)
2: {
3:     if(iN != 1 || fN != 2)
4:         throw std::invalid_argument( "Invalid mas length" );
5:     int i = *(int*)&x;
6:     i = iCoefs[0] - (i >> 1); // магічна константа R
7:     float y = *(float*)&i;
8:     y = fCoefs[0]*y*fmaf(-x, y*y, fCoefs[1]); // мод. ітерація NR (2.30)
9:     float c = x*y;
10:    float r = fmaf(y, -c, 1.0f);
11:    y = fmaf(0.5f*y, r, y); // класична ітерація NR у формі (2.27), (2.28)
12:    return y;
13: }
```

У лістингу 2.1 *iCoefs* та *fCoefs* – масиви параметрів алгоритму типу *int* та типу *float* розмірів *iN* та *fN* відповідно ($R = iCoefs[0]$, $k_1 = fCoefs[0]$ та $k_2 = fCoefs[1]$), які необхідно покращити відносно $\delta_{2\max}$. В цьому алгоритмі застосовується друга класична ітерація NR у формі (2.27), (2.28) із FMA для зменшення похибок заокруглення відповідних FP операцій, що стають істотними на цьому етапі [8,82]. На рис. 2.8 наведені варіанти реалізації даного алгоритму при різних формах обчислення другої класичної ітерації NR (рядки 9-11 лістингу 2.1). Таким чином, на практиці ми можемо покращувати теоретичні параметри алгоритмів шляхом їх послідовної чи одночасної оптимізації – мінімізації максимальної відносної похибки. За умови $iN + fN > 1$ для переходу від задачі чисельної оптимізації функції з однією змінною до багатовимірного випадку зазвичай застосовується послідовний метод багатовимірного прямого пошуку [187,188] (див. рис. 2.9а), що належить до класу жадібних евристичних алгоритмів.

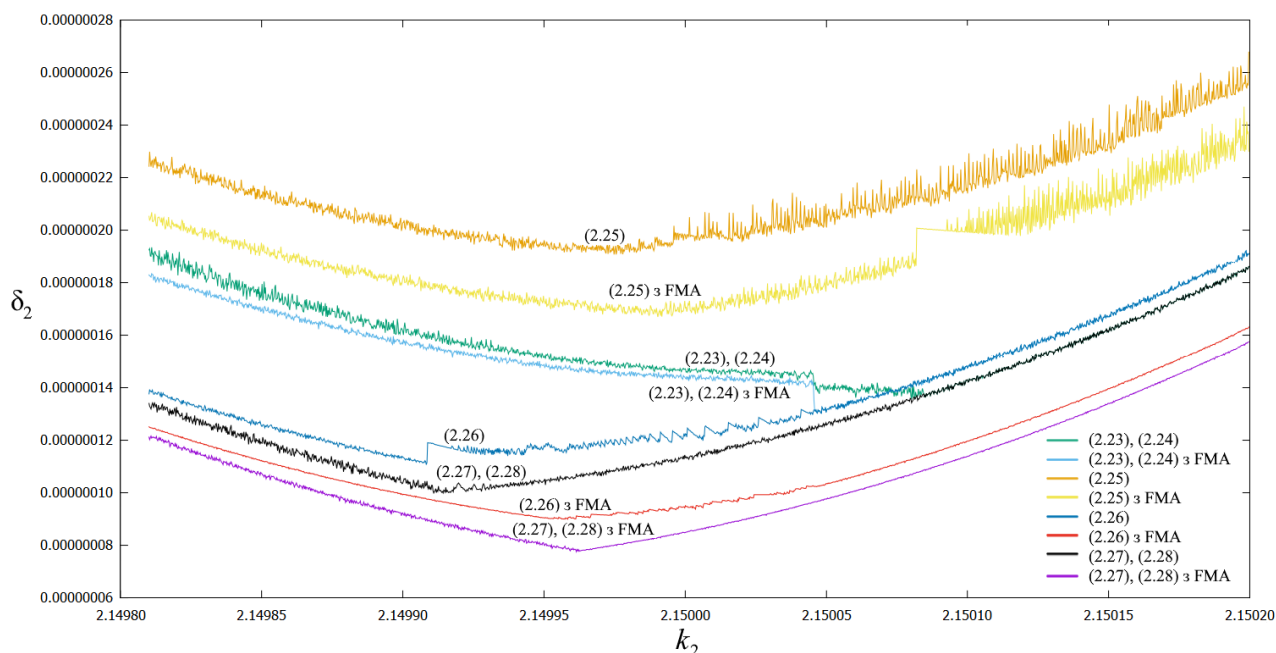


Рисунок 2.8 – Максимальна відносна похибка $\delta_{2 \max}$ алгоритму InvSqrt2f на проміжку $x \in [2; 4)$ із фіксованими $R = 0x5f19d200$ та $k_1 = 0,824212492$ при зміні параметра k_2 залежно від форми запису другої ітерації

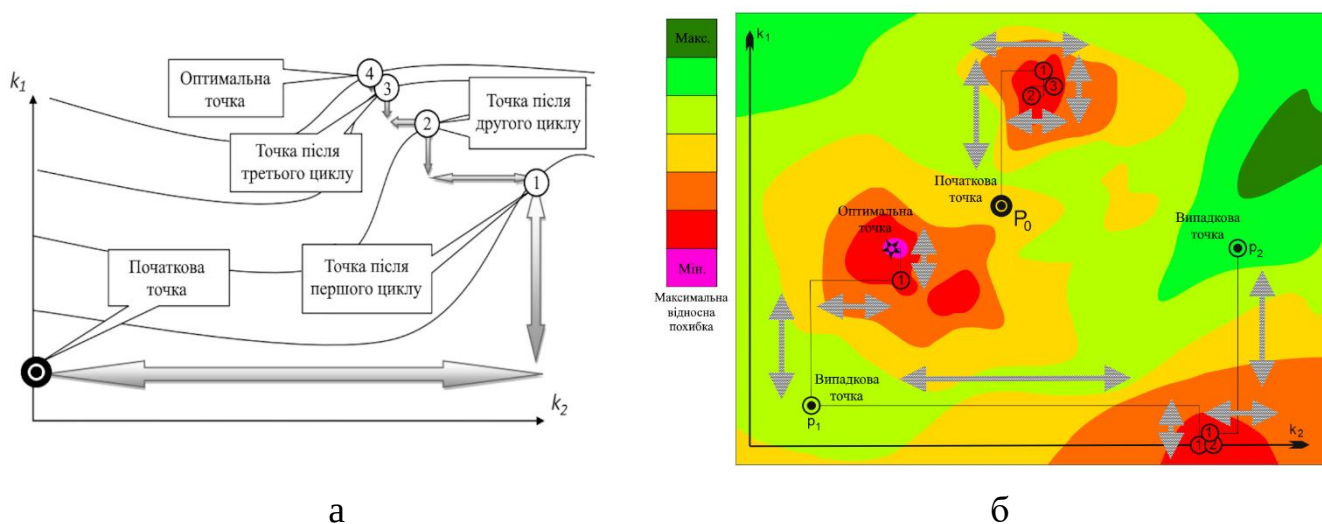


Рисунок 2.9 – (а) Базовий метод багатовимірний прямого пошуку [188].
(б) Метод рандомізованої багатовимірної жадібної оптимізації [8,80] з випадковим вибором порядку оптимізації параметрів та проведенням періодичних спроб випадкової зміни поточних найкращих параметрів

Також для вирішення поставленої задачі практичної оптимізації параметрів можна застосовувати методи повного перебору та випадкового пошуку, методи оптимізації важкої кулі, багатокутника, генетичні алгоритми, методи імітації різноманітних фізичних та біологічних процесів (наприклад відпалу, рою частинок, колонії мурах або голубів) тощо [186,187,189,190]. Як бачимо, багато з таких евристичних алгоритмів оптимізації широко використовують ГПВЧ [189-191]. Для цього задамо структуру алгоритму в вигляді шаблону, проміжок $x \in [a; b)$ та крок зміни вхідного аргументу типу *double* для обчислення $\delta_{\max}$, початкові значення параметрів $R_j = R_j^0, j = \overline{1, iN}, k_j = k_j^0, j = \overline{1, fN}$ та межі їх зміни $R_j \in [R_j^{\min}; R_j^{\max}], j = \overline{1, iN}, k_j \in [k_j^{\min}; k_j^{\max}], j = \overline{1, fN}$. Деякі методи використовують також значення кроку для кожного з параметрів: $R_j^{\text{step}}, j = \overline{1, iN}$, та $k_j^{\text{step}}, j = \overline{1, fN}$. Було розроблено наступний набір методів для алгоритмів одинарної та подвійної точності:

- 1) Метод багатовимірного випадкового пошуку із двома центрами ваги та з фокусуванням – зважена сума останніх і попередніх покращених значень параметрів або лише поточні найкращі значення (див. додаток Б.3.1).
- 2) Методи рандомізованої багатовимірної жадібної оптимізації (див. рис. 2.9б), що базуються на основі методу багатовимірного прямого пошуку (рис. 2.9а) та методу золотого перетину (рис. 2.7):
 - а. основний метод;
 - б. з випадковим розбиттям проміжків $[R_j^{\min}; R_j^{\max}], j = \overline{1, iN}$, та $[k_j^{\min}; k_j^{\max}], j = \overline{1, fN}$, на визначену кількість частин, на кожній з яких застосовується метод золотого перетину (див. додаток Б.3.2);
 - с. з випадковим розбиттям проміжків та випадковим вибором серед рівнозначних параметрів (формування списку рівнозначних параметрів) (див. додаток Б.3.3).
- 3) Методи рандомізованої багатовимірної жадібної оптимізації (див. рис. 2.9б), що базуються на багатовимірному прямому пошуку з повним

перебором та випадковим вибором рівнозначних параметрів (див. додаток Б.3.4);

4) Метод багатовимірного повного перебору (див. рис. 2.10).

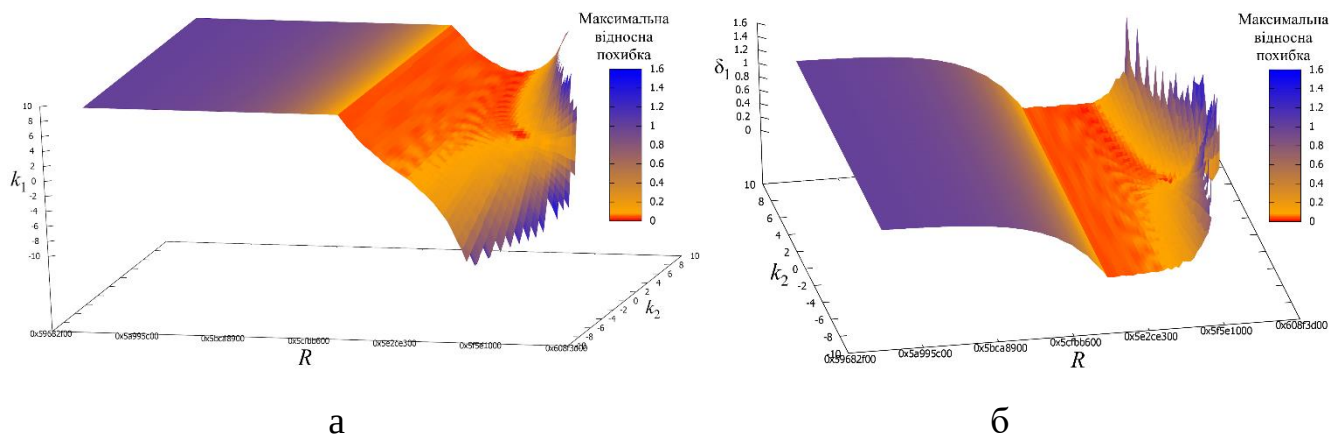


Рисунок 2.10 – Приклад використання методу багатовимірного повного перебору параметрів R , k_1 , k_2 для однієї ітерації алгоритму FISR у формі (2.31):

(а) На осі k_1 показані ті $k_1 \in [-10; 10]$, що забезпечують мінімальне значення $\delta_{1 \max}$.

(б) Відповідні значення похибок

При виконанні експериментальних досліджень ці методи оптимізації зазвичай використовувались або в комбінації, або як альтернативні способи вирішення задач [7,8,80,82]. Два перші класи методів в загальному дають грубші результати, а два останні частіше дозволяють знайти оптимальні значення, в тому числі зважаючи на можливі недоліки ГПВЧ, який в них використовується. Методи випадкового пошуку та повного перебору через їх низьку швидкодію погано підходять для задач глобального пошуку та високої розмірності. Істотна перевага методів, що базуються на основі багатовимірного прямого пошуку, в тому числі методу 2b та методу 3a у випадку вибору оптимального кроку, є їх висока швидкодія та непогана точність [7,8,80]. На противагу цьому, метод повного перебору дає детермінований і гарантовано оптимальний розв'язок задачі у визначених межах параметрів [8,81,120,149].

2.2 Методи проєктування та дослідження ГПВЧ на основі арифметики з рухомою комою для застосувань у сфері кібербезпеки та захисту інформації

Використання FP арифметики під час генерації ПВП має свої переваги та недоліки зважаючи на особливості даної арифметики [15,89], а тому виступає предметом дослідження цієї дисертації.

2.2.1 Особливості проєктування ГПВП на основі FP арифметики

Серед таких особливостей стандарту IEEE 754, що були розглянуті детальніше у п. 1.1, можна виділити, зокрема, формат представлення FP числа в пам'яті; округлення чисел; обмежену точність та широкий діапазон можливих значень; широку апаратну підтримку; високу швидкодію на сучасних обчислювальних пристроях, зокрема на GPU; наявність денормалізованих чисел, спеціальних значень; можливість появи виняткових ситуацій; залежність часу виконання обчислень від вхідних аргументів тощо. Багато науковців висловлюють серйозні застереження щодо використання FP арифметики в критичних системах [89,91,92,192,193]. Крім того, питання, що пов'язані з безпекою FP арифметики, все ще мало досліджені. Помилки та похибки обчислень, дефекти реалізації, що можуть виникати через вказані особливості арифметики, інколи можуть мати істотні практичні наслідки. Використання FP арифметики може бути причиною серйозних загроз ІБ та КБ [67,89,165,194-196].

Деякі особливості реалізації обчислень можуть бути використані для зламу та компрометації безпеки реальних інформаційних систем [165,194,195,197]. Зокрема, на початку 2011 року було виявлено баг у PHP [89,197], що при спробі визначити певне FP значення на основі його символьного представлення змушував PHP сервер на 32-бітних системах зависнути, тобто потрапити у нескінченний цикл із майже 100% завантажкою CPU. Наведемо інший приклад. Практична реалізація механізму Лапласа для забезпечення конфіденційності чутливих даних на основі FP арифметики часто містить відому вразливість – отриманий розподіл вихідних даних є нерівномірним та містить багато прогалів [194,195]. Причинами такого серйозного недоліку багатьох систем диференціальної приватності є скінченна точність типів даних стандарту IEEE 754

та механізми заокруглення FP арифметики [194]. Аналіз молодших бітів результату лише кількох запитів до бази даних дозволяє зламати систему захисту диференційної приватності та навіть отримати її вміст.

Застосування FP арифметики може також служити джерелом загроз при генерації ПВП [51,165]. При цьому, числа формату IEEE 754 можуть використовуватись як у внутрішній структурі генераторів [56,165,198,199], так і в якості формату представлення вихідних даних [165,166,194,198,199]. Зокрема, при генерації дійсних випадкових та ПВЧ з інтервалу $[0;1)$ у FP форматі потрібно пам'ятати про нерівномірну щільність таких чисел на числовій осі.

Використання чисельних методів апроксимації елементарних функцій в FP арифметиці як ключових блоків ГПВП може виступати як додатковий елемент нелінійності, що підвищує ентропію та зменшує передбачуваність згенерованих ПВП. Як бачимо, вибраний підхід до проєктування ГПВЧ та ПВБП є неоднозначним щодо можливих статистичних характеристик сформованих ПВП та можливих ризиків щодо їх криптографічної стійкості. Тому важливо ретельно тестувати програми, що використовують FP обчислення, зокрема ГПВП, досліджувати точність, незмінність, передбачуваність одержаних результатів, їх ентропію та статистичні характеристики. Для критичних застосувань тестування здійснюється при всіх можливих значеннях вхідних аргументів та при різних виняткових ситуаціях.

Розглянемо детальніше використану методику дослідження ГПВЧ, побудованих на основі чисельних методів апроксимації елементарних функцій в FP арифметиці, зокрема методи і засоби оцінки випадковості, визначення періоду повторення, статистичних характеристик сформованих ГПВЧ послідовностей, визначення розмірності простору початкових значень, оцінки швидкодії та криптографічної стійкості запропонованих ГПВЧ на основі FP арифметики.

2.2.2 Методика дослідження статистичних характеристик ПВБП для задач у сфері кібербезпеки та захисту інформації на основі тестів NIST

Якість статистичних характеристик згенерованих ПВБП визначалася на основі методики досліджень та пакету тестів NIST STS (Statistical Test Suite)

[78,200]. Перелік реалізованих в даному пакеті статистичних тестів наведений у табл. 2.1. Можна побачити, що 5 із 15 тестів містять декілька підтестів – від 2 до 148 – всього 188 тестів (підтестів).

Таблиця 2.1 – Перелік статистичних тестів NIST STS та їх параметри за замовчуванням

№ тесту	Статистичний тест NIST	Параметри за замовчуванням	Кількість підтестів (всього 188)
1.	Монобітний частотний тест (Frequency)		1
2.	Частотний блоковий тест (Block Frequency)	$M = 128$	1
3.	Тест накопичених сум (Cumulative Sums)		2
4.	Тест серій однакових бітів (Runs)		1
5.	Тест найдовшої серії одиниць (Longest Run)		1
6.	Тест перевірки рангів бінарних матриць (Rank)		1
7.	Спектральний тест на основі дискретного перетворення Фур'є (FFT)		1
8.	Тест на відповідність шаблону без перекриття (Non-Overlapping Template)	$M = 9$	148
9.	Тест на відповідність шаблону з перекриттям (Overlapping Template)	$M = 9$	1
10.	Універсальний статистичний тест Маєра (Universal)		1
11.	Тест наближеної ентропії (Approximate Entropy)	$M = 10$	1
12.	Тест на довільні відхилення (Random Excursions)		8
13.	Варіант тесту на довільні відхилення (Random Excursions Variant)		18
14.	Тест на підпоследовності (Serial)	$M = 16$	2
15.	Тест лінійної складності (Linear Complexity)	$M = 500$	1

NIST рекомендує під час тестування для задач у сфері КБ та ЗІ, зокрема для криптографічних застосувань, наступні параметри: $n \geq 10^6$ біт, $m \geq 1000$ та $\alpha = 0,01$, де n – розмір підпоследовності, m – їх кількість, а α – заданий рівень значущості [78,159]. Тому для статистичного тестування ми генерували последовності розміру $125 \cdot 10^6$ байт та використовували параметри $n = 10^6$, $\alpha = 0,01$ та, як правило, $m = 1000$.

Для кожної підпослідовності значення P -value має бути не менше за обраний рівень значущості: $P\text{-value} \geq \alpha$. Статистичний тест вважається пройденим згідно з стратегією 1, якщо серед, наприклад, $k = 1000$ протестованих підпослідовностей частка проходження P потрапляє у довірчий інтервал (1.56). Результати тестів NIST у цьому випадку представляються графічно у вигляді т. зв. статистичного портрету послідовності (див. рис. 2.11). Варто зауважити, що в залежності від характеристик вхідних послідовностей статистичні тести Random Excursions та Random Excursions Variant не завжди застосовні [37,168]. У даному випадку результат отримується на основі меншої кількості значень P -value. Зазвичай це практично не впливає на якість результатів тестування, якщо дослідження повторюються для достатньої кількості послідовностей, що мають непогані статистичні характеристики [168]. Згідно з стратегією 2 досліджується розподіл значень P -value при розбитті проміжку $[0;1]$ на 10 рівних частин з кроком 0,1. Визначається кількість тестів, що потрапляє в кожен з проміжків, а рівномірність оцінюється або графічно у вигляді гістограми, або за допомогою тесту χ^2 . Початкова гіпотеза H_0 приймається, якщо отримане значення задовольняє нерівності $P(\chi^2) \geq 0,0001$ [78,164]. Детальнішу інформацію щодо інтерпретації результатів тестування можна знайти в роботах [37,78,159,164].

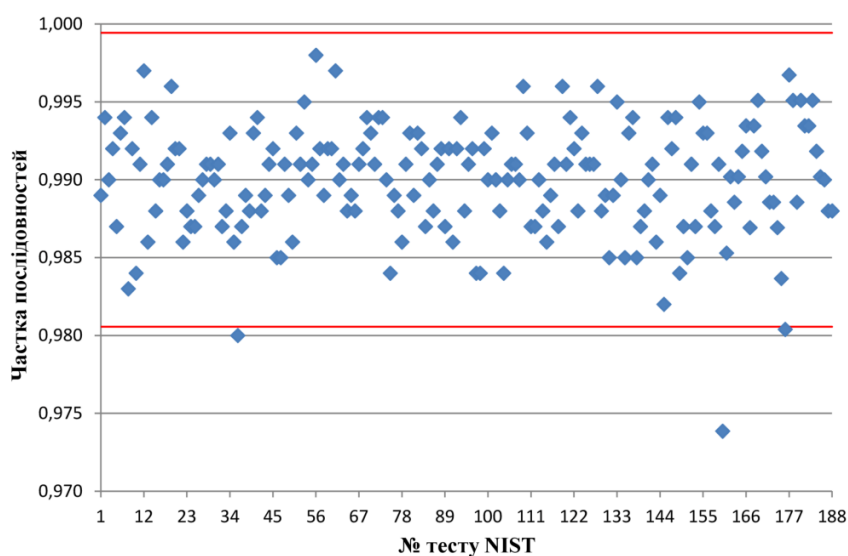


Рисунок 2.11 – Приклад статистичного портрету ПВБП, побудований на основі результатів тестів NIST STS за методикою оцінки 1

Дослідження виконувалось в три етапи. На першому етапі швидкого статистичного тестування генераторів при фіксованому початковому значенні $x_0 = 0$ ми поступово збільшували значення параметра $m \in \{10, 100, 1000\}$ та відкидали послідовності, що провалили більше ніж 2 статистичні тести із 188. На другому етапі при різних початкових значеннях x_0 ми використовували параметр $m = 1000$. На третьому етапі для ГПВЧ із найкращими статистичними характеристиками параметр m визначається на основі довжини послідовності, визначеної в байтах, що рівна періоду L ПБП, таким чином, що

$$m = \lfloor L / (1,25 \cdot 10^5) \rfloor. \quad (2.48)$$

Крім того, для сформованих на третьому етапі послідовностей ПБЧ розміру L 8-бітових цілих значень застосовувались графічні тести та будувались їх гістограми.

Аналіз одержаних результатів статистичних тестів NIST здійснювався на основі класичного методу оцінки, реалізованого в пакеті NIST STS [200], а також на основі стратегії 1. В обох випадках для оцінки статистичних характеристик сформованих ПБП будувались їх статистичні портрети та визначалась загальна кількість пройдених тестів, а для оцінки якості ГПВЧ обчислювались усереднені результати для різних початкових значень та режимів роботи, загальна кількість послідовностей, що успішно пройшли тестування тощо.

Для автоматизації процесу тестування згенерованих ПБП статистичними тестами NIST STS, що реалізовані в вигляді виконуваного файлу *assess.exe* [200], у ОС Windows нами було розроблено декілька bat-файлів, що реалізують:

- 1) Тестування з заданими параметрами однієї ПБП, збереженої у вигляді бінарного файлу достатнього розміру, та збереження результату в окремому файлі (див. додаток В.1).
- 2) Виведення в окремий файл значень P , що характеризують рівень проходження кожного тесту, та підрахунок кількості не пройдених тестів (див. додаток В.2).
- 3) Статистичне тестування та визначення періоду повторення всіх ПБП, що знаходяться в певній папці:

- a. із фіксованим значенням параметра m (див. додаток В.3);
- b. із автоматичним збільшенням параметра m (див. додаток В.4);
- c. із автоматичним вибором параметра m на основі періоду згідно з формулою (2.48).

2.2.3 Визначення періоду повторення ГПВЧ

Алгоритми генерації ПВЧ зазвичай формують періодичні послідовності, в тому числі запропоновані генератори на основі FP арифметики. Для більшості генераторів це є наслідком практичного обмеження розрядності регістрів внутрішнього стану ГПВЧ. Максимальний період повторення ГПВЧ у більшості випадків можна оцінити на основі теоретичного аналізу, проте важливо також визначити реальний період повторення сформованих послідовностей в залежності від початкових значень та інших параметрів алгоритму генерації ПВЧ.

Для визначення періоду повторення L запропонованих генераторів було сформовано набір ПБП достатньо великого розміру – 2 Гб. Тоді, мовою C++ було реалізовано програму *calculate_period_of_the_sequence_from_input_file.exe* для визначення періоду повторення L та зміщення L_0 , якщо вони присутні в сформованих послідовностях (див. додаток Б.4). Програма опрацьовує вхідний файл починаючи із кінця послідовності та здійснює спроби підбору необхідного розміру L , починаючи із значення $L=1$ байт. Програма завершує свою роботу, якщо виявлено достатню кількість повторів значень ПБП, що відповідають періоду L . Після того, як знайдено період L , визначається значення зміщення L_0 .

Для подальшого використання у згаданих вище bat-файлах результатів виконання цієї програми (значень L_0 , L та розміру вхідного файлу), зокрема для скрипта 3с, було розроблено додатковий допоміжний bat-файл, що опрацьовує виведені дані в консоль (див. додаток В.5).

2.2.4 Аналіз швидкодії генераторів ПБП

Оцінку швидкодії ГПВЧ будемо виконувати як на основі теоритичного аналізу середньої кількості основних арифметичних операцій необхідних для генерації одного ПВЧ, так і на основі практичного вимірювання пропускну

здатності генератора – середньої кількості згенерованих ПВЧ (байт) за одиницю часу. Для практичного заміру часу на різних платформах будемо використовувати ці ж методи та бібліотеки, які розглядались у п. 2.1.2.

Висновки до розділу 2

1. Запропоновано методику проєктування та дослідження вдосконалених алгоритмів обчислення елементарних функцій, придатних для задач у сфері інформаційної та кібербезпеки, зокрема використання імітаційного моделювання (побудова аналітичної моделі алгоритмів на основі методу МС, полінома найкращого рівномірного наближення щодо відносної похибки згідно методом Ремеза, спеціальних апаратних інструкцій наближення функцій RCP і RSQRT та ітераційних методів). Побудована математична модель (2.11) описує алгоритми обчислення RSQRT на основі наближених апаратних інструкцій, зокрема FRSQRTE від ARM та RSQRTSS від Intel, та легко модифікується для RCP інструкцій. Розроблено методику покращення алгоритмів, що застосовує спеціальну форму запису ітераційних та поліноміальних методів з використанням FMA та модифікацію їх параметрів з метою мінімізації максимальних відносних похибок $\delta_{\max}$ на основі теоретичного аналізу та чисельної оптимізації. Запропоновано набір методів рандомізованої чисельної багатовимірної жадібної оптимізації, що застосовують методи багатовимірного прямого пошуку та золотого перетину.
2. Сформульовано практичні та теоретичні методи визначення абсолютних та відносних похибок, оцінки точності алгоритмів, аналізу збіжності ітераційних методів, вимірювання їх швидкодії на різних платформах. З'ясовано, що в загальному випадку модифіковані методи на основі алгоритму FISR є більш ефективними за поліноми найкращого рівномірного наближення функцій SQRT, RCP та RSQRT та мають швидшу збіжність (приблизно на 5 біт вища точність для однієї ітерації). Проведено теоретичний аналіз збіжності ітераційних методів NR, звичайного й

модифікованого методу Н різних порядків та їх комбінацій для початкового наближення функції RSQRT методом МС, здійснено аналіз та порівняння згаданих методів.

3. Проаналізовано можливі загрози використання FP арифметики у критичних системах, зокрема у сфері кібербезпеки та захисту інформації.
4. Розроблено методи проєктування та дослідження ГПВЧ для застосувань у сфері кібербезпеки та захисту інформації, зокрема на основі FP арифметики, а саме: дослідження статистичних характеристик та випадковості сформованих ПВП з використанням тестів NIST для криптографічних цілей, автоматизоване тестування зміщення та періоду повторення періодичних ПВП, а також швидкодії ГПВЧ. Запропоновано новий спосіб генерації ПВП, що використовує чисельні методи апроксимації елементарних функцій в FP арифметиці. Методика дослідження передбачає три етапи статистичного тестування для різних довжин ПВП при зміні початкових значень та параметрів роботи ГПВЧ.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ГЕНЕРАТОРІВ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ НА ОСНОВІ ВДОСКОНАЛЕНИХ АЛГОРИТМІВ ОБЧИСЛЕННЯ ЕЛЕМЕНТАРНИХ ФУНКЦІЙ

3.1 Проєктування ефективних модифікованих алгоритмів обчислення елементарних функцій на основі методу МС для задач у сфері кібербезпеки та захисту інформації

Таким чином існує три основних підходи для підвищення точності існуючих методів на основі МС для задач у сфері кібербезпеки та захисту інформації, зокрема алгоритму FISR: модифікація їх параметрів для мінімізації $\delta_{\max}$, застосування ітераційних методів вищих порядків та використання різних МС залежно від значення вхідного аргументу. Комбінації зазначених підходів в поєднанні з записом ітераційних методів NR та H в особливій формі на основі використання FMA були використані нами, зокрема, при проєктуванні модифікацій алгоритму FISR [7,8,80-83]. У цих роботах наведені покращені алгоритми для обчислення функцій RSQRT [7,8,81-83], SQRT [8,82] та RCP [80] для чисел одинарної [7,8,80-83] та/або подвійної [7,8,82] точності. Як було розглянуто у п. 1.4 та п. 2.1.1, більшість з запропонованих методів можуть бути використані також для інших елементарних функцій, зокрема DIV, RCBRT, CBRT, $y = 2^x$, $y = e^x$, $y = \log_2 x$ та $y = \ln x$.

Для прикладу, в роботах [7,81,82] для підвищення точності модифікованого алгоритму FISR до необхідного рівня FP типів *float* та *double* крім модифікованого методу NR було використано метод Ho2 (1.39), (1.43). При цьому, в першому випадку [7,81] цей метод застосовувався для останньої ітерації алгоритмів у класичному або модифікованому вигляді у формі (2.27) та (2.32). У другому випадку [82] модифікований метод Ho2 у формі (1.39) та (2.34) використовується лише для першої ітерації, де він у поєднанні з МС формує нове початкове наближення. Нагадаємо, що цей ітераційний метод містить на одне $\times$ та $-$ більше (або одне FMA) ніж метод NR, забезпечуючи кубічну збіжність ітерації. Як альтернативний метод підвищення точності алгоритмів, у роботах [8,83] нами

було вперше запропоновано виконувати розбиття вхідного проміжку [1; 4) для функції RSQRT методу MC на декілька частин, зокрема $x \in [1; 2)$ та $x \in [2; 4)$, та, відповідно, використання кількох MC R . Ці алгоритми базуються лише на модифікованих та класичних ітераціях NR, а тому не вимагають додаткових операцій в FP арифметиці. При цьому, як завжди, визначаються такі константи основного алгоритму, що мінімізують $\delta_{\max}$ після першої ітерації.

У випадку функції RCP для типів *float* та *double* достатньо використовувати лише модифіковані та класичні ітерації NR, щоб одержати алгоритми з високим рівнем точності [80]. При цьому необхідно дві ітерації для чисел одинарної точності та три ітерації для чисел подвійної точності. При застосуванні в алгоритмі модифікованої ітерації Ho2 можна також отримати результат RCP повної точності із коректним заокругленням.

Як наслідок, запропоновані нами алгоритми на практиці дозволяють одержати таку точність результату, що гарантує коректне обчислення функції, за винятком, можливо, останнього біту. Для FP чисел типу *float* це як мінімум 23 коректних біти, для чисел типу *double* – 52 біти. Для того, щоб одержати правильно округлений результат, крім функції RCP, застосовується додатковий крок коригування похибок [201] або на останній ітерації використовується проміжний тип даних вищої точності. У роботах [7,83] також розглядалися методи, що дозволяють зменшити кількість $\times$ в FP арифметиці, виконуючи операції з цілочисловим представленням числа (зокрема, експонентою).

Розглянемо деякі з цих методів детальніше. Запропоновані в цьому розділі алгоритми будемо називати відповідно до наступної схеми: *<функція><метод><кількість_ітерацій*><версія_алгоритму*><тип_даних>*, де * позначено елементи, що можуть пропускатися. Надалі будемо використовувати такі покращені алгоритми обчислення RCP та RSQRT як базові блоки для побудови генераторів ПВП, придатних до використання у задачах кібербезпеки та інформаційної безпеки.

3.1.1 Покращені алгоритми обчислення функцій RCP та DIV підвищеної точності та швидкодії

Для забезпечення високої точності алгоритмів для RCP першу та другу ітерації будемо проводити методом NR згідно з формулами відповідно (2.21) для модифікованої першої ітерації із параметрами k_1 і k_2 та (2.18), (2.19) для класичної останньої ітерації. Тоді, для знаходження оптимальних теоретичних значень МС R та коефіцієнтів першої ітерації, як було зазначено у п. 2.1.3, використаємо наступний метод для кожного з проміжків $[1; x')$ та $[x'; 2)$. Після проведення першої ітерації з лінійним початковим наближенням y_0 змінна y_1 описується поліномами 3-го порядку, а для забезпечення найменшої можливої відносної похибки необхідно, щоб це були поліноми P_3^* [80]. Для цього виберемо таке значення t , що вирівнює ліву і праву частину рис. 2.5а. У даному випадку, $t = \sqrt{2} - 1$ (і відповідно $x' = 1 + t = \sqrt{2}$). Тоді, для чисел типу *float* МС набере вигляду $R = 0x7eb504f3$, а для типу *double* $R = 0x7fd6a09e667f3bcd$. Підставивши значення $t = \sqrt{2} - 1$ у формули відповідних поліномів для y_{1i} та порівнявши їх з P_3^* , отриманими з допомогою метода Ремеза, знайдемо наступні значення k_1 і k_2 : $k_1 = 1,940908883185$ та $k_2 = 1,43566017177982$. Після першої ітерації практичні похибки цього алгоритму для типу *float* складають: $\delta_{1\max}^+ = 1,1173 \cdot 10^{-4}$ та $\delta_{1\max}^- = -1,1170 \cdot 10^{-4}$, $\alpha_1 = 13,13$; після другої ітерації – $\delta_{2\max}^+ = 5,8954 \cdot 10^{-8}$ та $\delta_{2\max}^- = -7,1665 \cdot 10^{-8}$, $\alpha_2 = 23,73$ [80]. Це на 4 біти точніше, ніж алгоритм Мороза та Гринчишина [152] після однієї та після двох ітерацій. При порівнянні запропонованого алгоритму для однієї ітерації з P_3^* , що задається виразом (2.8), спостерігається зменшення $\delta_{1\max}$ більше ніж у 15 раз. Після застосування методів чисельної оптимізації на основі комбінації методів 1 та 3 для практичного вдосконалення параметрів R , k_1 та k_2 ми отримали алгоритм Rcp12f, що наведений у лістингу 3.1 (див. табл. Г.7). Похибки алгоритму складають $\delta_{2\max}^+ = 5,9020 \cdot 10^{-8}$ та $\delta_{2\max}^- = -6,8615 \cdot 10^{-8}$, $\alpha_2 = 23,8$ (див. рис. 3.1).

Лістинг 3.1 – Оптимізований алгоритм RCP для чисел одинарної точності

```

1: float Rcp12f (float x)
2: {
3:     int i = *(int*)&x;
4:     i = 0x7eb53567 - i; // модифікована магічна константа R
5:     float y = *(float*)&i;
6:     y = 1.9395974f*y*fmaf(y, -x, 1.436142f); // мод. ітер. NR у формі (2.21)
7:     float r = fmaf(y, -x, 1.0f);
8:     y = fmaf(y, r, y); // класична ітерація NR у формі (2.18), (2.19)
9:     return y;
10: }

```

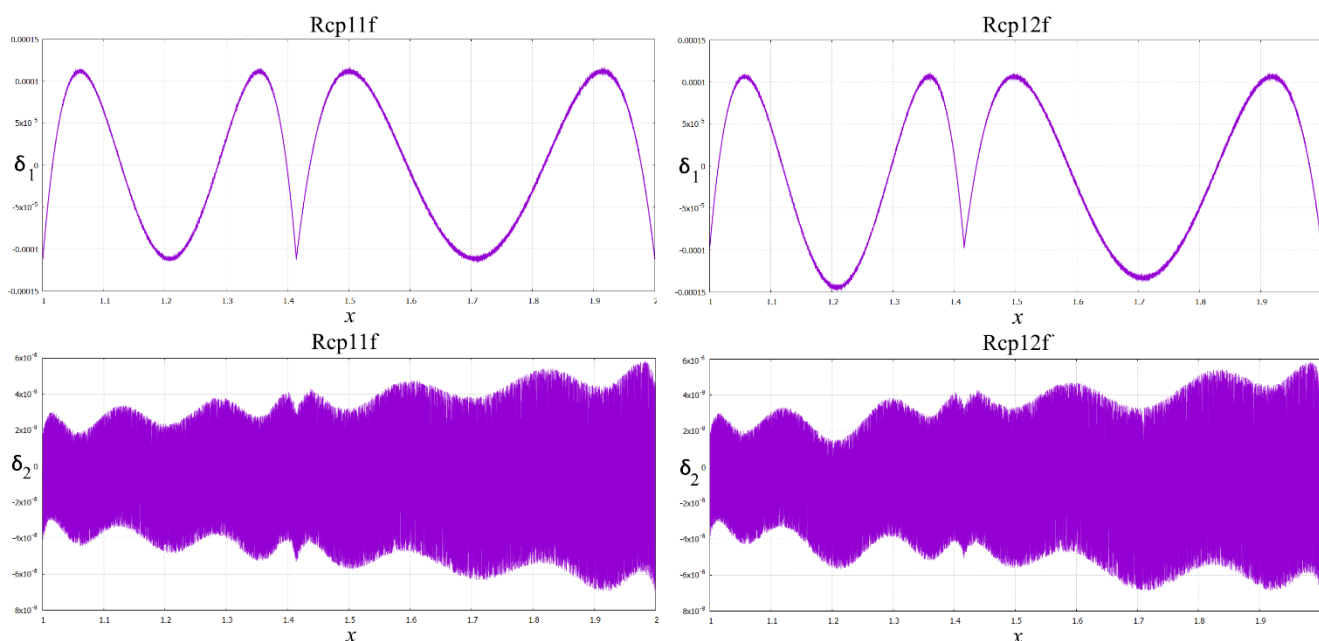


Рисунок 3.1 – Практичні похибки після першої та другої ітерацій базового та оптимізованого алгоритмів (Rcp11f та Rcp12f) для обчислення RCP типу *float*

Для випадку чисел подвійної точності (аргумент типу *double*) отримаємо наступний алгоритм на основі визначених теоретичних параметрів R , k_1 та k_2 :

Лістинг 3.2 – Версія алгоритму RCP для чисел подвійної точності

```

1: double Rcp131d (double x)
2: {
3:     uint64_t i = *(uint64_t*)&x;
4:     i = 0x7fd6a09e667f3bcd - i;

```

```

5:  double y = *(double*)&i;
6:  y = 1.940908883185003*y*fma(y, -x, 1.435660171779821);
    // мод. ітерація NR у формі (2.21)
7:  y = y*fma(y, -x, 2.0000000062735); // мод. ітерація NR у формі (2.20)
8:  double r = fma(y, -x, 1.0);
9:  y = fma(y, r, y); // класична ітерація NR у формі (2.18), (2.19)
10: return y;
11: }

```

Алгоритм Rsp131d на відміну від Rsp12f містить три ітерації, друга з яких також є модифікованою. Відповідно, в алгоритмі появляється додатковий параметр k_3 . Похибки цього алгоритму складають $\delta_{3\max}^+ = 1,1070 \cdot 10^{-16}$ та $\delta_{3\max}^- = -1,4919 \cdot 10^{-16}$, $\alpha_3 = 52,57$. У цьому записі алгоритм містить $3\times$ та 4FMA. Версія цього алгоритму з оптимізованими параметрами, Rsp132d, має похибки $\delta_{3\max} = 1,4864 \cdot 10^{-16}$. Якщо порівнювати цей алгоритм із відомим методом Гольдшмідта [154] ($3\times$ та 5FMA) для чотирьох ітерацій, що має значно вищу похибку $\delta_{4\max} = 4,4290 \cdot 10^{-16}$, $\alpha_4 = 51$ точний біт, то він використовує і меншу кількість арифметичних операцій.

Для підвищення швидкодії алгоритмів на деяких платформах, зокрема на МК ESP-32 чи при їх апаратній реалізації на ПЛІС, можна спробувати підібрати такі значення певних констант алгоритму, що є степенями двійки або представляються у вигляді їх простої комбінації [7]. Це дозволить реалізовувати швидке $\times$ на такі константи, наприклад шляхом модифікації експоненти числа. Аналогічним чином ми реалізували алгоритми Rsp22f та Rsp23d при фіксованому значенні $k_1 = 2 - 2^{-4} = 1,9375$. У даному випадку отримаємо наступні аналітичні значення МС та параметра k_2 : $R = 0x7eb52a74$ для чисел одинарної точності, $R = 0x7fd6a54e79bcc0f6$ для чисел подвійної точності, а $k_2 = 1,4369233123786418$. Проте, отримані алгоритми дещо поступаються в точності вищенаведеним алгоритмам: $\delta_{1\max} = 1,1285 \cdot 10^{-4}$ та $\delta_{2\max} = 7,0927 \cdot 10^{-8}$ для чисел типу *float*; $\delta_{3\max} = 1,5067 \cdot 10^{-16}$ для чисел типу *double* (див. додаток Д).

Алгоритми на основі модифікованого методу Но2

Інший варіант оптимізованого алгоритму, Rcp31f, що базується на модифікованому методі Но2 (2.36), (2.41) із параметрами $R = 0x7ef3344b$, $k_1 = 3,0019228458$ і $k_2 = -3,0019216537$, має значно вищу точність після першої ітерації за Rcp11f ($\alpha_1 = 14,9$), проте вимагає додаткового $-$. Похибки цього алгоритму для типу *float* складають: $\delta_{1\max}^+ = 3,2622 \cdot 10^{-5}$ та $\delta_{1\max}^- = -3,2684 \cdot 10^{-5}$, а після другої ітерації NR у формі (2.18), (2.19) – $\delta_{2\max}^+ = 5,8555 \cdot 10^{-8}$ та $\delta_{2\max}^- = -6,0643 \cdot 10^{-8}$, $\alpha_2 = 23,98$ (див. табл. Г.7). Версія цього алгоритму для типу *double*, Rcp331d, що використовує $R = 0x7fde668c791d0766$ та $k_3 = 2,0000000005276211$ у другій модифікованій ітерації NR у формі (2.20), має наступні вирівняні похибки: $\delta_{1\max} = 3,2484 \cdot 10^{-5}$, $\delta_{2\max} = 5,2762 \cdot 10^{-10}$ та майже повну точність після третьої ітерації, $\delta_{3\max} = 1,1129 \cdot 10^{-16}$. Після оптимізації параметрів для двох ітерацій, алгоритм для чисел одинарної точності забезпечує коректне заокруглення результату ($e_2 = 24$) і має наступний вигляд:

Лістинг 3.3 – Оптимізований алгоритм RCP повної точності типу *float*, що використовує модифікований метод Но2

```

1: float Rcp321f (float x)
2: {
3:   int i = *(int*)&x;
4:   i = 0x7ef311c3 - i; // модифікована магічна константа R
5:   float y = *(float*)&i;
6:   float c = x*y;
7:   y = y*fmaf(c - 3.003093f, c, 3.0030725f);
      // мод. ітер. Н 2-го порядку у формі (2.36), (2.41)
8:   float r = fmaf(y, -x, 1.0f);
9:   y = fmaf(y, r, y); // класична ітерація NR у формі (2.18), (2.19)
10:  return y;
11: }
```

Аналогічно, версія алгоритму для типу *double* із оптимізованими параметрами $R = 0x7fde669bbebc1fc9$, $k_1 = 3,0019250985340205$, $k_2 = -3,0019212251167886$ та $k_3 = 2,00000000105761888$ також забезпечує повну точність результату ($e_3 = 53$).

Реалізація $DIV(a, b)$ на основі запропонованих алгоритмів $RCP(b)$ відповідно до формули (2.14) передбачає модифікацію останньої ітерації:

```
float r = fmaf(y, -b, 1.0f);
y = a*fmaf(y, r, y); // модифікація алгоритму для операції DIV
```

3.1.2 Прості модифікації алгоритму FISR

Як зазначено в розділі 1, багато науковців займалися визначенням оптимальних параметрів алгоритму FISR щодо $\delta_{\max}$ [7,81,148,149]. Метою є визначення оптимізованих значень R , k_1 та k_2 для першої модифікованої ітерації NR у формі (2.30) або (2.31). Аналогічно для другої модифікованої ітерації в формі (2.31) з параметрами k_3 та k_4 або в модифікованій особливій формі:

$$r_n = k'_3 - x \cdot y_n \cdot y_n \quad (3.1)$$

$$y_{n+1} = y_n + k'_4 \cdot y_n \cdot r_n, \quad (3.2)$$

де параметри k'_3 та k'_4 необхідно додатково оптимізувати (див. табл. 3.1).

Як видно із табл. 3.1 найвищу точність після першої ітерації ($\alpha_1 = 10,59$) мають алгоритм Кадлеця [149] та запропоновані алгоритми на основі MC $R = 0x5e9fead4$ (InvSqrt31f) та, у випадку використання FMA, $R = 0x5f5ffff8$ (InvSqrt41f) [81]. Зауважте, що параметри R , k_1 та k_2 запропонованого алгоритму 1 (InvSqrt31f) отримані виключно на основі чисельних методів пошуку, описаних у п. 2.1.3, та аналізу отриманих результатів (див. рис. 2.10 та 3.2). Після другої ітерації NR у класичній формі запису найвищу точність ($\alpha_2 = 21,02$) має алгоритм 1 (InvSqrt321f), а у формі запису (3.1), (3.2) з використанням FMA на основі параметрів k'_3 та k'_4 має також оптимізований алгоритм із роботи [7] (InvSqrt521f), що має трохи нижчі похибки, $\delta_{2\max} = 3,9734 \cdot 10^{-7}$, $\alpha_2 = 21,26$.

Таблиця 3.1 – Результати покращення параметрів найкращих модифікацій алгоритму FISR для однієї та двох ітерацій у класичній та в особливій формі запису з використанням FMA

Алгоритм		Вальчика та ін. [148]	Кадлеця [149]	Запропоно- ваний 1	Запропоно- ваний 2 [81]	Запропоно- ваний 3 [7]	Запропоно- ваний 4 [7]
Перша модифікована ітерація без FMA та з FMA							
Структура ітерації NR		(2.31)	(2.30)	(2.31)	(2.30)	(2.30)	(2.30)
R		0x5F200000	0x5F1FFFF9	0x5E9FEAD4	0x5F5FFFF8	0x5F5FB6D3	0x5F5FB432
k_1		1,68191391	0,703952253	3,36527634	0,248884737	0,25	0,25
k_2		0,703952009	2,38924456	5,63889456	4,778488636	4,764266968	4,76405191
$\delta_{1\max}$	без FMA	$6,5029 \cdot 10^{-4}$	$6,5020 \cdot 10^{-4}$	$6,5020 \cdot 10^{-4}$	$6,5021 \cdot 10^{-4}$	$6,5026 \cdot 10^{-4}$	$7,2045 \cdot 10^{-4}$
	з FMA	$6,5026 \cdot 10^{-4}$	$6,5020 \cdot 10^{-4}$	$6,5020 \cdot 10^{-4}$	$6,5020 \cdot 10^{-4}$	$6,5024 \cdot 10^{-4}$	$7,2042 \cdot 10^{-4}$
Друга модифікована ітерація в класичній формі (2.31) без FMA							
k_3		1,50000036	1,5000021	1,50000846	1,50001025	1,50000703	1,50010848
k_4		0,500000053	0,50000184	0,500008166	0,500009954	0,500006735	0,500108182
$\delta_{2\max}$		$4,8605 \cdot 10^{-7}$	$4,8125 \cdot 10^{-7}$	$4,6906 \cdot 10^{-7}$	$4,7980 \cdot 10^{-7}$	$4,7562 \cdot 10^{-7}$	$4,7446 \cdot 10^{-7}$
Друга модифікована ітерація в особливій формі (3.1), (3.2) з FMA							
k'_3		1,00000006	1,00000006	1,00000006	1,00000006	1,00000006	1,00000006
k'_4		0,500002265	0,500001788	0,500000834	0,499998182	0,499998093	0,500097573
$\delta_{2\max}$		$4,0094 \cdot 10^{-7}$	$4,0100 \cdot 10^{-7}$	$3,9923 \cdot 10^{-7}$	$4,0681 \cdot 10^{-7}$	$4,0715 \cdot 10^{-7}$	$3,9734 \cdot 10^{-7}$

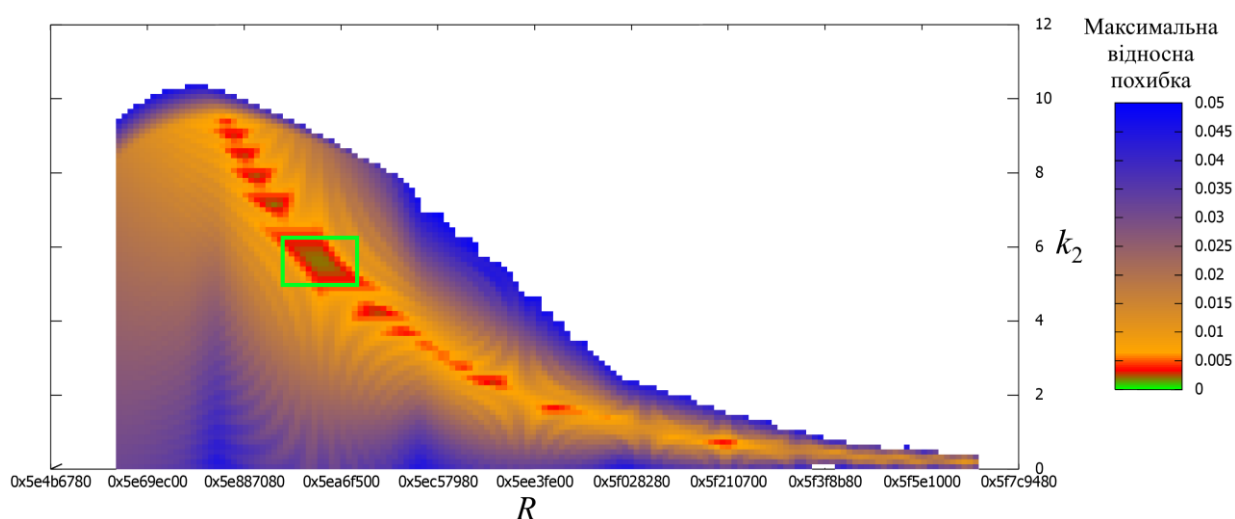


Рисунок 3.2 – Локалізація області дослідження для задачі 3D оптимізації алгоритму FISR, $3 \leq k_1 \leq 3,6$. Результат чисельної оптимізації:

$R = 0x5e9fead4$, $k_1 = 3,36527634$ та $k_2 = 5,63889456$ при $\delta_{1 \max} = 6,5020 \cdot 10^{-4}$

3.1.3 Покращені алгоритми обчислення функцій RSQRT та SQRT підвищеної точності

Далі детальніше розглянемо покращені алгоритми на основі модифікацій FISR, що описані у [7,81].

Алгоритми на основі простих модифікацій FISR

МС Кадлеця $R=0x5f1ffff9$ має параметри $Q=190$ і $t=0,249999167 < 1/2$, а МС $R=0x5e9fead4$ – $Q=189$ і $t=0,24935389 < 1/2$. Тому, варто розглянути детальніше випадок $Q=190$ та $t \geq 1/2$, зокрема коли $t=0,75$. В цьому випадку, відповідно до формул аналітичної моделі (2.1)-(2.4), $x' = 2t + 1/N_m \approx 1,5$ – середина проміжку $[1; 2)$, а МС $R=0x5f600000$ та $R=0x5febffff9a15000$ для чисел одинарної та подвійної точності [81]. Модифікацію першої ітерації NR виконуємо у вигляді (2.30) із двома параметрами k_1 та k_2 , які необхідно знайти шляхом розв'язку відповідної системи із двох рівнянь. Виявляється, що в цьому випадку для вирівнювання більшості точок максимумів достатньо зрівняти похибки δ_{1i} у трьох ключових точках, що визначаються параметром t : точка максимуму x_{11}^+ на першому проміжку, точка з'єднання першого та другого проміжків x' , що є точкою мінімуму x_{20}^- , та точка мінімуму на третьому проміжку x_{32}^- [7,81] (див. рис. 3.3). Звідси отримаємо значення невідомих параметрів модифікованої ітерації NR: $k_1=0,2488850264045049$ та $k_2=4,778490637430022$. На практиці для чисел типу *float* кращими будуть трохи модифіковані параметри $R=0x5f5ffff8$, $k_1=0,248884737$ та $k_2=4,778488636$ (InvSqrt41f) [81], як показано у табл. 3.1. Похибка цього алгоритму теж складає $\delta_{1\max} = 6,5020 \cdot 10^{-4}$ у випадку застосування FMA. Графік відносних похибок δ_1 алгоритму наведений на рис. 3.3. Для підвищення точності алгоритму після першої ітерації застосовуємо метод Но2 у формі (2.27), (2.32), що містить $2\times$ та $3FMA$. Максимальні відносні похибки цього алгоритму (InvSqrt42f) рівні $\delta_{2\max}^+ = 8,9589 \cdot 10^{-8}$ та $\delta_{2\max}^- = -8,7765 \cdot 10^{-8}$, а бітова похибка рівна $\alpha_2 = 23,41$ [81].

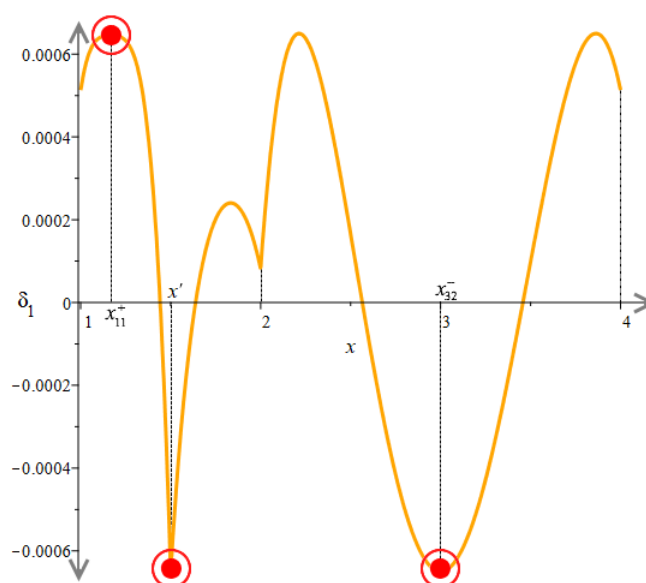


Рисунок 3.3 – Вибір точок максимумів x_{ij}^+ та мінімумів x_{ij}^- похибок δ_{1i} після першої ітерації модифікованого алгоритму FISR для процесу вирівнювання їх максимальних значень: $|\delta_{1i}(x_{ij}^+)| = |\delta_{1i}(x_{ij}^-)|$

Аналогічним чином розглядається випадок із фіксованим значенням параметра $k_1 = 2^{-2} = 0,25$. Тоді будується подібна система рівнянь для знаходження невідомого значення t магічної константи R та параметра k_2 , розв'язком якої є наступні теоретичні значення параметрів для чисел одинарної точності: $t = 0,7477678099609422$ та $k_2 = 4,764267011868366$ [7]. Звідки отримаємо теоретичне значення MC $R = 0x5f5fb6db$. На практиці менші похибки мають параметри $R = 0x5f5fb6d3$ та $k_2 = 4,764266968$ для типу *float* (див. табл. 3.1) і $R = 0x5febf6db526de7d9$ та $k_2 = 4,7642670066528519$ для типу *double*. Якщо порівнювати цей алгоритм (InvSqrt51f) з запропонованим алгоритмом 2 (InvSqrt41f), то цей алгоритм має трохи вищу похибку після першої ітерації, $\delta_{1\max} = 6,5024 \cdot 10^{-4}$ при реалізації з FMA, проте дозволяє використати згаданий раніше метод для зменшення кількості $\times$ в алгоритмі (розглянемо детальніше в п. 3.1.4). У випадку реалізації цього алгоритму для чисел подвійної точності отримаємо максимальну відносну похибку $\delta_{1\max} = 6,5014 \cdot 10^{-4}$. Інші варіанти

алгоритмів із [7], побудованих на основі цих параметрів, включають, зокрема, використання другої модифікованої ітерації NR в формі (3.1), (3.2) та ітерації Ho2 в формі (2.27), (2.32) з модифікованими параметрами, при використанні FMA на останній ітерації та багатовимірній оптимізації параметрів всього алгоритму. Перший з цих двох вдосконалених алгоритмів (InvSqrt521f), що наведений у лістингу 3.4, має похибки $\delta_{2\max}^+ = 3,7567 \cdot 10^{-7}$ та $\delta_{2\max}^- = -3,9734 \cdot 10^{-7}$. Такий варіант фінальної оптимізації алгоритмів, як правило, дозволяє ще трохи підвищити їх точність (для порівняння див. $\delta_{2\max} = 4,0715 \cdot 10^{-7}$ із табл. 3.1). Другий вдосконалений алгоритм (InvSqrt522f), що використовує метод Ho2 із модифікованими параметрами $R = 0x5f5fb414$, $k_2 = 4,76410007$, $k_3 = 0,5$ та $k_4 = 0,374000013$, має максимальні похибки $\delta_{2\max}^+ = 8,6041 \cdot 10^{-8}$ та $\delta_{2\max}^- = -8,1762 \cdot 10^{-8}$, що забезпечує бітову похибку $\alpha_2 = 23,47$. Проте, цей алгоритм містить вже $6\times$, 3FMA та одне $-$.

Лістинг 3.4 – Покращений оптимізований алгоритм для функції RSQRT одинарної точності

```

1: float InvSqrt521f (float x)
2: {
3:     int i = *(int*)&x;
4:     i = 0x5f5fb432 - (i>>1); // модифікована магічна константа R
5:     float y = *(float*)&i;
6:     y = 0.25f*y*(4.76405191f - x*y*y); // мод. ітерація NR (2.30) із  $k_1 = 1/4$ 
7:     float c = x*y;
8:     float r = fmaf(y, c, -1.0000006f);
9:     y = fmaf(-0.500097573f*y, r, y); // мод. ітер. NR у формі (3.1), (3.2)
10:    return y;
11: }
```

При застосуванні після першої ітерації алгоритму InvSqrt31f класичного методу Ho2 в особливій формі (2.27), (2.32) отримаємо алгоритм InvSqrt322f із

наступними значеннями похибок: $\delta_{2\max}^+ = 7,4352 \cdot 10^{-8}$ та $\delta_{2\max}^- = -7,4388 \cdot 10^{-8}$, бітова похибка $\alpha_2 = 23,68$, що є значно кращими за усі попередні алгоритми. Код алгоритму наведений у лістингу 3.5. У випадку модифікації методу Н та подальшої оптимізації параметрів алгоритму (InvSqrt323f) можна отримати максимальну похибку $\delta_{2\max} = 7,3902 \cdot 10^{-8}$, $\alpha_2 = 23,69$.

Лістинг 3.5 – Покращений алгоритм RSQRT для чисел одинарної точності, що використовує метод Ho2

```

1: float InvSqrt322f (float x)
2: {
3:     int i = *(int*)&x;
4:     i = 0x5e9fead4 - (i>>1); // модифікована магічна константа R
5:     float y = *(float*)&i;
6:     y = y*(3.36527634f - 5.63889456f*x*y*y); // мод. ітерація NR (2.31)
7:     float c = x*y;
8:     float r = fmaf(y, -c, 1.0f);
9:     float t = fmaf(0.375f, r, 0.5f);
10:    y = fmaf(y*r, t, y); // ітер. Н 2-го порядку у формі (2.27), (2.32)
11:    return y;
12: }
```

Для ефективної реалізації обчислення функції SQRT достатньо модифікувати останню ітерацію наведених вище алгоритмів. Для прикладу, версія алгоритму InvSqrt521f для SQRT буде мати другу ітерацію, що записана в вигляді:

```

7:     float c = x*y;
8:     float r = fmaf(y, c, -1.0000006f);
9:     y = fmaf(-0.500097573f*c, r, c); // модифікація алгоритму для SQRT
```

Похибки цього алгоритму трохи вищі, ніж у InvSqrt521f, а саме: $\delta_{2\max}^+ = 3,7782 \cdot 10^{-7}$ та $\delta_{2\max}^- = -4,0835 \cdot 10^{-7}$, бітова похибка $\alpha_2 = 21,22$. Кількість арифметичних операцій алгоритму при цьому не змінюється. Аналогічно, при

модифікації алгоритму `InvSqrt322f` для функції `SQRT` отримаємо наступні похибки: $\delta_{2\max}^+ = 8,8708 \cdot 10^{-8}$ та $\delta_{2\max}^- = -8,8947 \cdot 10^{-8}$, бітова похибка $\alpha_2 = 23,42$.

Алгоритми на основі модифікованого методу Но2

У даному випадку, на відміну від попередніх запропонованих алгоритмів для `RSQRT` на основі першої модифікованої ітерації `NR`, для підвищення точності алгоритмів у [82] ми пропонуємо використовувати також модифікований метод Но2 (1.39), (2.34). Цей метод застосовується лише для першої ітерації, а його параметри R , k_1 , k_2 та k_3 модифікуються таким чином, щоб мінімізувати максимальну відносну похибку алгоритму. Для уточнення результату після першої модифікованої ітерації ми використовуємо класичні та модифіковані методи `NR` в особливій формі запису. Основна відмінність запропонованого методу від алгоритмів Леметри та ін. [1] полягає в тому, що вони застосовують для початкового наближення класичні значення `MC` та стандартні параметри ітераційних методів `N` вищих порядків (див. табл. Г.3-Г.4). Зокрема, похибки цього алгоритму для Но2 типу `float` складають $\delta_{1\max}^+ = 1,0072 \cdot 10^{-4}$ та $\delta_{1\max}^- = -9,9025 \cdot 10^{-5}$, $\alpha_1 = 13,28$ одиниць бітової похибки. Після другої класичної ітерації `NR` у формі (2.27), (2.28) з використанням `FMA` практичні похибки цього алгоритму будуть рівні $\delta_{2\max}^+ = 7,1644 \cdot 10^{-8}$ та $\delta_{2\max}^- = -8,7479 \cdot 10^{-8}$, $\alpha_2 = 23,45$ (графік теоретичних похибок алгоритму показаний на рис. 1.4е).

Використовується модель початкового наближення (1.27)-(1.30). Застосувавши метод послідовних наближень, що був розглянутий у п. 2.1.3, для покращення параметрів цього алгоритму ми отримаємо наступний аналітичний розв'язок задачі вирівнювання відносних похибок після першої ітерації для чисел одинарної точності [82]: $t = 0,3233064922027810$, звідки отримаємо `MC` $R = 0x5f29621b$, $k_1 = 2,00718651889802436$, $k_2 = -1,53237125952242026$ та $k_3 = 0,52595012062439589$. Для чисел подвійної точності отримаємо $t = 0,32330655180742558$, звідки $R = 0x5fe52c437da986d9$. Остаточний алгоритм для типу `float` показаний у лістингу 3.6. Похибки після першої ітерації

складають $\delta_{1\max}^+ = 1,9730 \cdot 10^{-5}$ та $\delta_{1\max}^- = -1,9755 \cdot 10^{-5}$, бітова похибка $\alpha_1 = 15,63$, якщо трохи модифікувати параметр $k_2 = -1,5323712$ (InvSqrt61f). Таким чином, максимальна похибка зменшилась в 5,1 раз порівняно з базовим алгоритмом.

Лістинг 3.6 – Покращений алгоритм RSQRT для чисел одинарної точності на основі методу Ho2

```

1: float InvSqrt62f (float x)
2: {
3:     int i = *(int*)&x;
4:     i = 0x5f29621b - (i>>1); // модифікована магічна константа R
5:     float y = *(float*)&i;
6:     float c = x*y*y;
7:     y = y*(2.00718652f + c*(-1.5323713f + 0.52595012f*c));
        // мод. ітерація Н 2-го порядку (1.39), (2.34)
8:     c = x*y;
9:     float r = fmaf(y, c, -1.0f);
10:    y = fmaf(-0.5f*y, r, y); // ітер. NR у формі (2.27), (2.28)
11:    return y;
12: }
```

Якщо порівнювати цей метод із найкращими модифікаціями алгоритму FISR для однієї ітерації із табл. 3.1, то за рахунок використання додаткової операції $\times$ та одного $+$ похибка зменшилась в 32,9 раз, а точність зросла на 5 біт. Покращений алгоритм InvSqrt62f використовує $7\times$, 2FMA та $2+$ і має похибки $\delta_{2\max}^+ = 7,4139 \cdot 10^{-8}$, $\delta_{2\max}^- = -7,4134 \cdot 10^{-8}$, $\alpha_2 = 23,69$. Відповідні версії алгоритму для SQRT мають похибки $\delta_{1\max}^+ = 1,9770 \cdot 10^{-5}$ та $\delta_{2\max}^- = 8,8749 \cdot 10^{-8}$ ($\alpha_2 = 23,43$).

У випадку використання чисел типу *double* отримаємо наступні похибки після першої ітерації: $\delta_{1\max}^+ = 1,9741 \cdot 10^{-5}$, $\delta_{1\max}^- = -1,9770 \cdot 10^{-5}$, $\alpha_1 = 15,63$, після другої модифікованої ітерації NR у формі (2.29) з $k_4 = 1,500000000285$ – $\delta_{2\max}^+ = 2,8829 \cdot 10^{-10}$, $\alpha_2 = 31,69$, а після третьої класичної ітерації (2.26) – $\delta_{3\max}^+ = 1,3867 \cdot 10^{-16}$ та $\delta_{3\max}^- = -1,3889 \cdot 10^{-16}$, $\alpha_3 = 52,68$. Остаточний алгоритм

(InvSqrt63d) має високу точність порівняно з класичними методами та використовує $10\times$, 2FMA та $3+/-$ (див. табл. Г.2).

Алгоритми на основі методу переключення магічних констант

У роботі [83] пропонуються два підходи до побудови покращених алгоритмів обчислення RSQRT, що використовують розбиття проміжку $x \in [1; 4)$ на дві частини. Один з підходів передбачає модифікацію значення експоненти вхідного аргументу x , що належить діапазону $x \in [1; 2)$, та подальше коригування результату в цьому діапазоні шляхом множення, відповідно до формули (1.26), на константу $1/\sqrt{2}$. При цьому застосовується одна МС та одна модифікована ітерація NR у формі (2.30). Проте, кращі результати дає запропонований метод переключення магічних констант (DC) [8,83].

Основна ідея методу DC полягає в тому, щоб розбити вхідний проміжок $x \in [1; 4)$, на якому описується поведінка початкового наближення y_0 , на дві нерівні частини $x \in [1; 2)$ та $x \in [2; 4)$ та застосувати метод МС та відповідну модифіковану першу ітерацію NR окремо для кожної з цих ділянок [8]. Тоді, відповідно до використаного загального підходу покращення алгоритму FISR, визначаються такі два значення МС R_1 та R_2 , а також відповідні значення параметрів k_{i1} та k_{i2} , де $i=1, 2$, модифікованих формул методу NR для двох різних перших ітерацій у формі (2.30), що мінімізують $\delta_{1\max}$ на кожній з ділянок.

Для цього, на першому проміжку $x \in [1; 2)$ будемо розглядати МС із параметрами $Q=189, 1/2 \leq t < 1$, що описуються на основі скорегованої аналітичної моделі (2.1)-(2.4), а на другому проміжку $x \in [2; 4)$ – із параметрами $Q=190, 0 \leq t < 1/2$ та аналітичною моделлю (1.27)-(1.30). Потрібно знайти таке значення t , що задовольняє рівність $\delta_{01\max}^+ - \delta_{01\max}^- = \delta_{02\max}^+ - \delta_{02\max}^-$ для двох сусідніх кусково-лінійних наближень $y_{01}, x \in [1; x')$, та $y_{02}, x \in [x'; 2)$, із спільною точкою в x' (див. рис. 3.4а). Звідси отримаємо для проміжку $x \in [1; 2)$ значення $t=0,70241439342498779$, що відповідає наступним МС для чисел одинарної та подвійної точності: $R_1=0x5ed9e8b7$ та $R_1=0x5fdb3d16dd72c671$ [8].

Аналогічно для проміжку $x \in [2; 4)$, для якого отримаємо значення $t = 0,20241439342498779$, що визначає МС $R_2 = 0x5f19e8b7$ та $R_2 = 0x5fe33d16dd72c671$ для одинарної та подвійної точності [8]. Графіки отриманих початкових кусково-лінійних наближень, що відповідають комбінації цих констант показані на рис. 3.4б.

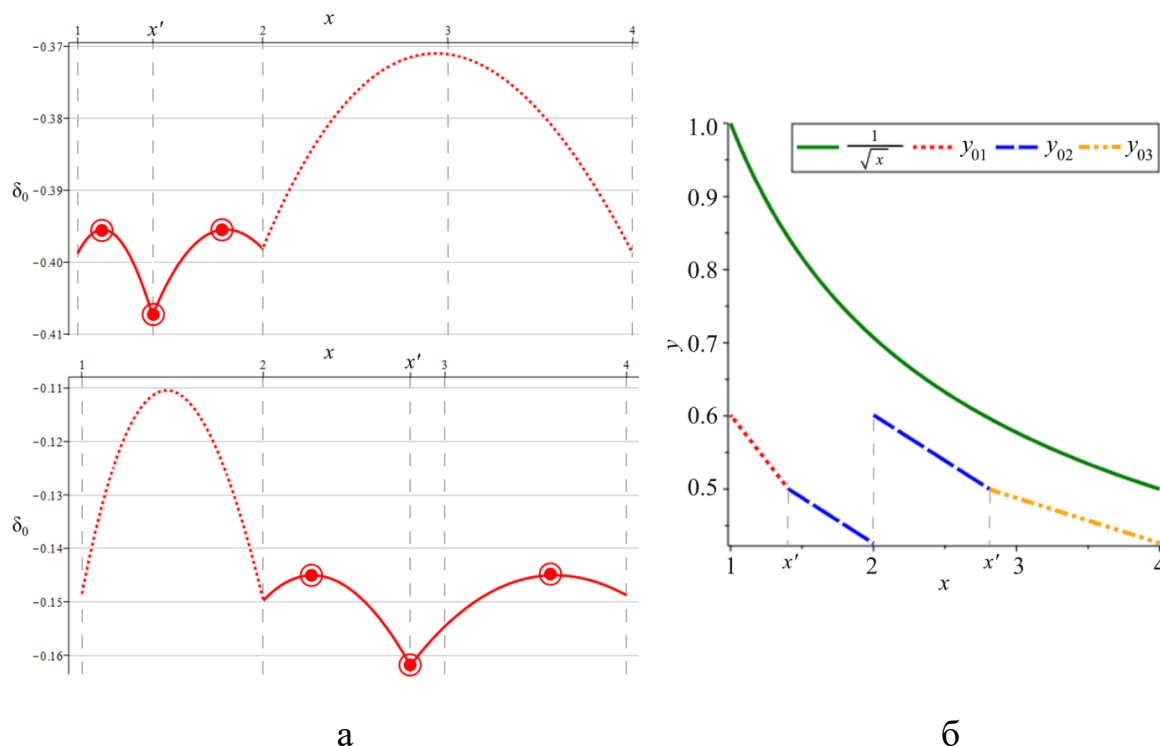


Рисунок 3.4 – (а) Процедура вирівнювання розмаху максимальних відносних похибок на проміжках $x \in [1; 2)$ та $x \in [2; 4)$. (б) Початкове наближення y_0 методу DC на проміжку $x \in [1; 4)$ для RSQRT до застосування модифікованого методу NR

Підбираються такі коефіцієнти модифікованого методу NR, щоб мінімізувати похибку алгоритму після першої ітерації при відомому t на кожному з проміжків: $k_{11} = 2,33124253$, $k_{12} = 1,07497365$ для $x \in [1; 2)$, та $k_{21} = 0,8242186$, $k_{22} = 2,1499476$ для $x \in [2; 4)$. Молодший біт зміщеної експоненти E числа x у форматі IEEE 754 визначає проміжок, в який потрапляє це число, $x \in [1; 2)$ або $x \in [2; 4)$. У випадку програмної реалізації методу застосовується бітова маска до значення $t(x)$, а у випадку апаратної реалізації для вибору необхідних значень параметрів R , k_1 та

k_2 можна використати невеликий LUT із однобітним адресним входом. Таким чином, алгоритм з оптимізованими параметрами InvSqrt71f, що реалізує даний метод для чисел типу *float* наведений у лістингу 3.7. Максимальні похибки алгоритму складають $\delta_{1\max}^+ = 7,4593 \cdot 10^{-5}$, $\delta_{1\max}^- = -7,4504 \cdot 10^{-5}$, а бітова похибка рівна $\alpha_1 = 13,71$.

Лістинг 3.7 – Покращений оптимізований алгоритм для RSQRT одинарної точності на основі методу DC

```

1: float InvSqrt71f (float x)
2: {
3:     int i = *(int*)&x;
4:     int k = i & 0x00800000; // бітова маска над LSB експоненти E числа x
5:     float y;
6:     if (k != 0) {
7:         i = 0x5ed9e91f - (i >> 1); // модифікована магічна константа  $R_1$ 
8:         y = *(float*)&i;
9:         y = 2.33124256f*y*fmaf(-x, y*y, 1.0749737f);
           // мод. ітер. NR у формі (2.30)
10:    } else {
11:        i = 0x5f19e8fc - (i >> 1); // модифікована магічна константа  $R_2$ 
12:        y = *(float*)&i;
13:        y = 0.824218631f*y*fmaf(-x, y*y, 2.1499474f);
           // мод. ітер. NR у формі (2.30)
14:    }
15:    return y;
16: }
```

Порівняно з іншими модифікаціями алгоритму FISR, що використовують лише класичні та модифіковані ітерації NR, запропоновані алгоритми на основі методу DC характеризуються підвищеною точністю та низькою надлишковістю. Метод DC дозволяє зменшити $\delta_{1\max}$ в 11,8 раз у порівнянні з методом Вальчика та ін. [140] (див. табл. Г.1) та в 8,7 раз у порівнянні із найкращими модифікаціями алгоритму FISR. Друга спільна для двох проміжків ітерація для

типу *float* реалізується в вигляді класичного методу NR у формі (2.27), (2.28) з FMA. При цьому параметри першої ітерації алгоритму InvSqrt72f оптимізуються для зменшення похибки всього алгоритму з $\delta_{2\max} = 8,0380 \cdot 10^{-8}$ до $\delta_{2\max}^+ = 7,3624 \cdot 10^{-8}$, $\delta_{2\max}^- = -7,7542 \cdot 10^{-8}$, $\alpha_2 = 23,62$ біти. Оптимізовані параметри алгоритму мають вигляд $R_1 = 0x5ed9dbc6$, $k_{11} = 2,33124018$, $k_{12} = 1,07497406$, $R_2 = 0x5f19d200$, $k_{21} = 0,824212492$ та $k_{22} = 2,14996147$.

Оптимізований базовий алгоритм методу DC для типу *double* InvSqrt71d має наступні похибки для однієї ітерації: $\delta_{1\max}^+ = 7,4379 \cdot 10^{-5}$, $\delta_{1\max}^- = -7,4379 \cdot 10^{-5}$, а після другої модифікованої ітерації NR у формі (3.1), (3.2) бітова похибка алгоритму вже сягає $\alpha_2 = 27,84$. Алгоритм InvSqrt72d містить 8 оптимізованих параметрів подвійної точності, $5 \times$ і 3FMA [8]. При цьому похибки рівні $\delta_{2\max}^+ = 4,1492 \cdot 10^{-9}$ та $\delta_{2\max}^- = -4,1492 \cdot 10^{-9}$. Версія алгоритму InvSqrt731d, що забезпечує максимальну точність для трьох модифікованих ітерацій NR, після першої ітерації методу DC застосовує модифікований метод NR у формі (2.31) та третю модифіковану ітерацію у формі (3.1), (3.2). Хоча цей алгоритм містить вже 9 оптимізованих параметрів та використовує $8 \times$ і 4FMA [8, алгоритм 10], його точність не перевищує бітову похибку $\alpha_3 = 52,47$, $\delta_{3\max}^+ = 1,3639 \cdot 10^{-16}$ та $\delta_{3\max}^- = -1,6062 \cdot 10^{-16}$.

Відповідні алгоритми SQRT для FP чисел типу *float* мають похибки: $\delta_{1\max} = 7,4511 \cdot 10^{-5}$ та $\delta_{2\max} = 9,0380 \cdot 10^{-8}$ ($\alpha_2 = 23,4$), а для чисел типу *double* – $\delta_{1\max} = 7,4379 \cdot 10^{-5}$, $\delta_{2\max} = 4,1492 \cdot 10^{-9}$ та $\delta_{3\max} = 1,8475 \cdot 10^{-16}$ ($\alpha_3 = 52,27$).

3.1.4 Покращені алгоритми обчислення функцій RSQRT та SQRT підвищеної швидкодії

Як було зазначено раніше, для зменшення в алгоритмі кількості операцій в FP арифметиці, наприклад $\times$ на константу, у деяких випадках їх можна реалізувати на основі простіших цілочислових операцій. Для прикладу, $\times$ на параметри $k_1 = 1/4$ у формулі (2.30) або $k_2 = 1/2$ в ітерації вигляду (2.29), як було показано,

зокрема, в роботі [7] для функції RSQRT. Використання методу DC, зокрема для подальшого розбиття проміжків та формування LUT, також дозволяє зменшити кількість FP операцій та підвищити швидкодію алгоритму. Крім того, на багатьох платформах доцільно широко використовувати FMA операції в обчисленнях.

Алгоритми на основі методу двох магічних констант

Множення y_0 на коефіцієнт $k_1 = 1/4$ в запропонованому методі [7], зокрема алгоритмі InvSqrt521f, при програмній або апаратній його реалізації можна замінити на цілочисельне віднімання двійки від експоненти числа, поданого в форматі IEEE-754. Цю операцію можна реалізувати через модифікацію МС $R' = R - 2 \cdot N_m$ (наприклад, для запропонованих алгоритмів 3 та 4). Тоді перша ітерація алгоритму (див. лістинг 3.4) запишеться в альтернативному вигляді:

```

3:   int i = *(int*)&x;
4:   i = i>>1;
5:   int ii = 0x5e5fb432 - i; // додаткова магічна константа R'
6:   i = 0x5f5fb432 - i; // основна магічна константа R
7:   float y = *(float*)&i;
8:   float yy = *(float*)&ii;
9:   y = yy*(4.76405191f - x*y*y); // альтернативний запис
                                     // мод. ітерації NR у формі (2.30) із  $k_1 = 1/4$ 

```

Наведений алгоритм містить лише $3 \times$ і $1-$ для FP чисел та використовує додаткову МС. Будемо називати його методом двох магічних констант (2МС).

Розглянемо ще один покращений алгоритм для *float* з оптимізованими параметрами, що базується на методі 2МС та інших методах підвищення швидкодії алгоритмів. Програмний код алгоритму наведений у лістингу 3.8. Алгоритм InvSqrt523f містить всього $3 \times$ та 3FMA в FP арифметиці, а множення $-0,5 \cdot x$ виконується в цілочисловому представленні над значенням I_x . Похибки алгоритму складають $\delta_{2\max}^+ = 3,7860 \cdot 10^{-7}$ та $\delta_{2\max}^- = -4,1143 \cdot 10^{-7}$, що відповідає $\alpha_2 = 21,21$ одиницям бітової похибки.

Лістинг 3.8 – Швидкий покращений оптимізований алгоритм для RSQRT одинарної точності на основі методу 2МС

```

1: float InvSqrt523f (float x)
2: {
3:     int i = *(int*)&x;
4:     int ix = i - 0x80800000; // альтернативне множення x на -1/2
5:     i = i>>1;
6:     int ii = 0x5e5fb436 - i; // мод. магічна константа R'
7:     i = 0x5f5fb436 - i; // мод. магічна константа R
8:     float y = *(float*)&i;
9:     float yy = *(float*)&ii;
10:    y = yy*fmaf(-x, y*y, 4.76424933f); // мод. NR у формі (2.30) із  $k_1 = 1/4$ 
11:    float mhalf = *(float*)&ix;
12:    float t = fmaf(mhalf, y*y, 0.500000298f);
13:    y = fmaf(y, t, y); // мод. ітер. NR у формі (2.23), (2.26)
14:    return y;
15: }
```

Версія цього алгоритму для чисел подвійної точності із трьома модифікованими ітераціями NR, дві останні з яких записані в формі (2.23), (2.26), InvSqrt531d, має максимальні похибки $\delta_{3\max}^+ = 7,5541 \cdot 10^{-14}$ та $\delta_{3\max}^- = -7,5496 \cdot 10^{-14}$, $\alpha_3 = 43,59$. Алгоритм містить $4\times$ та 5FMA. Програмний код цього алгоритму з оптимізованими параметрами наведений в роботі [7]. Для порівняння, базовий алгоритм Робертсона [23] та алгоритм Вальчика та ін. [140] у випадку їх реалізації для трьох ітерацій типу *double*, використовують $10\times$ і $3-$, а їх бітова похибка складає $\alpha_3 = 34,88$ та $\alpha_3 = 41,85$ відповідно. Якщо останню ітерацію реалізувати у вигляді класичного методу Но2 в особливій формі (2.27), (2.32), то ми отримаємо алгоритм InvSqrt532d, наведений у лістингу 3.9. Цей алгоритм містить $5\times$ і 6FMA та забезпечує майже повну точність RSQRT для чисел подвійної точності, $\delta_{3\max}^+ = 1,3878 \cdot 10^{-16}$ та $\delta_{3\max}^- = -1,3872 \cdot 10^{-16}$, звідки отримуємо $\alpha_3 = 52,68$. Він використовує меншу кількість операцій та має значно вищу точність за інші відомі методи, наприклад алгоритми Робертсона [23] і Вальчика та ін. [140] для

чотирьох ітерацій (13×, 4–), а також Леметри та ін. [1,151] для двох класичних ітерацій методу Ho3 (6×, 6FMA), як показано у табл. Г.4 та у роботі [7].

Лістинг 3.9 – Швидкий покращений оптимізований алгоритм для RSQRT подвійної точності на основі методу 2МС, що використовує метод Ho2

```

1: double InvSqrt532d (double x)
2: {
3:     uint64_t i = *(uint64_t*)&x;
4:     uint64_t ix = i - 0x8010000000000000;
5:     i = i>>1;
6:     uint64_t ii = 0x5fcbf6d9db9a45cd - i; // мод. магічна константа R'
7:     i = 0x5febf6d9db9a45cd - i; // мод. магічна константа R
8:     double y = *(double*)&i;
9:     double yy = *(double*)&ii;
10:    y = yy*fma(-x, y*y, 4.7642670025852993); // мод. NR (2.30) із  $k_1 = 1/4$ 
11:    double mhalf = *(double*)&ix;
12:    double t = fma(mhalf, y*y, 0.50000031697852854);
13:    y = fma(y, t, y); // мод. ітер. NR у формі (2.23), (2.26)
14:    double c = x*y;
15:    double r = fma(y, -c, 1.0);
16:    t = fma(0.375, r, 0.5);
17:    y = fma(y*r, t, y); // ітер. Н 2-го порядку у формі (2.27), (2.32)
18:    return y;
19: }
```

Алгоритм для функції SQRT, побудований на основі InvSqrt532d, не потребує додаткових операцій та має похибки $\delta_{3\max} = 1,6653 \cdot 10^{-16}$, $\alpha_3 = 52,42$.

Алгоритми на основі методу переключення магічних констант

Для трьох модифікованих ітерацій NR методу DC для типу *double* можна реалізувати також варіант алгоритму, що спроектований для підвищення швидкодії та зменшення кількості FP операцій (див. лістинг 3.10).

Лістинг 3.10 – Швидкий покращений оптимізований алгоритм для RSQRT подвійної точності на основі методу DC

```

1: double InvSqrt732d (double x)
2: {
3:     uint64_t i = *(uint64_t*)&x;
4:     uint64_t ix = i - 0x8010000000000000; // множення x на -1/2
5:     uint64_t k = i & 0x0010000000000000; // бітова маска над x
6:     double y;
7:     if (k != 0) {
8:         i = 0x5fdb3d14170034b6 - (i >> 1); // модифікована константа R1
9:         y = *(double*)&i;
10:        y = 2.33124735553421569*y*fma(-x, y*y, 1.07497362654295614);
           // мод. ітер. NR у формі (2.30)
11:    } else {
12:        i = 0x5fe33d18a2b9ef5f - (i >> 1); // модифікована константа R2
13:        y = *(double*)&i;
14:        y = 0.82421942523718461*y*fma(-x, y*y, 2.1499494964450325);
           // мод. ітер. NR у формі (2.30)
15:    }
16:    double mxhalf = *(double*)&ix;
17:    y = y*fma(mxhalf, y*y, 1.5000000034937999); // мод. ітер. NR (2.29)
18:    double t = fma(mxhalf, y*y, 0.5);
19:    y = fma(y, t, y); // мод. ітер. NR у формі (2.26)
20:    return y;
21: }

```

Цей алгоритм містить 7 модифікованих параметрів та використовує лише $6 \times i$ 4FMA, що на $2 \times$ менше ніж InvSqrt731d. Максимальні похибки алгоритму InvSqrt732d складають $\delta_{3\max}^+ = 1,6035 \cdot 10^{-16}$ та $\delta_{3\max}^- = -1,8263 \cdot 10^{-16}$, $\alpha_3 = 52,28$.

Метод DC можна узагальнити також на випадок подальшого розбиття проміжків та збільшення кількості MC, і відповідно розміру LUT [8]. При цьому, в залежності від значення старших бітів мантиси M , кожен з проміжків $x \in [1; 2)$ та $x \in [2; 4)$ ділиться на рівні частини, кількість яких є степенем двійки. Наприклад, під час розбиття на 8 піддіапазонів похибка методу 8DC, алгоритму InvSqrt81f, складає $\delta_{1\max} = 8,6888 \cdot 10^{-6}$, а бітова похибка $\alpha_1 = 16,81$ (див. лістинг 3.11).

Лістинг 3.11 – Швидкий покращений оптимізований алгоритм для RSQRT одинарної точності на основі узагальненого методу DC (8DC) із 96-байтною LUT

```

1: const int DCR0[8] = { 0x5f178854, 0x5f3e839e, 0x5f0b642f, 0x5f27c05d,
    0x5ed6372c, 0x5e873530, 0x5eb4e1cb, 0x5ed39c80 };
2: const float DCK1[8] = { 0.846447587f, 0.401537776f, 0.98020374775f,
    0.672251999f, 2.4631722f, 7.77609873f, 3.78227472f, 2.48082948f };
3: const float DCK2[8] = { 2.11204123f, 3.47231483f, 1.9152530432f,
    2.46271038f, 1.03619039f, 0.481497437f, 0.778513789f, 1.03126919f };
    // LUT для методу DC із 8-ми піддіапазонами
4:
5: float InvSqrt81f (float x)
6: {
7:     int i = *(int*)&x;
8:     int j = (i & 0x00e00000) >> 21; // бітова маска для піддіапазону числа x
9:     float y;
10:    i = DCR0[j] - (i >> 1); // модифікована MC R для j-го піддіапазону
11:    y = *(float*)&i;
12:    y = DCK1[j]*y*fmaf(-x, y*y, DCK2[j]);
    // мод. ітер. NR у формі (2.30) для j-го піддіапазону
13:    return y;
14: }
```

Запропоновані методи DC та 2MC можуть також бути ефективним при їх апаратній реалізації, зокрема на FPGA-платформах, що мають FPU, можливість розпаралелення обчислень та швидкі механізмами виконання цілочислових операцій, наприклад Intel Cyclone [202].

3.1.5 Покращені алгоритми обчислення інших степеневих, показникових та логарифмічних функцій

Покращені алгоритми обчислення логарифмічних та експоненціальних функцій

Застосуємо запропоновані підходи підвищення точності алгоритмів для згаданих методів апроксимації трансцендентних функцій $y = 2^x$ та $y = \log_2 x$ на основі MC для мінімізації їх максимальних відносних або абсолютних похибок (див. табл. Г.8). Порівняно з методом Бліна [6], точність запропонованих базових алгоритмів збільшена на 1 біт або абсолютна похибка зменшена вдвічі. Як

зазначалося у п. 2.1.1, ці алгоритми можна досить просто модифікувати для обчислення інших показникових та логарифмічних функцій. Якщо необхідна більша точність, рекомендується застосовувати мінімаксну поліноміальну апроксимацію (див. п. 3.2).

Покращені алгоритми обчислення функцій RCBRT, CBRT та SQRT підвищеної точності та швидкодії

Аналогічні підходи застосуємо під час проєктування алгоритмів для функцій RCBRT, CBRT та SQRT (див. табл. Г.9-Г.11), в тому числі без та з використанням операції $\div$. Для прикладу, запропонований алгоритм RCBRT із табл. Г.9, що використовує $R=0x548C3300$ і потребує $9\times$, $2FMA$ та $2-$, має максимальні похибки $\delta_{2\max}^+ = 8,7057 \cdot 10^{-8}$ та $\delta_{2\max}^- = -8,6954 \cdot 10^{-8}$, $\alpha_2 = 23,45$.

Обчислення CBRT на основі RCBRT вимагає модифікації останньої ітерації NR:

```

8:   float d = x*y*y;
9:   float t = fmaf(-0.33333334f*y, d, 0.33333334f);
10:  t = t*(t + 2.00028062);
11:  y = fmaf(d, t, d); // модифікація алгоритму для CBRT

```

Точність алгоритму складає $e_2 = 23$ біт. Цей алгоритм містить на $1\times$ та $1+$ більше, ніж версія для RCBRT, на відміну від використання формули (2.16), що потребує $2\times$ та має значно гіршу точність. Для підвищення швидкодії, $\div$ в запропонованих алгоритмах CBRT та SQRT (табл. Г.10 та Г.11) можна замінити множенням на функцію RCP повної або мінімальної допустимої точності (див. табл. Г.7).

Розглянемо тепер приклади реалізації алгоритмів на основі альтернативних методів отримання початкового наближення – з використанням поліноміальної апроксимації та наближених апаратних інструкцій Intel та ARM.

3.2 Проєктування ефективних модифікованих алгоритмів обчислення елементарних функцій на основі методу поліноміальної апроксимації

Приклад використання кусково-поліноміальної апроксимації для отримання початкового наближення функції RSQRT показано у лістингу 3.12. У цьому

випадку експонента та мантиса FP числа x опрацьовуються окремо. Початкове наближення y_0 алгоритму InvSqrt90f отримано на основі восьми поліномів найкращого рівномірного наближення другого степеню для приведених вхідних аргументів з діапазону $x' \in [1; 2)$, записаних за схемою Горнера (у разі необхідності, з FMA). Слід зазначити, що для підвищення швидкодії, обчислення таких поліномів можна виконувати також в арифметиці з фіксованою комою.

Лістинг 3.12 – Швидкий алгоритм для RSQRT одинарної точності на основі мінімаксної кусково-поліноміальної апроксимації із 96-байтною LUT

```

1: const float PK0[8] = { 1.82069285f, 1.7218863f, 1.6376214f, 1.5646443f,
    1.5006382f, 1.44390206f, 1.3931553f, 1.34741234f };
2: const float PK1[8] = { 1.1437567656f, 0.9675948f, 0.83245683f,
    0.726104128f, 0.64062472f, 0.57069961f, 0.512634497f, 0.46379168f };
3: const float PK2[8] = { 0.32304797f, 0.24450855f, 0.190316328f,
    0.151561647f, 0.12301841f, 0.10147076f, 0.08485931f, 0.0718199135f };
// LUT на основі коефіцієнтів 8-ми мінімаксних поліномів другого порядку
4:
5: float InvSqrt90f (float x)
6: {
7:     int R;
8:     int i = *(int*)&x;
9:     int k = i & 0x00800000; // бітова маска над LSB експоненти E
10:    int j = (i & 0x00700000) >> 20; // бітова маска над 3 MSB мантиси M
11:    int e = i & 0x7f800000; // бітова маска над експонентою E
12:    if (k != 0) { R = (0xbe800000 - e) >> 1; } // R1 для експоненти
13:    else { R = (0xbe6a09e5 - e) >> 1; }; // R2 для експоненти
14:    int Mx = (i | 0x3f800000) & 0xbfffffff; // приведення аргументу x
15:    float y;
16:    float ey = *(float*)&R; // масштабуючий коефіцієнт для експоненти
17:    float yy = *(float*)&Mx; // значення  $x' \in [1; 2)$ 
18:    y = PK0[j] - yy*(PK1[j] - PK2[j]*yy); // вибір j-го полінома
19:    y = y*ey; // коригування експоненти результату  $y_0$ 
20:    return y;
21: }
```

Похибки такого початкового наближення рівні $\delta_{0\max}^+ = 1,6087 \cdot 10^{-5}$ та $\delta_{0\max}^- = -1,6101 \cdot 10^{-5}$, що відповідає бітовій похибці $\alpha_0 = 15,92$. Після застосування додаткової класичної ітерації NR (2.23), (2.24) отримаємо наступні похибки: $\delta_{1\max}^+ = 1,4453 \cdot 10^{-7}$ та $\delta_{1\max}^- = -1,4370 \cdot 10^{-7}$, $\alpha_0 = 22,72$, а у випадку реалізації ітерації NR у вигляді (2.27), (2.28) з використанням FMA, алгоритм InvSqrt91f, отримаємо похибки $\delta_{1\max}^+ = 7,4098 \cdot 10^{-8}$ та $\delta_{1\max}^- = -7,4125 \cdot 10^{-8}$, $\alpha_0 = 23,69$.

3.3 Проєктування ефективних модифікованих алгоритмів RCP, DIV, RSQRT та SQRT на основі швидких наближених апаратних інструкцій Intel та ARM

Наведемо у лістингу 3.13 приклад реалізації покращеного оптимізованого алгоритму RSQRT для FP чисел одинарної точності, що використовує для початкового наближення апаратну інструкцію ARM NEON FRSQRTE (викликається з допомогою функції *vrsqrtes_f32(x)*) та застосовує метод Ho2 у формі (2.27), (2.32) з модифікованими параметрами.

Лістинг 3.13 – Швидкий покращений оптимізований алгоритм RSQRT для одинарної точності на основі наближеної інструкції ARM NEON та модифікованого методу Ho2

```

1: float InvSqrt101f (float x)
2: {
3:     float32_t a = x;
4:     a = vrsqrtes_f32(a); // апаратна інструкція FRSQRTE від ARM
5:     float y = (float)a; // початкове наближення  $y_0$ 
6:     float c = x*y;
7:     float r = fmaf(y, -c, 1.0f);
8:     float t = fmaf(0.374771f, r, 0.50000815f);
9:     y = fmaf(y*r, t, y); // мод. ітер. Н 2-го порядку (2.27), (2.32)
10:    return y;
11: }
12: }
```

Похибки цього покращеного алгоритму складають $\delta_{1\max}^+ = 7,3669 \cdot 10^{-8}$ та $\delta_{1\max}^- = -7,7825 \cdot 10^{-8}$, $\alpha_0 = 23,62$, а використовує він, крім інструкції FRSQRTE, лише $2\times$ та 3 FMA .

3.4 Проєктування ГПВП для задач у сфері кібербезпеки та захисту інформації, побудованих на основі чисельних методів в FP арифметиці

Зважаючи на ускладнену структуру формату FP даних, нелінійність більшості арифметичних операцій та накопичення похибок при виконанні достатньої їх кількості, можна припустити, що при генерації ПВП можна використовувати й “неточні” біти результату наближеного обчислення функцій. Оскільки числа подвійної точності оперують 64-бітними FP значеннями із 52 бітами мантиси та 11 бітами експоненти, мають широкий діапазон значень, тому можуть бути використані для проєктування ГПВП з достатнім рівнем ентропії та періодом повторення для застосування у сфері кібербезпеки та захисту інформації, зокрема під час розробки новітніх методів стеганографічного вбудовування даних.

У статті [15] ми запропонували новий клас ГПВЧ, зокрема для застосувань у сфері КБ та ЗІ, що використовують чисельні методи швидкого наближення елементарних функцій, зокрема RCP та RSQRT, в FP арифметиці (для 64-бітних чисел типу *double*). Алгоритми ефективної реалізації даних методів, в тому числі на основі FISR, були детально розглянуті в пп. 1.2-1.4 даної дисертації, а у пп. 3.1-3.3 запропоновано вдосконалені щодо точності та швидкодії алгоритми на основі методу МС, кусково-поліноміальної апроксимації та наближених апаратних інструкцій. Запропонований базовий алгоритм генерації ПВБП використовує рекурентне рівняння вигляду $x_i = a \cdot (x_{i-1} + b)$ над числами формату IEEE 754 [39], щоб сформувати вхідну послідовність для відповідних алгоритмів обчислення елементарних функцій $y(x_i)$. Отримані результати обробляються для формування вихідної послідовності ПВЧ. Як альтернативний підхід для підвищення ентропії

послідовності вхідних даних є використання рекурентних рівнянь із властивостями хаотичних систем [56].

Перевагами використання в ГПВЧ запропонованих покращених методів обчислення елементарних функцій в FP арифметиці на основі MC у порівнянні з методами довільної точності є їх простота, достатня гнучкість, висока точність та швидка збіжність, а тому за одну ітерацію методу кількість точних біт зазвичай подвоюється або потроюється, що підвищує швидкодію генераторів. За умови незалежності та випадковості вхідних аргументів такі алгоритми можуть виступати в ролі додаткового елементу нелінійності [15].

3.4.1 Загальна структура запропонованих ГПВЧ

Алгоритм генерації на основі початкового значення x_0 типу *double* формує послідовність FP чисел вигляду $\{x_i\}_{i=1}^N$, використовуючи рекурентну формулу

$$x_i = 1,000003174189 \cdot (x_{i-1} + 0,000358474), \quad i = \overline{1, N}. \quad (3.3)$$

Параметри a і b рівняння (3.3) вибрані таким чином, щоб досягти компромісу між довжиною послідовності (періодом генератора) та різномірністю її елементів (випадковістю генератора). Отримані числа x_i подаються на вхід відповідного алгоритму у обчислення функції в FP арифметиці $fpFunction(y, x_i) = y(x_i)$, після чого результат розглядається як ціле 8-байтове число відповідно до формули $I_{y(x_i)} = \iota(y(x_i))$, що є основою для процесу генерації ПВП.

Опрацювання результату обчислення функції $y(x_i)$ відбувається побайтово відповідно до процедури $transform(I_{y(x_i)}, mode, bytesCount, offset)$, що визначається параметрами: *mode* – режим генерації; *bytesCount* – кількість байт, що генерується на основі $I_{y(x_i)}$; *offset* – зміщення в байтах, що разом із *bytesCount* визначає діапазон байтів FP числа $I_{y(x_i)}$, що використовуються в процесі генерації ПВП, як правило, це молодші байти мантиси.

Таким чином, запропонований ГПВП дозволяє за один виклик функції $y(x_i)$ генерувати відразу *bytesCount* байтів ПВП, а тому його можна розглядати як ГПВЧ. Варіанти реалізації запропонованого ГПВЧ для фіксованих a і b будемо

позначати відповідно до схеми: $\langle y \rangle_{\langle mode \rangle}$, а значення параметрів генерації будемо записувати в вигляді: $\langle bytesCount \rangle / \langle offset \rangle$. Кожен з таких генераторів формує набір детермінованих ПВП для будь-якого вибраного початкового значення x_0 [85]. Основна блок-схема алгоритму, що визначає загальну структуру роботи запропонованих ГПВЧ, показана на рис. 3.5а [15]. Генератор формує вихідний бінарний файл із ПВБП розміру $fileSize$ байт шляхом дописування до нього згенерованих на кожному кроці даних $generatedBytes$.

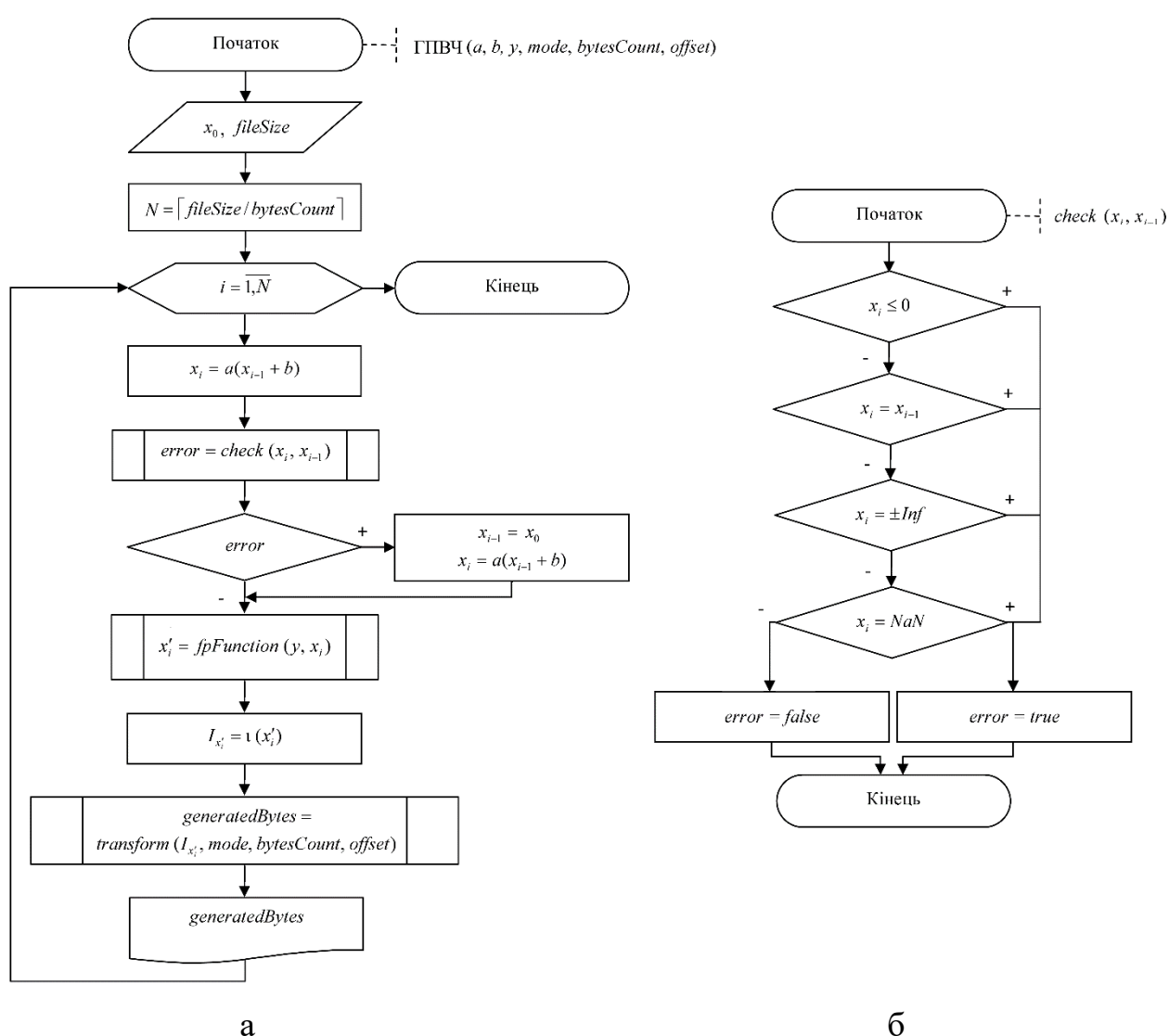


Рисунок 3.5 – Блок-схеми алгоритмів роботи: (а) загальної структури запропонованих ГПВЧ на основі чисельних методів в FP арифметиці та (б) процедури *check* для перевірки елемента послідовності $\{x_i\}_{i=1}^N$ для $y = 1/\sqrt{x}$

Перевагою такого простого ГПВЧ є можливість розпаралелення та векторизації значної частини операцій алгоритму, починаючи з обчислення функції для всіх значень $\{x_i\}_{i=1}^N$. Процедура $check(x_i, x_{i-1})$ використовується під час обчислення цієї послідовності для визначення моменту переповнення аргументу x_i та інших виняткових ситуацій в FP арифметиці. Завдяки цьому формується періодичність згенерованих ПВП. Приклад реалізації цієї процедури у випадку функції RSQRT наведений на рис. 3.5б.

3.4.2 Опис можливих варіантів реалізації запропонованих ГПВЧ

Розглянемо детальніше можливі варіанти реалізації інших базових блоків алгоритму, які будемо досліджувати в цій дисертації: процедуру *fpFunction* та процедуру *transform*. Процедура $fpFunction(y, x_i)$ виконує обчислення певної елементарної функції $y(x_i)$ алгоритмом y . Насамперед ми розглянули використання 13 типових швидких алгоритмів наближеного обчислення функцій RCP та RSQRT у запропонованих ГПВЧ, різної точності та, відповідно, з різною обчислювальною складністю [15,85]. Всі ці алгоритми базуються на методі МС та реалізовані з використанням чисел подвійної точності. Перелік цих алгоритмів та їх порівняння наведено у табл. 3.2. Програмну реалізацію вибраних алгоритмів можна знайти у додатку Д. Також ми дослідили використання стандартної реалізації функції SQRT з бібліотеки *cmath*, що має точність $e = 53$ біти, та використання тотожної функції. Таким чином блок-схема алгоритму узагальненої процедури $fpFunction(y, x_i)$ буде мати вигляд як на рис. 3.6а. Процедура $transform(I_{y(x_i)}, mode, bytesCount, offset)$ формує вихідну ПВП використовуючи один з режимів *mode* ГПВЧ [85] – наприклад *everyBit* (див. рис. 3.6б) або *xored* (див. рис. 3.6в) – та зазначені параметри генерації *bytesCount/offset*. При цьому в режимі *xored* для підвищення стійкості генератора до криптоаналізу застосовується ланцюжок операцій XOR. Досліджувалось також використання складніших режимів генерації, а також різноманітних варіантів реалізації алгоритмів на основі методу 8DC, оскільки за результатами досліджень [85] ГПВЧ на основі методу DC показали одні з найкращих результатів. Зокрема, як

буде показано в розділі 4, хороші статистичні характеристики має генератор *rsqrt2dc_everyBit* (1/0).

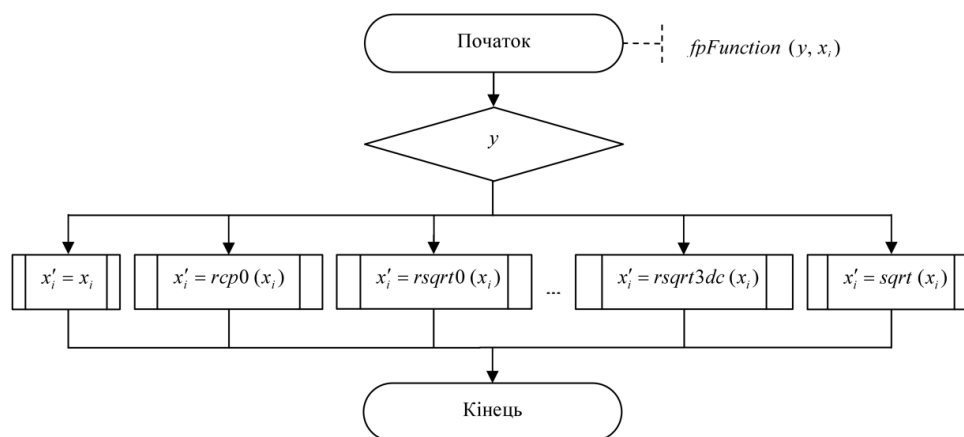
Таблиця 3.2 – Перелік вибраних для тестування ГПВЧ алгоритмів наближеного обчислення функцій $y(x) = 1/x$ та $y(x) = 1/\sqrt{x}$

Алгоритм обчислення функції y	Кількість ітерацій	Позначення алгоритму (y)	Кількість FP операцій	Точність (бітова похибка) (α)
$y(x) = 1/x$				
Початкове наближення методом МС	0	$rcp0(x)$	–	$\alpha_0 = 4,3$
Покращені алгоритми для RCP [80]	1–3	$rcp1(x)$	2×, 1FMA	$\alpha_1 = 13,1$
		$rcp2(x)$	3×, 2FMA	$\alpha_2 = 27,2$
		$rcp3(x)$	3×, 4FMA	$\alpha_3 = 52,3$
$y(x) = 1/\sqrt{x}$				
Початкове наближення методом МС	0	$rsqrt0(x)$	–	$\alpha_0 = 4,9$
Вальчика та ін. [140]	1–3	$rsqrt1w(x)$	4×, 1–	$\alpha_1 = 10,2$
		$rsqrt2w(x)$	7×, 2–	$\alpha_2 = 20,7$
		$rsqrt3w(x)$	10×, 3–	$\alpha_3 = 41,9$
Алгоритми на основі методу Ho2 [82]	1–2	$rsqrt1h(x)$	5×, 2+	$\alpha_1 = 15,6$
		$rsqrt2h(x)$	9×, 2+, 1–	$\alpha_2 = 31,7$
Алгоритми на основі методу переключення МС (DC) [8]	1–3	$rsqrt1dc(x)$	3×, 1FMA	$\alpha_1 = 13,7$
		$rsqrt2dc(x)$	5×, 3FMA	$\alpha_2 = 27,5$
		$rsqrt3dc(x)$	7×, 4FMA	$\alpha_3 = 52,3$
Алгоритми на основі узагальненого методу DC (8DC) [8]	1–3	$rsqrt1dc8(x)$	3×, 1FMA	$\alpha_1 = 16,8$
		$rsqrt2dc81(x)$	5×, 2FMA, 1–	$\alpha_2 = 34,1$
		$rsqrt2dc82(x)$	5×, 4FMA	$\alpha_2 = 49,1$
		$rsqrt3dc8(x)$	8×, 2FMA, 2+, 1–	$\alpha_3 = 52,4$

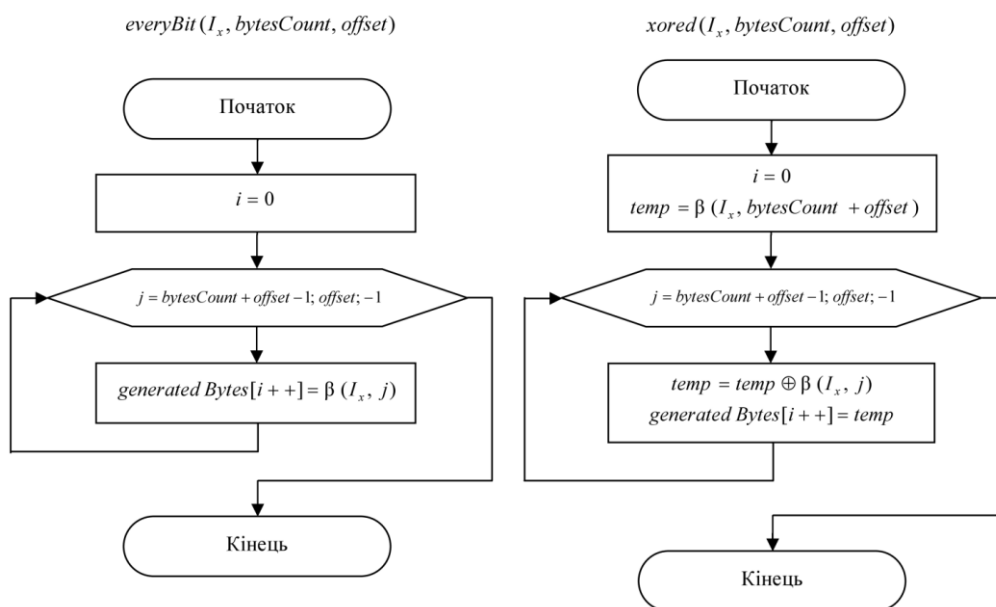
На блок-схемах режимів генерації використовуються наступні позначення: $\beta(i, j)$ – функція отримання j -го байту цілого числа i , а $\oplus$ – операція побітового додавання за модулем два (XOR) окремих байтів числа $I_{y(x_i)}$. У режимі *everyBit* використовується *bytesCount* байт із $I_{y(x_i)}$, а в режимі *xored* – *bytesCount* + 1 байт.

Запропонований алгоритм генерації ПВЧ на основі наближених методів обчислення елементарних функцій був реалізований мовою C++ відповідно до

блок-схем на рис. 3.5 та 3.6. Генератор цього типу формує детерміновані послідовності, що визначаються його параметрами та початковим значенням.



а



б

в

Рисунок 3.6 – Блок-схеми роботи процедур *fpFunction* та *transform*:

(а) вибір алгоритму y в процедурі *fpFunction* та вибір режиму генерації ПВБП

(б) *everyBit* або (в) *xored* в процедурі *transform*

Для визначення придатності запропонованих ГПВЧ для практичного застосування, в тому числі для криптографічних та некриптографічних задач інформаційної та кібербезпеки, необхідно провести ретельне їх тестування та аналіз ефективності, зокрема властивостей сформованих ПВП.

Висновки до розділу 3

1. Вперше запропоновано набір наступних вдосконалених методів наближеного обчислення функцій RCP, RSQRT, DIV та SQRT для задач у сфері кібербезпеки та захисту інформації на основі алгоритму FISR: метод швидкого обчислення RCP на основі тотожності з двома мінімаксними поліномами 3-го порядку на проміжку [1; 2]; методи швидкого обчислення RCP та RSQRT на основі модифікованого методу Ho2; методи додаткових магічних констант (2MC та 3MC) для функцій RSQRT та RCP, що завдяки цілочисловим операціям дозволяють зменшити кількість $\times$; методи переключення магічних констант (DC та 8DC) для обчислення RSQRT та SQRT на основі розбиття інтервалу визначення функції $x \in [1; 4)$ на дві, чотири або більше нерівних частини та використання різних MC та оптимізованих коефіцієнтів на кожній з них. На їх основі реалізовано набір алгоритмів із підвищеною точністю і/або швидкодією для FP чисел одинарної та подвійної точності. Як було показано, запропоновані методи мають похибку на рівні 1 ULP або, для функції RCP, забезпечують повну точність результату з коректним заокругленням.
2. Досліджено прості модифікації алгоритмів на основі методів MC, кусково-поліноміальної апроксимації та наближених апаратних інструкцій Intel і ARM з використанням методів рандомізованої чисельної багатовимірної оптимізації коефіцієнтів з метою зменшення максимальної відносної похибки. Реалізовано набір покращених алгоритмів для функцій RCP, DIV, RSQRT, SQRT, RCBRT, CBRT, логарифмічних та показникових функцій із підвищеною точністю і/або швидкодією, в тому числі без операції $\div$.
3. Вперше запропоновано новий клас алгоритмів генерації ПВЧ на основі рекурентного рівняння $x_i = a \cdot (x_{i-1} + b)$ і покращених алгоритмів швидкої апроксимації елементарних функцій RCP та RSQRT для FP чисел подвійної точності, що в залежності від початкового значення та обраних параметрів генератора формує послідовність псевдовипадкових байтів. Вони будуть корисними для багатьох задач у сфері кібербезпеки та захисту інформації.

РОЗДІЛ 4. ПРАКТИЧНЕ ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ ЗАПРОПОНОВАНИХ АЛГОРИТМІВ ТА ГЕНЕРАТОРІВ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ НА ПК, МІНІ-КОМП'ЮТЕРІ ТА МК ДЛЯ ЗАДАЧ У СФЕРІ КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ

Запропоновані покращені алгоритми обчислення елементарних функцій в FP арифметиці на основі методу MC та відповідні прості ГПВЧ на їх основі, можуть бути ефективно реалізовані як програмно, так і апаратно на різноманітних пристроях, що підтримують FP обчислення. Проте, особливо корисними у сфері кібербезпеки та захисту інформації ці алгоритми будуть для платформ, що не мають апаратних засобів для обчислення функцій RCP, DIV, RSQRT, SQRT, RCBRT та CBRT чи засобів швидкої апаратної генерації надійних ПВП. Прикладом такого пристрою можна назвати МК ESP-WROOM-32. Також ці алгоритми можуть бути легко реалізовані на GPU з використанням технології CUDA та сучасних ПЛІС [7,8,33], наприклад Intel Cyclone, оскільки вони містять всі необхідні базові блоки алгоритмів для FP чисел: $\times$, FMA та $+$ [202].

У даному розділі ми будемо розглядати переважно програмну реалізацію запропонованих методів мовою C++ та дослідження їх ефективності на ПК, міні-комп'ютерах та МК, зокрема для застосувань у сфері КБ та ЗІ. Крім ПК з Intel Core i7, що використовувалися для тестування точності алгоритмів у попередніх розділах, для тестування продуктивності алгоритмів будемо розглядати ще ПК з Intel Core i5, МК ESP-WROOM-32, а також міні-комп'ютер Raspberry Pi 3 Model B. CPU Intel Core i7-7700HQ (Kaby Lake, 2.8 ГГц, 16 Гб RAM) та Intel Core i5-10210U (Comet Lake, 1.6 ГГц, 16 Гб RAM) працювали під 64-бітними ОС Windows 10 та використовували наступні параметри компіляції програм: `-std=c++11 -Os -ffp=contract=on -mfma`. ESP-WROOM-32 (ESP-32) – це 32-бітний Wi-Fi/Bluetooth/BLE модуль від компанії Espressif Systems, що використовує два енергоефективні мікропроцесори (Xtensa LX6, 240 МГц, 520 Кб RAM). Програмування цього МК виконувалось через Arduino IDE (GCC 5.2.0) із наступними параметрами компіляції: `-std=gnu++11 -Os -ffp-contract=fast`.

Raspberry Pi 3 Model B – це компактний одноплатний комп'ютер, що працює на основі чотирьохядерного 64-бітного SoC Broadcom BCM2837 (ARM Cortex-A53, 1.2 ГГц, 1 Гб RAM). Для програмування даного обчислювального пристрою ми використовували компілятор GNU (GCC 6.3.0) для 32-бітної ОС Raspbian із параметрами компіляції: `-std=c++11 -Os -ffp-contract=on -mfpu=neon-fp-armv8 -mcpu=cortex-a53`.

4.1 Аналіз та тестування покращених алгоритмів обчислення елементарних функцій для задач інформаційної та кібербезпеки

Зазначимо, що всі розглянуті платформи мають інструкції FMA, а також операції $\times$ й $+$ як мінімум для FP чисел одинарної точності (тип *float*). Такі інструкції визначені стандартом IEEE 754, а тому присутні на багатьох CPU, FPU, МК та ПЛІС. На платформах Intel та Raspberry Pi також присутні аналогічні інструкції для типу *double*. При цьому, МК ESP-32 не має окремих інструкцій для RCP та DIV і, оскільки він є 32-розрядний, всі операції із числами подвійної точності емулюються програмно.

Слід зазначити, що на швидкодію алгоритмів при їх програмній реалізації сильно впливає рівень оптимізації коду компілятором. Цим процесом можна керувати вручну, використовуючи відповідні прапорці компілятора – *O0-O3*, *Os* та *Og*. Опція *-mfma* вказує на необхідність використання апаратних інструкцій FMA. Крім того, на різних платформах відносні похибки алгоритмів можуть відрізнятися через різницю в параметрах компіляції та особливостях окремих компіляторів або архітектур. Наприклад, для платформи Intel компілятор C++ може автоматично замість функції *fmaf* з бібліотеки *cmath* використовувати потрібні апаратні інструкції FMA (наприклад, *VFMADD132SS* або *VFNMADD213SS*) [112]. Для архітектури Xtensa апаратні інструкції *MADD.S* та *MSUB.S* для поєданого $+$ та $-$ при вибраних опціях компіляції, більш того, автоматично використовуються для всіх послідовних обчислень операції $\times$ й $+/-$, а тому впливають на результат обчислень.

В цьому розділі обмежимося аналізом методів обчислення RCP та RSQRT.

4.1.1 Порівняння запропонованих покращених алгоритмів обчислення функцій RCP та RSQRT

Порівняння запропонованих покращених алгоритмів на основі FISR для RCP та RSQRT, що забезпечують високу точність результату, за кількістю використаних арифметичних операцій та одержаною точністю наведено у табл. 4.1 та 4.2. Тут показано точність (бітову похибку) початкових наближень покращених алгоритмів і повних версій алгоритмів для чисел з одинарною (бітова точність мінімум $e = 23$ біти) та подвійною точністю (бітова точність мінімум $e = 52$ біти). Ця інформація буде корисною під час вибору алгоритму для конкретної задачі у сфері КБ та ЗІ, зокрема при апаратній реалізації на ПЛІС, та подальшого аналізу продуктивності програмних реалізацій на різних пристроях.

Для функції RCP в табл. 4.1 можна побачити, що для однієї ітерації запропоновані покращені алгоритми Rcp11f та Rcp31f забезпечують на 36,6% та на 55% більшу точність, порівняно з відомим алгоритмом М [183], проте використовують відповідно на 1 операцію $\times$ та FMA більше. Для FP чисел одинарної точності завдяки використанню оптимізації вдалося забезпечити коректне заокруглення результату ($e = 24$ точних біти в алгоритмі Rcp321f) без збільшення загальної кількості арифметичних операцій. Для чисел типу *double* також запропоновано алгоритм максимальної точності ($e = 53$ точних біти в Rcp332d), що потребує меншої кількості ітерацій та загальної кількості FP операцій порівняно з алгоритмом [153]. Інші запропоновані покращені алгоритми для RCP майже повної точності (без округлення результату) дозволяють досягти компромісу між точністю та кількістю і складністю використаних операцій.

Порівняємо тепер алгоритми обчислення функції RSQRT. Як бачимо з табл. 4.2, алгоритм InvSqrt51f з 2МС використовує меншу кількість $\times$ у порівнянні з алгоритмом Вальчика та ін. [140] і має при цьому на 4,3% вищу точність. Запропоновані алгоритми на основі модифікованих методів Н вищих порядків при однаковій кількості операцій забезпечують на 17,7% та на 8,7% вищу точність у порівнянні з алгоритмами Леметри та ін. [1]. Якщо ж порівнювати алгоритм InvSqrt61f з алгоритмом [140], то він має на 54% вищу точність, але використовує

Таблиця 4.1 – Порівняння найкращих запропонованих алгоритмів для RCP за точністю та кількістю операцій, в тому числі на основі модифікованого методу Хаусхолдера (Ho2) та із трьома магічними константами (ЗМС)

Алгоритм	Кількість ітерацій	Необхідна точність / тип даних	Кількість арифметичних операцій						Точність (бітова похибка)
			FP			$\iota(x)/\phi(i)$	int	Всього	
			FMA	×	+/-		+/-		
Мороза та ін. [183]	1	–	–	2	1	2	1	6	9,61
Покращений алгоритм RCP (Rcp11f) [80]	1	–	–	3 (+1)	1	2	1	7 (+1)	13,13
Покращений алгоритм RCP на основі Ho2 (Rcp31f)	1	–	1 (+1)	2	1	2	1	7 (+1)	14,9
Мороза та ін. [183] у спеціальній формі	2	float	2	3	1	2	1	9	23
Покращений алгоритм RCP (Rcp12f) [80]	2	float	3 (+1)	2 (–1)	– (–1)	2	1	8 (–1)	23,8
Покращений алгоритм RCP із ЗМС (Rcp22f)	2	float	3 (+1)	1 (–2)	1	4 (+2)	3 (+2)	12 (+3)	23,75
Покращений алгоритм RCP на основі Ho2 (Rcp321f)	2	float	3 (+1)	2 (–1)	1	2	1	9	24
Кхуонг [153]	4	double	2	6	3	2	1	14	53
Покращений алгоритм RCP (Rcp131d)	3	double	4 (+2)	3 (–3)	– (–3)	2	1	10 (–4)	52,57
Оптимізований алгоритм RCP (Rcp132d)	3	double	4 (+2)	3 (–3)	– (–3)	2	1	10 (–4)	52,58
Покращений алгоритм RCP із ЗМС (Rcp23d)	3	double	4 (+2)	2 (–4)	1 (–2)	4 (+2)	3 (+2)	14	52,56
Покращений алгоритм RCP на основі Ho2 (Rcp331d)	3	double	4 (+2)	3 (–3)	1 (–2)	2	1	11 (–3)	52,99
Оптимізований алгоритм RCP на основі Ho2 (Rcp332d)	3	double	4 (+2)	3 (–3)	1 (–2)	2	1	11 (–3)	53

Таблиця 4.2 – Порівняння найкращих запропонованих алгоритмів для RSQRT за точністю та кількістю операцій, в тому числі із двома магічними константами (2МС), на основі методу Н та методу переключення магічних констант (DC)

Алгоритм	Кількість ітерацій	Необхідна точність / тип даних	Кількість арифметичних операцій									Обчи. 0,5×х?	Точність (бітова похибка)
			FP			$t(x)/\phi(i)$	int		AND	>>	Всього		
			FMA	×	+/-		+/-	==					
Вальчика та ін. [140]	1	–	–	4	1	2	1	–	–	1	9	+	10,15
Запропонований алгоритм із 2МС [7] (InvSqrt51f)	1	–	–	3 (–1)	1	3 (+1)	2 (+1)	–	–	1	10 (+1)	–	10,59
Оптимізований алгоритм із 2МС [7] (InvSqrt523f)	2	–	3 (+3)	3 (–1)	– (–1)	4 (+2)	3 (+2)	–	–	1	14 (+5)	+	21,21
Леметри та ін. [1] (Ho2)	1	–	2 (+2)	3 (–1)	– (–1)	2	1	–	–	1	9	–	13,28
Леметри та ін. [1] (Ho4)	1	–	4 (+4)	3 (–1)	– (–1)	2	1	–	–	1	11 (+2)	–	20,54
Запропонований алгоритм на основі методу Ho2 [82] (InvSqrt61f)	1	–	–	5 (+1)	2 (+1)	2	1	–	–	1	11 (+2)	–	15,63
Оптимізований алгоритм на основі методу Ho4	1	–	5 (+5)	2 (–2)	– (–1)	2	1	–	–	1	11 (+2)	–	22,33
Запропонований алгоритм на основі методу DC [8] (InvSqrt71f)	1	–	1 (+1)	3 (–1)	– (–1)	2	1	1 (+1)	1 (+1)	1	10 (+1)	–	13,71
Запропонований алгоритм на основі методу DC із 8 МС [8] (InvSqrt81f)	1	–	1 (+1)	3 (–1)	– (–1)	2	1	–	1 (+1)	2 (+1)	10 (+1)	–	16,81
Вальчика та ін. [140]	3	float	2	8	2	2	1	–	–	1	16	+	23,42
Оптимізований алгоритм із 2МС [7] (InvSqrt522f)	2	float	3 (+1)	5 (–3)	1 (–1)	3 (+1)	2 (+1)	–	–	1	15 (–1)	–	23,47
Оптимізований алгоритм на основі методу Ho2 [82] (InvSqrt62f)	2	float	2	7 (–1)	2	2	1	–	–	1	15 (–1)	–	23,69

Продовження таблиці 4.2

Алгоритм	Кількість ітерацій	Необхідна точність / тип даних	Кількість арифметичних операцій									Обчи. 0,5×x?	Точність (бітова похибка)
			FP			$t(x)/\phi(i)$	int		AND	>>	Всього		
			FMA	×	+/-		+/-	==					
Оптимізований алгоритм на основі Ho5	1	float	6 (+4)	2 (-6)	- (-2)	2	1	-	-	1	12 (-4)	-	23,59
Оптимізований алгоритм на основі методу DC [8] (InvSqrt72f)	2	float	3 (+1)	5 (-3)	- (-2)	2	1	1 (+1)	1 (+1)	1	14 (-2)	-	23,62
Кусково-поліноміальна апроксимація (InvSqrt91f)	1	float	2	5 (-3)	2	3 (+1)	1	1 (+1)	5 (+5)	2 (+1)	21 (+5)	-	23,69
Вальчика та ін. [140]	4	double	2	11	3	2	1	-	-	1	20	+	52,42
Леметри та ін. [1] (Ho3)	2	double	7 (+5)	5 (-6)	- (-3)	2	1	-	-	1	16 (-4)	-	52,68
Оптимізований алгоритм із 2МС [7] (InvSqrt532d)	3	double	6 (+4)	5 (-6)	- (-3)	4 (+2)	3 (+2)	-	-	1	19 (-1)	+	52,68
Запропонований алгоритм на основі методу Ho2 [82] (InvSqrt63d)	3	double	5 (+3)	7 (-4)	- (-3)	2	1	-	-	1	16 (-4)	+	52,68
Оптимізований швидкий алгоритм на основі методу DC [8] (InvSqrt732d)	3	double	2	8 (-3)	2 (-1)	2	2 (+1)	1 (+1)	1 (+1)	1	19 (-1)	+	52,28
Оптимізований точний алгоритм на основі DC [8] (InvSqrt731d)	3	double	4 (+2)	8 (-3)	- (-3)	2	1	1 (+1)	1 (+1)	1	18 (-2)	-	52,47
Запропонований алгоритм на основі методу DC із 8 МС [8]	3	double	2	8 (-3)	3	3 (+1)	2 (+1)	-	1 (+1)	2 (+1)	21 (+1)	+	52,42
Кусково-поліноміальна апроксимація	2	double	6 (+4)	3 (-8)	- (-3)	4 (+2)	2 (+1)	1 (+1)	5 (+5)	2 (+1)	23 (+3)	+	52,42

на 1 FMA більше. А алгоритми *InvSqrt71f* та *InvSqrt81f* на основі запропонованого методу DC при однаковій кількості $\times$ мають відповідно на 35% та на 65,6% вищу точність без використання громіздких LUT. Для FP чисел одинарної та подвійної точності запропоновані алгоритми забезпечують майже повну точність. Порівняно з алгоритмами Вальчика та ін. [140] вони вимагають меншої кількості ітерацій та значно меншої кількості FP операцій (на 1–4 множень менше), в деяких випадках за рахунок використання простіших цілочислових операцій та додаткових констант (методи 2MC, DC та кусково-поліноміальної апроксимації).

4.1.2 Тестування алгоритмів обчислення елементарних функцій на ПК, міні-комп'ютері та МК

Для оцінки практичної ефективності розглянутих вище алгоритмів обчислення RCP та RSQRT для FP чисел одинарної та подвійної точності у різних варіантах їх реалізації, а також порівняння з іншими програмними, програмно-апаратними й апаратними методами на сучасних обчислювальних платформах (Intel Core i5/i7, ARM Cortex-A53, Xtensa LX6), проведено вимірювання точності (у бітах) на основі обчислення $\delta_{\max}$ та продуктивності (у циклах і/або ns) на основі вимірювання латентності та зворотної пропускної здатності алгоритмів [7,8,80,82]. Для прикладу, результати порівняння найкращих відомих в літературі, вдосконалених та запропонованих програмних методів на 64-розрядній Intel i5 та 32-бітному ESP-32 наведені на рис. 4.1 та 4.2 відповідно. Алгоритми розташовані в порядку зростання їх точності. Розглядаються зокрема алгоритми на основі методу MC, кусково-поліноміальної апроксимації, емуляція операції DIV у МК, а також використання стандартної функції $\text{sqrt}(x)$ бібліотеки *cmath*. Слід зазначити, що архітектура МК ESP-WROOM-32 не передбачає апаратних інструкцій для операцій RCP та DIV, на відміну від RCPSS та DIVSS в Intel. Саме тому всі наведені наближені алгоритми на рис. 4.2а,в для функцій RCP та RSQRT одинарної точності, що використовують MC, є значно швидшими за стандартний метод на основі ділення. Для FP чисел подвійної точності, виграш в швидкодії є не такий очевидний, оскільки всі 64-розрядні типи та операції в цьому випадку емулюються. В додатку Е (рис. Е.1, Е.2) наведені також розширені результати



Рисунок 4.1 – Порівняння відомих та запропонованих програмних методів обчислення функцій RCP та RSQRT на платформі Intel i5 на основі вимірювання точності та швидкодії: (а) RCP для типу *float*, (б) RCP типу *double*, (в) RSQRT типу *float* та (г) RSQRT типу *double*

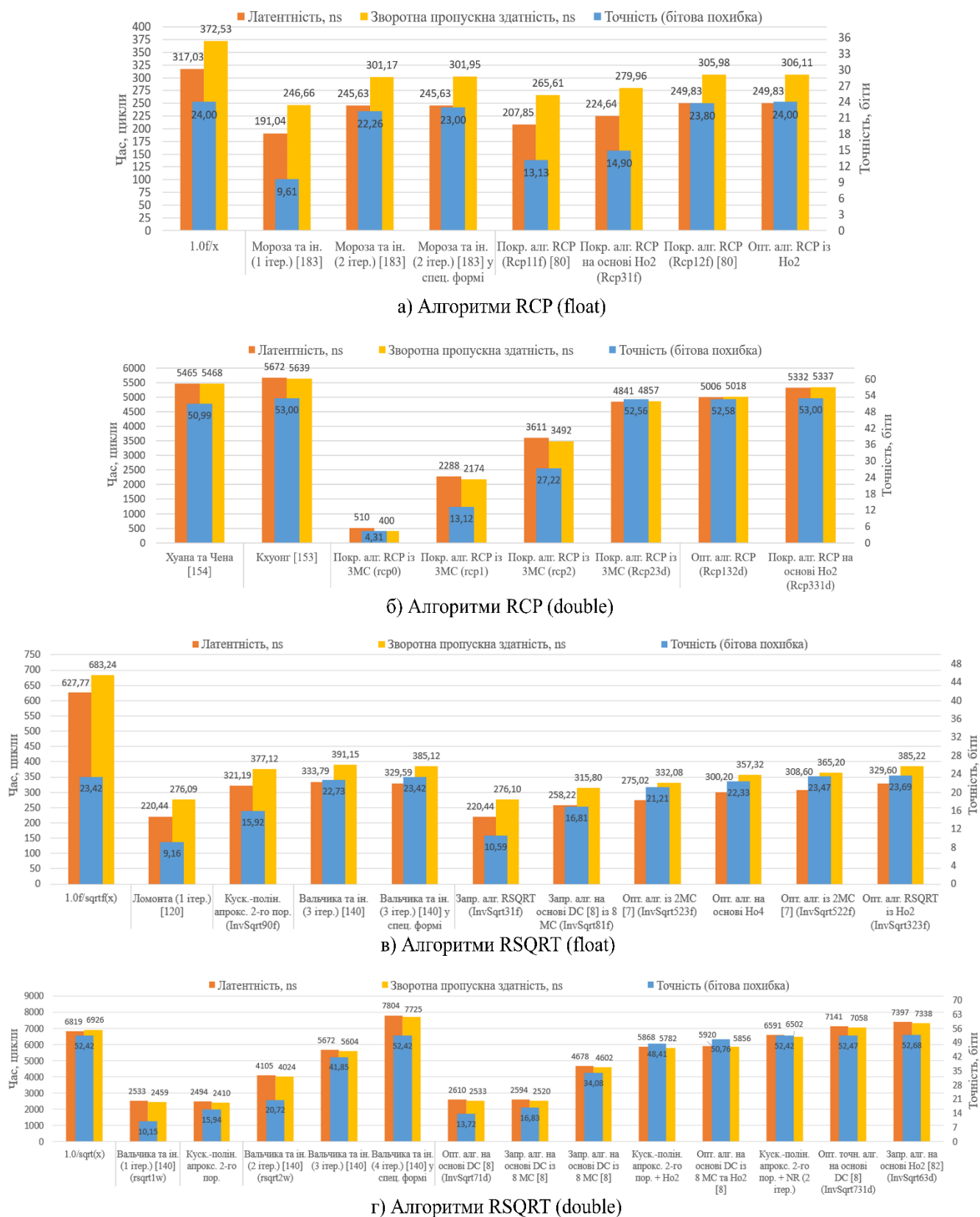


Рисунок 4.2 – Порівняння відомих та запропонованих програмних методів обчислення функцій RCP та RSQRT на МК ESP-32 на основі точності та швидкодії: (а) RCP для *float*, (б) RCP типу *double*, (в) RSQRT типу *float* та (г) RSQRT типу *double*

порівняння, з урахуванням також апаратних та програмно-апаратних методів на основі точних та наближених інструкцій Intel.

З рис. 4.2а видно, що як оптимізований алгоритм RCP із Ho2, так і метод $1.0f/x$ на основі $\div$ забезпечують повну точність результату для типу *float* – 24 коректних біти, але запропонований метод на МК ESP-32 має на 21% меншу латентність та на 18% меншу зворотну пропускну здатність. Інші покращені алгоритми мають трохи гіршу точність, але є ще швидшими. Порівняно з алгоритмом Мороза та ін. [183], покращені алгоритми Rcp11f та Rcp31f на ESP-32 мають відповідно на 36,5% та на 55% більшу точність, але є на 8,8% (7,5%) та 17,5% (13,5%) повільнішими щодо латентності (зворотної пропускну здатності).

Слід зауважити, що в роботах [7,8,82] нами також досліджувались інші методи підвищення швидкодії алгоритмів, зокрема на прикладі методів 2MC [7], Ho2 [82] та DC [8] на МК ESP-32 [7,8,82], Intel i7 [7,8,82] та Raspberry Pi [8]. Ці методи дозволяють знайти прийнятний компроміс між точністю та швидкодією, використовуючи альтернативні способи реалізації алгоритму (наприклад, використання лише однієї операції FMA на останній ітерації), що особливо актуально для чисел подвійної точності на ESP-32 [7] (див. табл. Е.1 в додатках). Зокрема, для оптимізованих алгоритмів з 2MC InvSqrt531d та InvSqrt532d вииграш у швидкодії складає 13,9% (зменшення латентності з 6106,0 ns до 5258,7 ns) та 7% (з 7286,9 ns до 6777,1 ns; що забезпечує вищу швидкодію та кращу точність за ділення $1.0/\sqrt{x}$ на ESP-32) відповідно.

Розглянемо тепер результати тестування точності та латентності алгоритмів на основі методу DC на міні-комп'ютері Raspberry Pi та МК ESP-32 [8] (див. табл. Е.2 в додатках), де розглядаються алгоритми обчислення RSQRT та SQRT із одинарною та подвійною точністю. За винятком чисел типу *double* на ESP-32, запропоновані алгоритми методом DC мають у 1,8-3,6 раз кращу швидкодію за використання бібліотечних функцій. При цьому на ESP-32, на противагу алгоритму InvSqrt732d, оптимізований під швидкодію алгоритм InvSqrt731d має трохи меншу латентність за наївний метод на основі $\sqrt{x}$, проте має дещо гіршу точність. Використання стандартних апаратних інструкцій ARM FRSQRTE

та FRSQRTS у Raspberry Pi 3 показує гірші результати щодо точності та латентності, ніж алгоритм InvSqrt72f (див. табл. Е.2 для RSQRT типу *float*), на відміну від оптимізованого алгоритму InvSqrt101f з розділу 3.3. Наша реалізація методу Вальчика та ін. [140] показує кращий компроміс між точністю та швидкодією обчислень, ніж в оригінальній версії алгоритму [8]. Також, цей метод був модифікований для обчислення функції SQRT. Як можна побачити із табл. Е.2, ці алгоритми є швидшими за DC, проте мають значно гіршу точність.

4.2 Дослідження основних характеристик запропонованих ГПВЧ для задач кібербезпеки та захисту інформації, побудованих на основі FP арифметики та покращених алгоритмів обчислення елементарних функцій

Далі наведемо результати дослідження якості сформованих такими генераторами послідовностей з використанням тестів NIST, призначених для дослідження ПВП для криптографічних застосувань, зокрема статистичних характеристик, періоду повторення, швидкодії, діапазонів початкових значень та стійкості до різноманітних статистичних та криптоаналітичних атак [15,85]. На основі одержаних результатів визначимо такі варіанти реалізації ГПВЧ, що мають найкращі характеристики для задач у сфері кібербезпеки та захисту інформації.

4.2.1 Дослідження статистичних характеристик сформованих ПВП на основі тестів NIST STS для криптографічних застосувань

На першому етапі дослідження для всіх 15 функцій y із рис. 3.6а (див. також част. 1 табл. 3.2), обох режимів генерації $mode \in \{everyBit, xored\}$ та вибраних значень параметрів *bytesCount* і *offset*, що задовольняють $bytesCount \in \{1, 2, 3, 5, 7\}$, $offset \in \{0, 1, 2, 4, 6\}$, $bytesCount + offset \leq 7$, при фіксованому значенні $x_0 = 0$ було сформовано набір ПВБП розміру 10^9 біт = 119,2 Мб кожна та проведено їх статистичне тестування. Для прикладу див. рис. 4.3, де пунктирною лінією позначено стандартну методику оцінювання результатів тестів NIST STS згідно із [200], а суцільною – додаткову оцінку згідно зі стратегією 1 [78], для якої із (1.56) при $k = 1000$ отримаємо довірчий інтервал

$0,99 \pm 0,0094392 = [0,9805607; 0,9994392]$. Варто зазначити, що згідно з стандартною методикою для тестів Random Excursions та Random Excursions Variant (тести № 160–185) нижня межа довірчого інтервалу може змінюватись. Як видно з рис. 4.3, обидва генератори проходять усі 188 тестів згідно з першою методикою оцінки, проте згідно з стратегією 1 для ГПВЧ $x_everyBit$ (2/1) вже 3 тести не потрапляють у визначений довірчий інтервал.

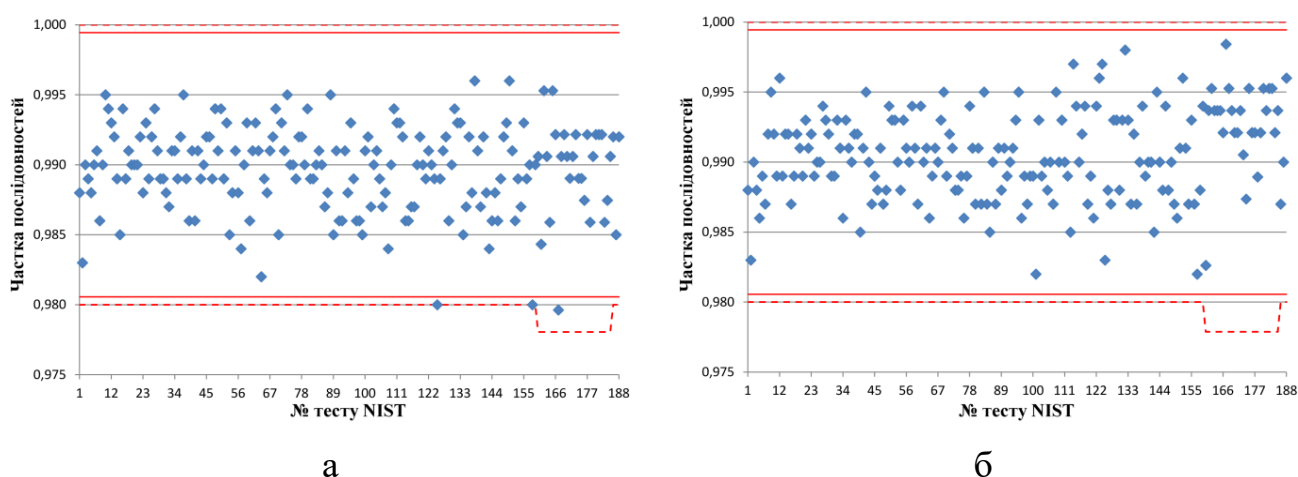


Рисунок 4.3 – Статистичні портрети ГПВЧ при $x_0 = 0$ та $m = 1000$:

(а) $x_everyBit$ (2/1) та (б) $rsqrt2w_xored$ (3/0)

В результаті було визначено такі комбінації параметрів, при яких сформовані ПВП проходять тести NIST з $m = 1000$. Для прикладу див. у додатках табл. Ж.1 для випадку використання функції $y = x$ та табл. Ж.2 для випадку використання алгоритмів Вальчика та ін. з різною кількістю ітерацій ($rsqrt1w$, $rsqrt2w$ і $rsqrt3w$). Як видно, всі тести проходять лише послідовності з параметрами $bytesCount + offset \leq 3$. Це справедливо й при використанні інших елементарних функцій. Узагальнені результати тестування всіх досліджених алгоритмів наведені в роботі [15] у табл. 2, де вказано лише ті послідовності, що успішно пройшли всі тести NIST згідно з двома стратегіями оцінки. Можна зробити висновок, що близько 56% послідовностей, що успішно пройшли тестування за першою методикою, проходять також оцінку за стратегією 1.

Одержані результати статистичного тестування сформованих ПВП при початковому значенні $x_0 = 0$ можна представити у вигляді графіків. Вони дозволяють визначити які алгоритми обчислення функцій y (див. рис. Ж.1а в додатках) та які значення параметрів *bytesCount* і *offset* (див. рис. Ж.1б) краще підходять для проєктування ГПВЧ для задач КБ та ЗІ з огляду на якість сформованих послідовностей. Ці рисунки містять також окремі графіки для кожного із режимів *everyBit* та *xored*, а також для двох стратегій оцінювання.

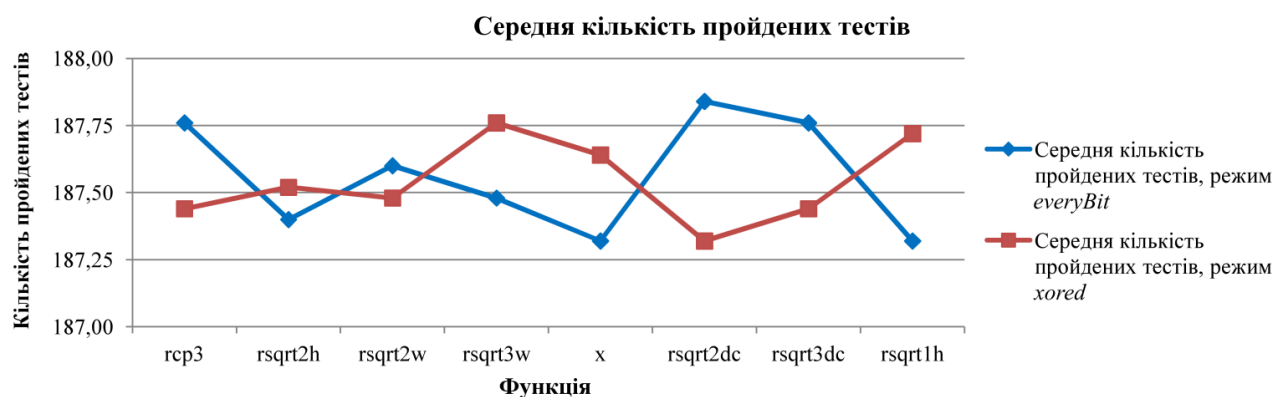
З рис. Ж.1 видно, що при фіксованому початковому значенні $x_0 = 0$ використання режиму *xored* є більш ефективним з погляду статистичних характеристик сформованих послідовностей, проте в контексті досліджень [85] ця закономірність вже не проявляється в загальному випадку. Згідно з рис. Ж.1а, найкращі результати мають ГПВЧ, що використовують наступні алгоритми: *rcp3*(x), *rsqrt2h*(x), *rsqrt2w*(x), *rsqrt3w*(x), x , *rsqrt2dc*(x) та *rsqrt3dc*(x), при цьому для режиму *everyBit* кращі статистичні результати має алгоритм *rsqrt3dc*(x), а для режиму *xored* – *rcp3*(x). При виборі алгоритму y слід враховувати також його криптографічну стійкість та обчислювальну складність (див. табл. 3.2). Найменшу обчислювальну складність має функція $y = x$, а найвищу – алгоритми *rsqrt3w*(x), *rsqrt3dc*(x) та функція $y = \sqrt{x}$. При виборі оптимального параметра генерації *bytesCount* слід пам'ятати, що це значення також прямо впливає на швидкодію запропонованого ГПВЧ. Згідно з результатами на рис. Ж.1б, оптимальна комбінація параметрів *bytesCount* і *offset* складає 3/0 (*bytesCount* = 3, *offset* = 0), а при $x_0 = 0$ у випадку використання режиму *everyBit* краще приймати *offset* = 0. Таким чином ми оцінили перспективні варіанти реалізації та напрямки подальшого дослідження запропонованих генераторів у випадку зміни початкових значень.

На другому етапі дослідження [85] ми обрали 8 алгоритмів обчислення функцій, що показали найкращі результати тестування на попередньому етапі, $y \in \{rcp3, rsqrt2h, rsqrt2w, rsqrt3w, x, rsqrt2dc, rsqrt3dc, rsqrt1h\}$. Розглянемо тепер

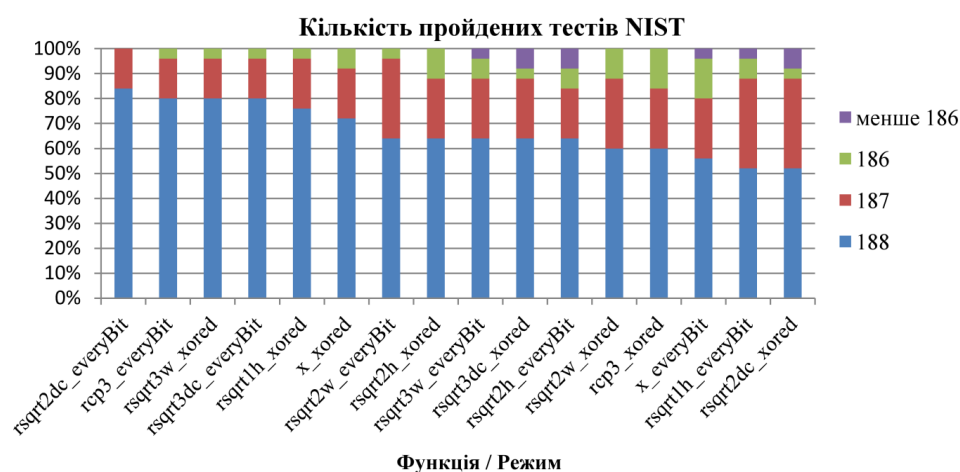
випадок модифікації початкового значення x_0 на прикладі ГПВЧ *rsqrt2dc_everyBit* (3/0) (див. рис. Ж.2). В даному випадку при $x_0 = 9,07225 \cdot 10^{-10}$ та $x_0 = 17,17$ цей генератор проходить усі тести NIST STS (див. рис. Ж.3б,д). З рис. Ж.3а,г можна побачити, що за класичною методикою оцінки при $x_0 = 0$ та $x_0 = 8,257638398$ згенеровані послідовності не проходять один тест – Non-Overlapping Template, а з рис. Ж.3е при $x_0 = 2108,573374$ – тест Block Frequency. Для деяких початкових значень цей ГПВЧ проходить тестування за класичною методикою оцінки, проте не проходить за уточненою, зокрема провалюються тести Non-Overlapping Template та Random Excursions Variant (див. рис. Ж.3в).

Отже, ми сформували масив значень x_0 , що будемо використовувати для тестування ГПВЧ. Прийmemo умову $x_0 \geq 0$ – скінченне число подвійної точності в FP арифметиці, включаючи діапазон денормалізованих чисел. Спершу розглянемо використання в якості x_0 , крім випадку $x_0 = 0$, ще 24 нормалізованих значень з діапазону $x_0 \in (10^{-10}; 10^6)$, що в своєму бітовому представленні мають різну кількість одиниць. В додатках на рис. Ж.2 наведено результати статистичного тестування для випадку стандартної методики оцінювання.

Тепер наведемо узагальнені результати тестування всіх обраних алгоритмів обчислення функцій у режимах $mode \in \{everyBit, xored\}$ та при значенні параметрів генерації $bytesCount = 3$ і $offset = 0$. Зокрема, результати порівняння запропонованих генераторів за середньою кількістю пройдених тестів та за їхнім відсотковим співвідношенням містяться на рис. 4.4а та 4.4б відповідно. Таким чином, згідно із рис. 4.4, найкращі результати дає використання функцій *rsqrt2dc*, *rcp3* і *rsqrt3dc* в режимі генерації *everyBit* та функцій *rsqrt3w* і *rsqrt1h* при застосуванні режиму *xored* [85]. Водночас усереднені статистичні результати різних ГПВЧ є приблизно однаковими в обох режимах (див. рис. 4.4а). Отже, найкращі статистичні характеристики згідно з тестами NIST STS показав ГПВЧ *rsqrt2dc_everyBit* (3/0), для якого близько 84% згенерованих послідовностей ПВЧ проходять всі статистичні тести NIST (див. рис. 4.4б).



а



б

Рисунок 4.4 – (а) Середня кількість пройдених тестів NIST для сформованого набору ПВП при різних початкових значеннях вказаних ГПВЧ та (б) відсоткове представлення результатів статистичних тестів NIST зазначеного набору ПВП

У випадку зміни значень параметрів *bytesCount* і *offset* запропоновані ГПВЧ в режимі *everyBit* зазвичай не проходять тестування NIST при параметрах 1/1 та 2/1. При цьому, як правило, провалюється один або кілька тестів, серед яких Block Frequency, Rank та Non-Overlapping Template. Винятком є генератор *rcp3_everyBit* (2/1). В режимі *xored* генератори, як правило, позбавлені цього недоліку (див. рис.4.5). Таким чином на основі одержаних результатів ми обрали по два ГПВЧ в кожному з режимів, що показали найкращі результати статистичного тестування ПВП. Розширивши діапазон вибору початкового значення до 34 чисел $x_0 \in [0; 1,7977 \cdot 10^{308})$, наведемо уточнені результати тестування NIST STS для вибраних генераторів (див. рис. Ж.4 в додатках).

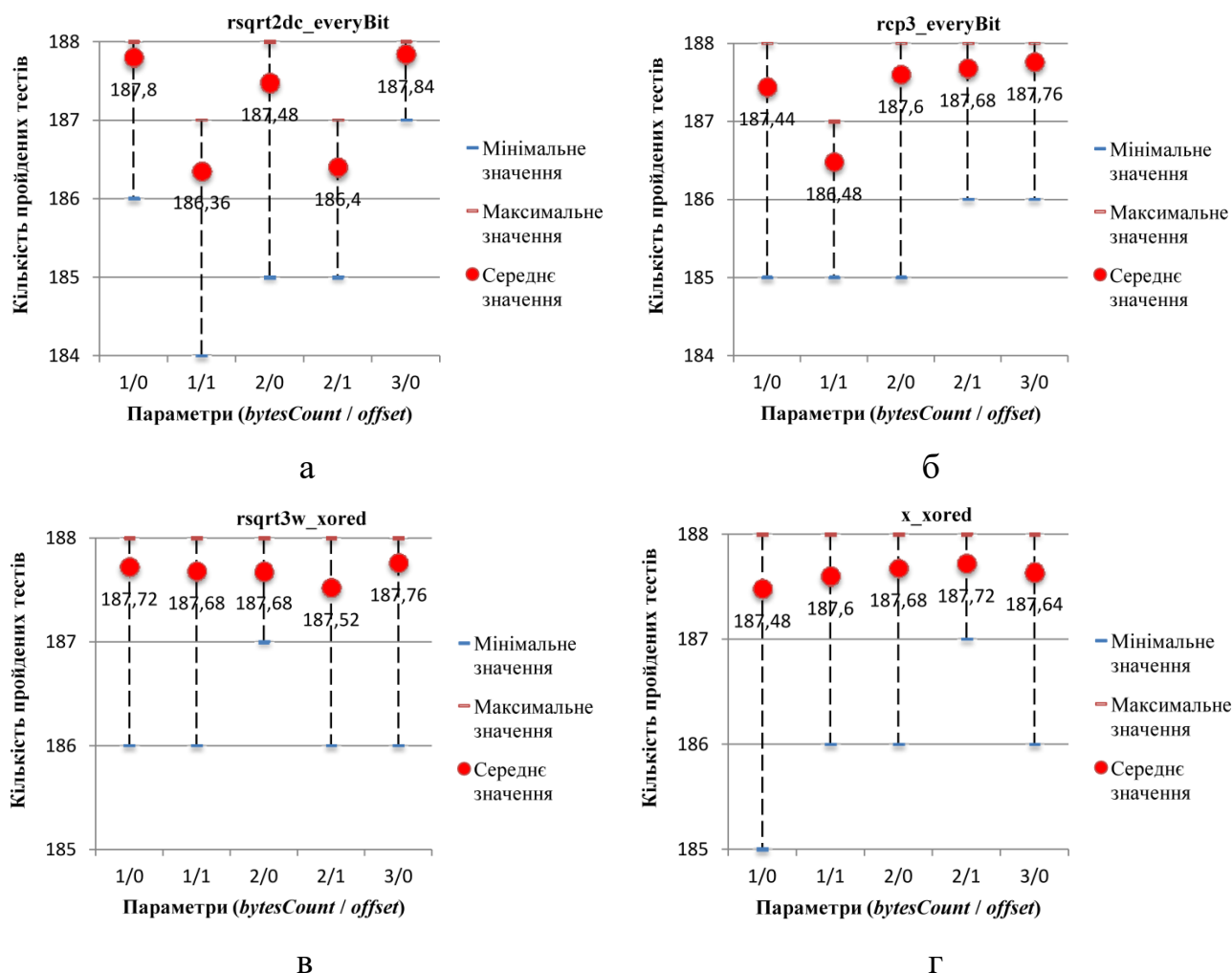


Рисунок 4.5 – Статистичні характеристики вибраних ГПВЧ згідно із тестами NIST в залежності від значень *bytesCount* та *offset*: (а) *rsqrt2dc_everyBit*, (б) *rcp3_everyBit*, (в) *rsqrt3w_xored* та (г) *x_xored*

Слід зазначити, що для всіх початкових значень $x_0 \leq 8,131516 \cdot 10^{-20}$ ГПВЧ формують однакові ПВП, такі самі як і при $x_0 = 0$. Крім того, було досліджено, що послідовності, сформовані на основі близьких початкових значень мають високий ступінь подібності (кореляції). Тому, краще використовувати початкові значення $x_0 \geq x_{\min}$, де $x_{\min} = 0,5$, з певним мінімальним кроком їх зміни. Врахувавши також вимоги щодо мінімального необхідного періоду послідовностей ГПВЧ, потрібно визначити також верхню межу $x_{\max}$, $x_0 \leq x_{\max}$ (див. п. 4.2.2). При цьому максимальний розмір множини початкових значень скоротиться в залежності від параметра *bytesCount* до значень $2^{61,7}$ у випадку

$bytesCount = 3$, $2^{61,52}$ для $bytesCount = 2$ та $2^{60,81}$ для $bytesCount = 1$. Таким чином, можна уточнити результати статистичного тестування генераторів з врахуванням вибраного діапазону для $x_0 \in [x_{\min}; x_{\max}]$ та можливості вибору початкових значень випадковим чином (окремо мантису M та експоненту E FP числа).

Окрім розглянутих вище алгоритмів, додатково аналізувалися алгоритми на основі узагальненого методу DC (див. алгоритми 14-17 із додатку Д, а також табл. 3.2.) та інші доступні методи генерації ПВП та ІВП на розглянутих платформах. Для ініціалізації/реініціалізації x_0 будемо використовувати наступні методи: 1) з масиву констант; 2) випадкове на основі мітки часу високої роздільної здатності; 3) випадкове на основі апаратних генераторів ІВП (наприклад, RDSEED і RDRAND від Intel або ESP_RANDOM від ESP-32); 4) випадкове на основі зчитування даних аналого-цифровим перетворювачем (АЦП) із відкритих аналогових входів МК; 5) випадкове на основі ГПВЧ *random_device* стандартної бібліотеки *random*.

Серед стандартних методів генерації ПВП на C++ розглядались:

- ГПВЧ *random_device* бібліотеки *random* у детермінованому варіанті реалізації на MinGW64;
- використання класу ГПВЧ по замовчуванню – *default_random_engine*;
- мінімальний стандартний лінійний конгруентний ГПВЧ Парка, Міллера та Стокмаєра – *minstd_rand*;
- ГПВЧ вихор Мерсена у 32-бітному та 64-бітному виконанні – *mt19937* та *mt19937_64*;
- ГПВЧ Люшера-Джеймса на основі віднімання із запозиченням з пропуском у 24-бітному та 48-бітному варіантах – *ranlux24* та *ranlux48*;
- ГПВЧ Д. Кнута на основі перемішаних лінійних конгруентних генераторів – *knuth_b*;
- застарілий механізм ГПВЧ у C++ на основі функції *rand* бібліотеки *cstdlib* (не рекомендовано до використання);
- ГПВЧ на основі функції *random* стандартної бібліотеки Arduino на МК.

Також досліджувались наступні апаратні генератори ІВП:

- використання апаратних інструкцій RDRAND та RDSEED від Intel у 32-бітному та 64-бітному варіантах;
- використання функцій *esp_random* та *esp_fill_random* для генерації ІВП на МК ESP-32.

Таким чином, уточнені результати статистичного тестування запропонованих ГПВЧ для вибраного діапазону x_0 оформлені в вигляді табл. 4.3, а в додатках на рис. Ж.5 наведено графіки порівняння та аналізу одержаних результатів. Пізніше в цьому розділі буде також розглянуто порівняння запропонованих алгоритмів з іншим розглянутими вище ГПВЧ за швидкодією та статистичними характеристиками.

Таблиця 4.3 – Узагальнені результати тестування запропонованих ГПВЧ статистичними тестами NIST STS при $m = 1000$ для випадкових x_0

Позначення ГПВЧ (<i>y_mode</i>)	Параметри ГПВЧ (<i>bytesCount / offset</i>)	Середня кількість пройдених тестів NIST	Відсоток проходження всіх тестів NIST
<i>rsqrt2dc81_everyBit</i>	2/0	187,78	81,19%
<i>rsqrt2dc_everyBit</i>	1/0	187,76	79,73%
<i>rsqrt1dc8_everyBit</i>	2/0	187,77	78,33%
<i>rsqrt2dc82_everyBit</i>	3/0	187,65	76,47%
<i>rcp3_everyBit</i>	3/0	187,72	75,53%
<i>rsqrt3w_xored</i>	3/0	187,69	74,32%
<i>rsqrt2dc81_xored</i>	3/0	187,65	74,07%
<i>rsqrt2dc81_everyBit</i>	1/0	187,61	72,13%
<i>rsqrt1dc8_everyBit</i>	3/0	187,71	70,59%
<i>rsqrt1dc8_everyBit</i>	1/0	187,68	70,59%
<i>rsqrt3w_xored</i>	2/0	187,66	68,97%
<i>rsqrt2dc81_everyBit</i>	3/0	187,63	68,32%
<i>x_xored</i>	2/1	187,53	64,52%

4.2.2 Визначення періоду повторення сформованих ПВП

Для того, щоб згенерована ПВП потрібного розміру мала хороші статистичні характеристики необхідно щоб вона була неперіодичною. Наприклад, для застосування тестів NIST STS необхідний мінімальний розмір послідовності має складати $125 \cdot 10^6 = 2^{26,9}$ байт, а тому до запропонованих ГПВЧ будемо висувати вимогу щодо мінімального періоду $L \geq 125 \cdot 10^6$ чисел (байт). Результати тестування періоду повторення запропонованих ГПВЧ на прикладі *rsqrt2dc_everyBit* показані на рис. 4.6.

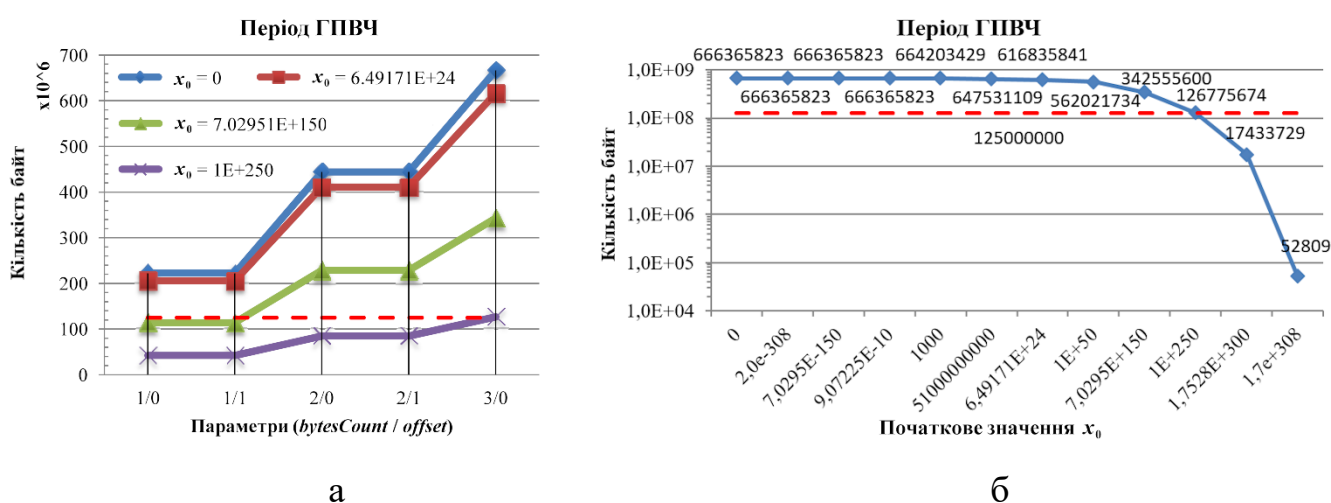


Рисунок 4.6 – Залежність періоду повторення ГПВЧ *rsqrt2dc_everyBit* від

(а) початкового значення і значень параметрів *bytesCount / offset* та

(б) початкового значення при фіксованих параметрах *bytesCount* = 3 і *offset* = 0

Як бачимо, період L сформованих ПВЧ залежить від початкового значення x_0 та параметра генерації *bytesCount*. При цьому L змінюється прямо пропорційно до значення *bytesCount* (див. рис. 4.6а). Зокрема, на рис. 4.6б наведено графік залежності періоду повторення згенерованих ПВП від вибору початкового значення x_0 . Таким чином, максимальне значення періоду цього генератора, $L = 666365823 = 2^{29,31}$ чисел (байт), отримане для початкових значень $x_0 \approx 0$. Період L також залежить від типу функції, зокрема для ГПВЧ

rcp_everyBit (3/0) при $x_0 \approx 0$ $L = 666039033 = 2^{29,31}$ байт, що трохи менше ніж для функції RSQRT. Таким чином, для забезпечення вимоги щодо періоду L $x_{\max}$ ГПВЧ необхідно вибирати з умови $x_{\max} \leq 10^{250}$ (див. рис. 4.6б). Тоді при значенні $bytesCount = 2$ вибір $x_{\max}$ має задовольняти умову $x_{\max} \leq 10^{220}$, а при $bytesCount = 1$ – $x_{\max} \leq 10^{135}$ відповідно.

При використанні випадкової реініціалізації початкового значення (модифікація x_0 на рис. 3.5) сформована послідовність стає неперіодичною. Для підвищення криптостійкості та покращення статистичних характеристик запропонованих ГПВЧ доцільно періодично застосовувати механізм реініціалізації, зокрема використовуючи апаратні генератори ІВП, наприклад 64-бітний RDSEED від Intel.

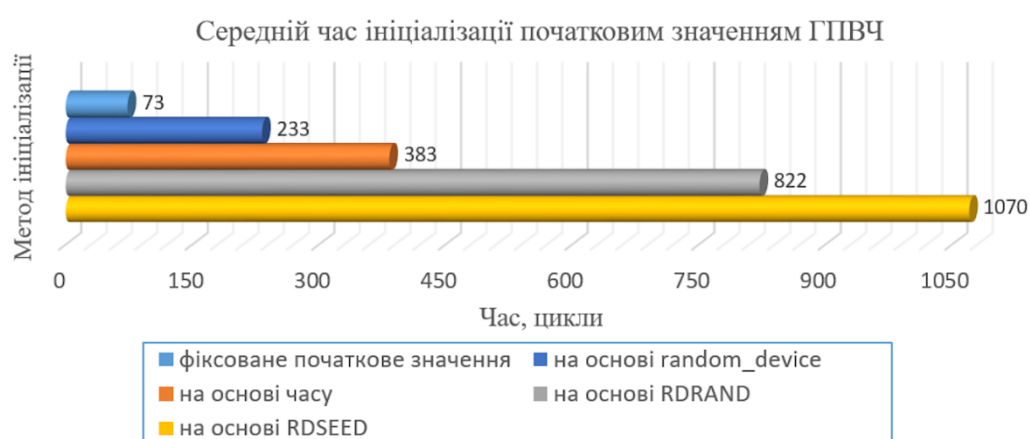
4.2.3 Дослідження швидкодії запропонованих ГПВЧ

Проведено аналіз реальної швидкодії досліджуваних ГПВЧ та генераторів ІВП на різних платформах (див. рис. Е.3 та Е.4 в додатках), зокрема часу генерації бінарних файлів із випадковими даними (малих, середніх та великих розмірів), середню швидкодію (див. рис. Е.3а,б та Е.4а для Intel) та часу ініціалізації генераторів (див. рис. Е.3в та Е.4б для ESP-32), а також обміну даними між МК та ПК, використовуючи послідовний COM порт і/або комунікацію з асинхронним веб-сервером ESPAsyncWebServer через WiFi-мережу. Для високоточного вимірювання часу та швидкодії використано бібліотеку *chrono* (C++), апаратний лічильник TSC (асемблер), а також бібліотеку *time* (Python). Тестування ГПВЧ виконувалось як для фіксованих, так і випадкових початкових значень.

Порівняння середньої швидкодії генерації ПВП (в Mbs/s) на платформах Intel i5, Intel i7 та ESP-32 найкращими з табл. 4.3 запропонованими ГПВЧ на основі обчислення елементарних функцій наведено на рис. 4.7а (для ESP-32 див. також рис. 4.8б). Тут *zeros* використовується для порівняння швидкодії генерації файлу, заповненого нулями. На рис. 4.7б показано залежність часу реініціалізації запропонованих ГПВЧ від вибору методу ініціалізації x_0 (див. рис. Е.5 для порівняння з іншими відомими ГПВЧ).



а



б

Рисунок 4.7 – Результати порівняння запропонованих ГПВЧ із найкращими статистичними характеристиками: (а) швидкодії на Intel та ESP-32 для фіксованих початкових значень та (б) часу ініціалізації на Intel i7 в залежності від способу вибору початкових значень

Порівняння пропонованих алгоритмів генерації ПВЧ із іншими доступними програмними методами можна побачити на рис. 4.8а для Intel i7 та на рис. 4.8б для МК ESP-32. Як бачимо, деякі варіанти запропонованих алгоритмів, зокрема *rsqrt2dc81_everyBit* (3/0), *rcp3_everyBit* (3/0), *rsqrt1dc81_everyBit* (3/0) та *x_xored* (2/1) є трохи швидшими за найшвидші некриптостійкі ГПВЧ на Intel i5 та Intel i7, *random_device* та *mt19937*, а також апаратних генераторів ІВП RDSEED та RDRAND (див. також рис. Е.3). На відміну від *random_device*, *mt19937*, *mt19937_64* та *knuth_b* запропоновані прості генератори не вимагають тривалих

термінів реініціалізації. На МК ESP-32 найвищу продуктивність серед запропонованих ГПВЧ мають *x_xored* (2/1) та *rsqrt1dc8_xored* (3/0), що є швидшими за *ranlux24*, *ranlux48* та *knuth_b*, а ГПВЧ *rand* та *random* у ESP-32 провалюють статистичні тести NIST (див. рис. Ж.9 в додатках).

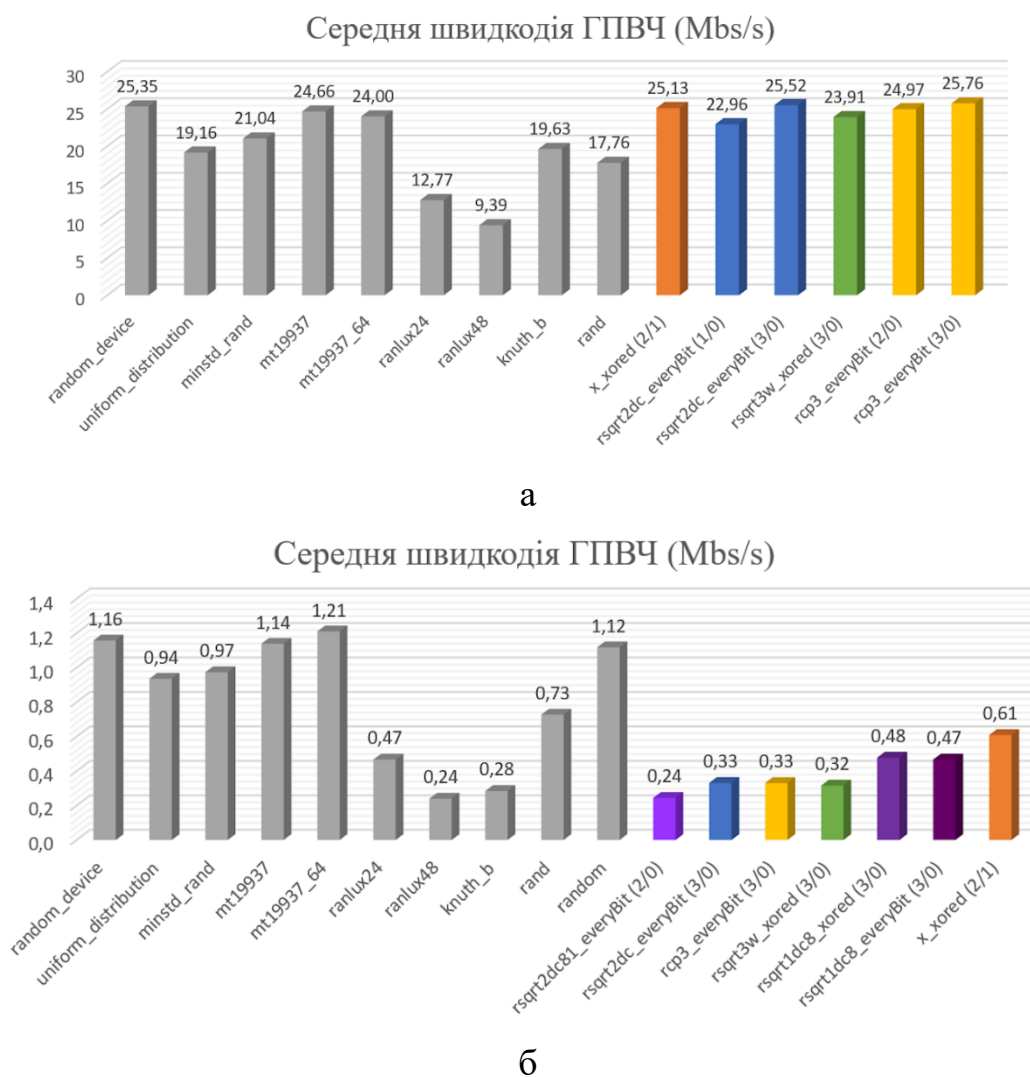


Рисунок 4.8 – Порівняння швидкодії доступних програмних та запропонованих методів ГПВЧ: (а) на платформі Intel i7 та (б) на МК ESP-32

4.2.4 Детальний аналіз результатів статистичного дослідження запропонованих ГПВЧ для сформованих ПВП максимальної довжини

Після визначення періоду повторення L , на третьому етапі дослідження ГПВЧ для задач КБ та ЗІ, ми дослідили статистичні характеристики сформованих ПВП відповідного розміру ($n \cdot m = L$), зокрема з використанням графічних тестів.

Продemonструємо одержані результати на прикладі ГПВЧ *rsqrt2dc81_everyBit* (2/0), *rsqrt2dc_everyBit* (1/0) та *rsqrt3w_xored* (3/0) із табл. 4.3, а також *rsqrt2dc_everyBit* (3/0) із рис. 4.5. Наведений в додатках рис. Ж.6 містить графічне представлення ПВЧ із початку та з кінця сформованих неперіодичних послідовностей для вказаних генераторів при початковому значенні $x_0 = 6,491708 \cdot 10^{24}$. Гістограми відповідних послідовностей розміру L байтів та обчислені значення ентропії наведені на рис. Ж.7. Порівняння графічних тестів між ГПВЧ *rsqrt2dc81_everyBit* (2/0) та генератором ІВЧ *rdrand64* показує подібні результати розподілу та ентропії $H(D_L)$ (див. рис. Ж.8). Такий підхід дозволяє наочно та кількісно оцінити рівномірність розподілу згенерованих ПВЧ та виявити можливі недоліки відповідних генераторів (див. рис. Ж.6д,е). Слід зазначити, що на відміну від інших трьох ГПВЧ, *rsqrt2dc_everyBit* (3/0) для вказаного початкового значення провалив статистичне тестування NIST STS [85]. При цьому згенерована ПВП не проходить 2 статистичні тести: FFT та Non-Overlapping Template. Помітний статистичний недолік вказує на наявність у структурі генератора внутрішніх патернів, що можуть бути використані зокрема для прогнозування вихідних значень. Незважаючи на це, для більшості початкових значень x_0 ГПВЧ *rsqrt2dc_everyBit* (3/0) проходить тести NIST STS.

Наведемо тепер в додатках на рис. Ж.10 приклад результатів тестів NIST для випадку динамічного вибору параметра m залежно від періоду L . У цьому випадку для кожного початкового значення x_0 ГПВЧ на випадковість тестуються неперіодичні послідовності максимальної довжини L . Кількісна оцінка відсотку проходження генераторами із табл. 4.3 всіх 188 тестів при зміні параметра m для випадкових x_0 наведена у табл. 4.4. Зауважимо, що через високу обчислювальну складність тестування, обсяг вибірки в цьому випадку є меншим. Як бачимо, майже всі ГПВЧ показують значно гірші результати статистичного тестування у випадку динамічного вибору параметра m (як і при $m=100$), ніж при $m=1000$. Ці результати також залежать від періоду L ГПВЧ: для параметрів (1/0) – в середньому на 22% менше, для (2/0) – на 23% менше, для (3/0) – від 18 до 52%

менше, і лише для ГПВЧ *x_xored* (2/1) та *rsqrt3dc_xored* (2/0) відповідно на 16% та на 3% більше ПБП проходить всі тести NIST (75% та 67% згенерованих ПБП). Отже, таке розширене тестування найкраще пройшли ГПВЧ *x_xored* (2/1), *rsqrt2dc_everyBit* (1/0) та *rsqrt3dc_xored* (2/0). Порівняння запропонованих ГПВЧ будемо виконувати також з використанням власної узагальненої метрики на основі статистичних результатів (див. табл. 4.4, де 100 – максимальне значення).

Таблиця 4.4 – Результати тестування запропонованих ГПВЧ статистичними тестами NIST STS при $m = 1000$ та при зміні параметра m (для ПБП розміру L)

Позначення ГПВЧ (<i>y_mode</i>)	Параметри ГПВЧ (<i>bytesCount / offset</i>)	Відсоток проходження всіх тестів NIST ($m = 1000$)	Оцінка відсотку проходження всіх тестів NIST при зміні m (згідно з формулою (2.48) на основі L)	Узагальнена оцінка (метрика)
<i>rsqrt2dc81_everyBit</i>	2/0	81,19%	42,86%	63,23
<i>rsqrt2dc_everyBit</i>	1/0	79,73%	68,42%	73,49
<i>rsqrt1dc8_everyBit</i>	2/0	78,33%	46,15%	63,16
<i>rsqrt2dc82_everyBit</i>	3/0	76,47%	58,33%	61,84
<i>rcp3_everyBit</i>	3/0	75,53%	61,54%	68,53
<i>rsqrt3w_xored</i>	3/0	74,32%	36,84%	49,61
<i>rsqrt2dc81_xored</i>	3/0	74,07%	50,00%	59,26
<i>rsqrt2dc81_everyBit</i>	1/0	72,13%	56,25%	62,05
<i>rsqrt1dc8_everyBit</i>	3/0	70,59%	33,33%	50,8
<i>rsqrt1dc8_everyBit</i>	1/0	70,59%	50,00%	61,33
<i>rsqrt3w_xored</i>	2/0	68,97%	63,16%	59,97
<i>x_xored</i>	2/1	64,52%	75,00%	54,34
<i>rsqrt3dc_xored</i>	2/0	64,48%	66,66%	63,79

Проаналізувавши всі одержані результати, можна зробити висновок, що найкращі характеристики серед досліджених має ГПВЧ *rsqrt2dc_everyBit* (1/0). Доля проходження тестів NIST STS цього генератора, зокрема для криптографічних застосувань, складає 79,73% (середня кількість пройдених тестів

рівна 187,76), проте його недолік – відносно невеликий період повторення ($2^{27,73}$) та трохи гірші результати тестування для довжини ПВП рівній періоду L – близько 68,42%. Генератор *rcp3_everyBit* (3/0) має трохи гірші статистичні характеристики як *rsqrt2dc_everyBit* (1/0), проте має більший період повторення ($2^{29,31}$) та кращу швидкодію, особливо на ESP-32. Ще декілька ГПВЧ на основі методу 8DC із близькими значеннями метрики (62-63 згідно з табл. 4.4) мають відносно непогані характеристики для сфери кібербезпеки та захисту інформації, зокрема два повільніші – *rsqrt2dc81_everyBit* (2/0) (має чудові статистичні результати при $m=1000$: 81,19% проходження та середнє значення 187,78) та *rsqrt2dc81_everyBit* (1/0); а також два швидші – *rsqrt2dc82_everyBit* (3/0) (краще проходить тестування для ПВП довжини L : 58,33%) та *rsqrt1dc8_everyBit* (2/0).

4.3 Аналіз і порівняння отриманих результатів тестування запропонованих ГПВЧ на основі чисельних методів у FP арифметиці

4.3.1 Порівняння з відомими методами за статистичними показниками

Всі розглянуті в пп. 4.2.1 та 4.2.3 альтернативні методи генерації випадкових та псевдовипадкових послідовностей тестувались на статистичні характеристики сформованих ними послідовностей (розміром до 2 Гб кожна) з використанням NIST STS, в тому числі для випадкових початкових значень ГПВЧ. При цьому досліджувалось використання різних параметрів генерації (*bytesCount / offset* в режимі *everyBit*).

Одержані результати тестування представлені у вигляді діаграм на рис. 4.9, де відповідні генератори посортовані за їх типами та результатами статистичного тестування. При цьому, на рис. 4.9а результати узагальнені за параметрами генерації, а на рис. 4.9б наведені лише основні версії цих генераторів. Також для порівняння у додатках на рис. Ж.9 наведено аналогічні результати для апаратних генераторів ІВП Intel та ESP-32. Можна побачити, що запропоновані в цій роботі ГПВЧ на основі обчислення елементарних функцій в FP арифметиці, зокрема *rsqrt2dc81_everyBit* (2/0), показують досить непогані результати проходження статистичних тестів у порівнянні з відомими методами (див. рис. 4.9а та Ж.9а).

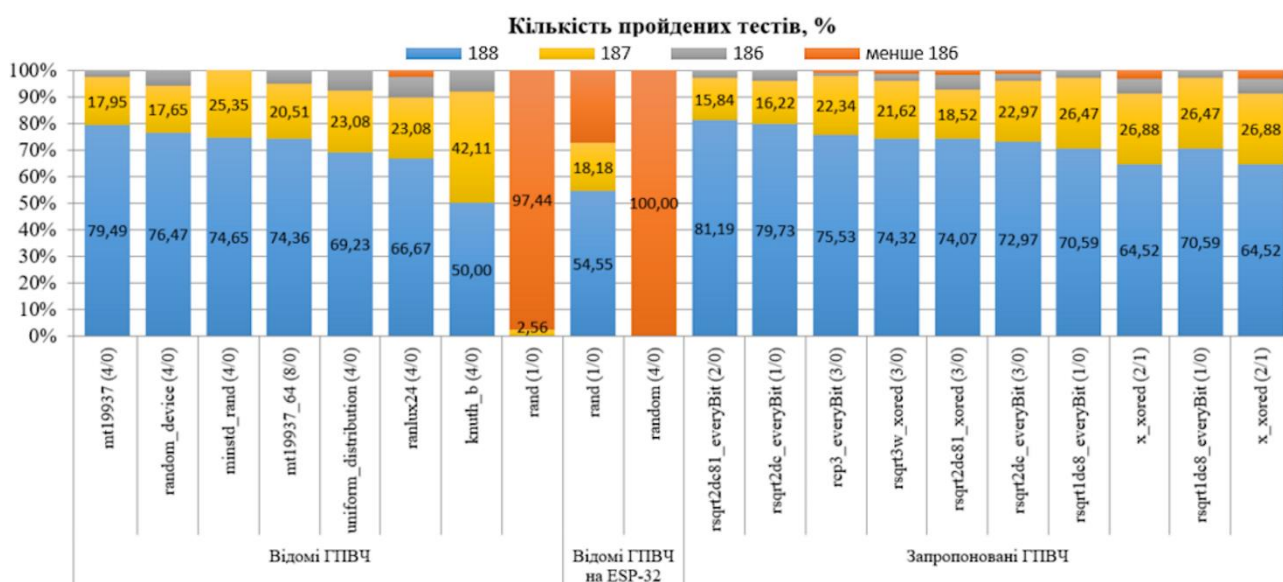
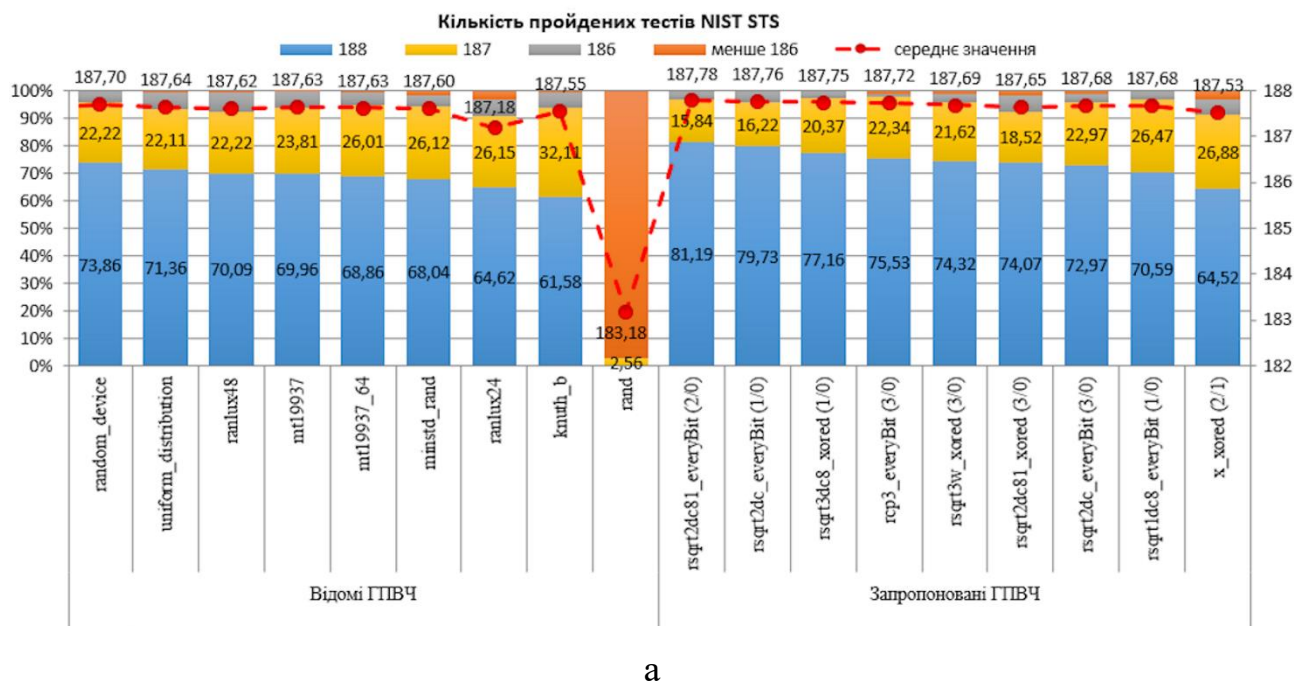


Рисунок 4.9 – Порівняння відомих ГПВЧ, доступних на Intel та ESP-32, із запропонованими ГПВЧ за результатами тестування NIST STS при $m = 1000$:
 (а) узагальнені результати відсотку проходження та середнього значення кількості пройдених тестів за типами і параметрами генерації та (б) результати тестування основних типів ГПВЧ із фіксованими параметрами

Крім того, у додатках на рис. Ж.11а та Ж.11б наведено детальні результати порівняння досліджуваних генераторів за відсотком проходження та середньою кількістю успішно пройдених тестів NIST STS.

Отже, найкращі статистичні характеристики мають запропоновані ГПВЧ *rsqrt2dc81_everyBit* (2/0) та *rsqrt2dc_everyBit* (1/0) із відповідно 81,19% та 79,73% проходження тестів NIST, а також апаратний метод генерації ІВЧ на ESP-32 з використанням фізичного джерела ентропії (у режимі з увімкненим WiFi-модулем) *esp_fill_random* з параметрами (8/0), що проходить 82,35% тестів, та 32-бітний генератор *mt19937* із показником в 79,49%. Як бачимо, повністю провалили тестування ГПВЧ *rand* на Intel, а також ГПВЧ *rand* і *random* на ESP-32.

4.3.2 Порівняння основних характеристик та оцінка ефективності запропонованих ГПВЧ для задач у сфері кібербезпеки та захисту інформації

Зведемо в одну таблицю всі отримані результати тестування найкращих запропонованих ГПВЧ після третього етапу статистичного дослідження (див. табл. Ж.3 в додатках). Теоретичний аналіз ефективності та швидкодії ГПВЧ проведено шляхом оцінки кількості необхідних арифметичних операцій. Одержані результати представлені у табл. Ж.4 в додатках.

Як бачимо з табл. Ж.4 згідно з використаною оцінкою швидкодії найвищу очікувану швидкість генерації ПВП має ГПВЧ *x_xored* (2/1), хоча він має деякі статистичні недоліки, як зазначено у табл. Ж.3. З іншого боку, найкращі статистичні характеристики сформованих ПВП згідно з табл. Ж.3 та Ж.4 має ГПВЧ *rsqrt2dc_everyBit* (1/0), хоча його реалізація вимагає великої кількості FP операцій (див. табл. Ж.4). Серед усіх досліджених генераторів найоптимальнішими варіантами як щодо статистичних характеристик, так і щодо кількості використаних арифметичних операцій є ГПВЧ *rsqrt1dc8_everyBit* з параметрами 2/0, а також ГПВЧ *rcp3_everyBit* з параметрами 3/0.

4.4 Сфери застосування та перспективні напрямки подальших досліджень

Запропоновані покращені алгоритми обчислення елементарних функцій, а також прості й швидкі ГПВЧ, створені на їх основі, можуть бути ефективно

використані під час проєктування новітніх інформаційних систем, технологій і комплексів для створення, обробки, передачі, зберігання, захисту та відображення інформації, а також автоматизованих систем управління інформаційною, кібербезпекою та захистом інформації. Наприклад, завдяки їх високій точності та малій обчислювальній складності, ці алгоритми можуть бути реалізованими у вигляді ефективних протоколів захищеного багатокористувацького обчислення. А запропоновані ГПВЧ можуть бути використані в задачах стеганографії, криптографії, моделювання та симуляції у сфері кібербезпеки та захисту інформації, а також для побудови генераторів шуму.

Отже, можна виділити наступні сфери застосування, в тому числі для задач КБ та ЗІ, а також перспективні напрямки подальших досліджень:

- розробка вдосконалених методів генерації ПВЧ та ПВБП із прийнятними статистичними характеристиками для криптографічних [51,54-56,66,78,160] і некриптографічних [2,15,16,55,85,162] застосувань, зокрема створення ефективних криптографічно стійких генераторів на основі чисельних методів та теорії хаосу [36,51,58]:
 - для приховування інформації в стеганографії [15,49,57,59,60,85-87];
 - для генерації ключів та інших випадкових даних для алгоритмів блокового та потокового шифрування, систем автентифікації, а також алгоритмів цифрового підпису [15,36,49,51,54,62];
 - для різноманітних криптографічних протоколів передачі та обміну інформацією [15,52,54], зокрема розподілу таємниці [67-69,71] та захищеного багатокористувацького обчислення [68-70,72,73];
 - для моделювання та симуляції, комп'ютерних ігор, статистичного аналізу, навчання нейронних мереж, методу Монте-Карло, оптимізації та рандомізованого пошуку [32,49,55,61,76,77];
 - для проєктування генераторів шуму, зашумлення та надійної передачі даних в безпроводних технологіях [15,49];
- використання для задач статистики, КБ, ІБ та цифрової обробки сигналів (наприклад, при обчисленні автокореляції, середнього значення –

функція DIV; середньоквадратичного відхилення, середньоквадратичної похибки (RMSE), пікового співвідношення сигналу до шуму (PSNR), коефіцієнта кореляції, лінійної регресії, центру мас вихідного розподілу – функції SQRT та DIV; відстані – SQRT), зокрема:

- для реалізації механізмів інтелектуального аналізу даних та систем прийняття рішень, наприклад для виявлення, класифікації та реагування на інциденти ІБ [190,203];
- для ефективної реалізації різноманітних статистичних тестів, наприклад із набору NIST STS [78] чи TestU01 [171];
- для систем біометричної автентифікації, наприклад для розпізнавання відбитків пальців за особливими точками [9,26];
- для попереднього опрацювання даних, отриманих з АЦП сенсорів та детекторів в IoT [31];
- для вибору областей стеганографічного вбудовування даних, оцінки якості стеганографічних методів приховування інформації, аналізу цифрових зображень тощо [86,87,204,205];
- розробка ефективних програмних та апаратних реалізацій даних методів для задач КБ та ЗІ на різноманітних платформах, що підтримують основні операції в FP арифметиці, зокрема для високопродуктивних систем, пристроїв з обмеженими ресурсами, а також систем, захищених від математичних, статистичних, криптографічних атак і атак через побічні канали витоку інформації [7,8,24,33,144,157,202,206];
- розробка математичних бібліотек або апаратних інструкцій для швидкого обчислення необхідних для сфери КБ та ЗІ елементарних функцій в FP арифметиці [5,21,22,99,106,108,109,131], зокрема методів коректного заокруглення та опрацювання виняткових ситуацій [18,22] для довільних FP типів відповідно до вимог стандарту IEEE 754 [39], а також обчислення з довільною точністю [84,128,160,207];
- реалізація швидких методів лінійної алгебри (розв'язання систем рівнянь, розклад матриці та знаходження обернених матриць з

використанням методів Гауса, Холецького, LU, QR тощо – функції DIV, SQRT та RCP) [1,25,208-212], зокрема для задач КБ та ЗІ:

- для збереження конфіденційності баз даних при їх інтелектуальному аналізі [12,13,211], для систем кодування та шифрування даних [209];
- для систем бездротового зв'язку, зокрема для відновлення отриманого сигналу від завад у системах MIMO-OFDM [208,210];
- для новітніх систем автентифікації користувачів [212];
- використання для задач комп'ютерної графіки та тривимірного моделювання (освітлення, тіні, нормалізація векторів – функція RSQRT; побудова перспективи – функція RCP) [6,11,137,156];
- розробка протоколів захищеного багатокористувацького обчислення в арифметиці з фіксованою [12,69,70,72,74,213,214] та рухомою [67,68,72,73,215-217] комою, зокрема для конфіденційного інтелектуального та статистичного аналізу, наукових та аутсорсингових обчислень [12,13,72,73,216,218,219];
- використання методів RSQRT для побудови функції активації нейронних мереж [14,220] та функції приналежності нечітких множин [221], зокрема для виявлення шкідливого програмного забезпечення, поведінкового аналізу користувачів, виявлення аномалій та вторгнень в мережі, прогнозування роботи систем тощо [191,203,222-224].

4.5 Практичне використання ГПВЧ для задач інформаційної та кібербезпеки

Серед широкого кола згаданих вище можливих застосувань запропонованих ГПВЧ на основі чисельних методів в FP арифметиці, серед яких стеганографія, криптографія, нейронні мережі, симуляція, генерація випадкових значень в криптографічних протоколах [85], виділимо наступні два приклади.

4.5.1 Приклад 1. Використання розробленого ГПВЧ для формування індивідуальних варіантів завдань для студентів

Типовою проблемою, з якою стикаються вчителі та викладачі вищих навчальних закладів, – це періодична необхідність формування та розподілу між

учнями чи студентами індивідувальних завдань. Це можуть бути як тести, так і варіанти завдань для контрольних робіт чи питань екзаменаційних білетів для об'єктивного оцінювання рівня засвоєння знань. Випадковість, непередбачуваність та унікальність варіантів завдань є елементом стимулювання академічної доброчесності, самостійності виконання завдань та протидії плагіату, можливим зловживанням чи фальсифікаціям.

Розглянемо приклад формування індивідуальних завдань курсового проєктування з дисципліни «Архітектура комп'ютерних систем» з використанням розробленого ГПВЧ *rsqrt2dc_everyBit* (1/0). Відповідно до завдання [225] кожен варіант має містити підмножину системи інструкцій прототипу скалярного RISC-комп'ютера, що складається з 6 апаратних інструкцій трьох типів: 2 інструкції пересилання даних, 3 арифметико-логічних інструкції та 1 інструкцію керування.

Для виконання цього завдання на базі згаданого ГПВЧ, що повертає вихідні значення з діапазону $[0; 255]$, реалізуємо методи для формування псевдовипадкових значень з довільного цілочислового інтервалу. Одержані результати генерації набору з 10000 варіантів завдань показані на рис. 4.10. Як бачимо, для кожного з трьох типів інструкцій обчислені значення ентропії становлять відповідно 0,999833 для набору з 18 інстр. пересилання (або 4,16923 біт/інстр.); 0,999912 для набору з 31 інстр. опрацювання (або 4,953762 біт/інстр.) та 0,999864 для набору з 10 інстр. керування (або 3,321478 біт/інстр.), забезпечуючи достатній рівень випадковості згенерованих варіантів завдань.



Рисунок 4.10 – Гістограми розподілу та одержані значення ентропії сформованих ГПВЧ *rsqrt2dc_everyBit* (1/0) 10000 варіантів індивідуальних завдань

4.5.2 Приклад 2. Використання розробленого ГПВЧ для вбудовування та шифрування даних у стеганографічних системах

Іншим прикладом застосування розроблених ГПВЧ на основі обчислення елементарних функцій є стеганографічні алгоритми та протоколи. Основними перевагами використання ГПВЧ є псевдовипадковий вибір пікселів для вбудовування, можливість суттєвого зменшення розміру стеганографічного ключа, а також шифрування даних методом одноразового блокноту [87]. Зокрема в роботах [86,87] ми розглядали методи аналізу та обробки цифрових зображень, визначення ділянок зображення, придатних для вбудовування інформації (висококонтрастних та контурних ділянок стеганоконтейнера C_v), захист конфіденційності та цілісності цієї інформації, а також приховування самого факту наявності в контейнері вбудованих даних. Була запропонована модифікована модель стеганосистеми (див. рис. 4.11), де використовується блок аналізу особливостей контейнерів з використанням псевдовипадкових масок (див. рис. 4.12), а також вибір пікселів для вбудовування на основі запропонованих ГПВЧ.

Ефективність розглянутих стеганографічних методів аналізувалась, зокрема, на основі наступних параметрів: стійкість до виявлення (візуальна якість, максимальна різниця, RMSE, PSNR); пропускна здатність (максимальна довжина повідомлення); довжина ключа K_l та максимальний коефіцієнт його стиску, довжина згенерованої ПВП та швидкодія алгоритмів в середовищі Matlab [86,87].

Як бачимо на рис. 4.13а-в, у запропонованому базовому методі 3а, що базується на основі стеганонезалежного аналізу особливостей модифікованого контейнера та не використовує бітову маску на основі ГПВЧ, розмір ключа залежить лише від максимального розміру повідомлення (в даному випадку, лише 19 бітів). Хоча розмір ключа K_l в методі 3а є мінімальним, при цьому не забезпечується конфіденційність вбудованих даних, а ділянки вбудовування визначаються лише особливостями контейнера та визначеними параметрами побудови стеганосистеми.



Рисунок 4.11 – Використана вдосконалена модель стеганосистеми на основі аналізу особливостей контейнера C_v , що використовує ключі мінімального розміру на основі ГПВЧ

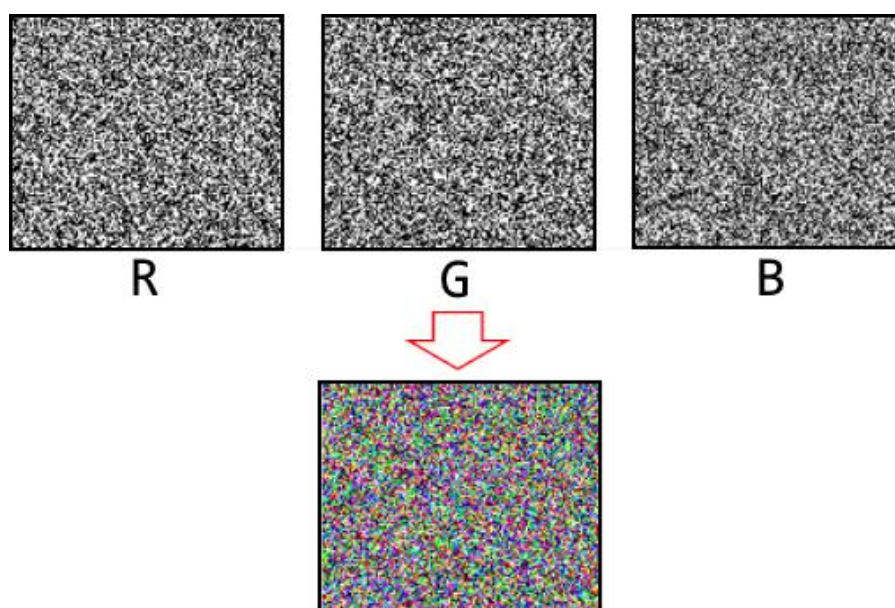
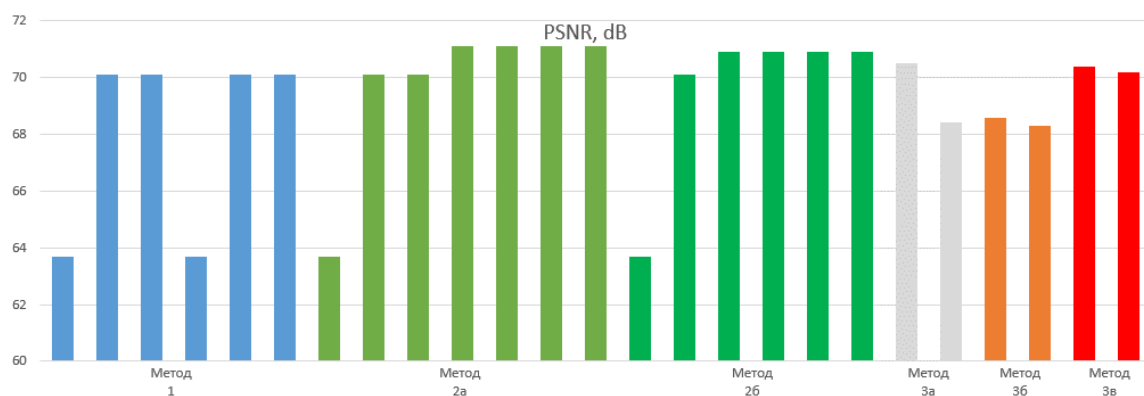
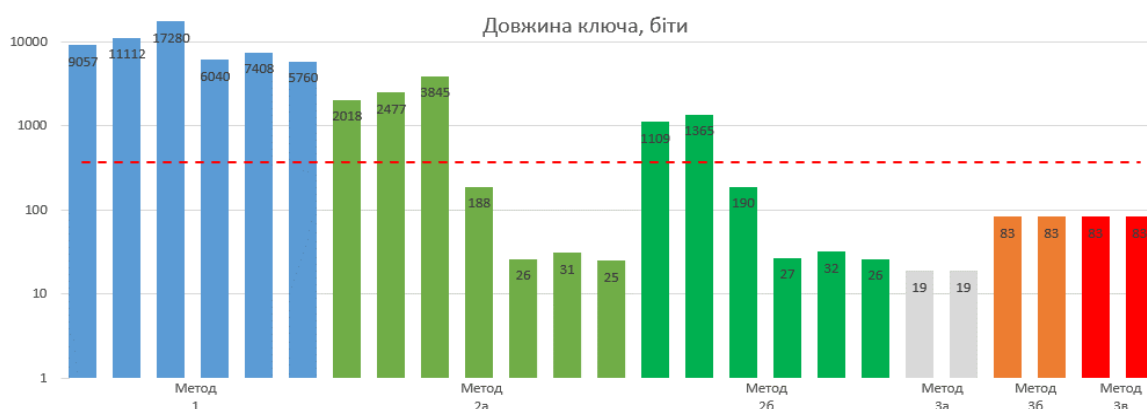


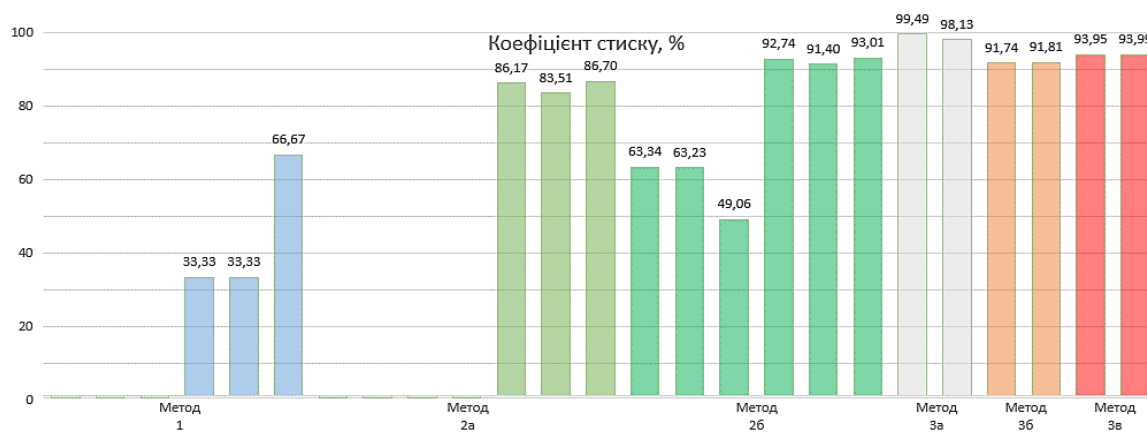
Рисунок 4.12 – Приклад побудови псевдовипадкової бітової маски, що використовується в блоці врахування особливостей контейнерів методів 3б та 3в вдосконаленої моделі стеганосистеми із [86,87], сформована ГПВЧ $rsqrt2dc_everyBit$ (1/0) із початковим значенням $x_0 = 0,5$



а



б



в

Рисунок 4.13 – Порівняння ефективності досліджуваних стеганографічних методів на основі використання ГПВЧ: (а) пікове співвідношення сигналу до шуму вбудованого контейнера, (б) довжини стеганографічного ключа K_l у випадку повідомлення довжини 368 біт та (в) коефіцієнти стиску ключа K_l

Для порівняння, вдосконалені стеганографічні методи Зб та Зв, при незначному підвищенні рівня шуму у вбудованому стеганоконтейнері (меншою мірою для методу Зв), завдяки використанню ГПВЧ, забезпечують високий коефіцієнт стиснення ключа – на рівні 92-94% (розмір ключа становить 83 біти) – при збереженні достатнього рівня безпеки як у варіанті без захисту конфіденційності, так і з шифруванням даних.

Слід зазначити, що для безпосереднього шифрування критично важливих даних доцільно використовувати ключову послідовність, згенеровану надійним генератором ІВП (зауважимо, що в разі великих повідомлень це призводить до суттєвого збільшення розміру ключа) або криптостійким ГПВЧ. Альтернативним підходом до зменшення обсягу ключа для шифрування великих повідомлень є використання симетричних блокових шифрів, зокрема AES-128. Крім того, ключові послідовності, призначені для криптографічного застосування, зокрема сформовані за допомогою запропонованих ГПВЧ, мають проходити ретельну перевірку на відповідність статистичним тестам NIST STS, а відповідні генератори – на відсутність вразливостей і стійкість до різних методів криптоаналізу. Розроблення та дослідження ефективних криптостійких варіантів запропонованих ГПВЧ на основі FP арифметики є метою найближчих подальших досліджень.

Висновки до розділу 4

1. Виконано практичну реалізацію та тестування ефективності (зокрема, точності, латентності та зворотної пропускну здатності) запропонованих алгоритмів для сфери інформаційної та кібербезпеки на прикладі функцій RCP та RSQRT для чисел одинарної та подвійної точності на ПК із CPU Intel Core i5/i7, міні-комп'ютері Raspberry Pi та МК ESP-32. Здійснено їх порівняння з відомими алгоритмами, апаратними та програмно-апаратними методами. Проведено аналіз отриманих результатів та запропоновано способи подальшого підвищення швидкодії, що важливо, зокрема, для систем аналізу та прийняття рішень, аутентифікації, виявлення загроз та

реагування на інциденти інформаційної безпеки. З'ясовано, що запропоновані алгоритми найкраще підходять для реалізації на Raspberry Pi та, у випадку типу *float*, на ESP-32. За винятком типу *double* на ESP-32, метод DC демонструє у 1,8-3,6 раз кращу швидкодію за використання бібліотечних функцій на Raspberry Pi та ESP-32. Крім того, на ESP-32 нам вдалося ще зменшити латентність запропонованих оптимізованих алгоритмів з 2MC InvSqrt531d та InvSqrt532d на 13,9% та 7% відповідно, що забезпечує вищу швидкодію та кращу точність за ділення $1.0/\sqrt{x}$.

2. Проведено дослідження основних характеристик запропонованих ГПВЧ щодо вимог задач кібербезпеки та захисту інформації, зокрема параметрів генерації, статистичних властивостей, періоду повторення, діапазонів початкових значень та їх чутливості, обчислювальної складності та швидкодії. Дослідження статистичних характеристик ПБП виконувалось на основі використання тестів NIST STS для криптографічних застосувань, графічних тестів, побудови гістограми розподілу та обчислення ентропії. Дослідження, аналіз та порівняння найкращих запропонованих та відомих програмних і апаратних генераторів виконувалось у три етапи; розглянуто різні методи ініціалізації початкових значень. Також реалізовано набір процедур для автоматизації процесу тестування ГПВЧ. Досліджено характеристики цих генераторів при фіксованому початковому значенні $x_0 = 0$ та в залежності від вибору початкового значення, для різних функцій, режимів роботи та значень параметрів. З'ясовано, що оптимальна комбінація параметрів *bytesCount* і *offset* складає *bytesCount* = 3 і *offset* = 0 (3/0), а у випадку використання режиму *everyBit* краще приймати *offset* = 0. З'ясовано, що період повторення генераторів залежить від початкового значення x_0 , параметра *bytesCount* та типу елементарної функції. Максимальне значення періоду для початкових значень близьких до нуля складає приблизно 2^{29} значень (байтів). Для покращення статистичних

характеристик та криптостійкості запропонованих ГПВЧ механізм реініціалізації можна застосовувати періодично з фіксованою частотою.

3. За результатами проведених досліджень було виявлено, що найкращі характеристики щодо випадковості згенерованих ПВП при різних початкових значеннях мають ГПВЧ *rsqrt2dc_everyBit* (1/0), *rsqrt1dc8_everyBit* (2/0) та *rcp3_everyBit* (3/0), доля проходження тестів NIST яких складає відповідно 79%, 78% та 75%, а ГПВЧ *rsqrt2dc_everyBit* (3/0) не проходить графічні тести. Найвищу швидкість генерації ПВП має ГПВЧ *x_xored* (2/1), хоча він має статистичні недоліки. Серед усіх досліджених генераторів найоптимальнішими варіантами для застосувань у сфері кібербезпеки та захисту інформації як щодо статистичних характеристик, так і щодо кількості використаних арифметичних операцій є ГПВЧ *rsqrt1dc8_everyBit* (2/0) та *rcp3_everyBit* (3/0). Вони є швидшими за найшвидші ГПВЧ *random_device* та *mt19937* на Intel, а також апаратні генератори ІВП RDSEED та RDRAND, не вимагаючи значних обсягів пам'яті та тривалих термінів реініціалізації.
4. Досліджено сфери застосування запропонованих алгоритмів обчислення елементарних функцій в FP арифметиці та ГПВЧ на їх основі, зокрема у галузі інформаційної та кібербезпеки, а також перспективні напрямки подальших досліджень. Наведено два приклади практичного застосування ГПВЧ *rsqrt2dc_everyBit* (1/0), зокрема для проектування вдосконаленої моделі стеганосистеми на основі аналізу висококонтрастних та контурних ділянок цифрового зображення з використанням псевдовипадкової бітової маски, псевдовипадкового вибору пікселів, шифрування даних, перевірки їх цілісності та мінімізації довжини стеганографічного ключа.

ВИСНОВКИ

У дисертаційній роботі вирішено актуальне науково-прикладне завдання для галузі знань 12 «Інформаційні технології» – розроблення та дослідження генераторів псевдовипадкових послідовностей для сфери кібербезпеки та захисту інформації на основі вдосконалених алгоритмів наближеного обчислення елементарних функцій в арифметиці з рухомою комою, які забезпечують задовільні статистичні характеристики відповідно до вимог тестів NIST STS.

За результатами проведених досліджень можна сформулювати наступні висновки:

1. Визначено основні характеристики арифметики з рухомою комою та проведено аналіз особливостей та можливих загроз щодо її використання в критичних системах і для задач кібербезпеки та інформаційної безпеки. Було виявлено широку апаратну підтримку FP чисел та наявність суттєвих недоліків, що пов'язані з накопиченням похибок, втратою точності обчислень тощо. Проведено огляд застосування чисельних методів та FP арифметики для генерації псевдовипадкових послідовностей у сфері кібербезпеки та захисту інформації, серед яких генератори на основі ірраціональних математичних констант, SQRT з простих чисел, раціональних чисел з великим простим знаменником тощо. При цьому, числа у FP форматі можуть використовуватись як при реалізації генератора, так і бути вихідним форматом псевдовипадкових даних. З'ясовано, що певні особливості та помилки реалізації FP обчислень можуть бути використані для зламу та компрометації інформаційних систем.

2. Проведено класифікацію та аналіз поширених методів обчислення елементарних функцій, визначено їх переваги та недоліки для сфери кібербезпеки та захисту інформації. Серед них можна виділити наступні: на основі бітової маніпуляції, рекурентні, цифра-за-цифрою, ітераційні, поліноміальні, дробово-раціональні, табличні, а також їх комбінації тощо. Досліджено теоретичні основи алгоритму FISR для обчислення функції RSQRT, проведено аналіз модифікацій алгоритму FISR та побудовано його узагальнену математичну модель для апроксимації інших логарифмічних, експоненціальних та степеневих функцій.

Виявлено, що найкращі модифіковані методи Кадлеця і Вальчика та ін. дозволяють зменшити максимальні відносні похибки базового алгоритму FISR після першої ітерації у 2,7 раз. Досліджено застосування методів початкового наближення на основі МС, поліноміальної апроксимації та наближених апаратних інструкцій, а також ітераційних методів NR та Н вищих порядків збіжності. Доведено ефективність модифікованого методу МС порівняно з поліномами найкращого рівномірного наближення для функцій SQRT, RCP та RSQRT – збільшення точності на 5 біт після однієї ітерації за умови використання приблизно такої ж кількості FP операцій. Проведено аналіз точності наближених апаратних інструкцій FRECPE і FRSQRTE від ARM (8 біт) та RCPSS і RSQRTSS від Intel (11 біт), а також застосування спеціальних інструкцій FRECPS і FRSQRTS технології ARM NEON. З'ясовано, що запропонована в літературі модифікація ітераційного методу Н дозволяє для функції RSQRT уникнути важкої операції $\div$. Здійснено порівняння відомих методів на основі МС, зокрема функцій RSQRT, RCP, RCBRT, SQRT та CBRT для FP чисел одинарної та подвійної точності. Сформульовано основні вимоги для застосувань у сфері інформаційної та кібербезпеки і запропоновано методи покращення щодо їх точності та швидкодії, зокрема: модифікація МС, комбінація модифікованих ітераційних методів NR і Н в особливій формі запису, оптимізація параметрів алгоритму для зменшення максимальної відносної похибки, мінімізація кількості $\div$ та $\times$, використання швидких операцій FMA тощо.

3. Проведено класифікацію та аналіз відомих генераторів ПБП, зокрема популярних методів генерації некриптостійких (наприклад Xorshift і вихор Мерсена) та криптостійких ГПВЧ (наприклад Блум-Блум-Шуба, AES-CTR DRBG і на основі теорії хаосу). Визначено області їх застосування, особливо у сфері кібербезпеки та захисту інформації, проаналізовано основні вимоги та методи оцінювання їх якості, зокрема на основі статистичних тестів NIST STS та графічних тестів.

4. Запропоновано методику проектування покращених алгоритмів обчислення елементарних функцій в FP арифметиці, придатних для задач у сфері

інформаційної та кібербезпеки, щодо точності та швидкодії. Розроблено математичну модель модифікованих алгоритмів та набір інструментів для їх вдосконалення та автоматизованого тестування, що можуть бути використані для подальших досліджень. Зокрема, здійснено імітаційне моделювання модифікованих алгоритмів для задач кібербезпеки та захисту інформації, що включає побудову аналітичної моделі методу МС, полінома найкращого рівномірного наближення для відносної похибки, запропонованої математичної моделі спеціальних апаратних інструкцій наближених функцій RCP і RSQRT, наприклад FRSQRTE від ARM, та комбінацій різноманітних ітераційних методів. Використовуються ефективні методи модифікації алгоритмів для обчислення інших поширених у сфері КБ та ЗІ елементарних функцій, наприклад DIV, SQRT і CBRT на базі швидких алгоритмів для RCP, RSQRT і RCBRT без $\div$, що потребують максимум одну-дві додаткові операції $\times$. Використовуються практичні та теоретичні методи визначення абсолютних та відносних похибок, оцінки точності та швидкодії алгоритмів на різних платформах. Для визначення оптимальних коефіцієнтів алгоритмів використано два типи методів: узагальнений аналітичний метод послідовного вирівнювання похибок, а також запропонований набір методів рандомізованої чисельної багатовимірної оптимізації на основі мінімізації максимальних відносних похибок з врахуванням похибок заокруглення. Для підвищення швидкодії чисельної оптимізації застосовується пошук за методом золотого перетину.

5. Запропоновано методику проектування та дослідження ГПВЧ, придатних для сфери кібербезпеки та захисту інформації, на основі FP арифметики, а саме: дослідження статистичних характеристик сформованих ПВП з використанням тестів NIST (для криптографічних застосувань) та графічних тестів, визначення зміщення та періоду повторення періодичних ПВП, а також швидкодії ГПВЧ. Реалізовано набір методів для автоматизації процесу тестування. Методика дослідження передбачала три етапи статистичного тестування для різних довжин ПВП при зміні початкових значень та параметрів ГПВЧ. Тестування виконувалось на ПК із CPU Intel Core i5/i7 та МК ESP-WROOM-32.

Запропоновані методи автоматизації статистичного тестування на основі тестів NIST STS, визначення періоду повторення ПБП, вимірювання швидкодії та продуктивності ГПВЧ є універсальними та ефективними інструментами, що можуть бути безпосередньо застосовані науковцями в інших дослідженнях у сфері кібербезпеки та захисту інформації.

6. Вперше запропоновано набір покращених методів для наближеного обчислення функцій RCP, RSQRT, DIV та SQRT на основі модифікації алгоритму FISR: метод швидкого обчислення RCP на основі тотожності з поліномами найкращого рівномірного наближення P_3^* (точність після першої ітерації – 13 біт); методи швидкого обчислення RCP та RSQRT на основі модифікованого методу Ho2 на першій ітерації (точність після першої ітерації – 14 біт для RCP та 15 біт для RSQRT); методи двох та трьох магічних констант (2MC та 3MC) для функцій RSQRT та RCP, що дозволяють на першій ітерації зменшити на одиницю кількість $FP \times$ в алгоритмі (точність після першої ітерації – 10 біт для RSQRT та 13 біт для RCP); методи переключення магічних констант (DC та 8DC) для обчислення RSQRT та SQRT на основі розбиття інтервалу $x \in [1; 4)$ на дві та вісім частин відповідно й використання оптимізованих MC та коефіцієнтів на кожній з них після першої ітерації (точність після першої ітерації – 13 біт для DC та 16 біт для 8DC), забезпечуючи якісний компроміс між точністю, швидкістю та розміром LUT. Ці методи не використовують складної операції $\div$ та громіздких LUT і дозволяють збільшити точність результату на 2-6 бітів (на 2-14 бітів вище) і/або зменшити кількість множень (до 5 множень менше) порівняно з відомими методами на основі FISR після першої ітерації (для двох і більше ітерацій). Розроблено також прості модифікації алгоритмів на основі методів MC, кусково-поліноміальної апроксимації та наближених апаратних інструкцій Intel і ARM з оптимізованими коефіцієнтами. На їх основі реалізовано набір покращених алгоритмів для обчислення функцій RCP, RSQRT, RCBRT із підвищеною точністю і/або швидкістю для FP чисел одинарної та подвійної точності, а також модифікації для DIV, SQRT і CBRT, інших логарифмічних та показникових функцій. Слід зазначити, що деякі з розглянутих методів обчислення RSQRT,

SQRT та CBRT вимагають операції $\div$. Запропоновані алгоритми будуть корисними та ефективними під час проектування сучасних інформаційних систем, технологій та комплексів обробки, аналізу, передачі, зберігання та захисту інформації, а також автоматизованих систем управління ІБ та КБ.

7. Вперше запропоновано новий клас методів генерації псевдовипадкових чисел на основі рекурентного рівняння та чисельних методів в FP арифметиці. За результатами проведених досліджень було виявлено, що найкращі характеристики для застосувань у сфері кібербезпеки та захисту інформації має ГПВЧ *rsqrt2dc_everyBit* (1/0), для якого відсоток проходження всіх тестів NIST при зміні початкових значень складає 79,73%, а середня кількість пройдених тестів рівна 187,76. Для покращення статистичних характеристик та криптостійкості запропонованих ГПВЧ можна застосовувати механізм реініціалізації. Здійснено оцінку швидкодії запропонованих ГПВЧ на основі підрахунку кількості арифметичних операцій, а також практичне тестування продуктивності. З'ясовано, що ГПВЧ *x_xored* (2/1) має досить високу швидкодію при деяких статистичних недоліках, а ГПВЧ *rsqrt2dc_everyBit* (1/0), навпаки, має кращі статистичні характеристики, проте значно гіршу швидкодію. Ще два генератори – *rsqrt1dc8_everyBit* (2/0) та *rcp3_everyBit* (3/0) – демонструють чудові компромісні результати дослідження. Зокрема, вони є швидшими за найшвидші ГПВЧ *random_device* та *mt19937* бібліотеки *random*, а також апаратні генератори істинно випадкових послідовностей RDSEED та RDRAND на Intel (швидкість генерації псевдовипадкових даних – до 40 Мб/с на Intel i5); доля проходження тестів NIST складає 78% та 75% відповідно, що також приблизно на 5-12% більше за згадані генератори. З'ясовано, що період повторення генераторів залежить від початкового значення x_0 , параметра *bytesCount* та типу елементарної функції; максимальний період рівний $L = 6,66 \cdot 10^8$ значень (байтів); множина початкових значень складає приблизно 2^{60} елементів (64-бітових FP чисел); значення ентропії згенерованих ПВП близьке до 8 біт. Відтак є придатними для некриптографічних застосувань у кібербезпеці, стеганографії, комп'ютерних іграх тощо.

8. Проведено аналіз сфер застосування запропонованих алгоритмів обчислення елементарних функцій та ГПВЧ на їх основі, зокрема для кібербезпеки та захисту інформації, а також сформульовано перспективні напрямки подальших досліджень. Наведено два приклади практичного застосування запропонованих ГПВЧ, зокрема у навчальному процесі та в стеганографії. З'ясовано, що запропоновані покращені алгоритми найкраще підходять для реалізації на міні-комп'ютері Raspberry Pi та, у випадку чисел одинарної точності, на мікроконтролері ESP-32, а запропоновані ГПВЧ на їх основі – на 64-бітних платформах Intel i5/i7 та Raspberry Pi.

9. На основі розроблених покращених алгоритмів буде створено спеціалізовану математичну бібліотеку для ПК, МК та GPU, що підтримує ефективні обчислення елементарних функцій в FP арифметиці з різною точністю: швидке наближення (наприклад, з точністю 13-14 біт для RCP і 10-16 для RSQRT), грубе обчислення (похибка на рівні 1 ULP) та з коректним заокругленням результату. Також можна розробити відповідну бібліотеку запропонованих легких та швидких ГПВЧ на їх основі для застосувань у сфері кібербезпеки та захисту інформації, що мають задовільні статистичні характеристики відповідно до тестів NIST. Більш того, запропоновані методи (особливо методи DC) підійдуть для апаратної реалізації на ПЛІС та у вигляді швидких апаратних інструкцій FPU, зокрема в режимі SIMD. Такі алгоритми будуть ефективними на платформах, що мають апаратну підтримку основних FP операцій.

10. Отримані результати досліджень впроваджено в ПП НВПІ «Спаринг-Віст Центр», використовувались при виконанні науково-дослідних робіт та у навчальному процесі кафедри безпеки інформаційних технологій Національного університету «Львівська політехніка» для студентів спеціальності 125 «Кібербезпека та захист інформації».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lemaitre F., Couturier B., Lacassagne L. Cholesky factorization on SIMD multi-core architectures. *Journal of Systems Architecture*. 2017. Vol. 79. 15 p.
2. Горпенюк А., Лужецька Н. Генератор псевдовипадкових чисел на обчислювачі кореня квадратного з простого числа. *Вісник НУ Львівська політехніка “Автоматика, вимірювання та керування”*. 2013. № 753. С. 45–50.
3. Parhami B. Function evaluation. *Computer arithmetic: algorithms and hardware designs* : textbook. New York, USA : Oxford University Press, 2010. P. 437–520.
4. Muller J.-M. Elementary functions and approximate computing, HAL Archives, HAL Id:hal-02517784, 2020. 13 p.
5. Kupriianova O. Towards a modern floating-point environment : thesis ... PhD. Université Pierre et Marie Curie, Paris, France, 2015. 140 p.
6. Blinn J. Floating-point tricks. *IEEE Computer Graphics and Applications*. 1997. Vol. 17, No 4. P. 80–84.
7. Horyachyy O., Moroz L., Otenko V. Simple effective fast inverse square root algorithm with two magic constants. *International Journal of Computing*. 2019. Vol. 18, No 4. P. 461–470.
8. Moroz L. V., Samotyy V. V., Horyachyy O. Y. Modified fast inverse square root and square root approximation algorithms: the method of switching magic constants. *Computation*. 2021. Vol. 9, Issue 2, 21. 23 p.
9. Chen W., Gao Y. A minutiae-based fingerprint matching algorithm using phase correlation. *Digital Image Computing Techniques and Applications (DICTA 2007)* : proc. 9th biennial conf. aust. pattern recognit. soc., Glenelg, SA, Australia, December 3-5, 2007. P. 233–238.
10. Борецький Т. Р. Розробка та реалізація методів обчислення елементарних функцій на основі програмних та апаратних засобів : дис. ... канд. техн. наук : 05.13.05. НУ «Львівська політехніка», Львів, 2019. 232 с.
11. Eberly D. H. Function approximations. *GPGPU programming for games and science* : book. Boca Raton, FL, USA : CRC Press, 2015. P. 107–122.

12. Du W., Zhan Z. A practical approach to solve secure multi-party computation problems. *New security paradigms* : proc. 2002 workshop, Virginia Beach, VA, USA, September 23-26, 2002. P. 127–135.
13. Du W. A study of several specific secure two-party computation problems : thesis ... PhD. Purdue University, USA, 2001. 171 p.
14. Carlile B., Delamarter G., Kinney P., Marti A., Whitney B. Improving deep learning by inverse square root linear units (ISRLUs), ArXiv Preprint, ArXiv: 1710.09967, 2018. 8 p.
15. Максимович В., Шабатура М., Горячий О., Лужецька Н. Генератори псевдовипадкових бітових послідовностей на основі чисельних методів в арифметиці з рухомою комою. *Сучасна спеціальна техніка*. 2021. № 1(64). С. 81–92.
16. Marsaglia G. On the randomness of pi and other decimal expansions. *InterStat: statistics on the Internet*. 2005. 17 p. URL: <http://interstat.statjournals.net/YEAR/2005/articles/0510005.pdf> (last accessed: 15.05.2021).
17. Lefèvre V., Muller J. M. Worst cases for correct rounding of the elementary functions in double precision. *Computer arithmetic* : proc. 15th IEEE symp. (ARITH'15), Vail, CO, USA, June 11-13, 2001. P. 111–118.
18. Iordache C., Matula D. W. On infinitely precise rounding for division, square root, reciprocal and square root reciprocal. *Computer arithmetic* : proc. 14th IEEE symp. (ARITH 1999), Adelaide, Australia, April 14-16, 1999. P. 233–240.
19. Ziv A. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Transactions on Mathematical Software*. 1991. Vol. 17, No 3. P. 410–423.
20. Thakkar S., Huff T. The internet streaming SIMD extensions. *Intel Technology Journal*. 1999. Issue Q2. 8 p.
21. Daramy-Loirat C., Defour D., de Dinechin F., Gallet M., Gast N., Lauter C., Muller J.-M. CR-LIBM A library of correctly rounded elementary functions in double-precision : research report, LIP, 2006. 163 p.

22. Fousse L., Hanrot G., Lefèvre V., Pélissier P., Zimmermann P. MPFR: A multiple-precision binary floating-point library with correct rounding : research report, INRIA, 2005. 15 p.
23. Robertson M. A brief history of InvSqrt : thesis ... Bachelor. University of New Brunswick, Canada, 2012. 48 p.
24. Hsu C., Chen J., Chen L. An efficient hardware implementation of HON4D feature extraction for real-time action recognition. *Consumer electronics : proc. 19th IEEE int. symp. (ISCE 2015)*, Madrid, Spain, June 24-26, 2015. 2 p.
25. Dongarra J., Moler C., Bunch J., Stewart G. LINPACK : users' guide. Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 1979. 344 p.
26. Garriss M., Watson C., McCabe R., Wilson C. User's guide to NIST fingerprint image software (NFIS) : technical report, NISTIR 6813, National Institute of Standards and Technology, 2001. 192 p. URL: <https://www.nist.gov/publications/users-guide-nist-fingerprint-image-software-nfis> (last accessed: 15.06.2021).
27. Гнатюк С., Хохлачова Ю., Охріменко А., Гребенькова А. Теоретичні основи побудови та функціонування систем управління інцидентами інформаційної безпеки. *Захист інформації*. 2012. Том 14, № 1(54). 6 с.
28. ДСТУ ISO/IEC 27001:2015. Інформаційні технології. Методи захисту системи управління інформаційною безпекою. [На заміну ДСТУ ISO/IEC 27001:2010; надано чинності від 18 грудня 2015 р.]. ДП «УкрНДНЦ», 2016. 28 с.
29. Cichonski P., Millar T., Grance T., Scarfone K. NIST SP 800-61 Rev. 2. Computer security incident handling guide: recommendations of the National Institute of Standards and Technology, U.S. Department of Commerce, 2012. URL: <https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-61r2.pdf> (last accessed: 05.07.2021).
30. Horyachyy I., Horyachyy O., Boretsky T. Information security management of a banking institution based on outsourcing. *Information protection and information*

- systems security* : materials 8th int. scient. techn. conf., Lviv, Ukraine, November 11-12, 2021. P. 11–12.
31. Horyachyy O. Comparison of wireless communication technologies used in a smart home: analysis of wireless sensor node based on Arduino in home automation scenario : thesis ... Master. Blekinge Institute of Technology, Sweden, 2017. 73 p.
 32. Jun S., Szmajda M., Khoma V., Khoma Y., Sabodashko D., Kochan O., Wang J. Comparison of methods for correcting outliers in ECG-based biometric identification. *Metrology and Measurement Systems*. 2020. Vol. 27, No 3. P. 387–398.
 33. Horyachyy O. Improving the efficiency of methods for calculating elementary functions in floating-point arithmetic. *Informatics education and educational software engineering research* : int. doc. cons., Druskininkai, Lithuania, December 2-6, 2019. P. 2–7.
 34. Shyshko D., Horyachyy O. Overview of main cybersecurity challenges of smart cities and methods of their protection. *Information protection and information systems security* : materials 8th int. scient. techn. conf., Lviv, Ukraine, November 11-12, 2021. P. 31–32.
 35. Mowery K., Wei M., Kohlbrenner D., Shacham H., Swanson S. Welcome to the entropics: boot-time entropy in embedded devices. *Security and Privacy* : proc. 2013 IEEE symp. (SP 2013), Berkeley, CA, USA, May 19-22, 2013. P. 1–15.
 36. Kietzmann P., Schmidt T., Wahlisch M. A guideline on pseudorandom number generation (PRNG) in the IoT. *ACM Computing Surveys*. 2021. Vol. 54, No 6. P. 1–38.
 37. Совин Я., Наконечний Ю., Чінка В., Тишик І. Тестування вбудованого генератора випадкових чисел мікроконтролерів родини STM32F4XX згідно з методикою NIST STS. *Вісник НУ «Львівська політехніка». Комп'ютерні системи та мережі*. 2012. № 745. С. 168–175.
 38. Horyachyy O., Boretsky T., Horiachyi I. Improving the methods of protection and hacking of modern Wi-Fi networks. *Information protection and information*

- systems security* : materials 7th int. scient. techn. conf., Lviv, Ukraine, May 30-31, 2019. P. 60–61.
39. IEEE 754-2019. Standard for floating-point arithmetic. [Revision of IEEE Std. 754-2008; active from 2019-07-22]. IEEE, 2019. 84 p.
 40. Loosemore S., Stallman R. M., McGrath R., Oram A., Drepper U. Known maximum errors in math functions. *The GNU C library* : reference manual, Free Software Foundation Inc., 2020. P. 556 –586.
 41. Lefèvre V. Test of mathematical functions of the standard C library, Vinc17.net, Inria, 2009. URL: <https://www.vinc17.net/research/testlibm/> (last accessed: 07.06.2024).
 42. Fog A. Instruction tables: lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs, Technical University of Denmark, 2019. 383 p. URL: https://www.agner.org/optimize/instruction_tables.pdf (last accessed: 13.02.2025).
 43. Nikmehr H. Architectures for floating-point division : thesis ... PhD. Adelaide, Australia, 2005. 258 p.
 44. ARM NEON intrinsics reference, ARM Ltd., Cambridge, England, IHI 0073B, 2016. 348 p.
 45. Intel 64 and IA-32 architectures software developer's manual. Combined volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4, Intel Corp., 325462-071US, 2019. 5038 p.
 46. Yi J. J., Joshi A., Sendag R., Eeckhout L., Lilja D. J. Analyzing the processor bottlenecks in SPEC CPU 2000. *SPEC benchmark* : proc. 2006 workshop, Austin, TX, USA, January 23, 2006.
 47. Xtensa instruction set architecture (ISA) : reference manual, Tensilica Inc., Santa Clara, CA, USA, PD-09-0801-10-01, 2010. P. 72 –74.
 48. Pareek V., Rathi A., Sharma D. Pseudo Random Number Generation. London : LAP LAMBERT Academic Publishing, 2013. 128 p.

49. Гарасимчук О., Максимович В. Генератори псевдовипадкових чисел, їх застосування, класифікація, основні методи побудови і оцінка якості. *Захист інформації*. 2003. Том 5, Вип. 16, № 3. С. 29–36.
50. Intel® Digital Random Number Generator (DRNG) : software implementation guide, Rev. 2.1. Intel Corp., 2018, 36 p. URL: <https://www.intel.com/content/dam/develop/external/us/en/documents/drng-software-implementation-guide-2-1-185467.pdf> (last accessed: 04.06.2024).
51. Bakiri M. Hardware implementation of a pseudo random number generator based on chaotic iteration : thesis ... PhD. Université Bourgogne Franche-Comté, France, 2018. 142 p.
52. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C. 2nd ed. New York : John Wiley & Sons Inc, 1996. 758 p.
53. Matsumoto M., Saito M., Haramoto H. Pseudorandom number generation: impossibility and compromise. *Journal of Universal Computer Science*. 2006. Vol. 12, No 6. P. 672–690.
54. Dodis Y., Pointcheval D., Ruhault S., Vergnaud D., Wichs D. Security analysis of pseudo-random number generators with input: /dev/random is not robust. *Computer & communications security : proc. 2013 ACM SIGSAC conf. (CCS'13)*, Berlin, Germany, November 4-8, 2013. P. 647–658.
55. O'Neill M. E. PCG: A family of simple fast space-efficient statistically good algorithms for random number generation : technical report, HMC-CS-2014-0905, Harvey Mudd College, 2014. 56 p. URL: <https://www.pcg-random.org/pdf/hmc-cs-2014-0905.pdf> (last accessed: 15.02.2025).
56. Wang L., Cheng H. Pseudo-random number generator based on logistic chaotic system. *Entropy*. 2019. Vol. 21, Issue 10, 960. 12 p.
57. Kordov K., Zhelezov S. Steganography in color images with random order of pixel selection and encrypted text message embedding. *PeerJ Computer Science*. 2021. Vol. 7, No 11. e380. 21 p.

58. Datcu O., Macovei C., Hobincu R. Chaos based cryptographic pseudo-random number generator template with dynamic state change. *Applied Sciences*. 2020. Vol. 10, No 2. 451. 17 p.
59. Pelosi M., Kessler G., Brown M. One-time pad encryption steganography system. *Digital Forensics, Security and Law : proc. 2016 ADFSL conf. (CDFSL 2016)*, Daytona Beach, FL, USA, May 24-26, 2016. P. 130–154.
60. Rani S., Kurniawardhani A., Rendani Y. Steganography on digital color image using modulo function and pseudo-random number generator. *International Journal on Advanced Science Engineering and Information Technology*. 2021. Vol. 11, No 6. P. 2470–2475.
61. Demchik V. Pseudo-random number generators for Monte Carlo simulations on Graphics Processing Units, ArXiv Preprint, ArXiv: 1003.1898v1, 2010. 31 p.
62. Горицький В., Снежок О., Височіненко М. Генерація випадкових послідовностей для систем управління ключами. *Сучасний захист інформації*. 2012. № 4. С. 88–95.
63. Checkoway S., Maskiewicz J., Garman C., Fried J., Cohnsey S., Green M., Heninger N., Weinmann R.-P., Rescorla E., Shacham H. A systematic analysis of the Juniper Dual EC incident. *Computer and communications security : proc. 2016 ACM SIGSAC conf. (CCS 2016)*, Vienna, Austria, October 24-28, 2016. P. 468–479.
64. Cohnsey S., Green M., Heninger N. Practical state recovery attacks against legacy RNG implementations. *Computer and communications security : proc. 2018 ACM SIGSAC conf. (CCS 2018)*, Toronto, Canada, October 15-19, 2018. P. 265–280.
65. Strenzke F. An analysis of OpenSSL's random number generator / M. Fischlin, J. Coron. *Advances in cryptology. EUROCRYPT 2016. Lecture Notes in Computer Science : book series*. Berlin : Springer, 2016. Vol. 9665. P. 644–669.
66. Shrestha B. Multiprime Blum-Blum-Shub pseudorandom number generator : thesis ... Master. Naval Postgraduate School, USA, 2016. 83 p.

67. Finamore T. Shamir's secret sharing scheme using floating point arithmetic : thesis ... Master. Florida Atlantic University, USA, 2012. 44 p.
68. Kamm L., Willemson J. Secure floating point arithmetic and private satellite collision analysis. *International Journal of Information Security*. 2015. Vol. 14, Issue 6. P. 531–548.
69. Shen E., Varia M., Cunningham R., Vesey W. Cryptographically secure computation. *Computer*. 2015. Vol. 48, No 4. P. 78–81.
70. Bogdanov D. Sharemind: programmable secure computations with practical applications : thesis ... PhD. University of Tartu, Estonia, 2013. 138 p.
71. Smart N. P. Secure MPC based on secret sharing. Homes.esat.kuleuven.be: FHE-MPC course materials, KU Leuven, 2018. URL: <https://web.archive.org/web/20230525043701/https://homes.esat.kuleuven.be/~nsmart/FHE-MPC/LSSS-MPC.pdf> (last accessed: 13.02.2025).
72. Krips T. Improving performance of secure real-number operations : thesis ... PhD. University of Tartu, Estonia, 2019. 146 p.
73. Guo C., Hannun A., Knott B., van der Maaten L., Tygert M., Zhu R. Secure multiparty computations in floating-point arithmetic, ArXiv Preprint, ArXiv: 2001.03192, 2020. 31 p.
74. Liedel M. Secure distributed computation of the square root and applications. *Information Security Practice and Experience (ISPEC 2012)* : proc. 8th int. conf., Hangzhou, China, April 9-12, Vol. 7232, 2012. P. 277–288.
75. Орленко В., Чешун В., Андрощук О., Катаєва А. Модель генератора криптоключів з джерелами ентропії для системи клієнт-банк. *Вимірювальна та обчислювальна техніка в технологічних процесах*. 2020. № 2. С. 103–111.
76. Mauricio G., Celso C. Greedy randomized adaptive search procedures: advances, hybridizations, and applications. *Handbook of Metaheuristics* : book series. New York : Springer, 2010. P. 283–319.
77. Хома Ю., Хома В., Сабодашко Д., Су Юн, Кочан О. Аналіз ефективності методів коригування промахів у системах біометричної ідентифікації на

- підставі електрокардіограми. *Науковий вісник НЛТУ України*. 2020. Том 30, № 3. С. 99–105.
78. NIST SP 800-22 Rev. 1a. A statistical test suite for random and pseudorandom number generators for cryptographic applications, 2010. URL: <http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf> (last accessed: 01.03.2021).
 79. Zhao Y., Gao C., Liu J., Dong S. A self-perturbed pseudo-random sequence generator based on hyperchaos. *Chaos, solitons & fractals: X*. 2019. Vol. 4. 100023.
 80. Moroz L., Samoty V., Horyachyy O. An effective floating-point reciprocal. *Wireless systems : proc. 2018 4th IEEE int. symp. within the int. conf. on intelligent data acquisition and advanced computing systems (IDAACS-SWS 2018)*, Lviv, September 20-21, 2018. P. 137–141.
 81. Hrynchyshyn A., Horyachyy O., Tymoshenko O., Moroz L. An efficient algorithm for fast inverse square root. *Processing, transmission and security of information : monograph*. Bielsko-Biala, Poland, 2018. Vol. 2. P. 105–114.
 82. Moroz L., Samoty V., Horyachyy O., Dzelendzyak U. Algorithms for calculating the square root and inverse square root based on the second-order Householder's method. *Intelligent data acquisition and advanced computing systems: technology and applications : proc. 2019 10th IEEE int. conf. (IDAACS)*, Metz, France, September 18-21, 2019. Vol. 1. P. 436–442.
 83. Мороз Л., Гринчишин А., Горячий О. Проста модифікація алгоритму швидкого обчислення зворотного квадратного кореня для чисел з рухомою комою одинарної точності. *Автоматика, вимірювання та керування*. 2019. Вип. 1, № 1. С. 39–45.
 84. Горячий О., Яремчук З. Аналіз ефективності генераторів псевдовипадкових чисел на основі квадратного кореня простого числа. *Інформаційна безпека та інформаційні технології : зб. тез доп. V міжнар. наук.-практ. конф. (ІБІТ-2024)*, Львів, 27 листопада, 2024. С. 234–237.

85. Горячий О., Максимович В., Шабатура М. Дослідження множини початкових значень генераторів псевдовипадкових чисел на основі арифметики з рухомою комою. *Сучасний захист інформації*. 2024. № 2(58). С. 91–102.
86. Горячий О., Журавель І. Дослідження ефективності простих стеганографічних методів обробки цифрових зображень із використанням генераторів псевдовипадкових послідовностей. *Інформаційна безпека та інформаційні технології* : зб. тез доп. V міжнар. наук.-практ. конф. (ІБІТ-2024), Львів, 27 листопада, 2024. С. 225–230.
87. Горячий О., Журавель І. Порівняння простих методів стеганографічного захисту інформації на основі аналізу цифрових зображень та використання псевдовипадкових послідовностей. *Автоматизація, електроніка, інформаційно-вимірювальні технології: освіта, наука, практика* : матер. V міжнар. наук.-техн. конф., Харків, 28–29 листопада, 2024. С. 27–28.
88. Brett M. Floating point error, Matthew Brett, 2016. URL: https://matthew-brett.github.io/teaching/floating_error.html (last accessed: 07.04.2020).
89. Yashkardin V. IEEE 754 – Standard binary arithmetic float, 2011. URL: https://web.archive.org/web/20241006173834/http://www.softelectro.ru/teoriy_en.html (last accessed: 24.02.2025).
90. Goldberg, D. What every computer scientist should know about floating-point arithmetic. *ACM computing surveys (CSUR)*. 1991. Vol. 23, Issue 1. P. 5–48.
91. Monniaux D. The pitfalls of verifying floating-point computations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*. 2008. Vol. 30, Issue 3. P. 1–41.
92. Sedgewick R., Wayne K. Floating point. *Introcs.cs.princeton.edu: Programming in Java, Computer Science, An Interdisciplinary Approach*; Princeton University. 2014. URL: <https://introcs.cs.princeton.edu/java/91float/> (last accessed: 08.07.2021).
93. Arnold J. Floating-point arithmetic. *Numerical computing* : CERN openlab/Intel workshop materials (Meyrin, 7-8 Feb. 2012). Meyrin, Switzerland, 2012. URL:

- <https://indico.cern.ch/event/166141/sessions/125684/attachments/201414/282782/Arnold-FPWorkshop-Print.pdf> (last accessed: 09.04.2020).
94. McIntosh F. A methodology for producing identical double precision floating-point results. *BE-KT innovation day : materials* (Meyrin, 3 June 2013). Meyrin, Switzerland, 2013.
 95. Cyganek B. How accurate we are? A refresher on the floating-point arithmetic and the standard library. *Meeting C++ 2019 : conf. materials* (Berlin, 14-16 Nov. 2019). Berlin, Germany, 2019. URL: http://home.agh.edu.pl/~cyganek/BC_Talk_CppMeeting_Berlin_2019_v11.pdf (last accessed: 12.04.2020).
 96. ARM developer suite : compilers and libraries guide, ARM Ltd., ARM DUI 0067D, 2001. 276 p.
 97. AMD math library (LibM), Advanced Micro Devices, Inc., 2020. URL: <https://developer.amd.com/amd-aocl/amd-math-library-libm/> (last accessed: 14.04.2020).
 98. Intel Math Kernel Library : developer reference, Intel Corp., Revision 27, 2020. 2666 p.
 99. The GNU MPFR library. URL: <https://www.mpfr.org/> (last accessed: 13.04.2020).
 100. Lefèvre V. Moyens arithmétiques pour un calcul fiable : thesis ... PhD. École Normale Supérieure de Lyon, Lyon, France, 2000.
 101. CR-libm : math. library source code, 2016. URL: <https://github.com/taschini/crlibm> (last accessed: 14.04.2020).
 102. IBM Accurate portable Mathlib : math. library source code, 2001. URL: <https://github.com/dreal-deps/mathlib> (last accessed: 13.04.2020).
 103. Sun libmcr : math. library source code, 2004. URL: <https://github.com/simonbyrne/libmcr> (last accessed: 13.04.2020).
 104. Intel C++ compiler 19.1 : developer guide and reference, Intel Corp. 2341 p.
 105. Math and floating-point support, Microsoft Corp., 2019. URL: <https://docs.microsoft.com/en-us/cpp/c-runtime-library/floating-point-support?view=vs-2019> (last accessed: 14.04.2020).

106. GoFast floating point library, Micro Digital Inc., 2017. URL: <http://www.smxrtos.com/ussw/gofast.htm> (last accessed: 14.04.2020).
107. Chamberlain S., Pesch R., Johnston J. The Red Hat newlib C math library, Red Hat Support, Red Hat, Inc., 2010. 100 p.
108. Raina S. K. FLIP: a floating-point library for integer processors : thesis ... PhD. École Normale Supérieure de Lyon, Lyon, France, 2006.
109. SEGGER floating-point library, SEGGER Microcontroller GmbH. URL: <https://www.segger.com/products/development-tools/runtime-library/technology/floating-point-library/> (last accessed: 24.03.2025).
110. Qfplib: an ARM Cortex-M0 floating-point library in 1 kbyte, Quinapalus.com. URL: <https://www.quinapalus.com/qfplib.html> (last accessed: 14.04.2020).
111. 3DNow! technology manual, Advanced Micro Devices, Inc., 21928G/0, 2000. 62 p.
112. X86 and AMD64 instruction reference, Felixcloutier.com, 2018. URL: <http://www.felixcloutier.com/x86/index.html> (last accessed: 24.03.2025).
113. AltiVec technology programming environments manual, Freescale Semiconductor, Inc., ALTIVECPEM, Rev. 3, 2006. 346 p.
114. Muller J.-M. Elementary functions: algorithms and implementation. 2nd ed. Basel, Switzerland : Birkhäuser, 2006. 283 p.
115. Muller J.-M., Brunie N., de Dinechin F., Jeannerod C.-P., Joldes M., Lefèvre V., Melquiond G., Revol N., Torres S. Handbook of floating-point arithmetic. 2nd ed. Basel, Switzerland : Birkhäuser, 2018. P. 375–433.
116. Cornea-Hasegan M. A., Golliver R. A., Markstein P. Correctness proofs outline for Newton-Raphson based floating-point divide and square root algorithms. *Computer Arithmetic* : proc. IEEE symp., Adelaide, Australia, April 14-16, 1999. P. 96–105.
117. Markstein P. Software division and square root using Goldschmidt's algorithms. *Real Numbers and Computers (RNC'6)* : proc. 6th conf., Wadern, Germany, November 15-17, Vol. 123, 2004. P. 146–157.

118. Allie M., Lyons R. A root of less evil [DSP tips and tricks]. *IEEE Signal Processing Magazine*. 2005. Vol. 22. P. 93–96.
119. McEniry C. The mathematics behind the fast inverse square root function code : technical report, 2007. 16 p.
120. Lomont C. Fast inverse square root : technical report, Purdue University, 2003. 12 p. URL: <http://www.lomont.org/papers/2003/InvSqrt.pdf> (last accessed: 28.04.2020).
121. Langhammer M., Pasca B. Elementary function implementation with optimized sub range polynomial evaluation. *Field-programmable custom computing machines* : proc. 21st IEEE annual int. symp. (FCCM), Seattle, WA, USA, April 28-30, 2013. P. 202–205.
122. Detrey J., de Dinechin F. Table-based polynomials for fast hardware function evaluation. *Application-specific systems, architecture processors* : proc. 2005 IEEE int. conf. (ASAP'05), Samos, Greece, July 23-25, 2005. P. 328–333.
123. Harrison J., Kubaska T., Story S., Tang P. T. P. The computation of transcendental functions on the IA-64 architecture. *Intel Technology Journal*. 1999. Issue Q4. 7 p.
124. Апроксимація функцій / за заг. ред. М. М. Москальков. *Чисельні методи* : конспект лекцій. Київ : КНУ ім. Т. Шевченка. 22 с.
125. Лисікова К. О. Вдосконалення методу апроксимації даних вимірювань : магістерська дис. : 151. Київ, 2018. 138 с.
126. Tasissa A. Function approximation and the Remez algorithm. Sites.tufts.edu, Tufts University. 2019. 12 p. URL: <https://sites.tufts.edu/atasissa/files/2019/09/remez.pdf> (last accessed: 24.02.2025).
127. de Groot E. Finding best minimax approximations with the Remez algorithm : thesis ... Bachelor. University of Groningen, Netherlands, 2017. 49 p.
128. Chevillard S., Joldeş M., Lauter C. Sollya: an environment for the development of numerical codes. *Mathematical Software*. Heidelberg : Springer, 2010. P. 28–31. URL: <https://www.sollya.org/> (last accessed: 13.06.2024).

129. Kahan W. Computing a real cube root, 1991. 3 p. URL: <https://csclub.uwaterloo.ca/~pbarfuss/qbrt.pdf> (last accessed: 23.02.2025).
130. Sasao T., Butler T. Numerical function generators using LUT cascades. *IEEE Transactions on Computers*. 2007. Vol. 56, No 6. P. 826–838.
131. Iordache C., Matula D. W. Analysis of reciprocal and square root reciprocal instructions in the AMD K6-2 implementation of 3DNow! *Electronic Notes in Theoretical Computer Science*. 2000. Vol. 25. P. 34–62.
132. Lynch T., Ahmed A., Schulte M., Callaway T., Tisdale R. The K5 transcendental functions. *Computer arithmetic : proc. 12th IEEE symp. (ARITH)*, Bath, UK, July 19-21, 1995. P. 222–229.
133. Sarma D., Matula D. Faithful bipartite ROM reciprocal tables. *Computer arithmetic : proc. 12th IEEE symp. (ARITH)*, Bath, UK, July 19-21, 1995. P. 17–28.
134. Muller J.-M. A few results on table-based methods. *Reliable Computing*. 1999. Vol. 5. P. 279–288.
135. Stine J. E., Schulte M. J. The symmetric table addition method for accurate function approximation. *Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology*. 1999. Vol. 21. P. 167–177.
136. Detrey J., de Dinechin F. Second order function approximation using a single multiplication on FPGAs / J. Becker, M. Platzner, S. Vernalde. *Field programmable logic and application. FPL 2004. Lecture Notes in Computer Science : book series*. Berlin : Springer, 2004. Vol. 3203. P. 221–230.
137. Quake III Arena, Id Software, 1999. URL: https://github.com/id-Software/Quake-III-Arena/blob/master/code/game/q_math.c (last accessed: 28.04.2020).
138. Moroz L., Walczyk C. J., Hrynychshyn A., Holimath V., Cieśliński J. L. Fast calculation of inverse square root with the use of magic constant – analytical approach. *Applied Mathematics and Computation*. 2018. Vol. 316. P. 245–255.
139. Lin J., Xu Z., Nukada A., Maruyama N., Matsuoka S. Optimizations of two compute-bound scientific kernels on the SW26010 many-core processor. *Parallel*

- processing* : proc. 46th int. conf. (ICPP-2017), Bristol, UK, August 14-17, 2017. P. 432–441.
140. Walczyk C., Moroz L., Cieśliński J. Improving the accuracy of the fast inverse square root algorithm, ArXiv Preprint, ArXiv:1802.06302, 2018. 21 p.
 141. Martin P. Eight rooty pieces. *Overload Journal*. 2016. Vol. 135. P. 8–12.
 142. Hasnat A., Bhattacharyya T., Dey A., Halder S., Bhattacharjee D. A fast FPGA based architecture for computation of square root and inverse square root. *Devices for integrated circuit* : proc. 2nd int. conf. (DevIC), Kalyani, India, March 23-24, 2017. P. 383–387.
 143. Li Z., Chen Y., Zeng X. OFDM synchronization implementation based on Chisel platform for 5G research. *ASIC* : proc. 2015 IEEE 11th int. conf. (ASICON), Chengdu, China, November 3-6, 2015. 4 p.
 144. Zafar S., Adapa R. Hardware architecture design and mapping of ‘Fast Inverse Square Root’ algorithm. *Advances in electrical engineering* : proc. 2014 int. conf. (ICAEE), Vellore, India, January 9-11, 2014. 4 p.
 145. Мороз Л., Гринчишин А. Швидке обчислення оберненого квадратного кореня з використанням магичної константи – аналітичний підхід. *Комп’ютерні технології друкарства*. 2014. №. 2 (32). С. 38–50.
 146. Мороз Л. Теорія та швидкодіючі апаратно-програмні засоби ітераційних методів обчислення функцій : автореф. дис. ... докт. техн. наук : 05.13.05. Національний університет «Львівська політехніка», Львів, 2013. 35 с.
 147. Walczyk C., Moroz L., Cieśliński J. A modification of the fast inverse square root algorithm. *Computation*. 2019. Vol. 7, Issue 3, 41. 14 p.
 148. Walczyk C., Moroz L., Cieśliński J. Improving the accuracy of the fast inverse square root by modifying Newton–Raphson corrections. *Entropy*. 2021. Vol. 23, Issue 1, 86. 20 p.
 149. Kadlec J. Improving the fast inverse square root, *Řřřlog*, 2010. URL: https://web.archive.org/web/20241113024543/http://rrrola.wz.cz/inv_sqrt.html (last accessed: 13.11.2024).

150. Householder A. The numerical treatment of a single nonlinear equation : book. New York, NY, USA : McGraw-Hill, 1970. 224 p.
151. Sebah P, Gourdon X. Newton's method and high order iterations : technical report, Numbers, constants and computation, 2001. 10 p.
152. Мороз Л., Гринчишин А. Швидке обчислення функції $y=1/x$ з використанням магичної константи. *Вісник НУ «Львівська політехніка». Автоматика, вимірювання та керування*. 2015. №. 821. С. 23–29.
153. Khuong P. 0x7fde623822fc16e6: a magic constant for double float reciprocal, Paul Khuong mostly on Lisp, 2011. URL: <https://pvk.ca/Blog/LowLevel/software-reciprocal.html> (last accessed: 23.02.2025).
154. Huang K., Chen Y. Improving performance of floating point division on GPU and MIC. *Algorithms and Architectures for Parallel Processing (ICA3PP 2015)* : proc. 15th int. conf., Zhangjiajie, China, November 18-20, 2015. P. 691–703.
155. Kahan W. Computing a real cube root : technical report, 1991. 3 p. URL: <https://csclub.uwaterloo.ca/~pbarfuss/qbrt.pdf> (last accessed: 23.02.2025).
156. Levin S. Two point raytracing for reflection off a 3D plane : technical report, Stanford Exploration Project, 2012. 14 p. URL: <http://sepwww.stanford.edu/data/media/public/oldsep/stew/ForDaveHale/sep148stew1.pdf> (last accessed: 23.02.2025).
157. Moroz L., Samoty V., Walczyk C., Cieśliński J. Fast calculation of cube and inverse cube roots using a magic constant and its implementation on microcontrollers. *Energies*. 2021. Vol. 14, Issue 4, 1058. 13 p.
158. Turkowski K. Computing the cube root : technical report, Apple Computer, 1998. 2 p.
159. Sýs M., Ríha Z., Matyás V., Marton K., Suci A. On the interpretation of results from the NIST statistical test suite. *Romanian Journal of Information Science and Technology*. 2015. Vol. 18, No 1, P. 18–32.
160. Kukharska N., Lagun A., Yashchuk O. Crypto-steganographic system based on the solver of the square root of a prime number. *Advanced computer information*

- technologies* : proc. 2024 14th int. conf. (ACIT), Ceske Budejovice, Czech Republic, September 19-21, 2024. P. 535–538.
161. Bailey D. H., Borwein J. Pi day is upon us again and we still do not know if pi is normal. *The American Mathematical Monthly*. 2014. Vol. 121, Issue 3, P. 191–206.
 162. Harase S., Kimoto T. Implementing 64-bit maximally equidistributed F_2 -linear generators with Mersenne prime period. *ACM Transactions on Mathematical Software*. 2018. Vol. 44, Issue 3, 30. 11 p.
 163. Matsumoto M., Nishimura T. Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *Association for Computing Machinery*. 1998. Vol. 8, Issue 1. P. 3–30.
 164. Blum L., Blum M., Shub M. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*. 1986. Vol. 15, Issue 2. P. 364–383.
 165. Goualard F. Generating random floating-point numbers by dividing integers: a case study. *Computational Science (ICCS 2020)* : proc. int. conf., Amsterdam, Netherlands, June 3-5, 2020. P. 15–28.
 166. Doornik J. Conversion of high-period random numbers to floating point. *ACM Transactions on Modeling and Computer Simulation*. 2007. Vol. 17, Issue 1, 3. 4 p.
 167. Doganaksoy A., Ege B., Koçak O., Sulak F. Cryptographic randomness testing of block ciphers and hash functions, IACR Cryptology ePrint Archive, IA.CR: 2010/564, 2010. 12 p.
 168. Obrátil L. The automated testing of randomness with multiple statistical batteries : thesis ... Master. Masaryk University, Czech Republic, 2017. 62 p.
 169. Liang L. Testing several types of random number generators : thesis ... Master. Florida State University, USA, 2012. 98 p.
 170. Семаньків М. Статистичне дослідження генераторів випадкових чисел. *Інформаційні технології та комп'ютерне моделювання* : матеріали конф. (ITCM-2016), Івано-Франківськ, 23-28 травня, 2016. С. 85–86.

171. L'ecuyer P., Simard R. TestU01: a C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)*. 2007. Vol. 33, Issue 4, 22. 40 p.
172. NIST FIPS 140-2: Security requirements for cryptographic modules, National Institute of Standards and Technology, 2001. URL: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf> (last accessed: 23.02.2025).
173. Knuth D. The art of computer programming, Vol. 2, Seminumerical Algorithms. 3rd ed. Boston, MA, USA : Addison-Wesley, Reading, 1998. 784 p.
174. Killmann W., Schindler W. Functionality classes for random number generators: a proposal, BSI Bonn, AIS31; Bonn, Germany : The University of Bonn, 2011. 133 p.
175. Marsaglia G. Diehard battery of tests of randomness. URL: <https://web.archive.org/web/20120102192622/www.stat.fsu.edu/pub/diehard/> (last accessed: 14.06.2021).
176. Brown R. Dieharder: a random number test suite, Version 3.31.1, 2004. URL: <http://www.phy.duke.edu/~rgb/General/dieharder.php> (last accessed: 23.02.2025).
177. Guyeux C., Wang Q., Bahi J. A pseudo random numbers generator based on chaotic iterations: application to watermarking. *Web information systems and mining* : proc. int. conf. (WISM 2010), Sanya, China, October 23-24, 2010. P. 202–211.
178. Schulte M., Stine J., Wires K. High-speed reciprocal approximations. *Signals, Systems and Computers* : conf. rec. 31st asilomar conf., Pacific Grove, CA, USA, November 2-5, Vol. 2, 1997. P. 1183–1187.
179. The Remez method. Boost.org: Boost C++ libraries. 2017. URL: https://www.boost.org/doc/libs/1_68_0/libs/math/doc/html/math_toolkit/remez.html (last accessed: 24.02.2025).
180. Selesnick I. The Remez algorithm. Eeweb.engineering.nyu.edu: EL 713 Lecture Notes, NYU Tandon School of engineering. URL:

- <https://web.archive.org/web/20250312175727/https://eeweb.engineering.nyu.edu/iselesni/EL713/remez/remez.pdf> (last accessed: 12.03.2025).
181. Testa F. Floating point math functions : application note, Microchip Technology Inc., Chandler, AZ, USA, AN660, 1997. 244 p. URL: <http://ww1.microchip.com/downloads/en/appnotes/00660.pdf> (last accessed: 24.02.2025).
 182. Niu M., Nandyala S., Sohn W., Sanger T. Multi-scale hyper-time hardware emulation of human motor nervous system based on spiking neurons using FPGA. *Neural Information Processing Systems (NIPS 2012)* : proc. 25th int. conf., Lake Tahoe, NV, USA, December 3-8, 2012. P. 37–45.
 183. Мороз Л., Гринчишин А., Мірецька Я. Прості алгоритми ділення чисел з плаваючою комою. *Вісник НУ «Львівська політехніка». Автоматика, вимірювання та керування*. 2016. № 852. С. 35–38.
 184. Thuning N., Barring L. Hardware architectures for the inverse square root and the inverse functions using harmonized parabolic synthesis : thesis ... Master. Lund University, Sweden, 2016. 137 p.
 185. Abel A. Latency, throughput, and port usage information for instructions on recent x86 microarchitectures, Uops.info, Saarland University, 2020. URL: <https://uops.info/index.html> (last accessed: 24.02.2025).
 186. Кононюк А. Основы теории оптимизации. Безусловная оптимизация. К. 2., Ч. 1. Киев : Изд-во «Освіта України», 2011. 544 с.
 187. Чисельні методи оптимізації. *Математичні методи дослідження операцій* : підручник / Є. А. Лавров, Л. П. Перхун, В. В. Шендрик та ін. Р. 8. Суми : Сумський державний університет, 2017. С. 91–144.
 188. Yalcin A. Multidimensional direct search method. Numerical methods with applications : textbook notes. University of South Florida. Ch. 09.03. 5 p. URL: http://mathforcollege.com/nm/mws/gen/09opt/mws_gen_opt_txt_multidirect.pdf (last accessed: 24.03.2025).
 189. Alba E., Garcia-Nieto J., Jourdan L., Talbi El-G. Gene selection in cancer classification using PSO/SVM and GA/SVM hybrid algorithms. *Evolutionary*

- computation* : proc. 2007 IEEE congress (CEC 2007), Singapore, September 25-28, 2007. P. 284–290.
190. Sun Y., Ye Z., Wang C., Yan L., Wang R. Research on network intrusion detection based on support vector machine optimized with pigeon-inspired optimization algorithm. *Wireless systems* : proc. 2018 4th IEEE int. symp. within the int. conf. on intelligent data acquisition and advanced computing systems (IDAACS-SWS 2018), Lviv, September 20-21, 2018. P. 62–67.
 191. Elmasry W., Akbulut A., Zaim A. Evolving deep learning architectures for network intrusion detection using a double PSO metaheuristic. *Computer Networks*. 2020. Vol. 168, 107042. 21 p.
 192. Wichmann B. A note on the use of floating point in critical systems. *The Computer Journal*. 1992. Vol. 35, Issue 1. P. 41–44.
 193. Goldberg D. What every computer scientist should know about floating-point arithmetic. *ACM computing surveys (CSUR)*. 1991. Vol. 23, Issue 1. P. 5–48.
 194. Mironov I. On significance of the least significant bits for differential privacy. *Computer and Communications Security (CCS '12)* : proc. 2012 ACM conf., Raleigh, NC, USA, October 16-18, 2012. P. 650–661.
 195. Andryscio M., Kohlbrenner D., Mowery K., Jhala R., Lerner S., Shacham H. On subnormal floating point and abnormal timing. *Security and Privacy* : proc. 2015 IEEE symp., San Jose, CA, USA, May 18-20, 2015. P. 623–639.
 196. Rane A., Lin C., Tiwari M. Secure, precise, and fast floating-point operations on x86 processors. *Security Symposium (SEC'16)* : proc. 25th USENIX conf., Berkeley, CA, USA, August 10, 2016. P. 71–86.
 197. Regan R. Exploring binary: PHP hangs on numeric value 2.2250738585072011e-308. Exploringbinary.com. URL: <https://www.exploringbinary.com/php-hangs-on-numeric-value-2-2250738585072011e-308/> (last accessed: 24.03.2025).
 198. Marsaglia G., Tsang W. The 64-bit universal RNG. *Statistics & Probability Letters*. 2004. Vol. 66, Issue 2. P. 183–187.

199. Downey A. Generating pseudo-random floating-point values. Allendowney.com, 2007. 8 p. URL: <https://allendowney.com/research/rand/downey07randfloat.pdf> (last accessed: 24.03.2025).
200. Random Bit Generation. NIST SP 800-22: Download documentation and software. National Institute of Standards and Technology, 2014. URL: <https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software> (last accessed: 13.04.2021).
201. Beebe N. Programming using the MathCW portable software library : the mathematical-function computation handbook. 1st ed. New York, NY, USA : Springer International Publishing, 2017. P. 215–242.
202. Intel® Cyclone® 10 GX : device overview, Intel Corp., 2019, 22 p. URL: <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-10/c10gx-51001.pdf> (last accessed: 24.03.2025).
203. Dubchak L., Komar M., Sachenko A., Kochan V. Speedy procesing method of fuzzy data for intelligent systems of intrusion detection. *Processing, transmission and security of information* : monograph. Bielsko-Biała, Poland, 2017. Vol. 2. P. 65–74.
204. Es-sabry M., Akkad N., Khrissi L., Satori K., El-Shafai W., Altameem T., Rathore R. S. An efficient 32-bit color image encryption technique using multiple chaotic maps and advanced ciphers. *Egyptian Informatics Journal*. 2024. Vol. 25, 100449. 17 p. URL: <https://doi.org/10.1007/s11075-024-01932-7>.
205. Rasika B., Udayprakash S. A Review on Applications of Chaotic Maps in Pseudo-Random Number Generators and Encryption. *Annals of Data Science*. 2024. Vol. 11, No 1. P 25–50. URL: <https://doi.org/10.1007/s40745-021-00364-7>.
206. Kokosiński Z., Gepner P., Moroz L., Samoty V, Wegrzyn M., Gavkalova N. Fast and accurate approximation algorithms for computing floating point square root. *Numerical Algorithms*. 2024. URL: <https://doi.org/10.1007/s11075-024-01932-7>.
207. Vestermarck H. Fast Square Root and inverse calculation for Arbitrary Precision number, 2023. 29p. URL: <https://doi.org/10.13140/RG.2.2.23648.76802/2>.

208. İrtürk A. U. GUSTO: general architecture design utility and synthesis tool for optimization : thesis ... PhD. University of California, San Diego, USA, 2009.
209. Triantafyllou D. Numerical linear algebra methods in data encoding and decoding. *Journal of Applied Mathematics and Bioinformatics*. 2013. Vol. 3, No 1. P. 193–203.
210. Cescato D., Bölcskei, H. Interpolation-based QR decomposition in MIMO-OFDM systems, ArXiv Preprint, ArXiv: 0908.3463v3, 2009. 46 p.
211. Maheswari N., Revathi M. Data security using decomposition. *International Journal of Applied Science and Engineering*. 2014. Vol. 12, No 4. P. 303–312.
212. Li X., Niu J., Khan M. K., Wang Z. Applying LU decomposition of matrices to design anonymity bilateral remote user authentication scheme. *Mathematical Problems in Engineering*. 2013. Vol. 2013, 910409. 10 p.
213. Catrina O., Saxena A. Secure computation with fixed-point numbers. *Financial Cryptography and Data Security (FC'10)* : proc. 14th int. conf., Tenerife, Spain, January 25-28, Vol. 6052, 2010. P. 35–50.
214. Bogdanov D., Niitsoo M., Toft T., Willemson J. High-performance secure multiparty computation for data mining applications. *International Journal of Information Security*. 2012. Vol. 11, Issue 6. P. 403–418.
215. Aliasgari M., Blanton M., Zhang Y., Steele A. Secure computation on floating point numbers. NDSS. 2013. 31 p.
216. Krips T., Willemson J. Hybrid model of fixed and floating point numbers in secure multiparty computations. *Information Security (ISC 2014)* : proc. 17th int. conf., Hong Kong, China, October 12-14, 2014. P. 179–197.
217. Pullonen P., Siim S. Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations. *Financial Cryptography and Data Security (FC'15)* : proc. 19th int. conf., San Juan, Puerto Rico, January 26-30, Vol. 8976, 2015. P. 172–183.
218. Feigenbaum J., Ishai Y., Malkin T., Nissim K., Strauss M. J., Wright R. N. Secure multiparty computation of approximations. *ACM Transactions on Algorithms*. 2006. Vol. 2, Issue 3. P. 435–472.

219. Lindell Y., Pinkas B. Privacy preserving data mining. *Journal of Cryptology*. 2002. Vol. 15, Issue 3. P. 177–206.
220. Nguyen A., Pham K., Ngo D., Ngo T., Pham L. An analysis of state-of-the-art activation functions for supervised deep neural network. *System science and engineering : proc. int. conf. (ICSSE 2021)*, Ho Chi Minh City, Vietnam, August 26-28, 2021. P. 215–220.
221. Reyes N. H. Fuzzy logic. Massey.ac.nz: Artificial Intelligence Lecture Notes, Massey University. URL: <https://www.massey.ac.nz/~nhreyes/MASSEY/159741/Lectures/Lec2012-3-159741-FuzzyLogic-v.2.pdf> (last accessed: 26.02.2025).
222. Шевченко А., Застело Г., Шпачинський Є. Аналіз застосування методів машинного навчання на основі штучних нейронних мереж для виявлення кіберзагроз. *Information Technology and Security*. 2019. Vol. 7, Issue 1. P. 79–90.
223. Maithem M., Al-sultany G. A. Network intrusion detection system using deep neural networks. *Journal of Physics: Conference Series*. 2021. Vol. 1804, No 1, 012138. 12 p.
224. Alzahrani A. O., Alenazi M. J. Designing a network intrusion detection system based on machine learning for software defined networks. *Future Internet*. 2021. Vol. 13, Issue 5, 111. 18 p.
225. Мельник В. А. Прототип скалярного RISC-комп'ютера із заданою підмножиною системи інструкцій. Методичні вказівки до виконання курсової роботи з дисципліни "Архітектура комп'ютерних систем": для студентів спец. 125 "Кібербезпека та захист інформації". Львів : НУ "Львів. політехніка", 2022. 25 с.

ДОДАТКИ

Додаток А

Акти впровадження

А.1 У ПП НВПІ «Спаринг-Віст Центр» – під час розроблення захищеного програмного забезпечення для моделювання та симуляції роботи вимірювальних пристроїв.

А.2 На кафедрі безпеки інформаційних технологій Національного університету «Львівська політехніка» – при виконанні науково-дослідних робіт.

А.3 На кафедрі безпеки інформаційних технологій Національного університету «Львівська політехніка» – у навчальному процесі для підготовки студентів спеціальності 125 «Кібербезпека та захист інформації».

A.1



Вих. № 342/1 від 05.03.2025 р.

А К Т

про використання результатів дисертаційної роботи

Горячого Олега Ярославовича

«Розроблення генераторів псевдовипадкових чисел на основі покращених методів обчислення елементарних функцій для задач кібербезпеки»,
представленої на здобуття наукового ступеня доктора філософії за спеціальністю
125 «Кібербезпека та захист інформації»

Науково-виробниче приватне підприємство «Спаринг-Віст Центр» (м. Львів), відомий в Україні та світі розробник та виробник приладів і систем радіаційного контролю торгової марки «ЕКОТЕСТ», у відповідності до угоди про співпрацю між підприємством та кафедрою безпеки інформаційних технологій Національного університету «Львівська політехніка» отримало теоретичні матеріали та коди програм, що реалізують генерацію псевдовипадкових чисел та бітових послідовностей на основі розроблених покращених алгоритмів наближеного обчислення елементарних функцій в арифметиці з рухомою комою. Розроблені алгоритми та відповідні теоретичні напрацювання впроваджені на підприємстві під час реалізації високоточного та ефективного моделювання й симуляції функціонування швидкодіючих апаратно-програмних засобів, зокрема пристроїв вимірювання іонізуючого випромінювання, їх налагодження та тестування. Запропоновані прості та швидкі генератори псевдовипадкових чисел можуть використовуватися як безпечна альтернатива традиційним методам генерації псевдовипадкових послідовностей, зокрема на персональних комп'ютерах і мікроконтролерах, для формування пуассонівських імпульсних послідовностей, а також для захисту інформації. Результати дисертаційної роботи Горячого О.Я. інтегровані в спеціалізоване програмне забезпечення підприємства для імітації вхідних даних дозиметричних пристроїв, що володіють необхідними статистичними характеристиками. Це дозволило підвищити їхню швидкодію та безпеку при збереженні точності вимірювань у межах допустимих похибок.

Директор-генеральний конструктор
ПП «Спаринг-Віст Центр»,
лауреат державної премії України
в галузі науки і техніки, к.т.н.



Горячий Ю.Б.

ПП «НВП «Спаринг-Віст Центр»
Україна, 79026, м. Львів, вул. Володимира Великого, 33
тел.: (032) 2421515, факс: (032) 2422015
e-mail: market@ecotest.ua, www.ecotest.ua

ЗАТВЕРДЖУЮ

Проректор з наукової роботи

Національного університету

«Львівська політехніка»

Іван ДЕМИДОВ

03

2025 р.

**А К Т**

про впровадження результатів дисертаційної роботи
здобувача кафедри безпеки інформаційних технологій

Горячого Олега Ярославовича

**«Розроблення генераторів псевдовипадкових чисел на основі покращених
методів обчислення елементарних функцій для задач кібербезпеки»,
представленої на здобуття наукового ступеня доктора філософії
за спеціальністю 125 «Кібербезпека та захист інформації»**

Комісія у складі: голови – начальника науково-дослідної частини, д.т.н., ст. досл. Небесного Р. В. та членів: завідувача кафедри безпеки інформаційних технологій, д.т.н., с.н.с. Журавля І. М., завідувача відділу науково-організаційного супроводу наукових досліджень, к.т.н. Лазько Г. В. і в. о. заступника начальника планово-фінансового відділу Фаст І. І., цим актом підтверджують, що результати дисертаційних досліджень, проведені здобувачем Горячим О. Я., виконувалися на кафедрі безпеки інформаційних технологій у рамках пріоритетних напрямів розвитку науки і техніки України, зокрема, при виконанні науково-дослідних робіт за темами «Розробка та покращення ефективності алгоритмів обчислення елементарних функцій для задач захисту інформації» (№ державної реєстрації 0120U103215, термін виконання 2020-2021), «Розробка та дослідження генераторів псевдовипадкових чисел і послідовностей» (№ державної реєстрації 0113U005267, термін виконання 2017-2018) та «Розроблення стеганографічних методів приховування даних у цифрових зображеннях» (№ державної реєстрації 0120U100024, термін виконання 2019-2024).

Отримані автором наукові результати, зокрема, використовуються:

- при реалізації покращених швидких алгоритмів апроксимації елементарних функцій на основі методу магічної константи, які забезпечують високу точність для чисел у форматі з рухомою комою;
- при проєктуванні нового класу генераторів псевдовипадкових чисел, призначених для задач захисту інформації, на основі чисельних методів в арифметиці з рухомою комою та дослідженні їх статистичних характеристик із використанням тестів NIST STS;
- при розробці простих стеганографічних методів приховування та захисту конфіденційності й цілісності даних на основі заміни найменш значущих бітів цифрових зображень, які передбачають аналіз висококонтрастних і

контурних ділянок стеганоконтейнера, псевдовипадкового вибору пікселів та накладання псевдовипадкової бітової маски з метою мінімізації довжини стеганографічного ключа та підвищення стійкості до виявлення факту наявності вбудованих даних.

Голова комісії
начальник науково-дослідної
частини, д.т.н., ст. досл.



Роман НЕБЕСНИЙ

Члени комісії:
зав. кафедри безпеки інформаційних
технологій, д.т.н., с. н. с.



Ігор ЖУРАВЕЛЬ

зав. відділу науково-організаційного
супроводу наукових досліджень, к.т.н.



Галина ЛАЗЬКО

в. о. заст. нач. планово-фінансового
відділу



Ірина ФАСТ

ЗАТВЕРДЖУЮ

проректор з науково-педагогічної
роботи

Національного університету
«Львівська політехніка»

Олег ДАВИДЧАК

» 03 2025 р.

**АКТ**

про впровадження результатів дисертаційної роботи в навчальний процес

Горячого Олега Ярославовича

«Розроблення генераторів псевдовипадкових чисел на основі покращених методів обчислення елементарних функцій для задач кібербезпеки», представленої на здобуття наукового ступеня доктора філософії за спеціальністю 125 – *Кібербезпека та захист інформації*

Комісія Національного університету «Львівська політехніка» у складі:

Голова комісії – голова науково-методичної ради інституту комп'ютерних технологій, автоматики та метрології, д.т.н., проф. Василь ЯЦУК.

Члени комісії:

Завідувач кафедри «Безпека інформаційних технологій», д.т.н., с.н.с. Ігор ЖУРАВЕЛЬ, доцент кафедри «Безпека інформаційних технологій», к.ф.-м.н. Степан ВОЙТУСІК, професор кафедри «Безпека інформаційних технологій», д.т.н., проф. Віктор МЕЛЬНИК, доцент кафедри «Безпека інформаційних технологій», к.ф.-м.н., доц. Моріка РУСИНКО.

Даним актом комісія підтверджує, що наукові дослідження здобувача проводились на кафедрі «Безпека інформаційних технологій» Національного університету «Львівська політехніка». Основні положення та результати дисертаційної роботи впроваджені у навчальний процес кафедри «Безпека інформаційних технологій» Національного університету «Львівська політехніка» і, зокрема, використовувались:

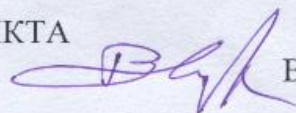
- під час виконання лабораторних робіт № 1 та № 2 з дисципліни «Прикладна криптологія» на теми: «Технологічне тестування джерел випадкових та псевдовипадкових послідовностей на основі методики Федерального стандарту США FIPS140-1» та «Дослідження властивостей симетричних криптоперетворень. Методи та засоби генерування ключів в симетричних криптосистемах» для здобувачів бакалаврського рівня спеціальності 125 «Кібербезпека та захист інформації» – дослідження та статистичне тестування альтернативних методів генерації псевдовипадкових послідовностей на основі теорії хаосу, чисельних методів та арифметики з рухомою комою;

- під час виконання лабораторних, практичних та курсових робіт з дисципліни «Архітектура комп'ютерних систем» для здобувачів бакалаврського рівня спеціальності 125 «Кібербезпека та захист інформації» – вивчення апаратних інструкцій та програмних методів обчислення в арифметиці з рухомою комою, зокрема особливості функціонування та реалізації машинної інструкції DIV.D для чисел у форматі з рухомою комою;
- під час виконання індивідуальних завдань та курсових робіт з дисциплін «Криптографічні системи та протоколи», «Технології програмування» та «Комп'ютерні методи дослідження інформаційних процесів і систем» для здобувачів бакалаврського рівня спеціальності 125 «Кібербезпека та захист інформації»;
- при виборі тем та керуванні виконанням студентами спеціальності 125 «Кібербезпека та захист інформації» бакалаврських і магістерських кваліфікаційних робіт;
- для генерації та випадкового розподілу між студентами варіантів індивідуальних завдань курсових робіт і білетів для контрольних робіт з курсу «Архітектура комп'ютерних систем» для здобувачів третього року навчання бакалаврського рівня цієї ж спеціальності, із використанням запропонованих швидкодійних генераторів псевдовипадкових чисел на основі покращених методів обчислення елементарних функцій в арифметиці з рухомою комою.

Голова комісії

голова науково-методичної ради ІКТА

д.т.н., проф.



Василь ЯЦУК

Члени комісії:

зав. каф. БІТ, д.т.н., с.н.с.



Ігор ЖУРАВЕЛЬ

доц. каф. БІТ, к.ф.-м.н.



Степан ВОЙТУСІК

проф. каф. БІТ, д.т.н., проф.



Віктор МЕЛЬНИК

доц. каф. БІТ, к.ф.-м.н., доц.



Моріка РУСИНКО

Додаток Б

Програмний код розроблених алгоритмів та методів дослідження мовою C++

Визначення основних типів та компонентів:

```
//Визначені типи для опису алгоритмів
typedef float (*float_function_type)(float);
typedef double (*double_function_type)(double);

//Вибір параметрів тестування швидкодії через директиви умовної компіляції
#ifdef rdtscTime
    int iterations = 100000;
    int tests = 10000;
#else
    int iterations = 1000000;
    int tests = 10000;
#endif

//Приклади реалізації алгоритмів для обчислення функції RCP (1/x) типу float мовою C++
float rcp_division_f (float x){
    return 1.0f/x;
}

float rcp_intel_instruction_f (float x){
    float y;
    _mm_store_ss(&y, _mm_rcp_ss(_mm_set_ss(x)));
    return y;
}

float rcp_nr_iter_2_r_form_optimized_f (float x){
    int i = *(int*)&x;
    i = 0x7eb53567 - i;
    float y = *(float*)&i;
    y = 1.9395974f*y*fmaf(-x, y, 1.436142f);
    float r = fmaf(y, -x, 1.0f);
    y = fmaf(y, r, y);
    return y;
}

...

//Приклади реалізації алгоритмів для обчислення функції RCP (1/x) типу double мовою C++
double rcp_division_d (double x){
    return 1.0/x;
}

double rcp_nr_iter_3_r_form_d (double x){
    uint64_t i = *(uint64_t*)&x;
    i = 0x7fd6a54e79bcc0f6 - i;
    double y = *(double*)&i;
    y = 1.9375*y*fma(y, -x, 1.43692333124);
    y = y*fma(y, -x, 2.00000000639);
    double r = fma(y, -x, 1.0);
    y = fma(r, y, y);
    return y;
}

double rcp_h_order_2_nr_iter_2_r_form_d (double x){
    uint64_t i = *(uint64_t*)&x;
    i = 0x7fd6a09e667f3bcd - i;
```

```

double y = *(double*)&i;
double c = x*y;
y = y*fma(c, c - 3.3766491563235857, 3.3017656219177971);
y = y*fma(y, -x, 2.0000000025);
double r = fma(y, -x, 1.0);
y = fma(r, y, y);
return y;
}

...

//Приклади реалізації алгоритмів для обчислення функції RSQRT (1/sqrtf(x)) типу
float мовою C++
float rsqrt_division_f (float x){
    return 1.0f/sqrtf(x);
}

float rsqrt_intel_instruction_f (float x){
    _mm_store_ss(&x, _mm_rsqrt_ss(_mm_set_ss(x)));
    return x;
}

float rsqrt_h_order_3_f (float x){
    int i =*(int*)&x;
    i = 0x5f375a86 - (i>>1);
    float y = *(float*)&i;
    float a = x*y*y;
    float t = fmaf(-0.3125f, a, 1.3125f);
    t = fmaf(t, a, -2.1875f);
    y = y*fmaf(t, a, 2.1875f);
    return y;
}

float rsqrt_dc_iter_1_optimized_f (float x){
    int i = *(int*)&x;
    int k = i & 0x00800000;
    float y;
    if (k == 0)
    {
        i = 0x5f19e8fc - (i >> 1);
        y = *(float*)&i;
        y = 0.824218631f*y*fmaf(-x, y*y, 2.1499474f);
    }
    else
    {
        i = 0x5ed9e91f - (i >> 1);
        y = *(float*)&i;
        y = 2.33124256f*y*fmaf(-x, y*y, 1.0749737f);
    }
    return y;
}

float rsqrt_2mc_iter_2_r_form_optimized_f (float x){
    int i = *(int*) &x;
    i = i>>1;
    int ii = 0x5e5fb432 - i;
    i = 0x5f5fb432 - i;
    float y = *(float*)&i;
    float yy = *(float*)&ii;
    y = yy*(4.76405191f - x*y*y);
    float c = x*y;
    float r = fmaf(y, c, -1.0000006f);
    y = fmaf(-0.500097573f*y, r, y);
    return y;
}

```

```

}

...

//Приклади реалізації алгоритмів для обчислення функції RSQRT (1/sqrt(x)) типу
double мовою C++
double rsqrt_division_d (double x){
    return 1.0/sqrt(x);
}

double rsqrt_walczyk_iter_3_d (double x){
    double xhalf = 0.5*x;
    long long i = *(long long*) &x;
    i = 0x5fe6ed2102dcbfda - (i>>1);
    x = *(double*)&i;
    x = x*(1.5008789551163346 - xhalf*x*x);
    x = x*(1.5000005796762577 - xhalf*x*x);
    x = x*(1.5000000000000252 - xhalf*x*x);
    return x;
}

double rsqrt_2mc_iter_1_optimized_d (double x){
    long long i = *(long long*) &x;
    i = i>>1;
    long long ii = 0x5fcbf6db526de7d9 - i;
    i = 0x5febf6db526de7d9 - i;
    double y = *(double*)&i;
    double yy = *(double*)&ii;
    y = yy*(4.7642670066528519 - x*y*y);
    return y;
}

double rsqrt_2mc_iter_3_int_t_form_optimized_d (double x){
    long long i = *(long long*) &x;
    long long ix = i - 0x8010000000000000;
    i = i>>1;
    long long ii = 0x5fcbf6da126a4db1 - i;
    i = 0x5febf6da126a4db1 - i;
    double y = *(double*)&i;
    double yy = *(double*)&ii;
    y = yy*(4.7642669737958503 - x*y*y);
    double mhalf = *(double*)&ix;
    double t = fma(mhalf, y*y, 0.50000031698540448);
    y = fma(y, t, y);
    t = fma(mhalf, y*y, 0.5);
    y = fma(y, t, y);
    return y;
}

double rsqrt_h_order_2_nr_iter_2_t_form_optimized_d (double x){
    int64_t i = *(int64_t*) &x;
    i = 0x5fe52c437da986d9 - (i >> 1);
    double y = *(double*) &i;
    double c = x*y*y;
    y = y*(2.00718651889802436 - c*(1.53237125952242026 - 0.52595012062439589*c));
    double xh = 0.5*x;
    y = y*(1.500000000285 - xh*y*y);
    double t = xh*y;
    t = fma(y, -t, 0.5);
    y = fma(y, t, y);
    return y;
}

double rsqrt_dc_iter_3_t_form_optimized_d (double x){

```

```

uint64_t i = *(uint64_t*)&x;
uint64_t k = i & 0x0010000000000000;
double y;
if (k != 0) {
    i = 0x5fdb3d14170034b6 - (i >> 1);
    y = *(double*)&i;
    y = 2.33124735553421569*y*fma(-x, y*y, 1.07497362654295614);
} else {
    i = 0x5fe33d18a2b9ef5f - (i >> 1);
    y = *(double*)&i;
    y = 0.82421942523718461*y*fma(-x, y*y, 2.1499494964450325);
}
double mxhalf = -0.5*x;
y = y*fma(mxhalf, y*y, 1.5000000034937999);
double t = fma(mxhalf, y*y, 0.5);
y = fma(y, t, y);
return y;
}

...

//Приклади реалізації алгоритмів для обчислення функції SQRT (sqrtf(x)) типу float
мовою C++
float sqrt_function_f (float x){
    return sqrtf(x);
}

float sqrt_intel_instruction_f (float x){
    _mm_store_ss(&x, _mm_sqrt_ss(_mm_set_ss(x)));
    return x;
}

float sqrt_dc_iter_1_optimized_f (float x){
    int i = *(int*)&x;
    int k = i & 0x00800000;
    float y;
    if (k == 0)
    {
        i = 0x5f19e8fd - (i >> 1);
        y = *(float*)&i;
        float c = x*y;
        y = 0.82421863f*c*fmaf(-c, y, 2.1499474f);
    }
    else
    {
        i = 0x5ed9e893 - (i >> 1);
        y = *(float*)&i;
        float c = x*y;
        y = 2.33130789f*c*fmaf(-c, y, 1.07495356f);
    }
    return y;
}

float sqrt_h_order_2_nr_iter_1_r_form_f (float x){
    int i = *(int*)&x;
    i = 0x5F29621B - (i >> 1);
    float y = *(float*)&i;
    float c = x*y*y;
    y = y*(2.00718652f - c*(1.5323713f - 0.52595012f*c));
    c = x*y;
    y = fmaf(y, c, -1.0f);
    y = fmaf(-0.5f*c, y, c);
    return y;
}

```

```

}

...

//Приклади реалізації алгоритмів для обчислення функції SQRT (sqrt(x)) типу double
мовою C++
double sqrt_function_d (double x){
    return sqrt(x);
}

double sqrt_intel_instruction_d (double x){
    _mm_store_sd(&x, _mm_sqrt_sd(_mm_set_sd(0), _mm_set_sd(x)));
    return x;
}

double sqrt_nr_iter_1_r_form_d (double x){
    uint64_t i =*(uint64_t*)&x;
    i = 0x5fe6eb50c7b537a9 - (i>>1);
    double y = *(double*)&i;
    double c = x*y;
    double r = fma(y, -c, 1.0);
    y = fma(0.5*c, r, c);
    return y;
}

double sqrt_dc_iter_2_r_form_optimized_d (double x){
    uint64_t i =*(uint64_t*)&x;
    uint64_t k = i & 0x0010000000000000;
    double y;
    if (k == 0)
    {
        i = 0x5fe33d2098e159e4 - (i >> 1);
        y = *(double*)&i;
        y = 0.82421861262961627*y*fma(-x, y*y, 2.1499474592540948);
    }
    else
    {
        i = 0x5fdb3d209f0f2455 - (i >> 1);
        y = *(double*)&i;
        y = 2.3312423972719842*y*fma(-x, y*y, 1.0749736939858485);
    }
    double c = x*y;
    double r = fma(y, -c, 1.000000008298416);
    y = fma(0.50000000057371974, c*r, c);
    return y;
}

...

#if RCP
//Задання списку алгоритмів обчислення функції RCP типу float для тестування
const int FLOAT_FUNCTIONS_COUNT = 8;
const float_function_type FLOAT_FUNCTIONS[FLOAT_FUNCTIONS_COUNT] {
rcp_division_f, rcp_intel_instruction_f, ... , rcp_nr_iter_2_r_form_optimized_f, ...
};
string FLOAT_FUNCTIONS_NAMES[FLOAT_FUNCTIONS_COUNT] { "1.0f/x", "RCP Intel
instruction f", ... , "RCP Newton-Raphson iter 2 (r-form) optimized f", ... };

//Задання списку алгоритмів обчислення функції RCP типу double для тестування
const int DOUBLE_FUNCTIONS_COUNT = 10;
const double_function_type DOUBLE_FUNCTIONS[DOUBLE_FUNCTIONS_COUNT] {
rcp_division_d, ..., rcp_nr_iter_3_r_form_d, ..., rcp_h_order_2_nr_iter_2_r_form_d, ...
};

```

```

    string DOUBLE_FUNCTIONS_NAMES[DOUBLE_FUNCTIONS_COUNT] { "1.0/x", ... , "RCP
Newton-Raphson iter 3 (r-form) modified d", ... , "RCP Householder order 2 + Newton-
Raphson iter 2 (r-form) optimized d", ... };

#elif RSQRT
    //Задання списку алгоритмів обчислення функції RSQRT типу float для тестування
    const int FLOAT_FUNCTIONS_COUNT = 10;
    const float_function_type FLOAT_FUNCTIONS[FLOAT_FUNCTIONS_COUNT] {
rsqrt_division_f, rsqrt_intel_instruction_f, ... , rsqrt_h_order_3_f,
rsqrt_dc_iter_1_optimized_f, rsqrt_2mc_iter_2_r_form_optimized_f, ... };
    string FLOAT_FUNCTIONS_NAMES[FLOAT_FUNCTIONS_COUNT] { "1.0f/sqrtf(x)", "RSQRT
Intel instruction f", ... , "RSQRT Householder order 3 f", "RSQRT Dynamic constants
iter 1 optimized f", "RSQRT 2 Magic constants iter 2 (r-form) optimized f", ... };

    //Задання списку алгоритмів обчислення функції RSQRT типу double для тестування
    const int DOUBLE_FUNCTIONS_COUNT = 20;
    const double_function_type DOUBLE_FUNCTIONS[DOUBLE_FUNCTIONS_COUNT] {
rsqrt_division_d, ..., rsqrt_walczyk_iter_3_d, rsqrt_2mc_iter_1_optimized_d,
rsqrt_2mc_iter_3_int_t_form_optimized_d, rsqrt_dc_iter_3_t_form_optimized_d,
rsqrt_h_order_2_nr_iter_2_t_form_optimized_d, ... };
    string DOUBLE_FUNCTIONS_NAMES[DOUBLE_FUNCTIONS_COUNT] { "1.0/sqrt(x)", ... ,
"RSQRT Walczyk iter 3 modified d", "RSQRT 2 Magic constants iter 1 optimized d",
"RSQRT 2 Magic constants iter 3 (int) (t-form) optimized d", "RSQRT Dynamic
constants iter 3 (t-form) optimized d", "RSQRT Householder order 2 + Newton-
Raphson iter 2 (t-form) optimized d", ... };

#elif SQRT
    //Задання списку алгоритмів обчислення функції SQRT типу float для тестування
    const int FLOAT_FUNCTIONS_COUNT = 8;
    const float_function_type FLOAT_FUNCTIONS[FLOAT_FUNCTIONS_COUNT] {
sqrt_function_f, sqrt_intel_instruction_f, ... , sqrt_dc_iter_1_optimized_f,
sqrt_h_order_2_nr_iter_1_r_form_f, ... };
    string FLOAT_FUNCTIONS_NAMES[FLOAT_FUNCTIONS_COUNT] { "sqrtf(x)", "SQRT Intel
instruction f", ... , "SQRT Dynamic constants iter 1 optimized f", "SQRT Householder
order 2 + Newton-Raphson iter 1 (r-form) modified f", ... };

    //Задання списку алгоритмів обчислення функції RSQRT типу double для тестування
    const int DOUBLE_FUNCTIONS_COUNT = 10;
    const double_function_type DOUBLE_FUNCTIONS[DOUBLE_FUNCTIONS_COUNT] {
sqrt_function_d, sqrt_intel_instruction_d, ..., sqrt_nr_iter_1_r_form_d,
sqrt_dc_iter_2_r_form_optimized_d, ... };
    string DOUBLE_FUNCTIONS_NAMES[DOUBLE_FUNCTIONS_COUNT] { "sqrt(x)", "SQRT Intel
instruction d", ... , "SQRT Newton-Raphson iter 1 (r-form) d", "SQRT Dynamic
constants iter 2 (r-form) optimized d", ... };
#endif

```

Б.1 Обчислення похибок алгоритмів

Опис: Процедура визначення максимальних абсолютних та відносних похибок алгоритму типу *float_function_type* для всіх можливих значень аргумента функції з інтервалу $x \in [\text{intervalA}; \text{intervalB}]$ із використанням функції *nextafterf*.

Аргументи: Вказівник на алгоритм *float_function* типу *float_function_type*.

Вихідний код:

```

void test_float_function_accuracy(float_function_type float_function){
    float a = intervalA;
    float b;
    double I_function_a, abs_er, rel_er;

```

```

double large = DBL_MAX;
double max_abs_er = -large, min_abs_er = large, max_rel_er = -large, min_rel_er
= large;
float x_max_abs_er = -1, x_min_abs_er = -1, x_max_rel_er = -1, x_min_rel_er = -
1;
float function_a;
do {
    b = nextafterf(a, FLT_MAX);

    function_a = float_function(a);
    #if RSQRT
        I_function_a = 1.0 / sqrt((double)a);
    #elif SQRT
        I_function_a = sqrt((double)a);
    #elif RCP
        I_function_a = 1.0 / a;
    #endif
    abs_er = function_a - I_function_a;
    #if RSQRT
        rel_er = sqrt((double)a) * function_a - 1.0;
    #elif SQRT
        rel_er = function_a / sqrt((double)a) - 1.0;
    #elif RCP
        rel_er = ((double)a) * function_a - 1.0;
    #endif

    if(abs_er > max_abs_er){
        max_abs_er = abs_er;
        x_max_abs_er = a;
    } else if(abs_er < min_abs_er){
        min_abs_er = abs_er;
        x_min_abs_er = a;
    }
    if(rel_er > max_rel_er){
        max_rel_er = rel_er;
        x_max_rel_er = a;
    } else if(rel_er < min_rel_er){
        min_rel_er = rel_er;
        x_min_rel_er = a;
    }
    a = b;
}
while(b < intervalB);
#if printShortResults
    #if printAbsErrors
        #if printMinMaxErrors
            printf("%.7e %.7e\n", min_abs_er, max_abs_er);
        #else
            double mod_max_abs_er = max(abs(max_abs_er), abs(min_abs_er));
            printf("%.7e\n", mod_max_abs_er);
        #endif
    #else
        #if printMinMaxErrors
            printf("%.7e %.7e\n", min_rel_er, max_rel_er);
        #else
            double mod_max_rel_er = max(abs(max_rel_er), abs(min_rel_er));
            printf("%.7e\n", mod_max_rel_er);
        #endif
    #endif
#else
    #if printAbsErrors
        #if printMinMaxErrors
            printf("Max_abs_er: er = %.7e x = %.7f\n", max_abs_er, x_max_abs_er);
            printf("Min_abs_er: er = %.7e x = %.7f\n\n", min_abs_er, x_min_abs_er);

```

```

        #else
            double mod_max_abs_er = max(abs(max_abs_er), abs(min_abs_er));
            float mod_x_max_abs_er = (mod_max_abs_er == abs(max_abs_er)) ? x_max_abs_er : x_min_abs_er;
            printf("Mod_max_abs_er: er = %.7e x = %.7f\n\n", mod_max_abs_er ,
mod_x_max_abs_er);
        #endif
    #endif
    #if printMinMaxErrors
        printf("Max_rel_er: er = %.7e x = %.7f\n", max_rel_er, x_max_rel_er);
        printf("Min_rel_er: er = %.7e x = %.7f\n\n", min_rel_er, x_min_rel_er);
    #else
        double mod_max_rel_er = max(abs(max_rel_er), abs(min_rel_er));
        float mod_x_max_rel_er = (mod_max_rel_er == abs(max_rel_er)) ? x_max_rel_er : x_min_rel_er;
        printf("Mod_max_rel_er: er = %.7e x = %.7f\n\n", mod_max_rel_er ,
mod_x_max_rel_er);
    #endif
    #endif
}

```

Опис: Основна процедура визначення максимальних абсолютних та відносних похибок алгоритму типу *double_function_type* методом повного перебору значень аргумента функції з інтервалу $x \in [\text{intervalAd}; \text{intervalBd}]$ або з визначеним кроком *doubleAccuracyStepValue*.

Аргументи: Вказівник на алгоритм *double_function* типу *double_function_type*.

Вихідний код:

```

void test_double_function_accuracy(double_function_type double_function){
    double a = intervalAd;
    double b;
    long double I_function_a, abs_er, rel_er;
    long double large = DBL_MAX;
    long double max_abs_er = -large, min_abs_er = large, max_rel_er = -large,
min_rel_er = large;
    double x_max_abs_er = -1, x_min_abs_er = -1, x_max_rel_er = -1, x_min_rel_er = -1;
    double function_a;
    int count = 0;
    do {
        #if doubleAccuracyAllPoints
            b = nextafter(a, DBL_MAX);
        #else
            b = a + doubleAccuracyStepValue;
        #endif
        #if !printShortResults
            count++;
            if(count%1000000 == 0){
                printf("\rProgress: %.5f%% (%.17Le)", (b - intervalAd)*100/(intervalBd - intervalAd), max(abs(max_rel_er), abs(min_rel_er)));
            }
        #endif

        function_a = double_function(a);
        #if RSQRT
            I_function_a = rsqrt_ld(a);
        #elif SQRT

```

```

    I_function_a = sqrt_ld(a);
#elif RCP
    I_function_a = rcp_ld(a);
#endif
    abs_er = function_a - I_function_a;
#if RSQRT
    rel_er = sqrt_ld(a) * function_a - 1.0L;
#elif SQRT
    rel_er = function_a / sqrt_ld(a) - 1.0L;
#elif RCP
    rel_er = ((long double)a) * function_a - 1.0L;
#endif

    if(abs_er > max_abs_er){
        max_abs_er = abs_er;
        x_max_abs_er = a;
    } else if(abs_er < min_abs_er){
        min_abs_er = abs_er;
        x_min_abs_er = a;
    }
    if(rel_er > max_rel_er){
        max_rel_er = rel_er;
        x_max_rel_er = a;
    } else if(rel_er < min_rel_er){
        min_rel_er = rel_er;
        x_min_rel_er = a;
    }
    a = b;
}
while(b < intervalBd);
#if !printShortResults
    printf("\rProgress: %.5f%%\n", 100.0);
#endif
#if printShortResults
    #if printAbsErrors
        ...
    #else
        #if printMinMaxErrors
            printf("%.17Le %.17Le \n", min_rel_er, max_rel_er);
        #else
            long double mod_max_rel_er = max(abs(max_rel_er), abs(min_rel_er));
            printf("%.17Le\n", mod_max_rel_er);
        #endif
    #endif
#else
    #if printAbsErrors
        ...
    #endif
    #if printMinMaxErrors
        printf("Max_rel_er: er = %.17Le x = %.17f\n", max_rel_er, x_max_rel_er);
        printf("Min_rel_er: er = %.17Le x = %.17f\n\n", min_rel_er, x_min_rel_er);
    #else
        ...
    #endif
#endif
}

```

Б.2 Вимірювання швидкодії алгоритмів

Опис: Процедура вимірювання латентності алгоритмів типу *float_function_type*, що описані в масиві *FLOAT_FUNCTIONS*, із використанням бібліотеки *chrono*.

Процедура виконується із найвищим можливим пріоритетом виконання для ОС Windows. Аналогічна процедура реалізується для алгоритмів типу *double_function_type* та у випадку використання апаратного лічильника `__rdtsc()` для вимірювання кількості тактів процесора.

Аргументи: Параметри тестування *iterations* та *tests*.

Вихідний код:

```
void test_float_function_elapsed_time_chrono(int iterations, int tests){
    long double sum_times[FLOAT_FUNCTIONS_COUNT + 1];
    long double min_times[FLOAT_FUNCTIONS_COUNT + 1];
    long double max_times[FLOAT_FUNCTIONS_COUNT + 1];
    long double times[FLOAT_FUNCTIONS_COUNT + 1];
    std::chrono::time_point<std::chrono::system_clock> t1,t2;
    volatile float y = 0.5f;
    float xf = y;

    for(int f=0;f<FLOAT_FUNCTIONS_COUNT+1;f++){
        sum_times[f] = 0;
        min_times[f] = DBL_MAX;
        max_times[f] = 0;
    }

    #if setRealTimePriority
        if(!SetPriorityClass(GetCurrentProcess(), REALTIME_PRIORITY_CLASS))
        {
            printf("Error: Failed to set priority mode for process!\n");
        }
    #endif

    for(int t=0; t<tests; t++){
        if(t%(tests/20) == 0)
            printf("\rProgress: %d%%", t*100/tests);
        t1 = std::chrono::high_resolution_clock::now();
        for(int i = 0; i < iterations; i++){
            xf = xf + i;
        }
        t2 = std::chrono::high_resolution_clock::now();
        times[0] = (long double) std::chrono::duration_cast<std::chrono::nanoseconds>
(t2-t1).count();

        for(int f=0;f<FLOAT_FUNCTIONS_COUNT;f++){
            t1 = std::chrono::high_resolution_clock::now();
            for(int i = 0; i < iterations; i++){
                xf = FLOAT_FUNCTIONS[f](xf);
                xf = xf + i;
            }
            t2 = std::chrono::high_resolution_clock::now();
            times[f+1] = (long double)
std::chrono::duration_cast<std::chrono::nanoseconds>(t2-t1).count();
        }

        for(int f=0;f<FLOAT_FUNCTIONS_COUNT+1;f++){
            sum_times[f] += times[f];
            if(times[f] < min_times[f])
                min_times[f] = times[f];
            if(times[f] > max_times[f])
                max_times[f] = times[f];
        }
    }
```

```

}
printf("\rProgress: %d%%\n", 100);

y = xf;

#ifdef setRealTimePriority
    if(!SetPriorityClass(GetCurrentProcess(), NORMAL_PRIORITY_CLASS))
    {
        printf("Error: Failed to set priority mode for process!\n");
    }
#endif

#ifdef printShortResults
    printf("Empty loop\n");
    for(int f=0; f<FLOAT_FUNCTIONS_COUNT; f++){
        printf("%s\n", FLOAT_FUNCTIONS_NAMES[f].c_str());
    }
    #if printShortResults
        #if printTimeMin
            printf("Min Time (nanoseconds)\n");
        #elif printTimeMax
            printf("Max Time (nanoseconds)\n");
        #elif printRatioTime
            printf("Ratio (nanoseconds)\n");
        #else
            printf("Avr Time (nanoseconds)\n");
        #endif
    #endif
#endif
printf("tests = %d iterations = %d\n", tests, iterations);

for(int f=0;f<FLOAT_FUNCTIONS_COUNT+1;f++){
    if(f == 0){
        sum_times[0] = sum_times[0] / ((long double)tests*iterations);
        min_times[0] = min_times[0] / iterations;
        max_times[0] = max_times[0] / iterations;
        #if printShortResults
            #if printTimeMin
                printf("%.5Lf\n", min_times[f]);
            #elif printTimeMax
                printf("%.5Lf\n", max_times[f]);
            #elif !printRatioTime
                printf("%.5Lf\n", sum_times[f]);
            #endif
        #else
            printf("Avr time of empty loop = %.5Lf nanoseconds [%.2Lf; %.2Lf]\n",
sum_times[f], min_times[f], max_times[f]);
        #endif
    } else {
        sum_times[f] = (sum_times[f] - sum_times[0]) / ((long
double)tests*iterations);
        min_times[f] = (min_times[f] - min_times[0]) / iterations;
        max_times[f] = (max_times[f] - min_times[0]) / iterations;
        #if printShortResults
            #if printTimeMin
                printf("%.5Lf\n", min_times[f]);
            #elif printTimeMax
                printf("%.5Lf\n", max_times[f]);
            #elif printRatioTime
                printf("%.5Lf\n", sum_times[f] / sum_times[1]);
            #else
                printf("%.5Lf\n", sum_times[f]);
            #endif
        #else

```

```

        printf("Avr time of %s = %.5Lf nanoseconds [%.5Lf; %.5Lf] (%.2Lf)\n",
FLOAT_FUNCTIONS_NAMES[f-1].c_str(), sum_times[f], min_times[f], max_times[f],
sum_times[f] / sum_times[1]);
    #endif
    }
}
printf("Dumb value: %.2f\n", y);
}

```

Б.3 Методи оптимізації параметрів алгоритмів

Визначення основних типів та компонентів:

```

//Визначені типи для опису шаблонів алгоритмів для оптимізації їх параметрів
typedef float (*float_parameterized_multidimensional_function_type)(float, int*,
int, float*, int);
typedef double (*double_parameterized_multidimensional_function_type)(double,
uint64_t*, int, double*, int);

```

```

//Приклади опису алгоритмів для функції RSQRT типу double у вигляді шаблонів
double rsqrt_dc_iter_3_t_form_template_d(double x, uint64_t* iCoefs, int iN,
double* fCoefs, int fN){

```

```

    if(iN != 2 || fN != 5)
        throw std::invalid_argument( "Invalid mas length" );

```

```

    uint64_t i = *(uint64_t*)&x;
    uint64_t k = i & 0x0010000000000000;
    double y;
    if (k == 0)
    {
        i = iCoefs[0] - (i >> 1);
        y = *(double*)&i;
        y = fCoefs[0]*y*fma(-x, y*y, fCoefs[1]);
    }
    else
    {
        i = iCoefs[1] - (i >> 1);
        y = *(double*)&i;
        y = fCoefs[2]*y*fma(-x, y*y, fCoefs[3]);
    }
    double mxHalf = -0.5*x;
    y = y*fma(mxHalf, y*y, fCoefs[4]);
    double r = fma(mxHalf, y*y, 0.5);
    y = fma(y, r, y);
    return y;
}

```

```

double rsqrt_dc_iter_3_t_form_template_one_part_d(double x, uint64_t* iCoefs, int
iN, double* fCoefs, int fN){

```

```

    if(iN != 1 || fN != 3)
        throw std::invalid_argument( "Invalid mas length" );

```

```

    uint64_t i = *(uint64_t*)&x;
    i = iCoefs[0] - (i >> 1);
    double y = *(double*)&i;
    y = fCoefs[0]*y*fma(-x, y*y, fCoefs[1]);
    double mxHalf = -0.5*x;
    y = y*fma(mxHalf, y*y, fCoefs[2]);
    double r = fma(mxHalf, y*y, 0.5);
    y = fma(y, r, y);
    return y;
}

```

```
}
```

Опис: Приклад реалізації методу пошуку оптимального параметра типу *int* для алгоритму одинарної точності на основі повного перебору значень або перебору із визначеним кроком. Аналогічно для типів *float*, *uint64_t* та *double*. Версія даного методу, що повертає набір таких значень (типу *vector<int>*) у випадку однакового значення максимальної відносної похибки, має назву *get_optimized_integer_parameters_with_bruteforce*.

Аргументи: Шаблон алгоритму *float_function* типу *float_parameterized_multidimensional_function_type*, межі проміжку [*c_min*; *c_max*], крок *step* для зміни значень параметра *iCoefs[iIndex]*, параметр *allPoints* визначає використання методу повного перебору, поточні значення всіх параметрів алгоритму.

Вихідний код:

```
int get_optimized_integer_parameter_with_bruteforce(
float_parameterized_multidimensional_function_type float_function, int c_min, int
c_max, int step, bool allPoints, int iIndex, int* iCoefs, int iN, float* fCoefs,
int fN){
    double large = DBL_MAX;
    int c = c_min;
    int best_c = c;
    double min_max_rel_er = large;
    float x_min_max_rel_er = -1;
    int percentage = 0;
    do {
        if(allPoints)
            c++;
        else {
            if(c + step <= c || c + step > c_max)
                break;
            else
                c = c + step;
        }
        iCoefs[iIndex] = c;
        float a = intervalA;
        float b;
        double rel_er, max_rel_er = -large;
        float x_max_rel_er = -1;
        float function_a;
        do {
            b = nextafterf(a, FLT_MAX);
            function_a = float_function(a, iCoefs, iN, fCoefs, fN);
            #if RSQRT
                rel_er = sqrt((double)a) * function_a - 1.0;
            #elif SQRT
                rel_er = function_a / sqrt((double)a) - 1.0;
            #elif RCP
                rel_er = ((double)a) * function_a - 1.0;
            #endif
            if(abs(rel_er) > max_rel_er){
                max_rel_er = abs(rel_er);
```

```

        x_max_rel_er = a;
    }
    a = b;
}
while(b < intervalB);
if(max_rel_er < min_max_rel_er){
    min_max_rel_er = max_rel_er;
    x_min_max_rel_er = x_max_rel_er;
    best_c = c;
}
if(c_max != c_min && (c - (double)c_min)*100.0/(c_max - (double)c_min) >
percentage){
    percentage++;
    while((c - (double)c_min)*100.0/(c_max - (double)c_min) > percentage){
        percentage++;
    }
    printf("%d % : c = 0x%x | Max_rel_er: er = %.10e x = %.7f\n", percentage -
1, c, max_rel_er, x_max_rel_er);
    if((percentage - 1)%10 == 0){
        printf("Best c = 0x%x | Max_rel_er: er = %.10e x = %.7f\n", best_c,
min_max_rel_er, x_min_max_rel_er);
    }
}
} while(c < c_max);
printf("\niIndex = %d : Best c = 0x%x | Max_rel_er: er = %.10e x = %.7f\n\n",
iIndex, best_c, min_max_rel_er, x_min_max_rel_er);
return best_c;
}

```

Опис: Приклад реалізації методу пошуку оптимального параметра типу *int* для алгоритму одинарної точності на основі методу золотого перетину та розбиття проміжку $[c_min; c_max]$ на *randomIntervalsCount* випадкових підінтервалів. Аналогічно для типів *float*, *uint64_t* та *double*. Версія даного методу, що повертає набір таких значень (типу *vector<int>*) у випадку однакового значення максимальної відносної похибки, має назву *get_optimized_integer_parameters_with_golden_search_and_random_intervals*.

Аргументи: Шаблон алгоритму *float_function* типу *float_parameterized_multidimensional_function_type*, межі проміжку $[c_min; c_max]$, крок *step* для методу золотого перетину, індекс параметра *iIndex* у масиві *iCoefs*, поточні значення всіх параметрів алгоритму, стандартне ядро ГПВЧ *eng* та кількість підінтервалів *randomIntervalsCount*.

Вихідний код:

```

int get_optimized_integer_parameter_with_golden_search_and_random_intervals(
float_parameterized_multidimensional_function_type float_function, int c_min, int
c_max, int step, int iIndex, int* iCoefs, int iN, float* fCoefs, int fN,
default_random_engine eng, int randomIntervalsCount){
    int temp_iCoefs0 = iCoefs[iIndex];
    uniform_int_distribution<int> unif(c_min + step, c_max - step);

```

```

int* intervals = new int[randomIntervalsCount + 1];
intervals[0] = c_min;
intervals[randomIntervalsCount] = c_max;
for(int i = 1; i < randomIntervalsCount; i++){
    int valueToAdd = unif(eng);
    if(i == 1)
        intervals[i] = valueToAdd;
    else for(int j = 1; j < i; j++){
        if(intervals[j] > valueToAdd){
            for(int k = i; k > j; k--){
                intervals[k] = intervals[k-1];
            }
            intervals[j] = valueToAdd;
            break;
        } else if(j == i - 1){
            intervals[i] = valueToAdd;
        }
    }
}
}
printf("\n[");
for(int i = 0; i <= randomIntervalsCount; i++){
    printf("0x%x ", intervals[i]);
}
printf("]\n");

int* c_best_in_interval = new int[randomIntervalsCount];
double gr = 1.6180339887498948; //gr = sqrt(5.0)/2.0 + 0.5
double rel_er1, rel_er2, best_rel_er = DBL_MAX;
int c1, c2, c_best;
for(int i = 0; i < randomIntervalsCount; i++){
    c_min = intervals[i];
    c_max = intervals[i + 1];
    printf("\n[0x%x; 0x%x]\n", c_min, c_max);
    c1 = c_max - (c_max - (double)c_min) / gr;
    c2 = c_min + (c_max - (double)c_min) / gr;
    while (abs(c1 - c2) > step){
        iCoefs[iIndex] = c1;
        rel_er1 = get_max_relative_error(float_function, iCoefs, iN, fCoefs, fN);
        iCoefs[iIndex] = c2;
        rel_er2 = get_max_relative_error(float_function, iCoefs, iN, fCoefs, fN);
        if(rel_er1 < rel_er2)
            c_max = c2;
        else
            c_min = c1;
        c1 = c_max - (c_max - (double)c_min) / gr;
        c2 = c_min + (c_max - (double)c_min) / gr;
    }
    c_best_in_interval[i] = get_optimized_integer_parameter_with_bruteforce
(float_function, c_min, c_max, step, true, iIndex, iCoefs, iN, fCoefs, fN);
    printf("c_best[%d] = 0x%x\n", i, c_best_in_interval[i]);
}

for(int i = 0; i < randomIntervalsCount; i++){
    iCoefs[iIndex] = c_best_in_interval[i];
    rel_er1 = get_max_relative_error(float_function, iCoefs, iN, fCoefs, fN);
    if(rel_er1 < best_rel_er){
        best_rel_er = rel_er1;
        c_best = c_best_in_interval[i];
    }
}
}
printf("c_best = 0x%x | Max_rel_er = %.10e\n\n", c_best, best_rel_er);
iCoefs[iIndex] = temp_iCoefs0;
return c_best;
}

```

Б.3.1 Метод багатовимірного випадкового пошуку із двома центрами ваги та з фокусуванням

Опис: Приклад реалізації методу пошуку оптимальних параметрів алгоритму для чисел типу *float* на основі генерації ПВЧ, використовуючи декілька стратегій випадкового вибору. Аналогічно для *double*.

Аргументи: Шаблон алгоритму *float_function* типу *float_parameterized_multidimensional_function_type*, параметри оптимізації, критерій зупинки *maxStopIterations*, стандартне ядро ГПВЧ *eng*.

Вихідний код:

```
float sigma(int number){
    if(number == 0)
        return 0;
    if(number < 5)
        return 0.3f;
    if(number < 10)
        return 0.5f;
    if(number < 15)
        return 0.8f;
    if(number < 25)
        return 0.9f;
    return 0.95f;
}

double get_optimized_multidimensional_parameters_with_random(
float_parameterized_multidimensional_function_type float_function, int iN, int fN,
int* iCoefs_min, int* iCoefs_max, float* fCoefs_min, float* fCoefs_max, int*
iCoefs_step, float* fCoefs_step, int maxStopIterations, default_random_engine eng
){
    uniform_real_distribution<float> unif(0.0f, 1.0f);
    uniform_int_distribution<int> unif_int_interval(0, 100);

    uniform_int_distribution<int> unif_ints[iN];
    uniform_real_distribution<float> unif_floats[fN];
    for(int i = 0; i < iN; i++){
        unif_ints[i] = uniform_int_distribution<int>(iCoefs_min[i], iCoefs_max[i]);
    }
    for(int f = 0; f < fN; f++){
        unif_floats[f] = uniform_real_distribution<float>(fCoefs_min[f],
fCoefs_max[f]);
    }

    int* iCoefs_best = new int[iN];
    float* fCoefs_best = new float[fN];
    double* Coefs_sum = new double[iN + fN] {0};
    int* iCoefs = new int[iN];
    float* fCoefs = new float[fN];

    double rel_er, best_rel_er = DBL_MAX;
    int iteration = 0;
    int iteration2 = 0;
    int stop = 0;
    do{
        ++iteration;
        for(int i = 0; i < iN; i++){
```

```

if(iteration2 > 0 && iteration%2 == 0){
    float s = sigma(iteration2);
    if(iteration2 > 10 && iteration%4 == 0)
        iCoefs[i] = (1 - s)*unif_ints[i](eng) + s*Coefs_sum[i];
    else {
        if(unif_int_interval(eng)%5 == 0)
            iCoefs[i] = iCoefs_best[i];
        else
            iCoefs[i] = (1 - s)*unif_ints[i](eng) + s*iCoefs_best[i];
    }
} else
    iCoefs[i] = unif_ints[i](eng);
}
for(int f = 0; f < fN; f++){
    if(iteration2 > 0 && iteration%2 == 0){
        float s = sigma(iteration2);
        if(iteration2 > 10 && iteration%4 == 0)
            fCoefs[f] = (1 - s)*unif_floats[f](eng) + s*Coefs_sum[iN + f];
        else {
            if(unif_int_interval(eng)%5 == 0)
                fCoefs[f] = fCoefs_best[f];
            else
                fCoefs[f] = (1 - s)*unif_floats[f](eng) + s*fCoefs_best[f];
        }
    } else
        fCoefs[f] = unif_floats[f](eng);
}
rel_er = get_max_relative_error(float_function, iCoefs, iN, fCoefs, fN);
if(rel_er < best_rel_er){
    ++iteration2;
    best_rel_er = rel_er;
    for(int i = 0; i < iN; i++){
        iCoefs_best[i] = iCoefs[i];
        if(iteration2 > 1)
            Coefs_sum[i] = 0.5*Coefs_sum[i] + 0.5*iCoefs[i];
        else
            Coefs_sum[i] = iCoefs[i];
    }
    for(int f = 0; f < fN; f++){
        fCoefs_best[f] = fCoefs[f];
        if(iteration2 > 1)
            Coefs_sum[iN + f] = 0.5*Coefs_sum[iN + f] + 0.5*fCoefs[f];
        else
            Coefs_sum[iN + f] = fCoefs[f];
    }
    printf("\n!!! New best (Iteration %d):\n", iteration);
    for(int i = 0; i < iN; i++){
        printf("iCoef[%d] = 0x%x\n", i, iCoefs[i]);
        printf("iCoef_avr[%d] = 0x%x\n", i, (int)(Coefs_sum[i]));
    }
    for(int f = 0; f < fN; f++){
        printf("fCoef[%d] = %.8e\n", f, fCoefs[f]);
        printf("fCoef_avr[%d] = %.8e\n", f, Coefs_sum[iN + f]);
    }
    printf("Max_rel_er = %.10e\n", rel_er);
    stop = 0;
} else {
    stop++;
}
} while(stop < maxStopIterations);
printf("The best coefficients (Iterations %d):\n", iteration);
for(int i = 0; i < iN; i++){
    printf("iCoef[%d] = 0x%x\n", i, iCoefs_best[i]);
}

```

```

for(int f = 0; f < fN; f++){
    printf("fCoef[%d] = %.8e\n", f, fCoefs_best[f]);
}
printf("Max_rel_er = %.10e\n\n", best_rel_er);
printf("Stop = %d\n", stop);
return best_rel_er;
}

```

Б.3.2 Метод рандомізованої багатовимірної жадібної оптимізації на основі методу багатовимірного прямого пошуку та методу золотого перетину з випадковим розбиттям проміжків

Опис: Приклад методу пошуку оптимальних параметрів алгоритму для чисел типу *float* на основі рандомізованого багатовимірного прямого пошуку, що включає випадковий вибір порядку оптимізації параметрів та проведення періодичних спроб випадкової модифікації поточних найкращих параметрів *iCoefs0* та *fCoefs0*. Для одновимірної оптимізації використовується розбиття проміжку зміни відповідного параметра на *randomIntervalsCount* випадкових підінтервалів, на кожному з яких застосовується метод пошуку на основі золотого перетину.

Аргументи: Шаблон алгоритму *float_function* типу *float_parameterized_multidimensional_function_type*, початкові значення параметрів *iCoefs0* та *fCoefs0*, параметри оптимізації, критерій зупинки *rel_er_eps*, стандартне ядро ГПВЧ *eng* та кількість інтервалів *randomIntervalsCount* для розбиття проміжків $[iCoefs_min[j]; iCoefs_max[j]]$, $j = \overline{1, iN}$, та $[fCoefs_min[j]; fCoefs_max[j]]$, $j = \overline{1, fN}$.

Вихідний код:

```

double
get_optimized_multidimensional_parameters_with_randomized_greedy_golden_search_and
_random_intervals( float_parameterized_multidimensional_function_type
float_function, int* iCoefs0, int iN, float* fCoefs0, int fN, int* iCoefs_min,
int* iCoefs_max, float* fCoefs_min, float* fCoefs_max, int* iCoefs_step, float*
fCoefs_step, double rel_er_eps, default_random_engine eng, int
randomIntervalsCount ){
    uniform_int_distribution<int> unif(0, iN + fN - 1);
    uniform_int_distribution<int> unif_int_interval(-100, 100);
    uniform_real_distribution<float> unif_float(1e-2, 1e2);
    uniform_real_distribution<float> unif_float2(-1.0f, 1.0f);

    double old_rel_er, new_rel_er, rel_er;
    int* indexes = new int[iN + fN];
    int* iCoefs_temp = new int[iN];

```

```

float* fCoefs_temp = new float[fN];
int iteration = 0;
int stop = 0;
do{
    old_rel_er = get_max_relative_error(float_function, iCoefs0, iN, fCoefs0, fN);
    bool jump = false;
    if(iteration > 0 && unif_int_interval(eng) > 40 - min(130, stop*15)){
        jump = true;
        printf("Jump!\n");
        for(int i = 0; i < iN; i++){
            iCoefs_temp[i] = iCoefs0[i];
            iCoefs0[i] += unif_float2(eng)*100*(iteration + 1)*iCoefs_step[i];
            if(iCoefs0[i] < iCoefs_min[i])
                iCoefs0[i] = iCoefs_min[i];
            if(iCoefs0[i] > iCoefs_max[i])
                iCoefs0[i] = iCoefs_max[i];
            printf("!!! New iCoef[%d] = 0x%x\n", i, iCoefs0[i]);
        }
        for(int f = 0; f < fN; f++){
            fCoefs_temp[f] = fCoefs0[f];
            fCoefs0[f] += unif_float2(eng)*100*(iteration + 1)*fCoefs_step[f];
            if(fCoefs0[f] < fCoefs_min[f])
                fCoefs0[f] = fCoefs_min[f];
            if(fCoefs0[f] > fCoefs_max[f])
                fCoefs0[f] = fCoefs_max[f];
            printf("!!! New fCoef[%d] = %.8e\n", f, fCoefs0[f]);
        }
        new_rel_er = get_max_relative_error(float_function, iCoefs0, iN, fCoefs0,
fN);
        printf("Max_rel_er = %.10e\n\n", new_rel_er);
    }
    int f = 0;
    int t_count = jump ? unif(eng) + 1 : 2;
    bool t_break = false;
    for(int t = 0; t < t_count; t++){
        {
            printf("t = %d/%d\n", t + 1, t_count);
            for(int i = 0; i < iN + fN; i++){
                indexes[i] = unif(eng);
                if(i == 0 && iN + fN > 1 && indexes[i] == indexes[iN + fN - 1]){
                    i--;
                } else {
                    for(int j = 0; j < i; j++){
                        if(indexes[j] == indexes[i]){
                            i--;
                            break;
                        }
                    }
                }
            }
        }
        for(int i = 0; i < iN + fN; i++){
            printf("%d ", indexes[i]);
        }
        printf("\n");
        for(int i = 0; i < iN + fN; i++){
            if(indexes[i] < iN){
                f = indexes[i];
                if(unif_int_interval(eng) > 50){
                    iCoefs_step[f] += unif_int_interval(eng)*0.2f*(iteration + 1);
                    if(iCoefs_step[f] < 20){
                        iCoefs_step[f] = 20;
                    }
                    if(iCoefs_step[f] > 300){
                        iCoefs_step[f] = 300;
                    }
                }
            }
        }
        iteration++;
        stop++;
    }
} while (iteration < 1000);

```

```

    }
    printf("!!! New iCoef_step[%d] = %d\n", f, iCoefs_step[f]);
}
iCoefs0[f] =
get_optimized_integer_parameter_with_golden_search_and_random_intervals
(float_function, iCoefs_min[f], iCoefs_max[f], iCoefs_step[f], f, iCoefs0, iN,
fCoefs0, fN, eng, randomIntervalsCount);
} else {
    f = indexes[i] - iN;
    if(unif_int_interval(eng) > 50){
        fCoefs_step[f] *= unif_float(eng)*0.2f*(iteration + 1);
        if(fCoefs_step[f] < 0.5e-6){
            fCoefs_step[f] = 0.5e-6;
        }
        if(fCoefs_step[f] > 2e-5){
            fCoefs_step[f] = 2e-5;
        }
        printf("!!! New fCoef_step[%d] = %.8e\n", f, fCoefs_step[f]);
    }
    fCoefs0[f] =
get_optimized_float_parameter_with_golden_search_and_random_intervals
(float_function, fCoefs_min[f], fCoefs_max[f], fCoefs_step[f], f, iCoefs0, iN,
fCoefs0, fN, eng, randomIntervalsCount);
}
}
if(jump){
    rel_er = get_max_relative_error(float_function, iCoefs0, iN, fCoefs0, fN);
    printf("Max_rel_er = %.10e\n\n", rel_er);
    if(rel_er > 3*old_rel_er){
        t_break = true;
        printf("t break!\n");
        break;
    }
}
else
{
    iteration++;
}
}
printf("Optimized coefficients (Iteration %d):\n", iteration);
for(int i = 0; i < iN; i++){
    printf("iCoef[%d] = 0xx\n", i, iCoefs0[i]);
    printf("iCoef_step[%d] = %d\n", i, iCoefs_step[i]);
}
for(int f = 0; f < fN; f++){
    printf("fCoef[%d] = %.8e\n", f, fCoefs0[f]);
    printf("fCoef_step[%d] = %.8e\n", f, fCoefs_step[f]);
}
if(!t_break){
    rel_er = get_max_relative_error(float_function, iCoefs0, iN, fCoefs0, fN);
}
if(jump && (t_break || rel_er > old_rel_er)){
    printf("Max_rel_er = %.10e\n\n", rel_er);
    printf("Jump back!\n");
    for(int i = 0; i < iN; i++){
        iCoefs0[i] = iCoefs_temp[i];
        printf("iCoef[%d] = 0xx\n", i, iCoefs0[i]);
        printf("iCoef_step[%d] = %d\n", i, iCoefs_step[i]);
    }
    for(int f = 0; f < fN; f++){
        fCoefs0[f] = fCoefs_temp[f];
        printf("fCoef[%d] = %.8e\n", f, fCoefs0[f]);
        printf("fCoef_step[%d] = %.8e\n", f, fCoefs_step[f]);
    }
}

```

```

    printf("Max_rel_er = %.10e\n\n", old_rel_er);
} else {
    if (abs(rel_er - old_rel_er) > rel_er_eps)
        stop = 0;
    else
        stop++;
    printf("Max_rel_er = %.10e, stop = %d\n\n", rel_er, stop);
}
} while (abs(rel_er - old_rel_er) > rel_er_eps || stop < 7);
printf("Done! Epsilon = %.6e, stop = %d", rel_er_eps, stop);
return rel_er;
}

```

Б.3.3 Метод рандомізованої багатовимірної жадібної оптимізації на основі методу багатовимірного прямого пошуку, методу золотого перетину з випадковим розбиттям проміжків та випадковим вибором серед рівнозначних параметрів

Опис: Версія методу *get_optimized_multidimensional_parameters_with_randomized_greedy_golden_search_and_random_intervals* (див. додаток Б.3.2) для пошуку оптимальних параметрів алгоритму для чисел типу *float*, що використовує постійний крок та випадковий вибір серед значень параметрів, що відповідають однаковим мінімальним значенням максимальних відносних похибок. Набір таких параметрів у форматі *vector<int>* та *vector<float>* формується відповідними методами *get_optimized_integer_parameters_with_golden_search_and_random_intervals* та *get_optimized_float_parameters_with_golden_search_and_random_intervals*. Аналогічно для алгоритмів подвійної точності.

Аргументи: Такий самий набір аргументів як у методі *get_optimized_multidimensional_parameters_with_randomized_greedy_golden_search_and_random_intervals*.

Вихідний код:

```

double
get_optimized_multidimensional_parameters_with_randomized_greedy_golden_search_and
_random_intervals_with_randomized_selection(
float_parameterized_multidimensional_function_type float_function, int* iCoefs0,
int iN, float* fCoefs0, int fN, int* iCoefs_min, int* iCoefs_max, float*
fCoefs_min, float* fCoefs_max, int* iCoefs_step, float* fCoefs_step, double
rel_er_eps, default_random_engine eng, int randomIntervalsCount ){
    uniform_int_distribution<int> unif(0, iN + fN - 1);
    ...
    uniform_real_distribution<float> unif_float3(0.0f, 1.0f);

    double old_rel_er, new_rel_er, rel_er;
    int* indexes = new int[iN + fN];
    int* iCoefs_temp = new int[iN];

```

```

float* fCoefs_temp = new float[fN];
int iteration = 0;
int stop = 0;
do{
    old_rel_er = get_max_relative_error(float_function, iCoefs0, iN, fCoefs0, fN);
    bool jump = false;
    if(iteration > 0 && unif_int_interval(eng) > 40 - min(130, stop*15)){
        jump = true;
        printf("Jump!\n");
        ...
        new_rel_er = get_max_relative_error(float_function, iCoefs0, iN, fCoefs0,
fN);
        printf("Max_rel_er = %.10e\n\n", new_rel_er);
    }
    int f = 0;
    int t_count = jump ? unif(eng) + 1 : 2;
    bool t_break = false;
    for(int t = 0; t < t_count; t++){
        {
            ...
            for(int i = 0; i < iN + fN; i++){
                printf("%d ", indexes[i]);
            }
            printf("\n");
            for(int i = 0; i < iN + fN; i++){
                if(indexes[i] < iN){
                    f = indexes[i];
                    ...
                    vector<int> v_iCoefs =
get_optimized_integer_parameters_with_golden_search_and_random_intervals(float_fun
ction, iCoefs_min[f], iCoefs_max[f], iCoefs_step[f], f, iCoefs0, iN, fCoefs0, fN,
eng, randomIntervalsCount);
                    int randomIndex = unif_float3(eng)*v_iCoefs.size();
                    iCoefs0[f] = v_iCoefs.at(randomIndex);
                } else {
                    f = indexes[i] - iN;
                    vector<float> v_fCoefs =
get_optimized_float_parameters_with_golden_search_and_random_intervals
(float_function, fCoefs_min[f], fCoefs_max[f], fCoefs_step[f], f, iCoefs0, iN,
fCoefs0, fN, eng, randomIntervalsCount);
                    int randomIndex = unif_float3(eng)*v_fCoefs.size();
                    fCoefs0[f] = v_fCoefs.at(randomIndex);
                }
            }
            ...
        }
        printf("Optimized coefficients (Iteration %d):\n", iteration);
        ...
    } while(abs(rel_er - old_rel_er) > rel_er_eps || stop < 7);
    printf("Done! Epsilon = %.6e, stop = %d", rel_er_eps, stop);
    return rel_er;
}

```

Б.3.4 Метод рандомізованої багатовимірної жадібної оптимізації на основі методу багатовимірного прямого пошуку з повним перебором та випадковим вибором серед рівнозначних параметрів

Опис: Приклад методу пошуку оптимальних параметрів алгоритму для чисел типу *float* на основі рандомізованого багатовимірного прямого пошуку з повним

перебором значень. При цьому використовується випадковий вибір серед значень параметрів, що відповідають однаковим мінімальним значенням максимальних відносних похибок. Аналогічно для алгоритмів подвійної точності, де може застосовуватись перебір значень із визначеним кроком.

Аргументи: Шаблон алгоритму *float_function* типу *float_parameterized_multidimensional_function_type*, початкові значення параметрів *iCoefs0* та *fCoefs0*, інші параметри оптимізації, стандартне ядро ГПВЧ *eng* та критерії зупинки методу *rel_er_eps* та *maxStopIterations*.

Вихідний код:

```
double
get_optimized_multidimensional_parameters_with_randomized_greedy_bruteforce_search
_with_randomized_selection( float_parameterized_multidimensional_function_type
float_function, int* iCoefs0, int iN, float* fCoefs0, int fN, int* iCoefs_min,
int* iCoefs_max, float* fCoefs_min, float* fCoefs_max, int* iCoefs_step, float*
fCoefs_step, double rel_er_eps, default_random_engine eng, int maxStopIterations
){
    uniform_int_distribution<int> unif(0, iN + fN - 1);
    uniform_int_distribution<int> unif_int_interval(-100, 100);
    uniform_real_distribution<float> unif_float(1e-2, 1e2);
    uniform_real_distribution<float> unif_float2(-1.0f, 1.0f);
    uniform_real_distribution<float> unif_float3(0.0f, 1.0f);

    double old_rel_er, new_rel_er, rel_er;
    int* indexes = new int[iN + fN];
    int* iCoefs_temp = new int[iN];
    float* fCoefs_temp = new float[fN];
    int iteration = 0;
    int stop = 0;
    do{
        old_rel_er = get_max_relative_error(float_function, iCoefs0, iN, fCoefs0, fN);
        bool jump = false;
        if(iteration > 0 && unif_int_interval(eng) > 40 - min(130, stop*15)){
            jump = true;
            printf("Jump!\n");
            for(int i = 0; i < iN; i++){
                iCoefs_temp[i] = iCoefs0[i];
                iCoefs0[i] += unif_float2(eng)*100*(iteration + 1)*iCoefs_step[i];
                ...
            }
            printf("!!! New iCoef[%d] = 0x%x\n", i, iCoefs0[i]);
        }
        for(int f = 0; f < fN; f++){
            fCoefs_temp[f] = fCoefs0[f];
            fCoefs0[f] += unif_float2(eng)*100*(iteration + 1)*fCoefs_step[f];
            ...
        }
        printf("!!! New fCoef[%d] = %.8e\n", f, fCoefs0[f]);
    }
    new_rel_er = get_max_relative_error(float_function, iCoefs0, iN, fCoefs0,
fN);
    printf("Max_rel_er = %.10e\n\n", new_rel_er);
} else {
    for(int i = 0; i < iN; i++){
        iCoefs_temp[i] = iCoefs0[i];
    }
}
```

```

    for(int f = 0; f < fN; f++){
        fCoefs_temp[f] = fCoefs0[f];
    }
}
int f = 0;
int t_count = 2;
for(int t = 0; t < t_count; t++)
{
    printf("t = %d/%d\n", t + 1, t_count);
    for(int i = 0; i < iN + fN; i++){
        indexes[i] = unif(eng);
        ...
    }
    for(int i = 0; i < iN + fN; i++){
        printf("%d ", indexes[i]);
    }
    printf("\n");
    for(int i = 0; i < iN + fN; i++){
        if(indexes[i] < iN){
            f = indexes[i];
            if(unif_int_interval(eng) > 50){
                iCoefs_step[f] += unif_int_interval(eng)*0.2f*(iteration + 1);
                ...
                printf("!!! New iCoef_step[%d] = %d\n", f, iCoefs_step[f]);
            }
            vector<int> v_iCoefs =
get_optimized_integer_parameters_with_bruteforce(float_function, iCoefs_min[f],
iCoefs_max[f], iCoefs_step[f], true, f, iCoefs0, iN, fCoefs0, fN);
            int randomIndex = unif_float3(eng)*v_iCoefs.size();
            iCoefs0[f] = v_iCoefs.at(randomIndex);
        } else {
            f = indexes[i] - iN;
            if(unif_int_interval(eng) > 50){
                fCoefs_step[f] *= unif_float(eng)*0.2f*(iteration + 1);
                ...
                printf("!!! New fCoef_step[%d] = %.8e\n", f, fCoefs_step[f]);
            }
            vector<float> v_fCoefs =
get_optimized_integer_parameters_with_bruteforce(float_function, fCoefs_min[f],
fCoefs_max[f], fCoefs_step[f], true, f, iCoefs0, iN, fCoefs0, fN);
            int randomIndex = unif_float3(eng)*v_fCoefs.size();
            fCoefs0[f] = v_fCoefs.at(randomIndex);
        }
    }
    if(!jump){
        iteration++;
    }
}
printf("Optimized coefficients (Iteration %d):\n", iteration);
for(int i = 0; i < iN; i++){
    printf("iCoef[%d] = 0x%x\n", i, iCoefs0[i]);
    printf("iCoef_step[%d] = %d\n", i, iCoefs_step[i]);
}
for(int f = 0; f < fN; f++){
    printf("fCoef[%d] = %.8e\n", f, fCoefs0[f]);
    printf("fCoef_step[%d] = %.8e\n", f, fCoefs_step[f]);
}
rel_er = get_max_relative_error(float_function, iCoefs0, iN, fCoefs0, fN);
if(rel_er > old_rel_er){
    printf("Max_rel_er = %.10e\n\n", rel_er);
    printf("Jump back!\n");
    for(int i = 0; i < iN; i++){
        iCoefs0[i] = iCoefs_temp[i];
        printf("iCoef[%d] = 0x%x\n", i, iCoefs0[i]);
    }
}

```

```

    printf("iCoef_step[%d] = %d\n", i, iCoefs_step[i]);
}
for(int f = 0; f < fN; f++){
    fCoefs0[f] = fCoefs_temp[f];
    printf("fCoef[%d] = %.8e\n", f, fCoefs0[f]);
    printf("fCoef_step[%d] = %.8e\n", f, fCoefs_step[f]);
}
printf("Max_rel_er = %.10e\n\n", old_rel_er);
} else {
    if(abs(rel_er - old_rel_er) > rel_er_eps)
        stop = 0;
    else
        stop++;
    printf("Max_rel_er = %.10e, stop = %d\n\n", rel_er, stop);
} while(abs(rel_er - old_rel_er) > rel_er_eps || stop < maxStopIterations);
printf("Done! Epsilon = %.6e, stop = %d", rel_er_eps, stop);
return rel_er;
}

```

Б.4 Визначення періоду повторення послідовності ПВЧ

Опис: Приклад реалізації консольної програми *calculate_period_of_the_sequence_from_input_file.exe* для визначення розміру в байтах періоду повторення *period* та зміщення *offset* у згенерованій послідовності ПВЧ на основі перебору значень. Результат виводиться в консоль. Значення *period* = 0 означає, що у вхідній послідовності періодичності не виявлено.

Аргументи: Перший аргумент *fileName* – повний шлях до вхідного бінарного файлу, другий аргумент *detailInfo* керує виведенням додаткової інформації, наприклад неперіодичних елементів (*offset*) та періодичних елементів (*period*) послідовності, їх гістограм тощо, наступні аргументи також керують процесом виведення інформації в консоль.

Вихідний код:

```

// calculate_period_of_the_sequence_from_input_file.cpp
#include <iostream>
#include <fstream>
#include <string>
#include <limits>
#include <stdexcept>
#include <sys/stat.h>

const int p=20;           // max periods for testing
int n;                    // generated sequence size
char *seq;                // generated sequence
int offset, period;       // generated properties
int i,j,k,ie,t0,T;
char e;
int *hcnt,hs;            // histogram

bool printPeriod = true;
bool printOffset = true;

```

```

...
bool printPeriodHistogram = true;
bool printOffsetHistogram = true;
bool printDetailInfo = false;

std::string trim_file_name(std::string fullFileName){
    int start_pos = fullFileName.rfind("/");
    if(start_pos != std::string::npos){
        fullFileName = fullFileName.substr(start_pos + 1, std::string::npos);
    }
    return fullFileName;
}

int main(int argc, char *argv[]){
    try {
        std::string fileName;
        bool detailInfo = false;
        ...

        if(argc == 1 && fileName == "")
            throw std::runtime_error("No file is selected! ");
        else {
            if(argc > 1 && argv[1] != "")
                fileName = argv[1];
            if(argc > 2 && argv[2] != ""){
                std::string detailInfoArg = std::string(argv[2]);
                if(detailInfoArg[0] == 'T' || detailInfoArg[0] == 't' || detailInfoArg[0] ==
'Y' || detailInfoArg[0] == 'y' || detailInfoArg[0] == '1' ){
                    detailInfo = true;
                } else if(detailInfoArg[0] == 'F' || detailInfoArg[0] == 'f' ||
detailInfoArg[0] == 'N' || detailInfoArg[0] == 'n' || detailInfoArg[0] == '0' ) {
                    detailInfo = false;
                }
            }
            ...
        }

        if(detailInfo){
            printPeriod = true;
            printOffset = true;
            ...
            printPeriodHistogram = true;
            printOffsetHistogram = true;
            printDetailInfo = false;
        } else {
            printPeriod = false;
            printOffset = false;
            ...
            printPeriodHistogram = false;
            printOffsetHistogram = false;
            printDetailInfo = false;
        }

        //read periodic sequence from file
        std::ifstream infile;
        infile.open(fileName, std::ios::in | std::ios::binary);
        if(!infile.is_open())
            throw std::runtime_error("Couldn't open the file! ");

        struct stat stat_buf;
        int rc = stat(fileName.c_str(), &stat_buf);
        unsigned long int tempN = (rc == 0 ? stat_buf.st_size : -1);
        printf("File: %s\n", trim_file_name(fileName).c_str());
    }
}

```

```

n = (tempN > std::numeric_limits<int>::max()) ? std::numeric_limits<int>::max()
: tempN;
printf("n = %d\n\n", n);
seq = new char[n];
if(!infile.read(seq, n))
    throw std::runtime_error("Failed to read data from the file! ");

// test possible periods
bool done = false;
for(int kk=1; kk <= maxK && kk <= (n>>1) && !done; kk++){
    bool ok = true;
    T = kk;
    if (printDetailInfo) printf("T = %d:\n", T);
    for(int s=0; s<T; s++){
        for (i=n-s-1, e=seq[i], k=0; (i>=(n>>1)) && (k<p) && (e==seq[i]); i-=T,
k++){
            if (printDetailInfo && (s < 100 || s >= T - 100)) printf("%hhu ", e);
            if (printDetailInfo && s == 100 && T > 200) printf("... ");
        }
        if (printDetailInfo && (s <= 100 || s >= T - 100)) printf("\n");
        if ((k < p) && (i >= (n>>1))) { if (printDetailInfo) printf("break\n"); ok =
false; break;}
    }
    if (ok) { if (printDetailInfo) printf("ok\n"); done = true; break; }
}

// search offset size O(n)
for (t0=-1, i=0; i<n-T; i++)
    if (seq[i]!=seq[i+T]) t0=i;
t0++;

...

printf("offset: %d\n", t0);
printf("period: %d\n", T);

if(printDetailInfo) printf("\n");
if(T > 1){
    if (printOffset && t0 != 0){
        printf("Offset: ");
        ...
        printf("\n\n");
    }
    hcnt = new int[256];
    if(printOffsetHistogram && t0 != 0){
        for (hs=0;hs<256;hs++) hcnt[hs] = 0;
        for (i=0;i<t0;i++){
            e=seq[i];
            hcnt[(unsigned char)e]++;
        }

        printf("\n");
        printf("Histogram (offset): \n");
        for(int s=0; s < hs; s++) printf("hval[%hhu] = %d\n", s, hcnt[s]);
        printf("\n");
    }
    if (printPeriod){
        printf("Period: ");
        ...
        printf("\n\n");
    }
}
if(printPeriodHistogram){
    // compute histogram O(T) on last multiple of period
    for (hs=0;hs<256;hs++) hcnt[hs] = 0;

```

```

    for (i=n-T; i<n; i++){
        e=seq[i];
        hcnt[(unsigned char)e]++;
    }

    printf("\n");
    printf("Histogram (period): \n");
    for(int s=0; s < hs; s++) printf("hval[%hu] = %d\n", s, hcnt[s]);
    printf("\n");
}
} catch(std::exception& e){
    std::cout << "Error: " << e.what() << std::endl;
}

delete[] seq;
delete[] hcnt;
return 0;
}

```

Додаток В

Програмний код розроблених bat-файлів для автоматизації дослідження ГПВБП

В.1 Статистичне тестування файлу із ПВБП з використанням набору тестів NIST STS

Опис: Скрипт *run_tests_one_file_with_termination.bat*, що реалізує тестування однієї ПВБП, збереженої у вигляді бінарного файлу *file*, із заданими параметрами (з можливістю примусової зупинки програми) та збереження результатів тестування NIST STS в окремому файлі.

Аргументи: Аргумент *tests* задає значення параметра *m*, а *folder* – папка розташування вхідного файлу *file*.

Вихідний код:

```
rem run_tests_one_file_with_termination.bat
@echo off
set tests=%1
set file=%2
set folder=%3
set /a counter=0
if exist %folder%\%file% ((echo 0 && echo %folder%\%file% && echo 1 && echo 0 &&
echo %tests% && echo 1) | (start /b assess.exe 1000000))
setlocal enabledelayedexpansion

:loop
timeout /t 1 /nobreak >nul 2>&1
tasklist | find /i "assess.exe" >nul 2>&1
if errorlevel 1 goto cont
set /a counter+=1
if %tests% == 10 if !counter! gtr 200 (
    taskkill /f /im "assess.exe"
    echo The time limit exceeded !counter! sec
)
goto loop

:cont
endlocal
if exist %folder%\%file% echo f | xcopy
experiments\AlgorithmTesting\finalAnalysisReport.txt
experiments\AlgorithmTesting\results\%folder%\%file%_%tests%.txt
```

В.2 Аналіз результатів статистичного тестування NIST STS

Опис: Скрипт *parse_into_file_nist_results_one_file_in_folder_with_return.bat*, що опрацьовує текстовий файл *file* із результатами тестування NIST STS, здійснює підрахунок кількості провалених тестів, а результати зберігає у вихідний файл

fileToSave.

Аргументи: Аргумент *folder* – папка розташування вхідного текстового файлу *file* із результатами.

Вихідний код:

```
rem parse_into_file_nist_results_one_file_in_folder_with_return.bat
@echo off
set file=%1
set fileToSave=%2
set folder=%3
setlocal enabledelayedexpansion
set /a index=0
set /a count=0
if exist %folder%\%file% (
    if %file% == %fileToSave% (
        echo. >> %folder%\%fileToSave%
        echo. >> %folder%\%fileToSave%
    ) else (
        break > %folder%\%fileToSave%
    )
    echo Proportion: >> %folder%\%fileToSave%
    echo. >> %folder%\%fileToSave%
    for /f "usebackq eol= tokens=* delims= " %%a in (`findstr /n ^^^^
"%folder%\%file%"`) do (
        set line=%%a
        set "line=!line:*=! "
        if !index! gtr 6 if !index! lss 195 (
            set t1=!line:~52,10!
            set t2=!line:~61,3!
            for /f "tokens=* delims= " %%b in ("!t1!") do set t1=%%b
            for /l %%b in (1,1,5) do if "!t1:~-1!"==" " set t1=!t1:~0,-1!
            echo !t2!|find "*" >nul
            if errorlevel 1 (set /a count+=0) else (set /a count+=1)
            echo =!t1!>> %folder%\%fileToSave%
        )
        set /a index+=1
    )
    echo. >> %folder%\%fileToSave%
    echo. >> %folder%\%fileToSave%
    echo Count: !count! >> %folder%\%fileToSave%
    echo Done file %folder%\%fileToSave%!
    if !index! geq 195 (
        set localReturn=!count!
    ) else (set localReturn=-1)
) else (set localReturn=-2)
endlocal & set result=%localReturn%
```

В.3 Статистичне тестування NIST STS набору файлів

Опис: Скрипт *run_tests_in_folder_with_delete.bat*, що виконує статистичне тестування усіх ПВБП, збережених у папці *folder*, обчислює кількість непройдених тестів NIST STS для кожної послідовності, знаходить їх період та зміщення, аналізує результати із збереженням їх в окремих файлах та видаленням вхідних послідовностей незадовільної якості.

Аргументи: Аргумент *tests* задає значення параметра *m* для статистичного тестування NIST, а *folder* – папка розташування вхідних файлів.

Вихідний код:

```
rem run_tests_in_folder_with_delete.bat
@echo off
set tests=%1
set folder=%2
setlocal enabledelayedexpansion
for /f %%G in ('dir /b /a-d %folder%') do (
    echo %%~G:
    if %tests% == 10 (call run_tests_one_file_with_termination.bat %tests% %%~G
%folder%) else (call run_tests_one_file.bat %tests% %%~G %folder%)
)
echo Analyzing results..
timeout 1 /nobreak
for /f %%G in ('dir /b /a-d %folder%') do (
    set input_file=%%~G_%tests%.txt
    set output_file=%%~G_%tests%.txt
    if not exist "experiments\AlgorithmTesting\results\%folder%" mkdir
"experiments\AlgorithmTesting\results\%folder%"
    call parse_into_file_nist_results_one_file_in_folder_with_return.bat
!input_file! !output_file! experiments\AlgorithmTesting\results\%folder%
    call parse_into_file_period_results_one_file_in_folder_with_return.bat %%~G
experiments\AlgorithmTesting\results\%folder%\!output_file! %folder%
    set toDel=false
    if not !periodRes! == 0 (
        set /a "sum = !offsetRes!+!periodRes!"
        if %tests% == 10 (if !sum! leq 625000 (
            set toDel=true
        ))
        if %tests% == 100 (if !sum! leq 6250000 (
            set toDel=true
        ))
        if %tests% == 1000 (if !sum! leq 62500000 (
            set toDel=true
        ))
        if "!toDel!" == "true" (
            del %folder%\%%~G
            echo Offset: !offsetRes!
            echo Period: !periodRes!
            if not exist %folder%\%%~G echo *** File %folder%\%%~G is deleted!
        )
    )
    if "!toDel!" == "false" (if !result! == -2 (echo *** File experiments\
AlgorithmTesting\results\%folder%\!input_file! is not found!) else (
        if !result! == -1 del %folder%\%%~G
        if !result! gtr 2 (
            del %folder%\%%~G
            echo Count: !result!
        )
        if not exist %folder%\%%~G echo *** File %folder%\%%~G is deleted!
    ))
)
endlocal
echo Done!
```

В.4 Статистичне тестування NIST STS набору файлів із автоматичною зміною параметра m

Опис: Скрипт *run_all_tests_in_folder_with_delete.bat*, що виконує статистичне тестування та визначення періоду повторення усіх ПВБП, збережених у папці *folder*, із поступовим збільшенням кількості протестованих підпоследовностей з множини $m \in \{10, 100, 1000\}$ та видаленням последовностей незадовільної якості.

Аргументи: *folder* – папка розташування вхідних бінарних файлів із ПВБП, а аргумент *tests_level* $\in \{1, 2, 3\}$ визначає максимальний рівень $m \in \{10, 100, 1000\}$ для тестування.

Вихідний код:

```
rem run_all_tests_in_folder_with_delete.bat
@echo off
set folder=%1
set tests_level=%2
if %tests_level% gtr 0 (
    echo Level 1: 10 bitstreams /at least small files/
    call run_tests_in_folder_with_delete.bat 10 %folder%
    if %tests_level% gtr 1 (
        echo Level 2: 100 bitstreams /at least big files/
        timeout 1 /nobreak
        call run_tests_in_folder_with_delete.bat 100 %folder%
        if %tests_level% gtr 2 (
            echo Level 3: 1000 bitstreams /only large files/
            timeout 1 /nobreak
            call run_tests_in_folder_with_delete.bat 1000 %folder%
        )
    )
)
echo Done!
```

В.5 Виклик процедури для визначення періоду повторення последовності ПВЧ та опрацювання результатів

Опис: Скрипт *parse_into_file_period_results_one_file_in_folder_with_return.bat*, що здійснює виклик програми *calculate_period_of_the_sequence_from_input_file.exe* для визначення періоду повторення *period*, зміщення *offset* та загального розміру в байтах последовності ПВЧ. Результат записується в файл та повертається скрипту вищого рівня для подальшого опрацювання.

Аргументи: Аргумент *file* – назва вхідного файлу, що містить последовність ПВЧ для тестування, *fileToSave* – вихідний текстовий файл для збереження результатів, а *folder* – папка розташування вхідного файлу.

Вихідний код:

```

rem parse_into_file_period_results_one_file_in_folder_with_return.bat
@echo off
set file=%1
set fileToSave=%2
set folder=%3
setlocal enabledelayedexpansion
set /a offset=0
set /a period=0
if exist %folder%\%file% (
    if exist %fileToSave% (
        echo. >> %fileToSave%
        echo. >> %fileToSave%
    ) else (
        break > %fileToSave%
    )
)

FOR /F "tokens=*" %%a IN ('calculate_period_of_the_sequence_from_input_file.exe
%folder%\%file%') do (
    SET VAR=%%a
    if not "!VAR!"=="!VAR:n=!" (
        set n=!VAR:n=!
        set n=!n:~2!
        set localFileSizeReturn=!n!
        echo FileSize: !n! >> %fileToSave%
    )
    if not "!VAR!"=="!VAR:offset=!" (
        set offset=!VAR:offset=!
        set localOffsetReturn=!offset!
        echo Offset: !offset! >> %fileToSave%
    )
    if not "!VAR!"=="!VAR:period=!" (
        set period=!VAR:period=!
        set localPeriodReturn=!period!
        echo Period: !period! >> %fileToSave%
    )
)
)
endlocal & set periodRes=%localPeriodReturn%& set offsetRes=%localOffsetReturn%&
set fileSizeRes=%localFileSizeReturn%

```

Додаток Г

Порівняння відомих та запропонованих алгоритмів обчислення елементарних функцій на основі методу МС

Таблиця Г.1 – Порівняння відомих модифікацій алгоритму FISR для FP чисел одинарної точності

Алгоритм	Бліна [6]	FISR [137]	Ломонта [120]	Ломонта [120]	Вальчика та ін. [147]	Вальчика та ін. [140]	Кадлеця [149]	Вальчика та ін. [148]
Метод магічної константи								
R	0x5F400000	0x5F3759DF	0x5F37642F	0x5F375A86	0x5F375A86	0x5F376908	0x5F1FFFF9	0x5F200000
$\delta_{0 \max}$	$8,8662 \cdot 10^{-2}$	$3,4376 \cdot 10^{-2}$	$3,4213 \cdot 10^{-2}$	$3,4365 \cdot 10^{-2}$	$3,4365 \cdot 10^{-2}$	$3,4332 \cdot 10^{-2}$	$1,3398 \cdot 10^{-1}$	$1,3397 \cdot 10^{-1}$
Перша модифікована ітерація NR								
Структура ітерації NR	(1.33)	(1.31)	(1.31)	(1.31)	(1.33)	(1.33)	(1.36)	(1.33)
k_1	1,47	1,5	1,5	1,5	1,5013145	1,5008789	0,703952253	1,68191391
k_2	0,47	0,5	0,5	0,5	0,50043818	0,5	2,38924456	0,703952009
Кількість FP операцій	4×, 1–	4×, 1–	4×, 1–	4×, 1–	4×, 1–	4×, 1–	4×, 1–	4×, 1–
$\delta_{1 \max}$	$6,0919 \cdot 10^{-3}$	$1,7523 \cdot 10^{-3}$	$1,7759 \cdot 10^{-3}$	$1,7513 \cdot 10^{-3}$	$8,7653 \cdot 10^{-4}$	$8,7922 \cdot 10^{-4}$	$6,5020 \cdot 10^{-4}$	$6,5029 \cdot 10^{-4}$
Друга модифікована ітерація NR								
Структура ітерації NR	(1.31)	(1.31)	(1.31)	(1.31)	(1.34)	(1.35)	(1.31)	(1.35)
k_3	1,5	1,5	1,5	1,5	1,5000008	1,5000006	1,5	1,50000036
k_4	0,5	0,5	0,5	0,5	0,99912498	0,5	0,5	0,500000053
Кількість FP операцій	4×, 1–	3×, 1–	3×, 1–	3×, 1–	4×, 1–	3×, 1–	4×, 1–	4×, 1–
$\delta_{2 \max}$	$5,5684 \cdot 10^{-5}$	$4,7330 \cdot 10^{-6}$	$4,8626 \cdot 10^{-6}$	$4,7348 \cdot 10^{-6}$	$7,3791 \cdot 10^{-7}$	$7,3675 \cdot 10^{-7}$	$7,6630 \cdot 10^{-7}$	$4,8605 \cdot 10^{-7}$
Загальна кількість FP операцій								
Кількість ×	8	7	7	7	8	7	8	8
Кількість +/-	2	2	2	2	2	2	2	2

Таблиця Г.2 – Порівняння відомих модифікацій алгоритму FISR для FP чисел подвійної точності

Алгоритм	Бліна [6]	Робертсона [23]	Вальчика та ін. [147]	Вальчика та ін. [140]	Вальчика та ін. [148]
Метод магічної константи					
R	0x5FE8000 000000000	0x5FE6EB5 0C7B537A9	0x5FE6EB5 0C7B537A9	0x5FE6ED2 102DCBFDA	0x5FE4000 00000000C
$\delta_{0 \max}$	$8,8662 \cdot 10^{-2}$	$3,4365 \cdot 10^{-2}$	$3,4365 \cdot 10^{-2}$	$3,4332 \cdot 10^{-2}$	$1,3397 \cdot 10^{-1}$
Перша модифікована ітерація NR					
Структура ітерації NR	(1.33)	(1.31)	(1.33)	(1.33)	(1.33)
k_1	1,47	1,5	1,50131453 875281472	1,50087895 51163346	1,68191390 868723079
k_2	0,47	0,5	0,50043817 9584271573	0,5	0,70395200 9104829370
Кількість FP операцій	4×, 1–	4×, 1–	4×, 1–	4×, 1–	4×, 1–
$\delta_{1 \max}$	$6,0918 \cdot 10^{-3}$	$1,7512 \cdot 10^{-3}$	$8,7636 \cdot 10^{-4}$	$8,7908 \cdot 10^{-4}$	$6,5007 \cdot 10^{-4}$
Друга модифікована ітерація NR					
Структура ітерації NR	(1.31)	(1.31)	(1.34)	(1.35)	(1.35)
k_3	1,5	1,5	1,50000086 42589575	1,50000057 96762577	1,50000036 976749938
k_4	0,5	0,5	0,99912498 3832536169	0,5	0,50000005 2823927419
Кількість FP операцій	4×, 1–	3×, 1–	4×, 1–	3×, 1–	4×, 1–
$\delta_{2 \max}$	$5,5552 \cdot 10^{-5}$	$4,5973 \cdot 10^{-6}$	$5,7617 \cdot 10^{-7}$	$5,7968 \cdot 10^{-7}$	$3,1694 \cdot 10^{-7}$

Продовження таблиці Г.2

Алгоритм	Бліна [6]	Робертсона [23]	Вальчика та ін. [147]	Вальчика та ін. [140]	Вальчика та ін. [148]
Третя модифікована ітерація NR					
Структура ітерації NR	(1.31)	(1.31)	(1.35)	(1.35)	(1.31)
k_5	1,5	1,5	1,50000000 000024925	1,50000000 0000252	1,5
k_6	0,5	0,5	0,5	0,5	0,5
Кількість FP операцій	3×, 1–	3×, 1–	4×, 1–	3×, 1–	4×, 1–
$\delta_{3 \max}$	$4,6290 \cdot 10^{-9}$	$3,1703 \cdot 10^{-11}$	$2,4962 \cdot 10^{-13}$	$2,5229 \cdot 10^{-13}$	$1,5093 \cdot 10^{-13}$
Загальна кількість FP операцій					
Кількість ×	11	10	12	10	12
Кількість +/–	3	3	3	3	3

Таблиця Г.3 – Порівняння відомих модифікацій алгоритму FISR на основі ітераційних методів Хаусхолдера вищого порядку для FP чисел одинарної точності

Алгоритм	Ломонта [120]	Метод Хаусхолдера [150]				Леметра та ін. [1,151]			
Метод магічної константи									
R	0x5F375A86								
$\delta_{0\max}$	$3,4365 \cdot 10^{-2}$								
Ітерація методу Н						Ітерація модифікованого методу Н			
Порядок	1	2	3	4	5	2	3	4	5
Структура ітерації Н	(1.31)	(1.38), (1.39)	(1.39), (1.41)	(1.39), (1.40)	(1.39), (1.41)	(1.39), (1.43)	(1.39), (1.43)	(1.39), (1.43)	(1.39), (1.43)
k_1	1,5	3	1	5	1	1,875	2,1875	2,4609375	2,70703125
k_2	-0,5	1	6	10	15	-1,25	-2,1875	-3,28125	-4,51171875
k_3	–	3	4	1	15	0,375	1,3125	2,953125	5,4140625
k_4	–	–	4	10	6	–	-0,3125	-1,40625	-3,8671875
k_5	–	–	–	5	20	–	–	0,2734375	1,50390625
k_6	–	–	–	–	6	–	–	–	-0,24609375
Перша ітерація Н									
Кількість FP операцій	3×, 1FMA	1÷, 3×, 1FMA, 1+	1÷, 4×, 2FMA, 1+	1÷, 3×, 3FMA, 1+	1÷, 4×, 4FMA, 1+	3×, 2FMA	3×, 3FMA	3×, 4FMA	3×, 5FMA
$\delta_{1\max}$	$1,7513 \cdot 10^{-3}$	$1,0848 \cdot 10^{-5}$	$3,7545 \cdot 10^{-7}$	$2,6767 \cdot 10^{-7}$	$2,9775 \cdot 10^{-7}$	$1,0072 \cdot 10^{-4}$	$6,2643 \cdot 10^{-6}$	$6,5782 \cdot 10^{-7}$	$5,2132 \cdot 10^{-7}$
Друга ітерація Н									
Кількість FP операцій	2×, 1FMA	1÷, 3×, 1FMA, 1+	1÷, 4×, 2FMA, 1+	1÷, 3×, 3FMA, 1+	1÷, 4×, 4FMA, 1+	3×, 2FMA	–	–	–
$\delta_{2\max}$	$4,7306 \cdot 10^{-6}$	$2,5364 \cdot 10^{-7}$	$1,9322 \cdot 10^{-7}$	$1,9186 \cdot 10^{-7}$	$1,8108 \cdot 10^{-7}$	$1,6519 \cdot 10^{-7}$	–	–	–
Третя ітерація Н									
Кількість FP операцій	2×, 1FMA	–	–	–	–	3×, 2FMA	–	–	–
$\delta_{3\max}$	$1,4639 \cdot 10^{-7}$	–	–	–	–	$1,0634 \cdot 10^{-7}$	–	–	–

Таблиця Г.4 – Порівняння відомих модифікацій алгоритму FISR на основі ітераційних методів Хаусхолдера вищого порядку для FP чисел подвійної точності

Алгоритм	Леметра та ін. [1,23]	Метод Хаусхолдера [150]				Леметра та ін. [1,151]			
Метод магічної константи									
R	0x5FE6EB50C7B537A9								
$\delta_{0\max}$	$3,4365 \cdot 10^{-2}$								
Ітерація методу Н						Ітерація модифікованого методу Н			
Порядок	1	2	3	4	5	2	3	4	5
Перша ітерація Н									
Кількість FP операцій	3×, 1FMA	1÷, 3×, 1FMA, 1+	1÷, 4×, 2FMA, 1+	1÷, 3×, 3FMA, 1+	1÷, 4×, 4FMA, 1+	3×, 2FMA	3×, 3FMA	3×, 4FMA	3×, 5FMA
$\delta_{1\max}$	$1,7512 \cdot 10^{-3}$	$1,0688 \cdot 10^{-5}$	$1,8686 \cdot 10^{-7}$	$3,2668 \cdot 10^{-9}$	$5,7115 \cdot 10^{-11}$	$1,0057 \cdot 10^{-4}$	$6,0612 \cdot 10^{-6}$	$3,7719 \cdot 10^{-7}$	$2,3875 \cdot 10^{-8}$
Друга ітерація Н									
Кількість FP операцій	2×, 1FMA	1÷, 3×, 1FMA, 1+	1÷, 4×, 2FMA, 1+	1÷, 3×, 3FMA, 1+	1÷, 4×, 4FMA, 1+	3×, 2FMA	3×, 3FMA	3×, 4FMA	3×, 5FMA
$\delta_{2\max}$	$4,5973 \cdot 10^{-6}$	$6,8934 \cdot 10^{-16}$	$4,5829 \cdot 10^{-16}$	$4,3845 \cdot 10^{-16}$	$3,8402 \cdot 10^{-16}$	$2,5434 \cdot 10^{-12}$	$4,9006 \cdot 10^{-16}$	$5,8807 \cdot 10^{-16}$	$8,7549 \cdot 10^{-16}$
Третя ітерація Н									
Кількість FP операцій	2×, 1FMA	1÷, 3×, 1FMA, 1+	1÷, 4×, 2FMA, 1+	—	—	3×, 2FMA	3×, 3FMA	—	—
$\delta_{3\max}$	$3,1703 \cdot 10^{-11}$	$4,0072 \cdot 10^{-16}$	$3,5963 \cdot 10^{-16}$	—	—	$2,9371 \cdot 10^{-16}$	$3,0986 \cdot 10^{-16}$	—	—
Четверта ітерація Н									
Кількість FP операцій	2×, 1FMA	—	—	—	—	3×, 2FMA	—	—	—
$\delta_{4\max}$	$2,7636 \cdot 10^{-16}$	—	—	—	—	$1,9873 \cdot 10^{-16}$	—	—	—

Таблиця Г.5 – Порівняння алгоритмів обчислення інших степеневих функцій для FP чисел одинарної точності
на основі методу МС та класичних ітераційних методів NR

Степенева функція	$y = 1/x$				$y = 1/\sqrt[3]{x}$				$y = \sqrt[3]{x}$			$y = \sqrt{x}$	
Степінь p	-1				-1/3				1/3			1/2	
Алгоритм	Бліна [6]	Хуана та Чена [154]	Запропонований	Мороза та Гринчишина [152]	Бліна [6]	Кахана [155]	Левіна [156]	Запропонований	Бліна [6]	Кахана [155]	Запропонований	Бліна [6]	Запропонований
Метод МС													
Формула	(1.49), (1.50)				(1.46)				(1.46)			(1.46)	
k	0	0,0430356660280040159705095	0,050510227680206298828125	0,050510227680206298828125	0	0,049622263759374619	0,05011372169748866176428873	0,05008985001643057652670048	0	0,05048386276117218153660137	0,04929778899646452060515235	0	0,0732920709942346873790556
R	0x7F000000	0x7EF4FB9D	0x7EF311C3	0x7EF311C3	0x54AA AAAA	0x54A232A3	0x54A21D2A	0x54A21E35	0x2A555556	0x2A51067F	0x2A512068	0x1FC00000	0x1FBB4F2E
$\delta_{0 \max}$	$1,2500 \cdot 10^{-1}$	$6,1373 \cdot 10^{-2}$	$5,0510 \cdot 10^{-2}$	$5,0510 \cdot 10^{-2}$	$8,1687 \cdot 10^{-2}$	$3,4241 \cdot 10^{-2}$	$3,4592 \cdot 10^{-2}$	$3,4575 \cdot 10^{-2}$	$5,8267 \cdot 10^{-2}$	$3,1555 \cdot 10^{-2}$	$3,2182 \cdot 10^{-2}$	$6,0660 \cdot 10^{-2}$	$3,4747 \cdot 10^{-2}$
Бітова точність	3	4,03	4,31	4,31	3,61	4,87	4,85	4,85	4,1	4,99	4,96	4,04	4,85
	Ітерація методу NR	Ітерація методу Гольдшмідта		Ітерація методу NR									
Структура ітерації NR	(1.47)	(1.53)-(1.55)		(1.47)	(1.47)				(1.47)			(1.48)	
k_1	1,0	–		1,0	1,33333333				0,66666666			0,5	
k_2	2,0	–		2,0	0,33333333				0,33333333			–	

Продовження таблиці Г.5

Степенева функція	$y = 1/x$				$y = 1/\sqrt[3]{x}$				$y = \sqrt[3]{x}$			$y = \sqrt{x}$	
Алгоритм	Бліна [6]	Хуана та Чена [154]	Запропонований	Мороза та Гринчишина [152]	Бліна [6]	Кахана [155]	Левіна [156]	Запропонований	Бліна [6]	Кахана [155]	Запропонований	Бліна [6]	Запропонований
Перша ітерація NR		Перша ітерація Гольдшмідта		Перша ітерація NR									
Кількість FP операцій	2×, 1–	2FMA		2×, 1–	5×, 1–				1÷, 3×, 1+			1÷, 1×, 1+	
$\delta_{1 \max}$	$1,5625 \cdot 10^{-2}$	$3,7666 \cdot 10^{-3}$	$2,5513 \cdot 10^{-3}$	$2,5514 \cdot 10^{-3}$	$1,4087 \cdot 10^{-2}$	$2,3989 \cdot 10^{-3}$	$2,3386 \cdot 10^{-3}$	$2,3363 \cdot 10^{-3}$	$3,1494 \cdot 10^{-3}$	$1,0394 \cdot 10^{-3}$	$9,9307 \cdot 10^{-4}$	$1,7347 \cdot 10^{-3}$	$6,2550 \cdot 10^{-4}$
Бітова точність	6	8,05	8,61	8,61	6,15	8,7	8,74	8,74	8,31	9,91	9,98	9,17	10,64
Друга ітерація NR		Друга ітерація Гольдшмідта		Друга ітерація NR									
Кількість FP операцій	2×, 1–	1×, 1FMA		2×, 1–	4×, 1–				1÷, 2×, 1+			1÷, 1×, 1+	
$\delta_{2 \max}$	$2,4427 \cdot 10^{-4}$	$1,4273 \cdot 10^{-5}$	$6,6186 \cdot 10^{-6}$	$6,6479 \cdot 10^{-6}$	$3,9330 \cdot 10^{-4}$	$1,1597 \cdot 10^{-5}$	$1,1037 \cdot 10^{-5}$	$1,1016 \cdot 10^{-5}$	$9,9933 \cdot 10^{-6}$	$1,2226 \cdot 10^{-6}$	$1,1308 \cdot 10^{-6}$	$1,5649 \cdot 10^{-6}$	$2,8161 \cdot 10^{-7}$
Бітова точність	12	16,1	17,21	17,2	11,31	16,4	16,47	16,47	16,61	19,64	19,75	19,29	21,76
Третя ітерація NR		Третя ітерація Гольдшмідта		Третя ітерація NR									
Кількість FP операцій	2×, 1–	1×, 1FMA		2×, 1–	4×, 1–				1÷, 2×, 1+			1÷, 1×, 1+	
$\delta_{3 \max}$	$1,9351 \cdot 10^{-7}$	$1,7594 \cdot 10^{-7}$	$1,7552 \cdot 10^{-7}$	$1,4826 \cdot 10^{-7}$	$4,2916 \cdot 10^{-7}$	$2,0055 \cdot 10^{-7}$	$1,9922 \cdot 10^{-7}$	$1,9789 \cdot 10^{-7}$	$1,5218 \cdot 10^{-7}$	$1,5232 \cdot 10^{-7}$	$1,5232 \cdot 10^{-7}$	$8,9363 \cdot 10^{-8}$	$8,9363 \cdot 10^{-8}$
Бітова точність	22,3	22,44	22,44	22,69	21,15	22,25	22,26	22,27	22,65	22,65	22,65	23,42	23,42

Таблиця Г.6 – Порівняння алгоритмів обчислення степеневих функцій RCP та RCBRT
для FP чисел одинарної точності на основі модифікованих ітераційних методів NR та H

Степенева функція	$y = 1/x$				$y = 1/\sqrt[3]{x}$		
Алгоритм	Мороза та Гринчишина [152]	Мороза та ін. [183]	Мороза та ін. [183]	Запропонований	Мороза та ін. [157]	Мороза та ін. [157]	Мороза та ін. [157]
Метод MC							
Формула	(1.49), (1.50)				(1.46)		
k	0,05051022768020 6298828125	0,04998723917 669184152223 414	0,29289318047 537905677020 96	0,05051022768020 6298828125	0,0499640 543038885318 477242	0,17836464110 696614789552 74	0,178696520058603874 1612874
R	0x7EF311C3	0x7EF33409	0x7EB504F3	0x7EF311C3	0x54A223B4	0x548C39CB	0x548C2B4B
$\delta_{0 \max}$	$5,0510 \cdot 10^{-2}$	$5,1268 \cdot 10^{-2}$	$2,9289 \cdot 10^{-1}$	$5,0510 \cdot 10^{-2}$	$3,4485 \cdot 10^{-2}$	$1,3676 \cdot 10^{-1}$	$1,3706 \cdot 10^{-1}$
Бітова точність	4,31	4,29	1,77	4,31	4,86	2,87	2,87
Перша модифікована ітерація NR				Перша ітерація модифікованого Ho2	Перша модифікована ітерація NR		Перша модифікована ітерація Ho2
Структура ітерації	$k_1 \cdot y_0 \cdot (k_2 - x \cdot y_0)$			$c_0 = x \cdot y_0$ $y_0 \cdot (k_1 + c_0 \cdot (k_2 + c_0))$	$y_0 \cdot (k_1 - k_2 \cdot x \cdot y_0 \cdot y_0 \cdot y_0)$		$c_0 = x \cdot y_0 \cdot y_0 \cdot y_0$ $y_0 \cdot (k_1 - c_0 \cdot (k_2 - k_3 \cdot c_0))$
k_1	1,0	1,0	1,96875	3,0	1,33451575396	1,5015480449	1,752319676
k_2	2,00130856	2,00128159	1,4255685	-3,0	0,33333333	0,534850249	1,2509524245
k_3	—	—	—	—	—	—	0,5093818292
Кількість FP операцій	2×, 1–		3×, 1–	3×, 1+, 1–	5×, 1–		6×, 2–
$\delta_{1 \max}$	$1,3090 \cdot 10^{-3}$	$1,2820 \cdot 10^{-3}$	$2,4594 \cdot 10^{-4}$	$1,2909 \cdot 10^{-4}$	$1,1828 \cdot 10^{-3}$	$8,0838 \cdot 10^{-4}$	$2,6854 \cdot 10^{-5}$
Бітова точність	9,58	9,61	11,99	12,91	9,72	10,27	15,18

Продовження таблиці Г.6

Степенева функція	$y = 1/x$				$y = 1/\sqrt[3]{x}$		
Алгоритм	Мороза та Гринчишина [152]	Мороза та ін. [183]	Мороза та ін. [183]	Запропонований	Мороза та ін. [157]	Мороза та ін. [157]	Мороза та ін. [157]
Друга модифікована ітерація NR							Друга модифікована ітерація NR у спеціальній формі
Структура ітерації	$k_1 \cdot y_1 \cdot (k_2 - x \cdot y_1)$				$y_1 \cdot (k_1 - k_2 \cdot x \cdot y_1 \cdot y_1 \cdot y_1)$		$c_1 = fmaf(-x \cdot y_1, y_1 \cdot y_1, k_1)$ $y_1 \cdot fmaf(k_2, c_1, k_3)$
k_1	1,0	1,0	1,0		1,333334485	1,333333985	1,0
k_2	2,000000084	2,000000082	2,0		0,33333333	0,33333333	0,33333333
k_3	—	—	—		—	—	1,0
Кількість FP операцій	2×, 1–				4×, 1–	5×, 1–	3×, 2FMA
$\delta_{2\max}$	$1,0947 \cdot 10^{-6}$	$1,0735 \cdot 10^{-6}$	$1,9911 \cdot 10^{-7}$	$1,6475 \cdot 10^{-7}$	$1,8490 \cdot 10^{-6}$	$8,1392 \cdot 10^{-7}$	$1,3602 \cdot 10^{-7}$
Бітова точність	19, 8	19, 83	22,26	22,53	19,04	20,23	22,81
Третя модифікована ітерація NR							
Структура ітерації	$k_1 \cdot y_2 \cdot (k_2 - x \cdot y_2)$		—		$y_2 \cdot (k_1 - k_2 \cdot x \cdot y_2 \cdot y_2 \cdot y_2)$		—
k_1	1,0		—		1,33333333		—
k_2	2,0		—		0,33333333		—
Кількість FP операцій	2×, 1–		—		4×, 1–		—
$\delta_{3\max}$	$1,4101 \cdot 10^{-7}$	$1,3715 \cdot 10^{-7}$	—		$2,0132 \cdot 10^{-7}$	$1,8773 \cdot 10^{-7}$	—
Бітова точність	22,76	22,8	—		22,24	22,34	—

Таблиця Г.7 – Порівняння запропонованих покращених алгоритмів обчислення функції RCP
для FP чисел одинарної точності

Степенева функція	y = 1/ x								
Алгоритм	Мороза та Гринчишина [152]	Мороза та ін. [183]	Запропонований	Мороза та ін. [183]	Запропонований [80]	Запропонований			
Метод MC									
Формула	(1.49), (1.50)								
k	0,050510227680206298828125	0,0499872391766918415222341	0,0501199192281165715158928	0,2928931804753790567702096	0,2928931804753790567702096	0,2921538438366240931059744	0,050510227680206298828125	0,0499833052668113239491111	0,0425177575934096714204808
R	0x7EF311C3	0x7EF33409	0x7EF32B57	0x7EB504F3	0x7EB504F3	0x7EB53567	0x7EF311C3	0x7EF3344B	0x7EF51D8E
δ _{0 max}	5,0510·10 ⁻²	5,1268·10 ⁻²	5,1076·10 ⁻²	2,9289·10 ⁻¹	2,9289·10 ⁻¹	2,9215·10 ⁻¹	5,0510·10 ⁻²	5,1274·10 ⁻²	6,2127·10 ⁻²
Бітова точність	4,31	4,29	4,29	1,77	1,77	1,78	4,31	4,29	4,01
Перша модифікована ітерація NR							Перша модифікована ітерація Ho2		Перша модифікована ітерація NR
Структура ітерації	k ₁ · y ₀ · (k ₂ – x · y ₀)						c ₀ = x · y ₀ y ₀ · (k ₁ + c ₀ · (k ₂ + c ₀))		k ₁ · y ₀ · (k ₂ – x · y ₀)
k ₁	1,0	1,0	1,0	1,96875	1,940909	1,93959737	3,0030725	3,001922846	1,0
k ₂	2,00130856	2,0012816	2,0013084	1,4255685	1,4356601	1,43614197	-3,003093	-3,00192165	2,0017604828
Кількість FP операцій	2×, 1–			3×, 1–			3×, 2+		2×, 1–
δ _{1 max}	1,3090·10 ⁻³	1,2820·10 ⁻³	1,3090·10 ⁻³	2,4594·10 ⁻⁴	1,1173·10 ⁻⁴	1,4438·10 ⁻⁴	9,0906·10 ⁻⁵	3,2684·10 ⁻⁵	1,9900·10 ⁻³
Бітова точність	9,58	9,61	9,58	11,99	13,13	12,76	13,42	14,9	8,97

Продовження таблиці Г.7

Степенева функція	$y = 1/x$									
Алгоритм	Мороза та Гринчишина [152]	Мороза та ін. [183]	Запропонований	Мороза та ін. [183]	Запропонований [80]	Запропонований				
Друга модифікована ітерація NR					Друга модифікована ітерація NR у спеціальній формі					Друга модифікована ітерація Но2 у спеціальній формі
Структура ітерації	$k_1 \cdot y_1 \cdot (k_2 - x \cdot y_1)$				$r_1 = fmaf(y_1, -x, k_1)$ $fmaf(y_1, r_1, y_1)$					$r_1 = fmaf(y_1, -x, k_1)$ $t_1 = fmaf(r_1, r_1, r_1)$ $fmaf(y_1, t_1, y_1)$
k_1	1,0	1,0	1,0	1,0	1,0					1,0
k_2	2,0000009	2,00000082	2,0000017	2,0	—					—
Кількість FP операцій	2×, 1—				2FMA					3FMA
$\delta_{2 \max}$	$1,0947 \cdot 10^{-6}$	$1,0735 \cdot 10^{-6}$	$1,8099 \cdot 10^{-6}$	$1,9911 \cdot 10^{-7}$	$1,1882 \cdot 10^{-7}$	$7,1744 \cdot 10^{-8}$	$6,8615 \cdot 10^{-8}$	$5,9605 \cdot 10^{-8}$	$5,9605 \cdot 10^{-8}$	$5,9605 \cdot 10^{-8}$
Бітова точність	19,8	19,83	19,08	22,26	23	23,73	23,8	24	23,98	24
Третя модифікована ітерація NR					—	—	—	—	—	—
Структура ітерації	$k_1 \cdot y_2 \cdot (k_2 - x \cdot y_2)$				—	—	—	—	—	—
k_1	1,0				—	—	—	—	—	—
k_2	2,0				—	—	—	—	—	—
Кількість FP операцій	2×, 1—				—	—	—	—	—	—
$\delta_{3 \max}$	$1,4101 \cdot 10^{-7}$	$1,3715 \cdot 10^{-7}$	$1,1373 \cdot 10^{-7}$	$1,1679 \cdot 10^{-7}$	—	—	—	—	—	—
Бітова точність	22,76	22,8	23,07	23,03	—	—	—	—	—	—

Продовження таблиці Г.7

Степенева функція	$y = 1/x$									
Алгоритм	Мороза та Гринчишина [152]	Мороза та ін. [183]	Запропонований	Мороза та ін. [183]	Запропонований [80]			Запропонований		
Третя модифікована ітерація NR у спеціальній формі				—	—	—	—	—	—	—
Структура ітерації	$r_3 = fmaf(y_2, -x, k_1)$ $fmaf(y_2, r_1, y_2)$			—	—	—	—	—	—	—
k_1	1,0			—	—	—	—	—	—	—
Кількість FP операцій	2FMA			—	—	—	—	—	—	—
$\delta_{3\max}$	$5,9605 \cdot 10^{-8}$	$5,9605 \cdot 10^{-8}$	$5,9605 \cdot 10^{-8}$	$5,9605 \cdot 10^{-8}$	—	—	—	—	—	—
Бітова точність	23,99	23,99	24	24	—	—	—	—	—	—

Таблиця Г.8 – Застосування методу МС для наближення логарифмічних та експоненціальних функцій
для FP чисел одинарної точності

Функція	$y = 2^x$		$y = \log_2 x$		$y = e^x$		$y = \ln x$	
Алгоритм	Бліна [6]	Запропонований	Бліна [6]	Запропонований	Бліна [6]	Запропонований	Бліна [6]	Запропонований
Метод МС								
Формула	(1.45)		(1.44)		(2.12)		(2.13)	
R	0x3F800000	0x3F7A68B0	0x3F800000	0x3F7A7DD0	0x3F800000	0x3F7A68B0	0x3F800000	0x3F7A7DD0
Кількість FP операцій	1×		1×		2×	1×	2×	1×
$\Delta_{0 \max}$	—	—	$8,6071 \cdot 10^{-2}$	$4,3036 \cdot 10^{-2}$	—	—	$5,9660 \cdot 10^{-2}$	$2,9830 \cdot 10^{-2}$
$\delta_{0 \max}$	$6,1476 \cdot 10^{-2}$	$2,9823 \cdot 10^{-2}$	—	—	$6,1476 \cdot 10^{-2}$	$2,9823 \cdot 10^{-2}$	—	—
Бітова точність	4,02	5,07	—	—	4,02	5,07	—	—
Уточнення результату								
Формула	$\tilde{y}_0(x) = y_0(k_1 \cdot x) \cdot y_0(k_1 \cdot x + k_2) \cdot k_3$		$\tilde{y}_0(x) = k_1(y_0(x) + y_0(k_2 \cdot x)) + k_3$		$\tilde{y}_0(x) = \tilde{y}'_0(C_e \cdot x)$		$\tilde{y}_0(x) = C_e \cdot \tilde{y}'_0(x)$	
k_1	0,5	0,5	0,5	0,5	$0,5C_e$	$0,5C_e$	$0,5C_e$	$0,5C_e$
k_2	0,5	0,5	0,6666	0,6666	0,5	0,5	0,6666	0,6666
k_3	0,657	0,6973643	0,344	0,3055452	0,657	0,6973643	$0,344C_e$	$0,3055452C_e$
Кількість FP операцій	3×, 1+		2×, 2+		3×, 1+		2×, 2+	
$\Delta_{0 \max}$	—	—	$1,9761 \cdot 10^{-2}$	$1,5181 \cdot 10^{-2}$	—	—	$1,3698 \cdot 10^{-2}$	$1,0523 \cdot 10^{-2}$
$\delta_{0 \max}$	$1,6360 \cdot 10^{-2}$	$1,5416 \cdot 10^{-2}$	—	—	$1,6361 \cdot 10^{-2}$	$1,5416 \cdot 10^{-2}$	—	—
Бітова точність	5,93	6,02	—	—	5,93	6,02	—	—

Таблиця Г.9 – Порівняння запропонованих покращених алгоритмів обчислення функції RCBRT
для FP чисел одинарної точності

Степенева функція	$y = 1/\sqrt[3]{x}$							
Алгоритм	Мороза та ін. [157]	Мороза та ін. [157]	Запропонований	Мороза та ін. [157]	Запропонований	Левіна [151,156]	Запропонований	
Метод МС								
Формула	(1.46)							
k	0,0499640543038885318477242	0,1783646411069661478955274	0,049622263759374619	0,1786965200586038741612874	0.1785201199585011699152845	0,05011372169748866176428873	0,05011372169748866176428873	0,05061449055743748354605572
R	0x54A223B4	0x548C39CB	0x54A232A3	0x548C2B4B	0x548C3300	0x54A21D2A	0x54A21D2A	0x54A20749
$\delta_{0\max}$	$3,4485 \cdot 10^{-2}$	$1,3676 \cdot 10^{-1}$	$3,4241 \cdot 10^{-2}$	$1,3706 \cdot 10^{-1}$	$1,3690 \cdot 10^{-1}$	$3,4592 \cdot 10^{-2}$	$3,4592 \cdot 10^{-2}$	$3,4950 \cdot 10^{-2}$
Бітова точність	4,86	2,87	4,87	2,87	2,87	4,85	4,85	4,84
Перша модифікована ітерація NR				Перша модифікована ітерація Но2		Перша ітерація Но3		Перша модифікована ітерація Но3 у спеціальній формі
Структура ітерації	$y_0 \cdot (k_1 - k_2 \cdot x \cdot y_0 \cdot y_0 \cdot y_0)$			$c_0 = x \cdot y_0 \cdot y_0 \cdot y_0$ $y_0 \cdot (k_1 - c_0 \cdot (k_2 - k_3 \cdot c_0))$		$c_0 = x \cdot y_0 \cdot y_0 \cdot y_0$ $y_0 \cdot (k_1 + c_0 \cdot (k_2 + c_0 \cdot (k_3 + k_4 \cdot c_0)))$		$r_0 = fmaf(-x \cdot y_0, y_0 \cdot y_0, k_1)$ $t_{01} = fmaf(k_2, r_0, k_3)$ $t_{02} = fmaf(t_{01}, r_0, k_4)$ $fmaf(r_0 \cdot y_0, t_{02}, y_0)$
k_1	1,334515754	1,50154804	1,334477305	1,752319676	1,75229812	1,728395062	1,7283951044	1,0
k_2	0,33333333	0,53485025	0,333299994	1,250952425	1,25095224	-1,29629629	-1,29628158	0,17382118106
k_3	—	—		0,509381829	0,50938183	0,740740741	0,740740716	0,22347638011
k_4	—	—		—	—	-0,17283951	-0,17283951	0,33333227038
Кількість FP операцій	5×, 1–			6×, 2–		7×, 3+		3×, 4FMA
$\delta_{1\max}$	$1,1828 \cdot 10^{-3}$	$8,0838 \cdot 10^{-4}$	$1,1777 \cdot 10^{-3}$	$2,6854 \cdot 10^{-5}$	$4,6181 \cdot 10^{-5}$	$1,6561 \cdot 10^{-5}$	$1,5385 \cdot 10^{-5}$	$2,7958 \cdot 10^{-6}$
Бітова точність	9,72	10,27	9,73	15,18	14,4	15,88	15,99	18,45

Продовження таблиці Г.9

Степенева функція	$y = 1 / \sqrt[3]{x}$							
Алгоритм	Мороза та ін. [157]	Мороза та ін. [157]	Запропонований	Мороза та ін. [157]	Запропонований	Левіна [151,156]	Запропонований	
Друга модифікована ітерація NR				Друга модифікована ітерація NR у спеціальній формі				
Структура ітерації	$y_1 \cdot (k_1 - k_2 \cdot x \cdot y_1 \cdot y_1 \cdot y_1)$			$r_1 = \text{fmaf}(-x \cdot y_1, y_1 \cdot y_1, k_1)$ $\text{fmaf}(k_2 \cdot y_1, r_1, y_1)$				
k_1	1,333334485	1,33333399	1,333301425	1,0	1,0	1,0	1,0	1,0
k_2	0,33333333	0,33333333	0,333299994	0,33333333	0,333332241	0,33333333	0,333343804	0,33333333
Кількість FP операцій	4×, 1–	5×, 1–	4×, 1–	3×, 2FMA				
$\delta_{2\max}$	$1,8490 \cdot 10^{-6}$	$8,1392 \cdot 10^{-7}$	$1,6006 \cdot 10^{-6}$	$8,8684 \cdot 10^{-8}$	$8,7057 \cdot 10^{-8}$	$8,8445 \cdot 10^{-8}$	$8,7348 \cdot 10^{-8}$	$8,8614 \cdot 10^{-8}$
Бітова точність	19,04	20,23	19,25	23,43	23,45	23,43	23,45	23,43
Третя модифікована ітерація NR				–	–	–	–	–
Структура ітерації	$y_2 \cdot (k_1 - k_2 \cdot x \cdot y_2 \cdot y_2 \cdot y_2)$			–	–	–	–	–
k_1	1,33333333		1,333146572	–	–	–	–	–
k_2	0,33333333		0,333146572	–	–	–	–	–
Кількість FP операцій	4×, 1–		5×, 1–	–	–	–	–	–
$\delta_{3\max}$	$2,0132 \cdot 10^{-7}$	$1,8773 \cdot 10^{-7}$	$1,7735 \cdot 10^{-7}$	–	–	–	–	–
Бітова точність	22,24	22,34	22,43	–	–	–	–	–

Продовження таблиці Г.9

Степенева функція	$y = 1 / \sqrt[3]{x}$							
Алгоритм	Мороза та ін. [157]	Мороза та ін. [157]	Запропонований	Мороза та ін. [157]	Запропонований	Левіна [151,156]	Запропонований	
	Третя модифікована ітерація NR у спеціальній формі			–	–	–	–	–
Структура ітерації	$c_2 = fmaf(-x \cdot y_2, y_2 \cdot y_2, k_1)$ $fmaf(k_2 \cdot y_2, c_2, y_2)$			–	–	–	–	–
k_1	1,0		1,0	–	–	–	–	–
k_2	0,33333333		0,333146572	–	–	–	–	–
Кількість FP операцій	3×, 2FMA			–	–	–	–	–
$\delta_{3 \max}$	$8,8206 \cdot 10^{-8}$	$8,7563 \cdot 10^{-8}$	$8,7896 \cdot 10^{-8}$	–	–	–	–	–
Бітова точність	23,43	23,45	23,44	–	–	–	–	–

Таблиця Г.10 – Порівняння запропонованих покращених алгоритмів обчислення функції CBRT
для FP чисел одинарної точності

Степенева функція	$y = \sqrt[3]{x}$								
Алгоритм	Запропонований					Бліна [6]	Запропонований		
Метод MC									
Формула	(1.46)								
k	0,04884968 0922537579 81000517	0,050086001 44254191686 413430	0,0488496 809225375 798100052	0,0061653 305191248 969177493	0,0496040 975082908 771822023	0	0,04871932 546128748 410383506	0,051034073672374 57452409853	0,0474309700452285 820970372
R	0x 2A512A32	0x 2A510F30	0x 2A512A32	0x 2A54CEA6	0x 2A5119B7	0x 2A555556	0x 2A512D0B	0x 2A50FA7A	0x 2A514930
$\delta_{0 \max}$	$3,2419 \cdot 10^{-2}$	$3,1765 \cdot 10^{-2}$	$3,2419 \cdot 10^{-2}$	$5,5005 \cdot 10^{-2}$	$3,2020 \cdot 10^{-2}$	$5,8267 \cdot 10^{-2}$	$3,2488 \cdot 10^{-2}$	$3,1877 \cdot 10^{-2}$	$3,3170 \cdot 10^{-2}$
Бітова точність	4,95	4,98	4,95	4,18	4,96	4,1	4,94	4,97	4,91
Перша модифікована ітерація NR				Перша ітерація Но2 у спеціальній формі		Перша модифікована ітерація Но2		Перша модифікована ітерація Но3 у спеціальній формі	Перша модифікована ітерація Но3
Структура ітерації	$k_1 \cdot y_0 + \frac{k_2 \cdot x}{y_0 \cdot y_0}$			$c_0 = y_0 \cdot y_0 \cdot y_0$ $t_0 = c_0 + x$ $\frac{y_0 \cdot (t_0 + x)}{c_0 + t_0}$		$c_0 = y_0 \cdot y_0 \cdot y_0$ $y_0 + \frac{y_0 \cdot (x - c_0)}{k_1 \cdot c_0 + x}$		$c_0 = y_0 \cdot y_0 \cdot y_0$ $r_0 = \frac{c_0 - x}{x}$ $y_0 + y_0 \cdot r_0 \cdot (r_0 \cdot (k_1 \cdot r_0 + k_2) + k_3)$	$c_0 = y_0 \cdot y_0 \cdot y_0$ $y_0 + k_1 \cdot \frac{y_0 \cdot (x - c_0) \cdot (k_2 \cdot c_0 + x)}{x \cdot x + c_0 \cdot (k_3 \cdot x + k_4 \cdot c_0)}$
k_1	0,6661712	0,6668238	0,6661712	—		2	1,99852693	-0,172989666	2,9992847443
k_2	0,333333194	0,332676529	0,33340367	—		—	—	0,22329226136	2,0006361008
k_3	—	—	—	—		—	—	0,3333350718	15,999718666
k_4	—	—	—	—		—	—	—	9,999540329
Кількість FP операцій	1÷, 3×, 1+			1÷, 3×, 3+		1÷, 4×, 2+, 1–		1÷, 6×, 3+, 1–	1÷, 10×, 4+, 1–

Продовження таблиці Г.10

Степенева функція	$y = \sqrt[3]{x}$								
Алгоритм	Запропонований					Бліна [6]	Запропонований		
$\delta_{1 \max}$	$4,9577 \cdot 10^{-4}$	$5,1312 \cdot 10^{-4}$	$5,7034 \cdot 10^{-4}$	$1,0230 \cdot 10^{-4}$	$2,1021 \cdot 10^{-5}$	$1,2087 \cdot 10^{-4}$	$5,3870 \cdot 10^{-6}$	$2,0645 \cdot 10^{-6}$	$1,7429 \cdot 10^{-7}$
Бітова точність	10,98	10,93	10,78	13,25	15,54	13,01	17,5	18,89	22,45
Друга модифікована ітерація NR		Друга модифікована ітерація NR у спеціальній формі	Друга модифікована ітерація Но2				Друга модифікована ітерація NR у спеціальній формі	—	—
Структура ітерації	$k_1 \cdot y_1 + \frac{k_2 \cdot x}{y_1 \cdot y_1}$	$fmaf(k_1, y_1, \frac{k_2 \cdot x}{y_1 \cdot y_1})$	$c_1 = y_1 \cdot y_1 \cdot y_1$ $y_1 + \frac{y_1 \cdot (x - c_1)}{k_1 \cdot c_1 + x}$				$fmaf(k_1, y_1, \frac{k_2 \cdot x}{y_1 \cdot y_1})$	—	—
k_1	0,666666666666	0,666666650772	2	2,00034571	2	2	0,66666662693	—	—
k_2	0,333333319426	0,333333337307	—	—	—	—	0,33333337307	—	—
Кількість FP операцій	1÷, 2×, 1+	1÷, 2×, 1FMA	1÷, 4×, 2+, 1–				1÷, 2×, 1FMA	—	—
$\delta_{2 \max}$	$2,4417 \cdot 10^{-7}$	$2,1860 \cdot 10^{-7}$	$9,5422 \cdot 10^{-8}$	$8,8211 \cdot 10^{-8}$	$9,6693 \cdot 10^{-8}$	$8,4936 \cdot 10^{-8}$	$1,0398 \cdot 10^{-7}$	—	—
Бітова точність	21,97	22,13	23,32	23,43	23,3	23,49	23,2	—	—

Таблиця Г.11 – Порівняння запропонованих покращених алгоритмів обчислення функції SQRT
для FP чисел одинарної точності

Степенева функція	$y = \sqrt{x}$							
Алгоритм	Запропонований							
Метод МС								
Формула	(1.46)							
k	0,06891470265619178789706755	-0,005461935253702480703794825	0,07179360912034609476868349e	-0,06938176499670223513247738	-0,09990461082364793266013992	0,07078843565012881441715987	0,07330470718823883730034186e	0,07235341635660556045053504
R	0x1FBB96E6	0x1FC0597D	0x1FBB67BB	0x1FC470C0	0x1FC664D6	0x1FBB7833	0x1FBB4EF9	0x1FBB5E8F
$\delta_{0\max}$	$3,6295 \cdot 10^{-2}$	$6,2591 \cdot 10^{-2}$	$3,5277 \cdot 10^{-2}$	$8,5190 \cdot 10^{-2}$	$9,5982 \cdot 10^{-2}$	$3,5633 \cdot 10^{-2}$	$3,4753 \cdot 10^{-2}$	$3,5079 \cdot 10^{-2}$
Бітова точність	4,78	3,99	4,83	4,81	3,38	4,18	4,85	4,83
Перша модифікована ітерація NR					Перша модифікована ітерація Но2		Перша модифікована ітерація Но3	
Структура ітерації	$k_1 \cdot (y_0 + \frac{x}{y_0})$	$k_1 \cdot y_0 + k_2 \cdot \frac{x}{y_0}$	$k_1 \cdot (y_0 + \frac{x}{y_0})$		$\frac{c_0 = y_0 \cdot y_0}{y_0 \cdot (c_0 + k_1 \cdot x)} \cdot \frac{y_0 \cdot (x - c_0)}{k_2 \cdot c_0 + x}$	$y_0 + \frac{c_0 = y_0 \cdot y_0}{k_1 \cdot c_0 + k_2 \cdot x}$	$k_1 \cdot y_0 + \frac{c_0 = y_0 \cdot y_0}{y_0 \cdot (c_0 + x)} \cdot \frac{(x - c_0) \cdot (k_2 \cdot c_0 + k_3 \cdot x)}{y_0 \cdot (c_0 + x)}$	
k_1	0,4998473525	0,48461231589	0,4998497963	0,49971687794	2,9963648319	1,4995536804	1	0,9999999404
k_2	—	0,51564925909	—	—	2,9966468811	0,5	0,75	0,74999946356
k_3	—	—	—	—	—	—	0,249999449	0,250000059
Кількість FP операцій	1÷, 1×, 1+	1÷, 2×, 1+	1÷, 1×, 1+		1÷, 4×, 2+	1÷, 4×, 2+, 1–	1÷, 5×, 3+, 1–	1÷, 6×, 3+, 1–
$\delta_{1\max}$	$3,3017 \cdot 10^{-4}$	$2,2017 \cdot 10^{-4}$	$3,0052 \cdot 10^{-4}$	$2,7757 \cdot 10^{-3}$	$2,0190 \cdot 10^{-4}$	$2,7000 \cdot 10^{-6}$	$2,6093 \cdot 10^{-7}$	$1,8146 \cdot 10^{-7}$
Бітова точність	11,56	12,15	11,7	8,49	12,27	18,5	21,86	22,39

Продовження таблиці Г.11

Степенева функція	$y = \sqrt{x}$							
Алгоритм	Запропонований							
	Друга ітерація NR		Друга модифікована ітерація Но2		Друга ітерація NR			
Структура ітерації	$k_1 \cdot (y_1 + \frac{x}{y_1})$		$c_1 = y_1 \cdot y_1$ $y_1 + \frac{y_1 \cdot (x - c_1)}{k_1 \cdot c_1 + k_2 \cdot x}$		$k_1 \cdot (y_1 + \frac{x}{y_1})$			
k_1	0,5		1,50000215	1,498283386	0,5			
k_2	–		0,499995887	0,501730263	–			
Кількість FP операцій	1÷, 1×, 1+		1÷, 4×, 2+, 1–		1÷, 1×, 1+			
$\delta_{2 \max}$	$1,1756 \cdot 10^{-7}$	$1,0567 \cdot 10^{-7}$	$8,8817 \cdot 10^{-8}$	$8,6263 \cdot 10^{-8}$	$8,9363 \cdot 10^{-8}$	$8,9370 \cdot 10^{-8}$		
Бітова точність	23,02	23,7	23,42	23,47	23,42	23,42		

Додаток Д

Алгоритми наближеного обчислення функцій RCP та RSQRT використані при проектуванні ГПВЧ

1. RCP – Початкове наближення методом МС

```
double rcp0 (double x){
    uint64_t i = *(uint64_t*)&x;
    i = 0x7fde6238a0ad0e00 - i;
    double y = *(double*)&i;
    return y;
}
```

$$\delta_{0\max}^+ = 5,0510 \cdot 10^{-2}; \delta_{0\max}^- = -5,0510 \cdot 10^{-2} \text{ (бітова похибка } \alpha_0 = 4,3)$$

2. RCP – Покращений алгоритм – 1 ітерація

```
double rcp1 (double x){
    uint64_t i = *(uint64_t*)&x;
    i = 0x7fd6a54e79bcc0f6 - i;
    double y = *(double*)&i;
    y = 1.9375*y*fma(y, -x, 1.43692333124);
    return y;
}
```

$$\delta_{1\max}^+ = 1,1263 \cdot 10^{-4}; \delta_{1\max}^- = -1,1263 \cdot 10^{-4} \text{ (бітова похибка } \alpha_1 = 13,1)$$

3. RCP – Покращений алгоритм – 2 ітерації

```
double rcp2 (double x){
    uint64_t i = *(uint64_t*)&x;
    i = 0x7fd6a54e79bcc0f6 - i;
    double y = *(double*)&i;
    y = 1.9375*y*fma(y, -x, 1.43692333124);
    y = y*fma(y, -x, 2.00000000639);
    return y;
}
```

$$\delta_{2\max}^+ = 6,3900 \cdot 10^{-9}; \delta_{2\max}^- = -6,2967 \cdot 10^{-9} \text{ (бітова похибка } \alpha_2 = 27,2)$$

4. RCP – Покращений алгоритм – 3 ітерації

```
double rcp3 (double x){
    uint64_t i = *(uint64_t*)&x;
    i = 0x7fd6a54e79bcc0f6 - i;
    double y = *(double*)&i;
    y = 1.9375*y*fma(y, -x, 1.43692333124);
    y = y*fma(y, -x, 2.00000000639);
    double r = fma(y, -x, 1.0);
    y = fma(r, y, y);
    return y;
}
```

$$\delta_{3\max}^+ = 1,1070 \cdot 10^{-16}; \delta_{3\max}^- = -1,5067 \cdot 10^{-16} \text{ (бітова похибка } \alpha_3 = 52,6)$$

5. RSQRT – Початкове наближення методом МС

```
double rsqrt0 (double x){
    int64_t i =*(int64_t*)&x;
    i = 0x5fe6ec85e138b400 - (i>>1);
    double y = *(double*)&i;
    return y;
}
```

$$\delta_{0\max}^+ = 3,4213 \cdot 10^{-2}; \delta_{0\max}^- = -3,4213 \cdot 10^{-2} \text{ (бітова похибка } \alpha_0 = 4,9)$$

6. RSQRT – Вальчика та ін. – 1 ітерація

```
double rsqrt1w (double x){
    double xhalf = 0.5*x;
    int64_t i =*(int64_t*)&x;
    i = 0x5fe6ed2102dcbfda - (i>>1);
    double y = *(double*)&i;
    y = y*(1.5008789551163346 - xhalf*y*y);
    return y;
}
```

$$\delta_{1\max}^+ = 8,7908 \cdot 10^{-4}; \delta_{1\max}^- = -8,7908 \cdot 10^{-4} \text{ (бітова похибка } \alpha_1 = 10,2)$$

7. RSQRT – Вальчика та ін. – 2 ітерації

```
double rsqrt2w (double x){
    double xhalf = 0.5*x;
    int64_t i =*(int64_t*)&x;
    i = 0x5fe6ed2102dcbfda - (i>>1);
    double y = *(double*)&i;
    y = y*(1.5008789551163346 - xhalf*y*y);
    y = y*(1.5000005796762577 - xhalf*y*y);
    return y;
}
```

$$\delta_{2\max}^+ = 5,7968 \cdot 10^{-7}; \delta_{2\max}^- = -5,7968 \cdot 10^{-7} \text{ (бітова похибка } \alpha_2 = 20,7)$$

8. RSQRT – Вальчика та ін. – 3 ітерації

```
double rsqrt3w (double x){
    double xhalf = 0.5*x;
    int64_t i =*(int64_t*)&x;
    i = 0x5fe6ed2102dcbfda - (i>>1);
    double y = *(double*)&i;
    y = y*(1.5008789551163346 - xhalf*y*y);
    y = y*(1.5000005796762577 - xhalf*y*y);
    y = y*(1.5000000000000252 - xhalf*y*y);
    return y;
}
```

$$\delta_{3\max}^+ = 2,5231 \cdot 10^{-13}; \delta_{3\max}^- = -2,5228 \cdot 10^{-13} \text{ (бітова похибка } \alpha_3 = 41,9)$$

9. RSQRT – Хаусхолдера 2-го порядку (Ho2) – 1 ітерація

```
double rsqrt1h (double x){
    int64_t i = *(int64_t*)&x;
    i = 0x5fe52c437da986d9 - (i >> 1);
```

```

double y = *(double*) &i;
double c = x*y*y;
y = y*(2.00718651889802436 + c*(-1.53237125952242026 + 0.52595012062439589*c));
return y;
}

```

$\delta_{1\max}^+ = 1,9550 \cdot 10^{-5}$; $\delta_{1\max}^- = -1,9541 \cdot 10^{-5}$ (бітова похибка $\alpha_1 = 15,6$)

10.RSQRT – Хаусхолдера 2-го порядку (Ho2) – 2 ітерації

```

double rsqrt2h (double x){
    int64_t i = *(int64_t*)&x;
    i = 0x5fe52c437da986d9 - (i >> 1);
    double y = *(double*)&i;
    double c = x*y*y;
    y = y*(2.00718651889802436 + c*(-1.53237125952242026 + 0.52595012062439589*c));
    double xh = 0.5*x;
    y = y*(1.500000000285 - xh*y*y);
    return y;
}

```

$\delta_{2\max}^+ = 2,8500 \cdot 10^{-10}$; $\delta_{2\max}^- = -2,8829 \cdot 10^{-10}$ (бітова похибка $\alpha_2 = 31,7$)

11.RSQRT – Переключення MC (DC) – 1 ітерація

```

double rsqrt1dc (double x){
    uint64_t i = *(uint64_t*)&x;
    uint64_t k = i & 0x0010000000000000;
    double y;
    if (k != 0) {
        i = 0x5fdb3d20982e5432 - (i >> 1);
        y = *(double*)&i;
        y = 2.331242396766632*y*fma(-x, y*y, 1.074973693828754);
    } else {
        i = 0x5fe33d209e450c1b - (i >> 1);
        y = *(double*)&i;
        y = 0.824218612684476826*y*fma(-x, y*y, 2.14994745900706619);
    }
    return y;
}

```

$\delta_{1\max}^+ = 7,4379 \cdot 10^{-5}$; $\delta_{1\max}^- = -7,4379 \cdot 10^{-5}$ (бітова похибка $\alpha_1 = 13,7$)

12.RSQRT – Переключення MC (DC) – 2 ітерації

```

double rsqrt2dc (double x){
    uint64_t i = *(uint64_t*)&x;
    uint64_t k = i & 0x0010000000000000;
    double y;
    if (k != 0) {
        i = 0x5fdb3d209f0f2455 - (i >> 1);
        y = *(double*)&i;
        y = 2.3312423972719842*y*fma(-x, y*y, 1.0749736939858485);
    } else {
        i = 0x5fe33d2098e159e4 - (i >> 1);
        y = *(double*)&i;
        y = 0.82421861262961627*y*fma(-x, y*y, 2.1499474592540948);
    }
    double c = x*y;
    double r = fma(y, -c, 1.0000000008298416);
}

```

```

    y = fma(0.500000000057372, y*r, y);
    return y;
}

```

$\delta_{2\max}^+ = 4,1492 \cdot 10^{-9}$; $\delta_{2\max}^- = -4,1492 \cdot 10^{-9}$ (бітова похибка $\alpha_2 = 27,9$)

13.RSQRТ – Переключення МС (DC) – 3 ітерації

```

double rsqrt3dc (double x){
    uint64_t i = *(uint64_t*)&x;
    uint64_t k = i & 0x0010000000000000;
    double y;
    if (k != 0) {
        i = 0x5fdb3d14170034b6 - (i >> 1);
        y = *(double*)&i;
        y = 2.33124735553421569*y*fma(-x, y*y, 1.07497362654295614);
    } else {
        i = 0x5fe33d18a2b9ef5f - (i >> 1);
        y = *(double*)&i;
        y = 0.82421942523718461*y*fma(-x, y*y, 2.1499494964450325);
    }
    double mxhalf = -0.5*x;
    y = y*fma(mxxhalf, y*y, 1.5000000034937999);
    double r = fma(mxxhalf, y*y, 0.5);
    y = fma(y, r, y);
    return y;
}

```

$\delta_{3\max}^+ = 1,6035 \cdot 10^{-16}$; $\delta_{3\max}^- = -1,8263 \cdot 10^{-16}$ (бітова похибка $\alpha_3 = 52,3$)

Додаткові алгоритми:

```

const uint64_t DCRD0[8] = {
    0x5FE2F10A9B53F17B, 0x5FE7D073DD7A6E10, 0x5FE16C85E6B986A3, 0x5FE4F80BB6D9D3CF,
    0x5FDAC6E58C7CA670, 0x5FD0E6A60669C355, 0x5FD69C3973EBC278, 0x5FDA73900811544F
};
const double DCKD1[8] = {
    0.846447587, 0.401537776, 0.98020374775, 0.672251999,
    2.4631722, 7.77609873, 3.78227472, 2.48082948
};
const double DCKD2[8] = {
    2.11204123, 3.47231483, 1.915253065, 2.46271038,
    1.03619039, 0.481497437, 0.778513789, 1.03126919
};

```

14.RSQRТ – Узагальнений метод DC (8DC) з LUT – 1 ітерація

```

double rsqrt1dc8 (double x){
    uint64_t i = *(uint64_t*)&x;
    uint64_t j = (i & 0x001C000000000000) >> 50;
    double y;
    i = DCRD0[j] - (i >> 1);
    y = *(double*)&i;
    y = DCKD1[j]*y*fma(-x, y*y, DCKD2[j]);
    return y;
}

```

$\delta_{1\max}^+ = 8,5686 \cdot 10^{-6}$; $\delta_{1\max}^- = -8,5696 \cdot 10^{-6}$ (бітова похибка $\alpha_1 = 16,8$)

15.RSQRT – Узагальнений метод DC (8DC) з LUT – 2 ітерації

```
double rsqrt2dc81 (double x){
    uint64_t i = *(uint64_t*)&x;
    uint64_t j = (i & 0x001C000000000000) >> 50;
    uint64_t ix = i - 0x8010000000000000;
    double y;
    i = DCRD0[j] - (i >> 1);
    y = *(double*)&i;
    y = DCKD1[j]*y*(DCKD2[j] - x*y*y);
    double mxhalf = *(double*)&ix;
    double r = fma(mxhalf, y*y, 0.50000000005507905);
    y = fma(y, r, y);
    return y;
}
```

$$\delta_{2\max}^+ = 5,5079 \cdot 10^{-11}; \delta_{2\max}^- = -5,5079 \cdot 10^{-11} \text{ (бітова похибка } \alpha_2 = 34,1)$$

16.RSQRT – Узагальнений метод DC (8DC) з LUT – 2 ітерації

```
double rsqrt2dc82 (double x){
    uint64_t i = *(uint64_t*)&x;
    uint64_t j = (i & 0x001C000000000000) >> 50;
    double y;
    i = DCRD0[j] - (i >> 1);
    y = *(double*)&i;
    y = DCKD1[j]*y*fma(-x, y*y, DCKD2[j]);
    double c = x*y;
    double r = fma(y, -c, 1.0);
    double t = fma(0.375, r, 0.5);
    y = fma(y*r, t, y);
    return y;
}
```

$$\delta_{2\max}^+ = 1,7024 \cdot 10^{-15}; \delta_{2\max}^- = -1,7006 \cdot 10^{-15} \text{ (бітова похибка } \alpha_2 = 49,1)$$

17.RSQRT – Узагальнений метод DC (8DC) з LUT – 3 ітерації

```
double rsqrt3dc8 (double x){
    uint64_t i = *(uint64_t*)&x;
    uint64_t j = (i & 0x001C000000000000) >> 50;
    uint64_t ix = i - 0x8010000000000000;
    double y;
    i = DCRD0[j] - (i >> 1);
    y = *(double*)&i;
    y = DCKD1[j]*y*(DCKD2[j] - x*y*y);
    double mxhalf = *(double*)&ix;
    double r = mxhalf*y*y + 0.50000000005507905;
    y = y*r + y;
    r = fma(mxhalf, y*y, 0.5);
    y = fma(y, r, y);
    return y;
}
```

$$\delta_{3\max}^+ = 1,6643 \cdot 10^{-16}; \delta_{3\max}^- = -1,6648 \cdot 10^{-16} \text{ (бітова похибка } \alpha_3 = 52,4)$$

Додаток Е

Порівняння швидкодії відомих та запропонованих апаратних, програмних та програмно-апаратних методів обчислення елементарних функцій та методів генерації ІВП та ПВП на різних обчислювальних платформах

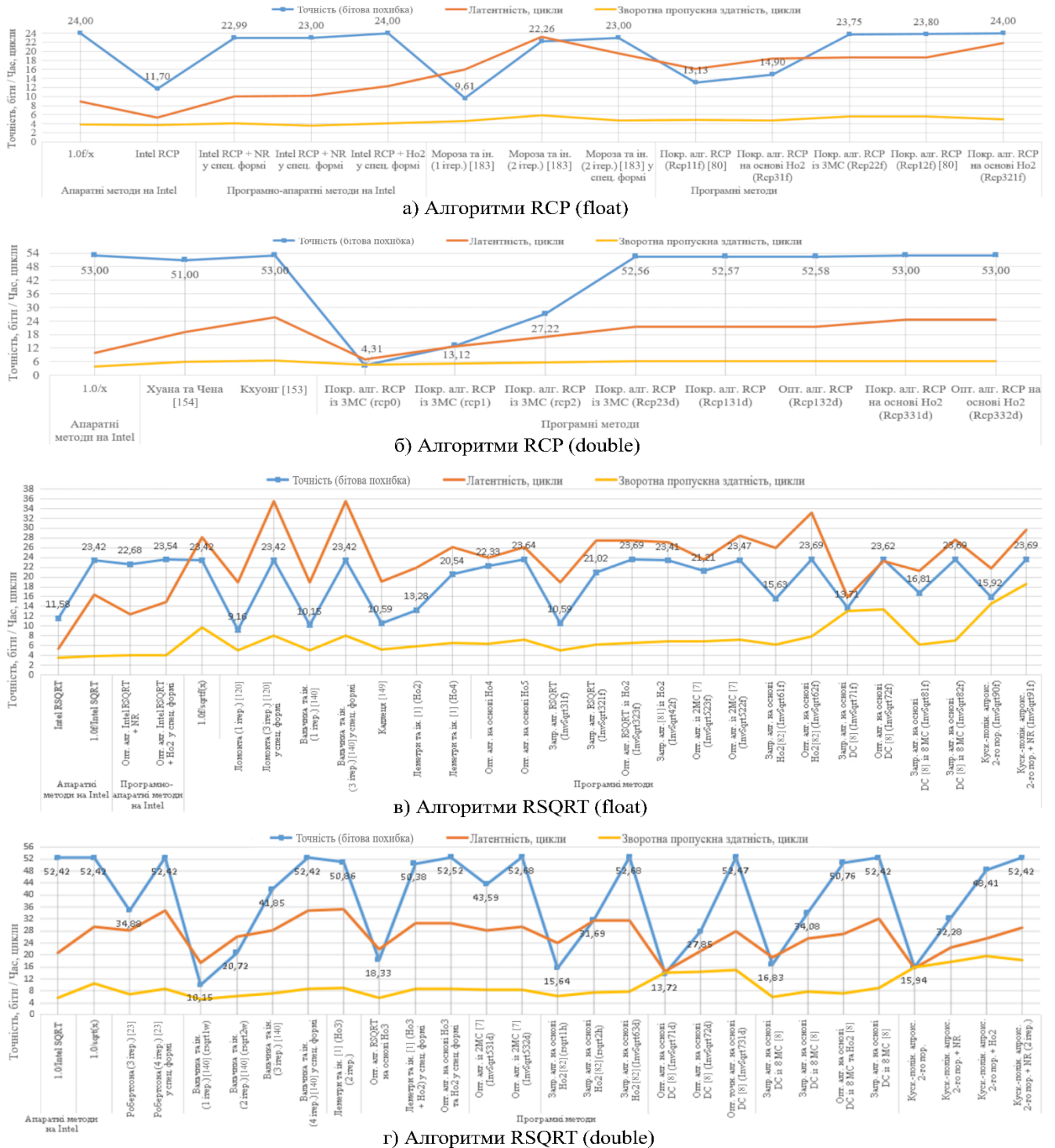


Рисунок Е.1 – Порівняння доступних (апаратних, програмних та програмно-апаратних), а також запропонованих методів обчислення функцій RCP та RSQRT на платформі Intel i5 на основі вимірювання точності та швидкодії: (а) RCP для типу *float*, (б) RCP типу *double*, (в) RSQRT типу *float* та (г) RSQRT типу *double*

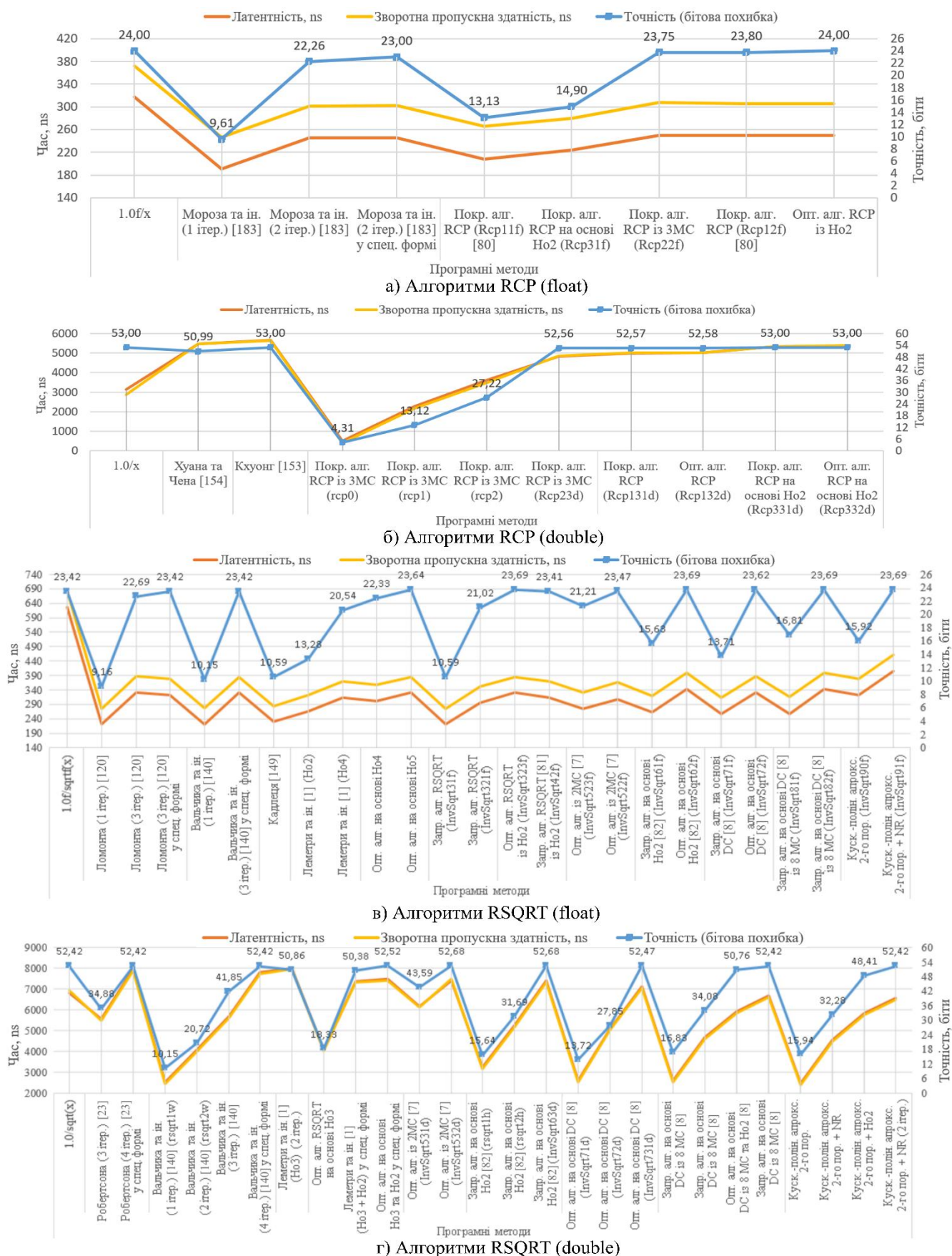


Рисунок Е.2 – Порівняння доступних і запропонованих методів обчислення функцій RCP та RSQRT на МК ESP-32 на основі точності та швидкодії: (а) RCP типу *float*, (б) RCP типу *double*, (в) RSQRT типу *float* та (г) RSQRT типу *double*

Таблиця Е.1 – Порівняння запропонованих алгоритмів RSQRT методом з 2МС та на основі Ho2 для чисел одинарної та подвійної точності з іншими відомими методами на ПК з Intel i7 та МК ESP-WROOM-32 [7,82]

Методи обчислення	Кількість та типи ітерацій	Точність (бітова похибка)	Латентність, ns	
			Intel i7	ESP-32
float				
1,0f/sqrtf(x)	—	23,42	15,3	801,5
Ломонта (FISR) (2 ітер.) [120]	2 NR	17,69	15,5	281,2
Вальчика та ін. (2 ітер.) [140]	2 NR	20,37/20,4	15,5	281,2
Кадлеця (2 ітер.) [149]	2 NR	21,25	15,9	302,1
Леметри та ін. (Ho4) [1]	Ho4	20,54	14,9	323,1
Леметри та ін. (Ho5) [1]	Ho5	20,87	15,5	348,3
Леметри та ін. [1] у спец. формі	Ho2 + NR	23,45	18,05	335,4
Запр. алг. з 2МС [7]	2 NR	21,26	14,2	306,4
Опт. алг. з 2МС [7] (InvSqrt523f)	2 NR	21,21	13,5	281,2
Опт. алг. з 2МС [7] (InvSqrt522f)	NR + Ho2	23,47	16,2	331,6
Опт. алг. на осн. Ho2 [82] (InvSqrt62f)	Ho2 + NR	23,69	18,05	335,4
double				
1.0/sqrt(x)	—	52,42	17,5	6893,5
Ломонта (FISR) (3 ітер.) [23,120]	3 NR	34,88	16,4	5373,9
Ломонта (FISR) (4 ітер.) [23,120]	4 NR	51,68	21,0	6849,9
Вальчика та ін. (3 ітер.) [140]	3 NR	41,85	16,4	5471,4
Вальчика та ін. (4 ітер.) [140]	4 NR	51,69	20,3	6982,3
Леметри та ін. (Ho3) (2 ітер.) [1]	2 Ho3	50,86	21,3	8010,5
Леметри та ін. (Ho3) (2 ітер.) [1] (*)	2 Ho3	50,71	—	7548,1
Леметри та ін. (Ho3) (2 ітер.) [1] (**)	2 Ho3	50,62	—	7455,6
Леметри та ін. [1]	Ho2 + 2 NR	50,92	18,2	7492,3
Запр. алг. на осн. Ho2 [82] (InvSqrt63d)	Ho2 + 2 NR	52,68	18,2	7492,3
Опт. алг. з 2МС [7] (InvSqrt531d)	3 NR	43,59	16,5	6106,0
Опт. алг. з 2МС [7] (InvSqrt531d) (**)	3 NR	43,59	—	5258,7
Опт. алг. з 2МС [7] (InvSqrt532d)	2 NR + Ho2	52,68	16,9	7286,9
Опт. алг. з 2МС [7] (InvSqrt532d) (*)	2 NR + Ho2	52,68	—	6777,1

Примітки:

Методи підвищення швидкодії запропонованих алгоритмів для чисел подвійної точності для МК ESP-32:

(*) – використання однієї FMA на останній ітерації алгоритму;

(**) – відсутність операції FMA.

Таблиця Е.2 – Порівняння запропонованих алгоритмів RSQRT та SQRT на основі методу DC з іншими відомими методами на МК ESP-32 та Raspberry Pi 3 [8]

Функція	Тип даних	Метод обчислення		Відносна похибка		Точність (бітова похибка)	Латентність, ns	
				$\delta_{\max}^+$	$\delta_{\max}^-$		RP 3	ESP-32
RSQRT	float	1.0f/sqrtf(x)		8,9407E-8	-8,9348E-8	23,42	178,6	797,3
		Вальчика та ін. [140] у спец. формі	1 ітер.	8,7919E-4	-8,7921E-4	10,15	25,9	234,9
			2 ітер.	6,7893E-7	-6,4727E-7	20,49	40,2	281,1
		Запр. метод DC [8]	1 ітер.	7,4593E-5	-7,4504E-5	13,71	28,5	248,0
			2 ітер.	7,3624E-8	-7,7542E-8	23,62	52,2	340,3
		FRSQRT [44]		3,2768E-3	-3,0354E-3	8,25	19,3	—
		FRSQRT + FRSQRTS [44]	1 ітер.	1,3127E-7	-1,6183E-5	15,92	40,2	—
			2 ітер.	1,6064E-7	-1,5772E-7	22,6	60,4	—
	double	1.0/sqrt(x)		1,6653E-16	-1,6653E-16	52,42	199,2	6984,8
		Запр. метод DC [8]	1 ітер.	7,4379E-05	-7,4379E-05	13,72	28,5	2744,1
			2 ітер.	4,1492E-09	-4,1492E-09	27,85	51,1	5128,36
			3 ітер.	1,6035E-16	-1,8263E-16	52,28	56,1	6828,2
				1,3639E-16	-1,6062E-16	52,47	64,5	7237,8
SQRT	float	sqrtf(x)		5,9565E-8	-5,9605E-8	24,00	172,1	604,3
		Вальчика та ін. [140] у спец. формі	1 ітер.	8,7919E-4	-8,7919E-4	10,15	28,4	234,9
			2 ітер.	6,8215E-7	-6,4493E-7	20,48	43,4	302,0
		Запр. метод DC [8]	1 ітер.	7,4504E-5	-7,4511E-5	13,71	27,5	264,8
			2 ітер.	8,7580E-8	-9,0380E-8	23,40	47,5	340,3
	double	sqrt(x)		1,1102E-16	-1,1102E-16	53,00	187,8	4403,3
		Запр. метод DC [8]	1 ітер.	7,4379E-05	-7,4379E-05	13,72	34,1	2650,4
			2 ітер.	4,1492E-09	-4,1492E-09	27,85	46,8	5047,6
			3 ітер.	1,6643E-16	-1,8475E-16	52,27	59,3	7151,9



а

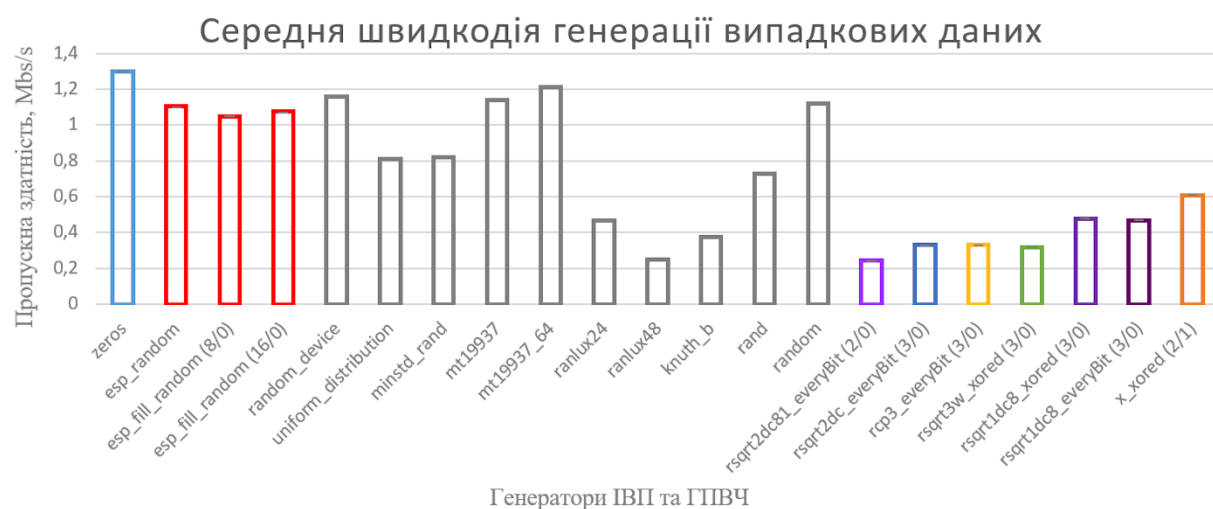


б



в

Рисунок Е.3 – Порівняння швидкодії доступних і запропонованих методів генерації ПВЧ та ІВП на Intel: (а) швидкість генерації ПВП на Intel i5, (б) швидкість генерації ПВП на Intel i7 та (в) час ініціалізації ГПВЧ на Intel i5



а



б

Рисунок Е.4 – Порівняння швидкості доступних і запропонованих методів генерації ПВЧ та ІВП на МК ESP-32: (а) швидкість генерації ПВП та (б) час ініціалізації ГПВЧ

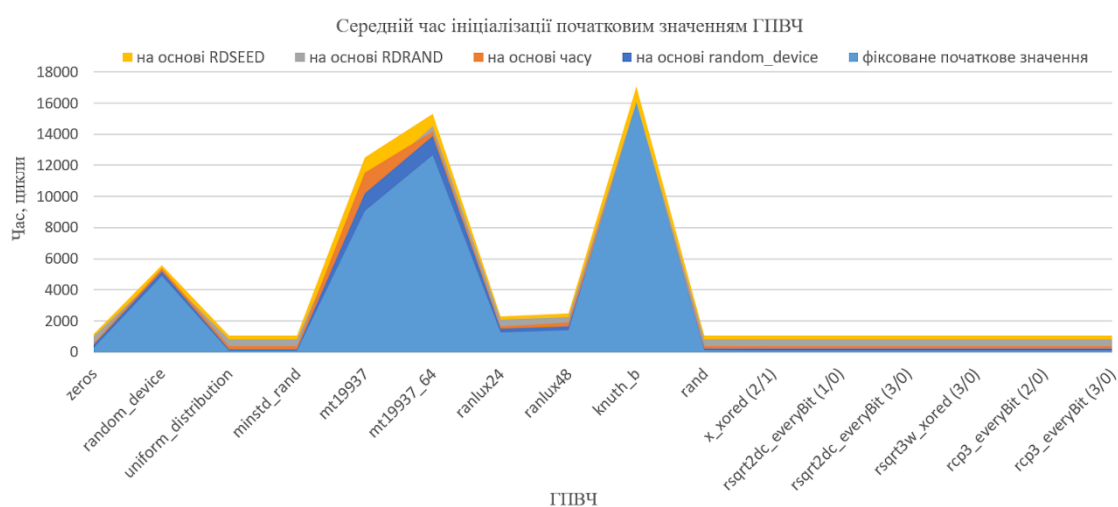


Рисунок Е.5 – Залежність часу ініціалізації доступних і запропонованих ГПВЧ від методу вибору початкових значень на Intel i7

Додаток Ж

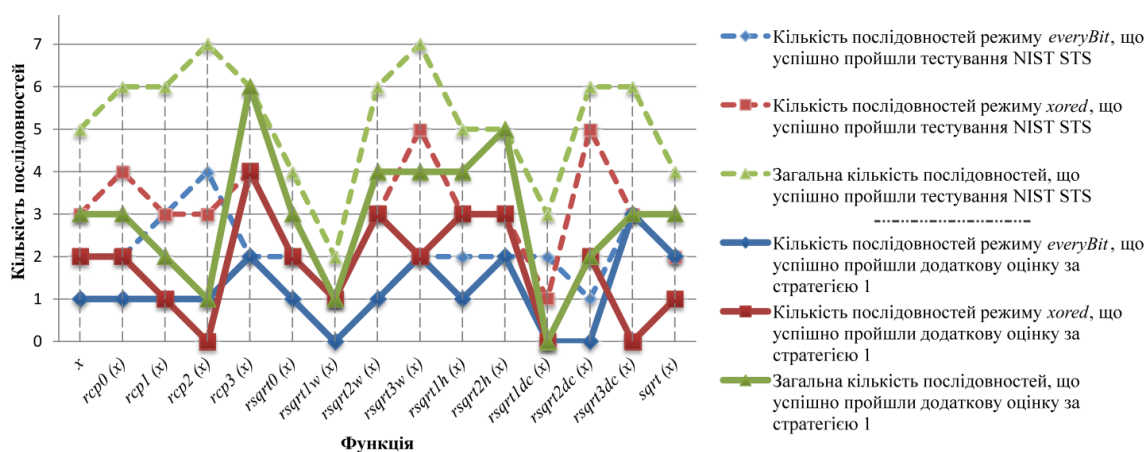
**Результати статистичного тестування ГПВЧ з використанням тестів
NIST STS та графічних тестів**

Таблиця Ж.1 – Приклад статистичного тестування ГПВЧ на основі тотожної функції ($y = x$) при фіксованому значенні $x_0 = 0$ згідно з двома методиками оцінки при $m = 1000$

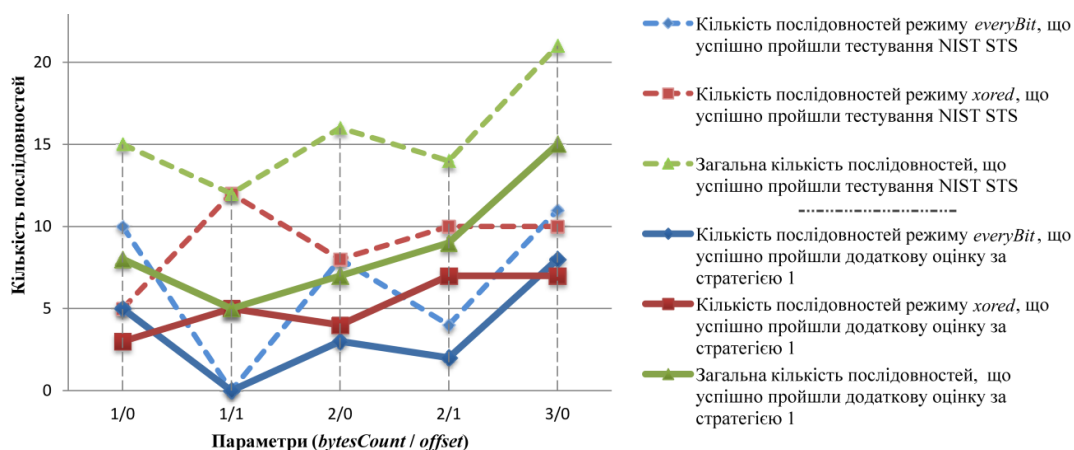
Режим роботи генератора	Кількість пройдених статистичних тестів NIST STS (макс. 188) ГПВЧ на основі функції $y = x$ згідно з двома методами оцінки (класичний / стратегія 1) при $x_0 = 0$ та вибраних параметрах ГПВЧ					
	<i>bytesCount</i> = 1		<i>bytesCount</i> = 2		<i>bytesCount</i> = 3	
	<i>offset</i> = 0	<i>offset</i> = 1	<i>offset</i> = 0	<i>offset</i> = 1	<i>offset</i> = 0	<i>offset</i> = 1
<i>everyBit</i>	185 / 185	184 / –	187 / 186	188 / 185	188 / 188	187 / 186
<i>xored</i>	187 / 184	188 / 188	187 / 187	188 / 188	188 / 187	186 / 185

Таблиця Ж.2 – Приклад статистичного тестування ГПВЧ на основі алгоритмів Вальчика та ін. [140] при фіксованому значенні $x_0 = 0$ згідно з стратегією 1 при $m = 1000$

Режим роботи генератора	Кількість ітерацій алгоритму Вальчика та ін. [140]	Кількість пройдених статистичних тестів NIST STS (макс. 188) ГПВЧ на основі алгоритмів $y \in \{rsqrt1w, rsqrt2w, rsqrt3w\}$ згідно з стратегією 1 при $x_0 = 0$ та вибраних параметрах ГПВЧ					
		<i>bytesCount</i> = 1		<i>bytesCount</i> = 2		<i>bytesCount</i> = 3	
		<i>offset</i> = 0	<i>offset</i> = 1	<i>offset</i> = 0	<i>offset</i> = 1	<i>offset</i> = 0	<i>offset</i> = 1
<i>everyBit</i>	1	187	185	186	185	184	185
	2	187	187	186	183	188	181
	3	–	187	188	183	188	180
<i>xored</i>	1	188	187	186	186	185	–
	2	187	188	187	188	188	185
	3	185	187	187	188	188	185



а



б

Рисунок Ж.1 – Порівняння результатів статистичного тестування NIST STS при $x_0 = 0$ в залежності від режиму роботи ГПВЧ та використаної стратегії оцінки:

(а) для вибору оптимального алгоритму обчислення елементарної функції та

(б) для вибору оптимальних значень параметрів *bytesCount* і *offset*

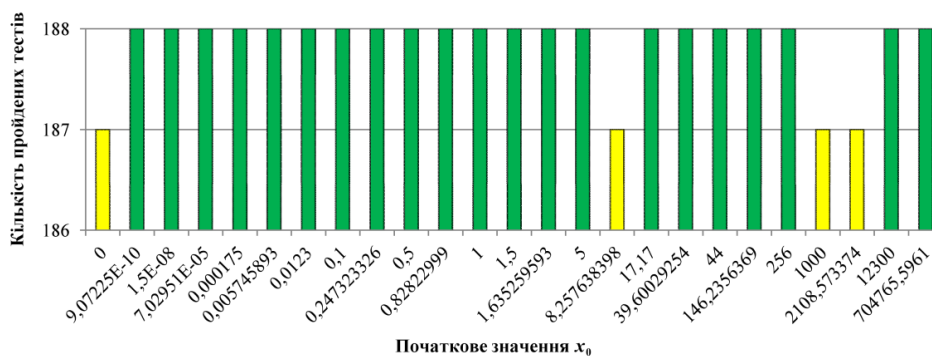
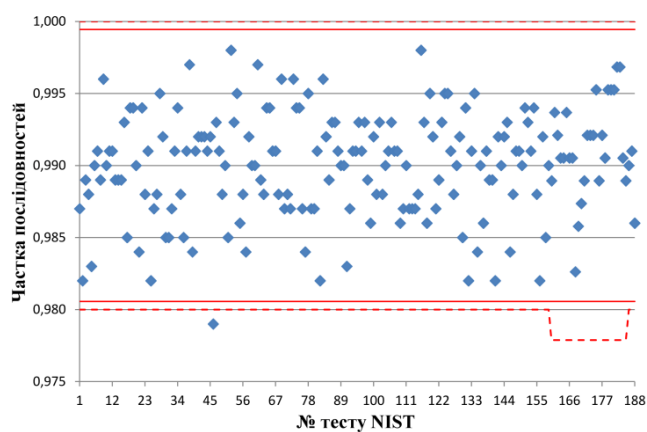
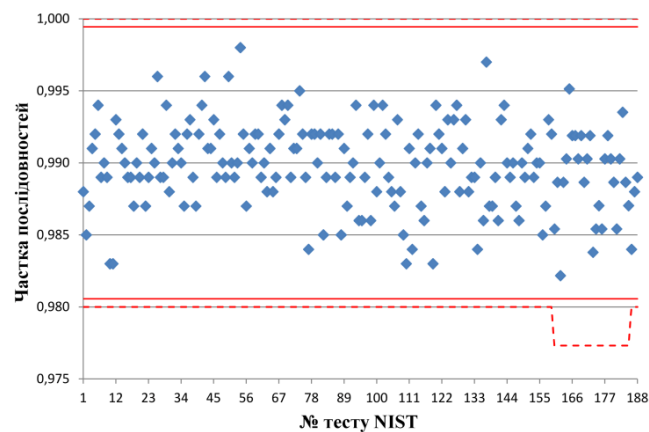


Рисунок Ж.2 – Приклад одержаних результатів тестів NIST для ГПВЧ

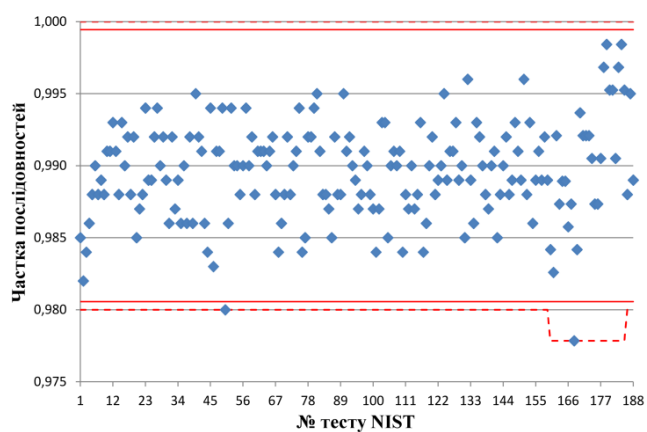
rsqrt2dc_everyBit (3/0) в залежності від $x_0 \in \{0\} \cup (10^{-10}; 10^6)$



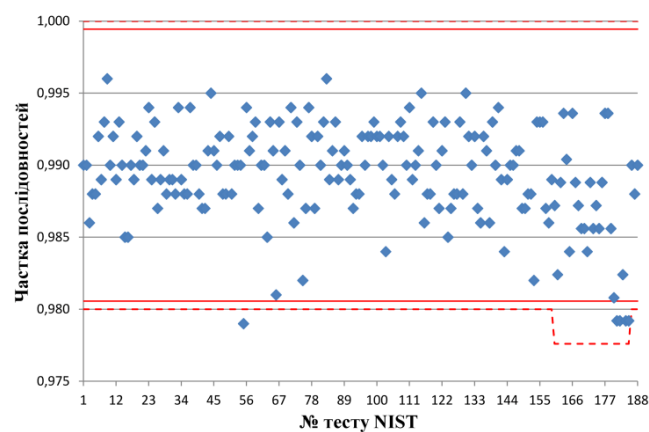
а



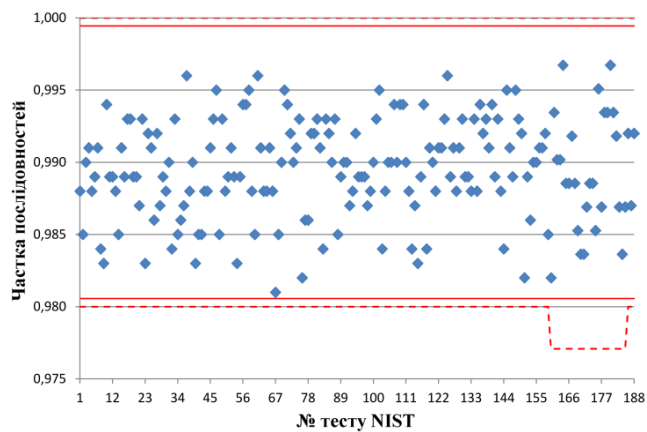
б



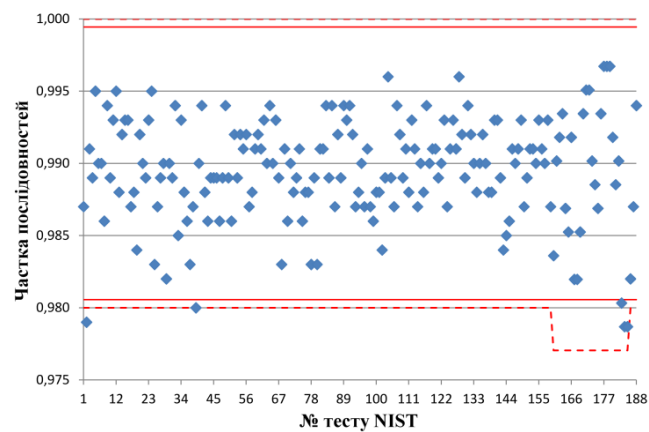
в



г



д



е

Рисунок Ж.3 – Приклади статистичних портретів ГПВЧ *rsqrt2dc_everyBit* (3/0) з початковим значенням: (а) $x_0 = 0$, (б) $x_0 = 9,07225 \cdot 10^{-10}$, (в) $x_0 = 7,02951 \cdot 10^{-5}$, (г) $x_0 = 8,257638398$, (д) $x_0 = 17,17$ та (е) $x_0 = 2108,573374$

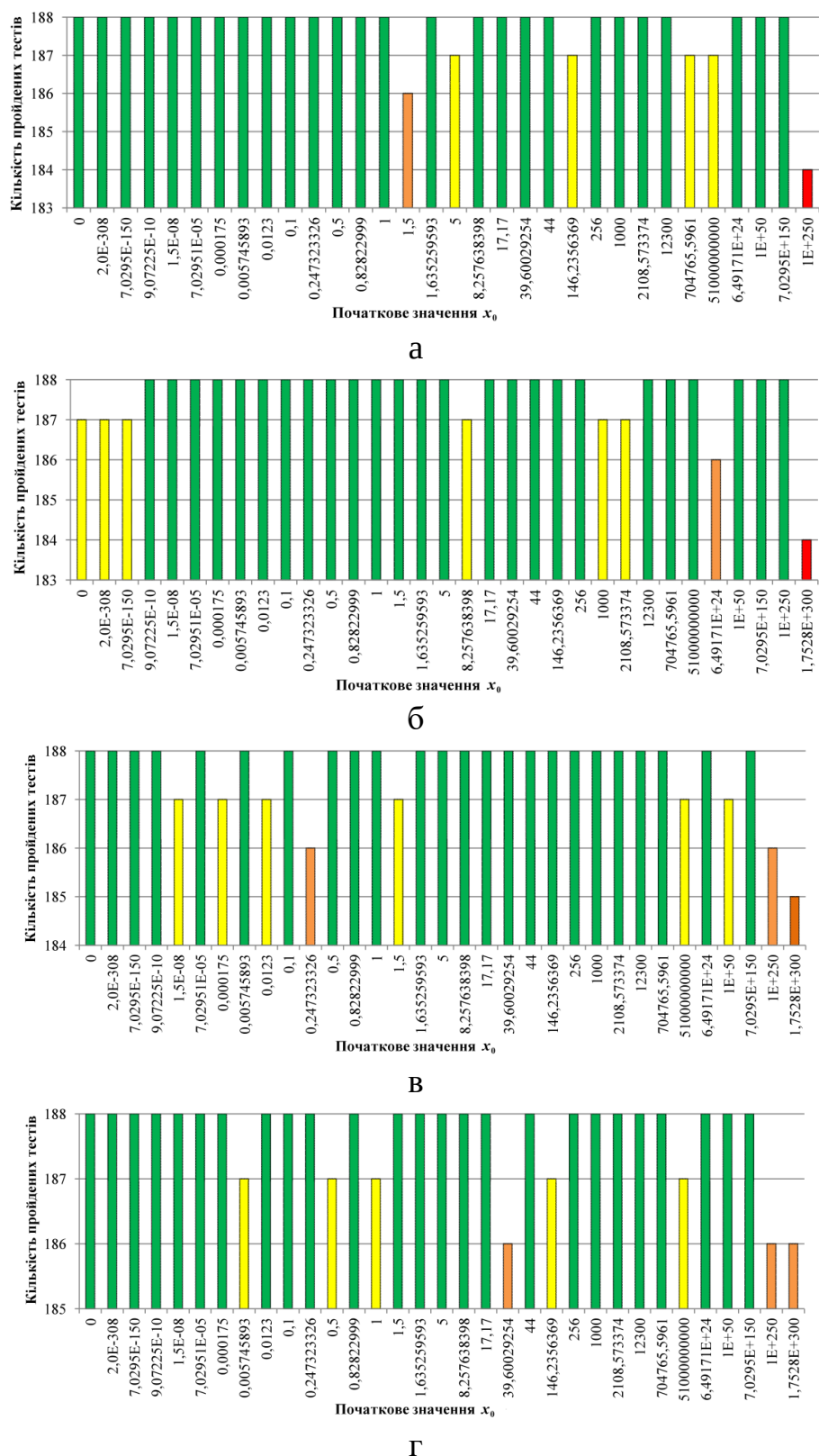
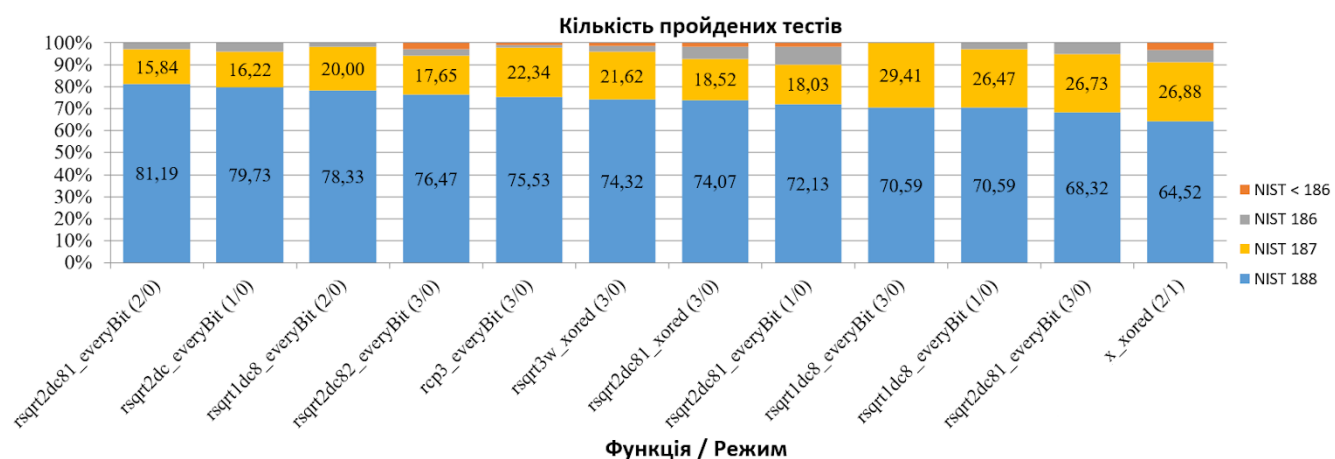
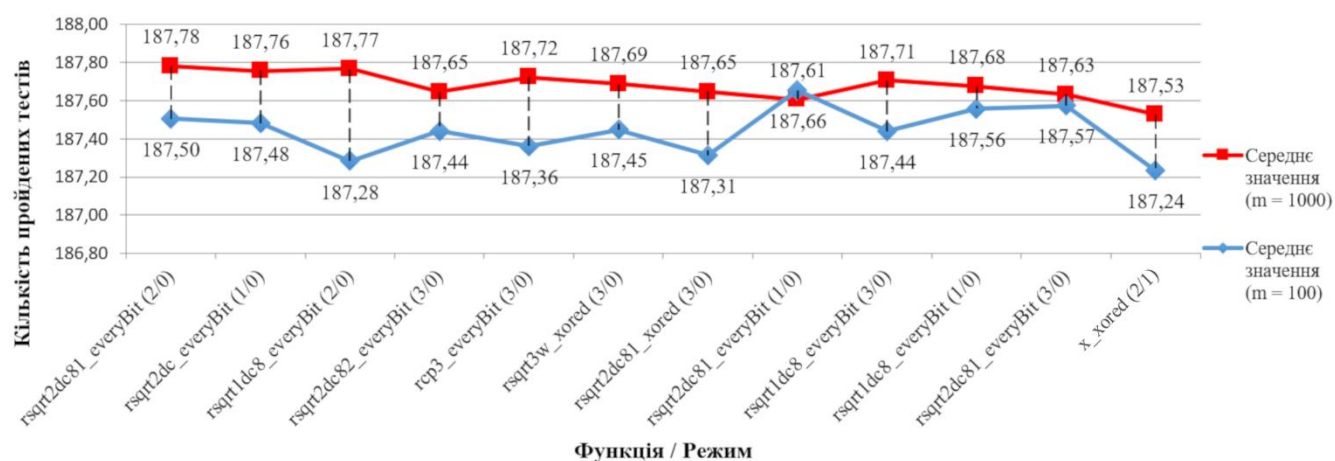


Рисунок Ж.4 – Результати тестів NIST для деяких інших ГПВЧ із найкращими характеристиками для розширеного діапазону початкових значень:

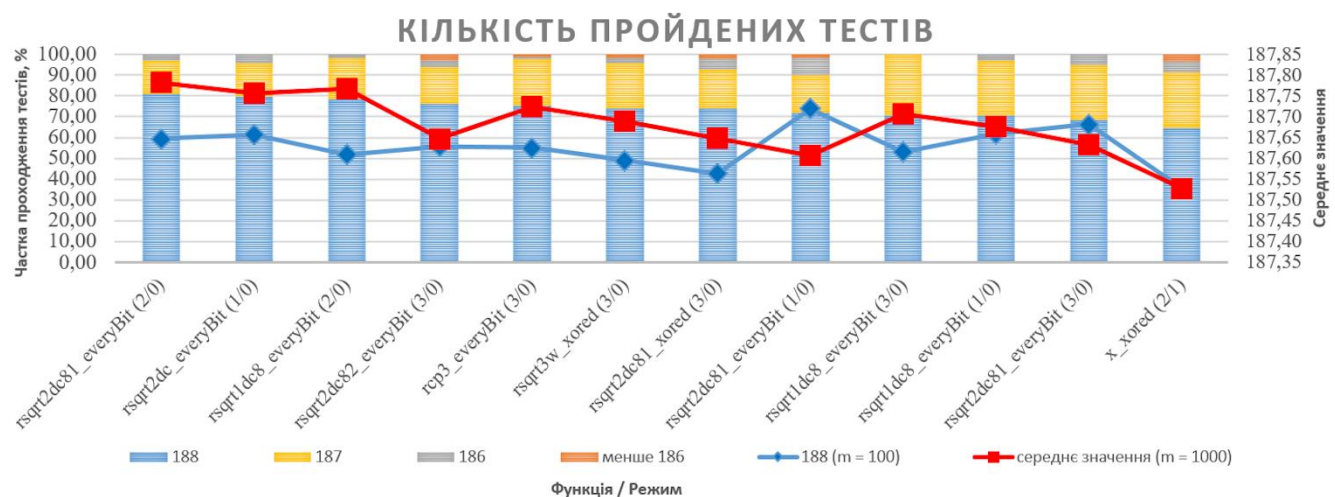
(а) *rsqrt2dc_everyBit* (1/0), (б) *rsqrt2dc_everyBit* (3/0), (в) *rcp3_everyBit* (3/0) та (г) *rsqrt3w_xored* (3/0)



а



б



в

Рисунок Ж.5 – Результати статистичних тестів NIST для запропонованих ГПВЧ із найкращими характеристиками для випадкових x_0 : (а) відсоток проходження тестів NIST STS при $m = 1000$, (б) середня кількість пройдених тестів NIST STS з параметрами $m = 1000$ і $m = 100$ та (в) відсоток проходження тестів NIST STS при $m = 1000$ і $m = 100$, а також середня кількість пройдених тестів при $m = 1000$

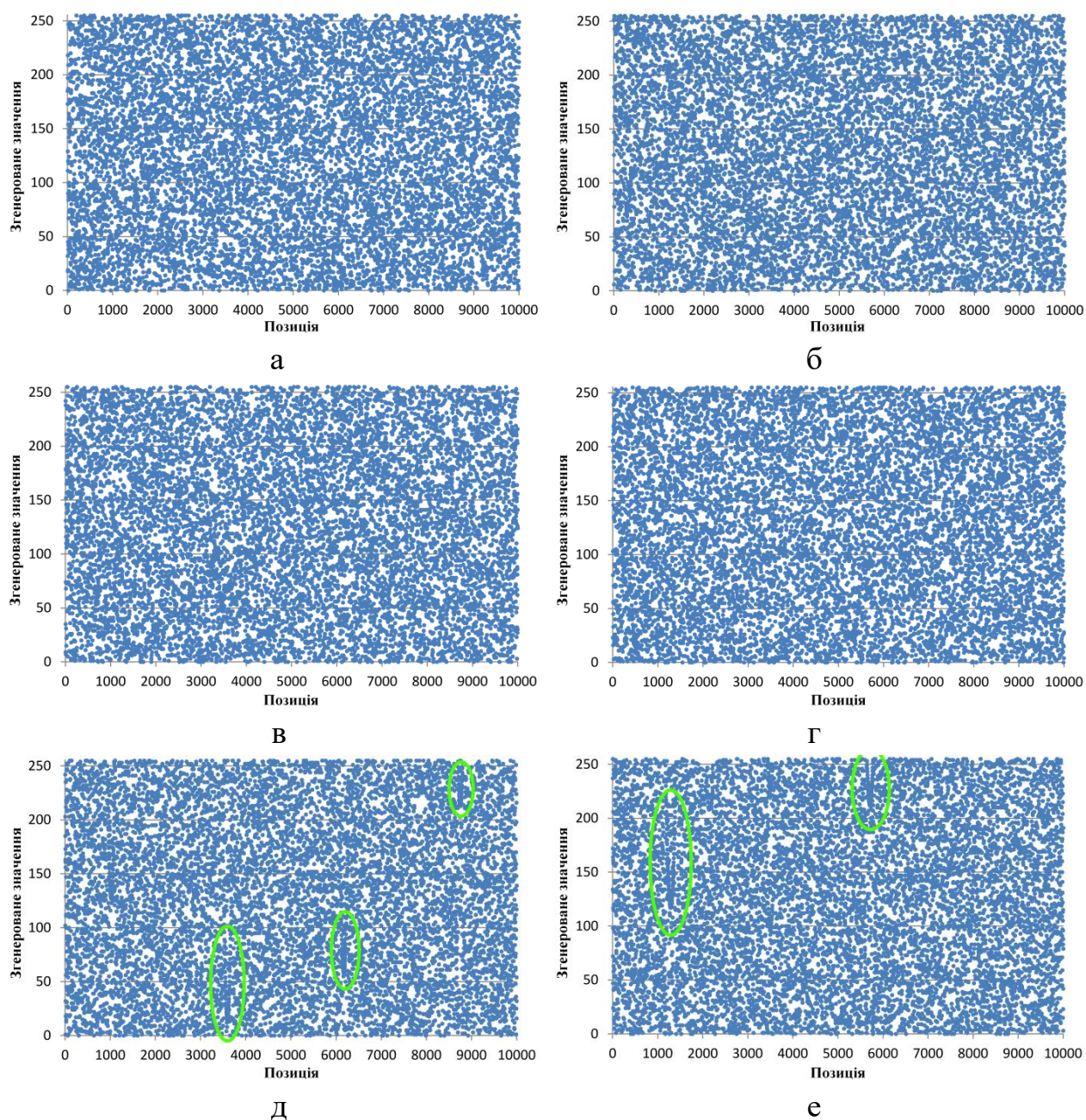
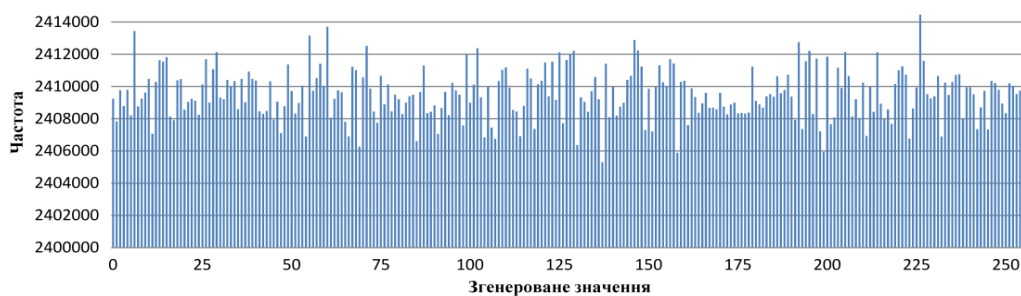


Рисунок Ж.6 – Графічне представлення перших і останніх 10000 елементів сформованої неперіодичної послідовності ПВЧ розміру L значень (байтів) для початкового значення $x_0 = 6,491708 \cdot 10^{24}$ вибраних генераторів:

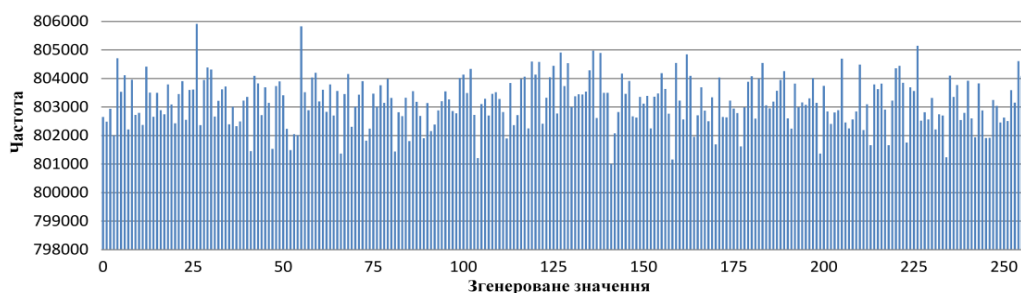
(а) і (б) ГПВЧ *rsqrt2dc_everyBit* (1/0), $L = 205,61 \cdot 10^6$ байт;

(в) і (г) ГПВЧ *rsqrt3w_xored* (3/0), $L = 616,84 \cdot 10^6$ байт;

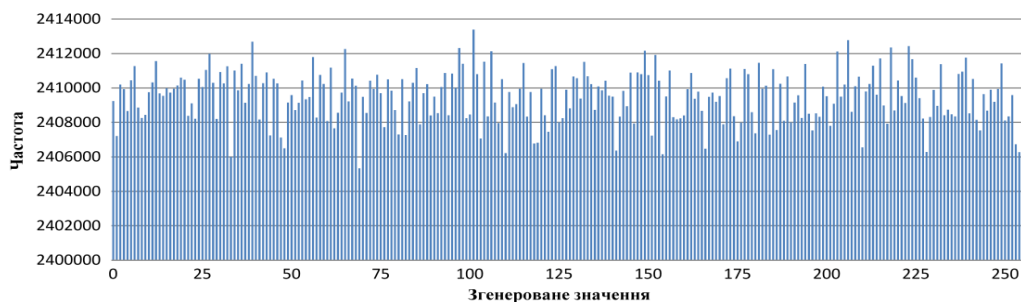
(д) і (е) ГПВЧ *rsqrt2dc_everyBit* (3/0), $L = 616,84 \cdot 10^6$ байт



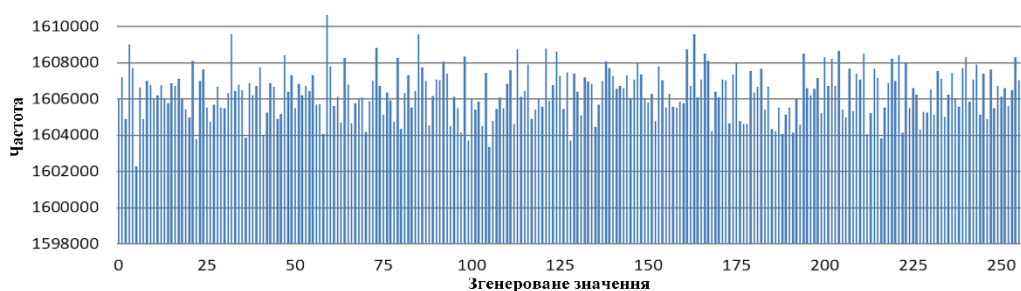
а



б



в



г

Рисунок Ж.7 – Гістограма сформованої неперіодичної послідовності ПВЧ розміру

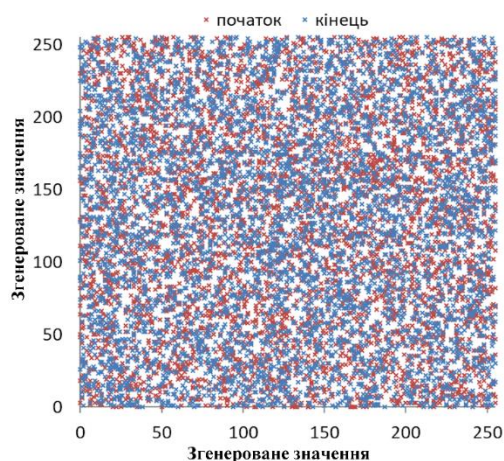
L значень (байтів) при $x_0 = 6,491708 \cdot 10^{24}$ для вибраних генераторів:

(а) ГПВЧ *rsqrt2dc_everyBit* (1/0), $L = 205,61 \cdot 10^6$ байт, $H(D_L) = 7,999999$ біт;

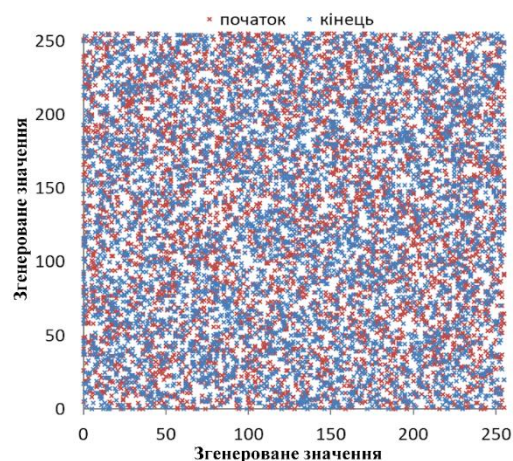
(б) ГПВЧ *rsqrt3w_xored* (3/0), $L = 616,84 \cdot 10^6$ байт, $H(D_L) = 8,0$ біт;

(в) ГПВЧ *rsqrt2dc_everyBit* (3/0), $L = 616,84 \cdot 10^6$ байт, $H(D_L) = 8,0$ біт;

(г) ГПВЧ *rsqrt2dc81_everyBit* (2/0), $L = 411,22 \cdot 10^6$ байт, $H(D_L) = 8,0$ біт



а

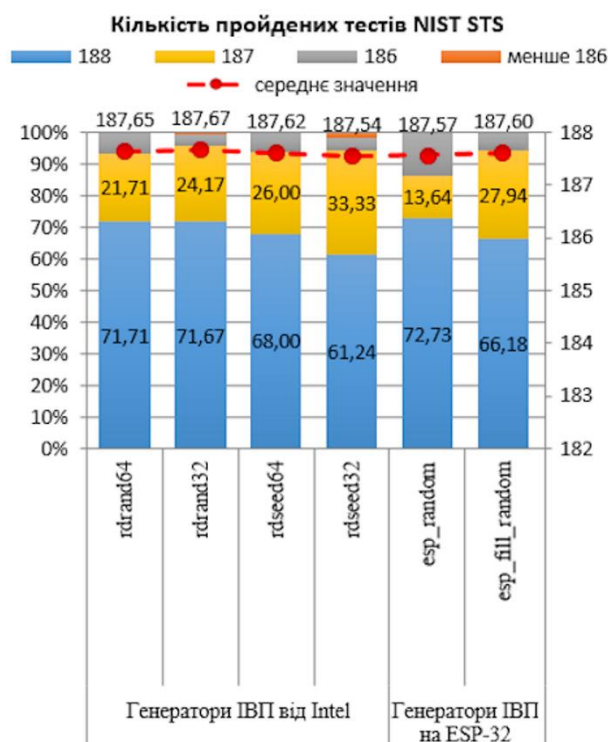


б

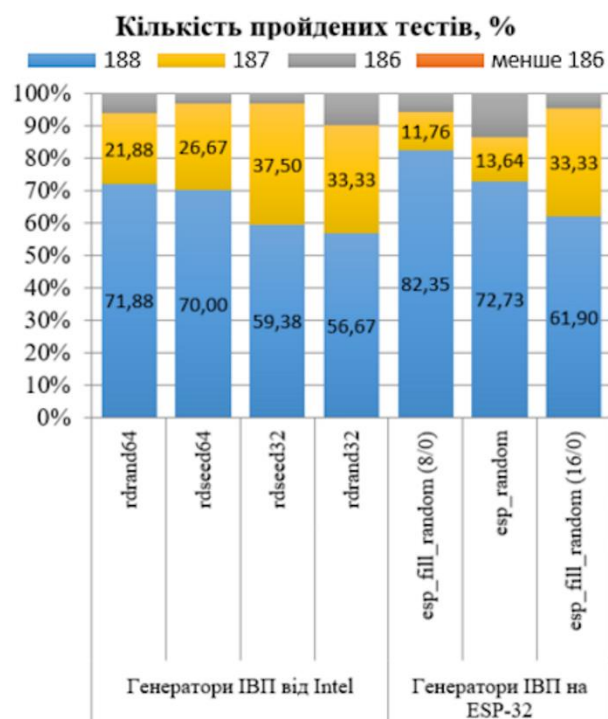
Рисунок Ж.8 – Результати графічного тесту: (а) ГПВЧ *rsqrt2dc81_everyBit* (2/0)

для $x_0 = 6,491708 \cdot 10^{24}$, $L = 411,22 \cdot 10^6$ значень (байт), $H(D_L) = 8,0$ біт та

(б) генератор ІВЧ *rdrand64* від Intel, $L = 214,75 \cdot 10^7$ байт (2 Гб), $H(D_L) = 8,0$ біт

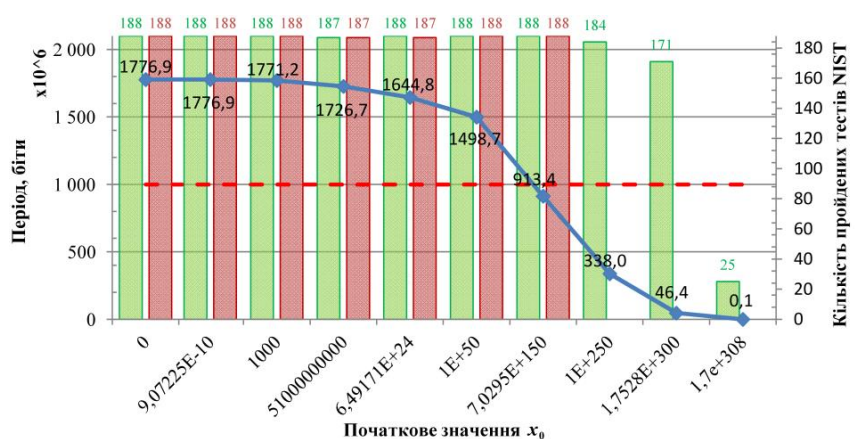


а

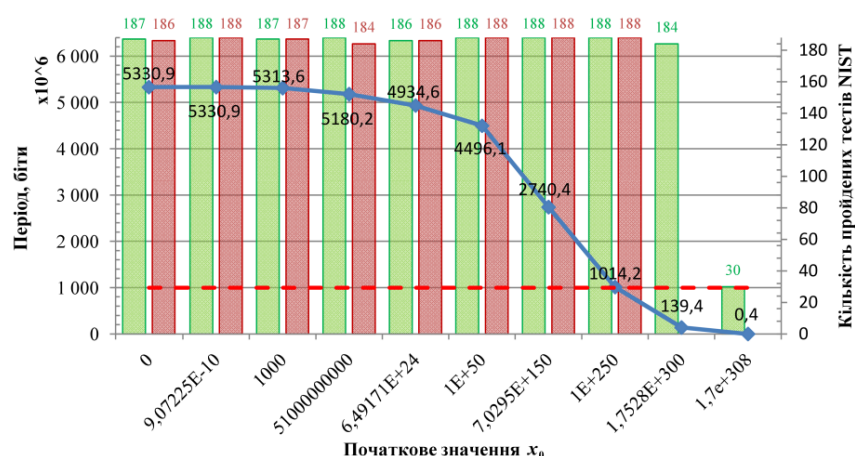


б

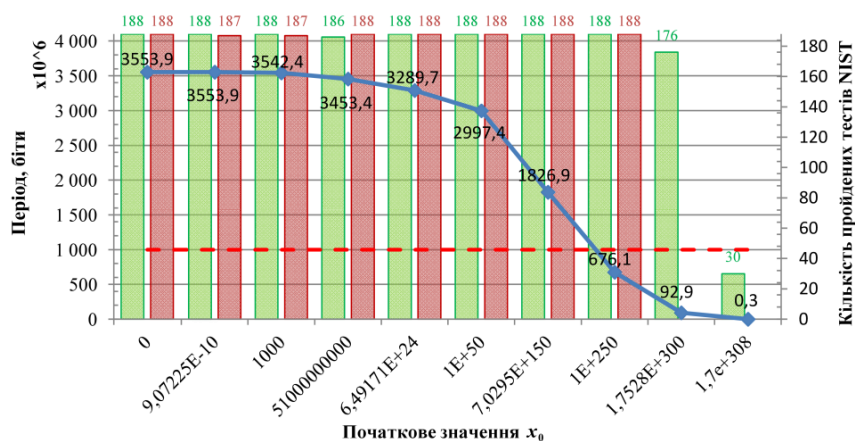
Рисунок Ж.9 – Результати статистичного тестування NIST STS відомих генераторів ІВП, доступних на Intel та ESP-32, при $m = 1000$: (а) узагальнені результати відсотку проходження та середнього значення кількості пройдених тестів за типами і параметрами генерації та (б) результати тестування основних типів генераторів ІВП із фіксованими параметрами



а



б



в

Рисунок Ж.10 – Залежність результатів статистичного тестування NIST та періоду повторення L ГПВЧ від початкового значення x_0 при фіксованому ($m = 1000$) та динамічному m : (а) *rsqrt2dc_everyBit* (1/0), (б) *rsqrt2dc_everyBit* (3/0) та (в) *rsqrt3w_xored* (2/0)

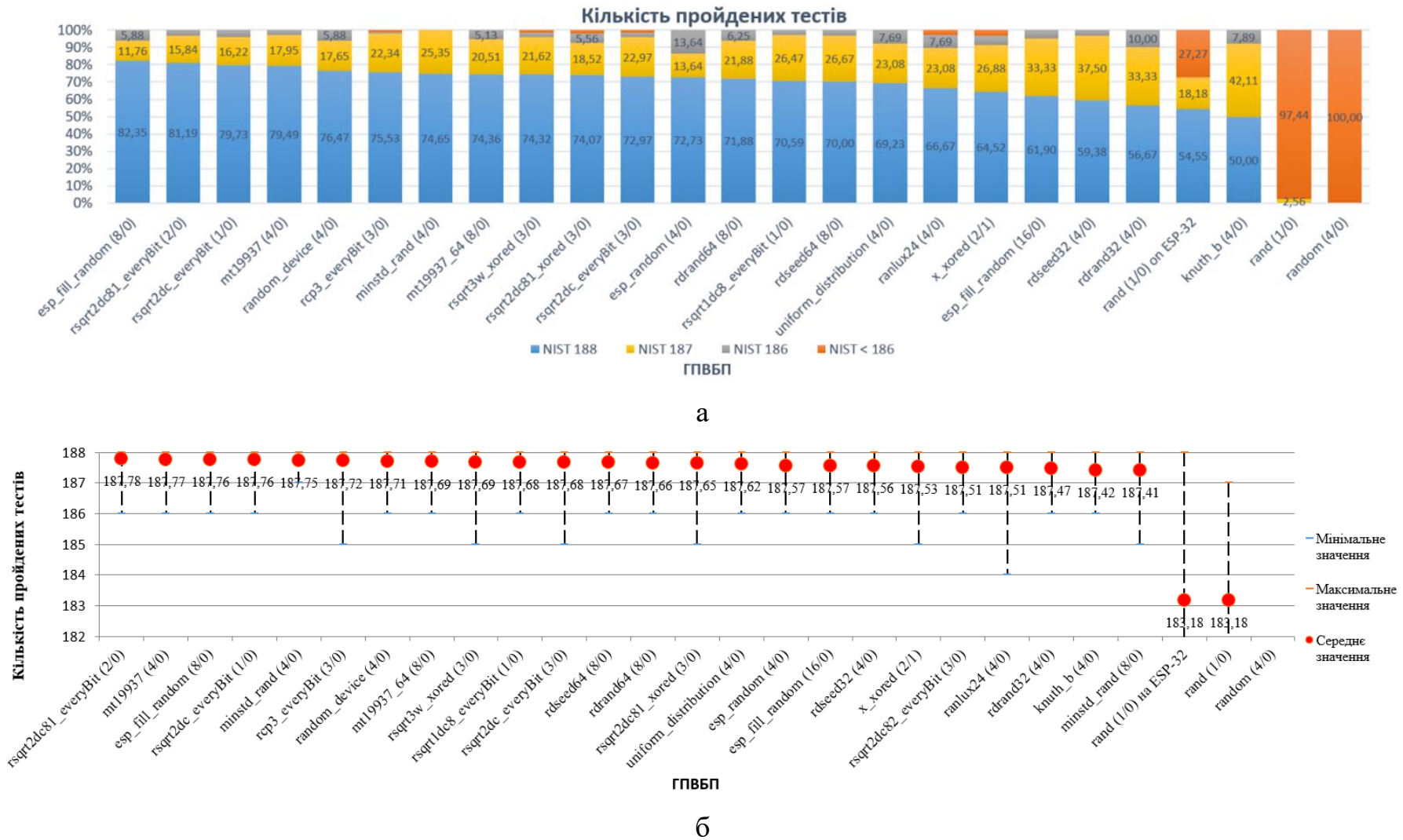


Рисунок Ж.11 – Порівняння відомих генераторів ІВЧ та ГПВЧ, доступних на Intel та ESP-32, із запропонованими ГПВЧ на основі чисельних методів в FP арифметиці: (а) за відсотком проходження статистичних тестів NIST та (б) за середнім значенням кількості пройдених тестів

Таблиця Ж.3 – Зведена таблиця із результатами статистичного тестування запропонованих ГПВЧ із найкращими характеристиками та оцінка отриманих результатів

ГПВЧ <i>y_mode</i> (<i>bytesCount / offset</i>)	Максимальний період повторення L_{max} , байт	Рекомендований діапазон початкових значень x_0 ($L \geq 125 \cdot 10^6$ байт)	Середня кількість пройдених тестів NIST	Відсоток проходження всіх тестів NIST		Проходження графічних тестів	Оцінка статистичних характеристик ГПВЧ	
				при $m = 1000$	при зміні m		Метрика 1	Метрика 2
<i>rsqrt2dc81_everyBit</i> (2/0)	$2^{28,73}$	$[0,5; 10^{220}]$	<u>187,78</u>	<u>81,19%</u>	42,86%	+	63,23	59,43
<i>rsqrt2dc_everyBit</i> (1/0)	$2^{27,73}$	$[0,5; 10^{135}]$	<u>187,76</u>	<u>79,73%</u>	<u>68,42%</u>	+	<u>73,49</u>	<u>67,57</u>
<i>rsqrt1dc8_everyBit</i> (2/0)	$2^{28,73}$	$[0,5; 10^{220}]$	<u>187,77</u>	<u>78,33%</u>	46,15%	+	63,16	54,86
<i>rsqrt2dc82_everyBit</i> (3/0)	$2^{29,31}$	$[0,5; 10^{250}]$	187,65	76,47%	58,33%	+	61,84	57,28
<i>rcp3_everyBit</i> (3/0)	$2^{29,31}$	$[0,5; 10^{250}]$	187,72	75,53%	61,54%	+	<u>68,53</u>	<u>61,18</u>
<i>rsqrt3w_xored</i> (3/0)	$2^{29,31}$	$[0,5; 10^{250}]$	187,69	74,32%	36,84%	+	49,61	46,56
<i>rsqrt2dc81_xored</i> (3/0)	$2^{29,31}$	$[0,5; 10^{250}]$	187,65	74,07%	50,00%	+	59,26	51,9
<i>rsqrt2dc_everyBit</i> (3/0)	$2^{29,31}$	$[0,5; 10^{250}]$	187,68	72,97%	40,00%	–	49,41	37,98
<i>rsqrt2dc81_everyBit</i> (1/0)	$2^{27,73}$	$[0,5; 10^{135}]$	187,61	72,13%	56,25%	+	62,05	<u>63,6</u>
<i>rsqrt1dc8_everyBit</i> (3/0)	$2^{29,31}$	$[0,5; 10^{250}]$	187,71	70,59%	33,33%	+	50,8	48,13
<i>rsqrt1dc8_everyBit</i> (1/0)	$2^{27,73}$	$[0,5; 10^{135}]$	187,68	70,59%	50,00%	+	61,33	59,93
<i>rsqrt2dc81_everyBit</i> (3/0)	$2^{29,31}$	$[0,5; 10^{250}]$	187,63	68,32%	40,00%	+	51,03	52,35
<i>x_xored</i> (2/1)	$2^{28,73}$	$[0,5; 10^{220}]$	187,53	64,52%	<u>75,00%</u>	+	54,34	45,79
<i>rsqrt3dc_xored</i> (2/0)	$2^{28,73}$	$[0,5; 10^{220}]$	187,64	64,48%	<u>66,66%</u>	+	63,79	60,67

Таблиця Ж.4 – Аналіз ефективності реалізації запропонованих ГПВЧ за середньою кількістю використаних операцій для формування одного байта вихідної ПВП та узагальнені рекомендації щодо використання ГПВЧ

ГПВЧ <i>y_mode</i> (<i>bytesCount / offset</i>)	Кількість ітерацій ГПВЧ	Кількість основних операцій на ітерацію ГПВЧ			Кількість основних операцій на один згенерований байт	Рекомендація щодо використання запропонованих ГПВЧ	
		Функція	Основний алгоритм ГПВЧ	Режим <i>mode</i>		Статистичні характери- стики	Швидко- дія
<i>rsqrt2dc81_everyBit</i> (2/0)	<i>fileSize/2</i>	1MC, 5×, 2FMA, 1+	1×,1+	–	1/2MC, 3×, FMA, 1+	++	+
<i>rsqrt2dc_everyBit</i> (1/0)	<i>fileSize</i>	1MC, 5×, 3FMA	1×,1+	–	1MC, 6×, 3FMA, 1+	+++	–
<i>rsqrt1dc8_everyBit</i> (2/0)	<i>fileSize/2</i>	1MC, 3×, 1FMA	1×,1+	–	1/2MC, 2×, 1/2FMA, 1/2+	++	++
<i>rsqrt2dc82_everyBit</i> (3/0)	<i>fileSize/3</i>	1MC, 5×, 4FMA	1×,1+	–	1/3MC, 2×, 4/3FMA, 1/3+	+	+
<i>rcp3_everyBit</i> (3/0)	<i>fileSize/3</i>	1MC, 3×, 4FMA	1×,1+	–	1/3MC, 4/3×, 4/3FMA, 1/3+	++	++
<i>rsqrt3w_xored</i> (3/0)	<i>fileSize/3</i>	1MC, 10×, 3+	1×,1+	3XOR	1/3MC, 11/3×, 4/3+, XOR	–	+
<i>rsqrt2dc81_xored</i> (3/0)	<i>fileSize/3</i>	1MC, 5×, 2FMA, 1+	1×,1+	3XOR	1/3MC, 2×, 2/3FMA, 2/3+, 1XOR	+–	++

Продовження таблиці Ж.4

ГПВЧ <i>y_mode</i> (<i>bytesCount / offset</i>)	Кількість ітерацій ГПВЧ	Кількість основних операцій на ітерацію ГПВЧ			Кількість основних операцій на один згенерований байт	Рекомендація щодо використання запропонованих ГПВЧ	
		Функція	Основний алгоритм ГПВЧ	Режим <i>mode</i>		Статистичні характери- стики	Швидко- дія
<i>rsqrt2dc_everyBit</i> (3/0)	<i>fileSize/3</i>	1MC, 5×, 3FMA	1×,1+	–	1/3MC, 2×, 1FMA, 1/3+	–	+
<i>rsqrt2dc81_everyBit</i> (1/0)	<i>fileSize</i>	1MC, 5×, 2FMA, 1+	1×,1+	–	1MC, 6×, 2FMA, 2+	+ –	–
<i>rsqrt1dc8_everyBit</i> (3/0)	<i>fileSize/3</i>	1MC, 3×, 1FMA	1×,1+	–	1/3MC, 4/3×, 1/3FMA, 1/3+	– –	+ +
<i>rsqrt1dc8_everyBit</i> (1/0)	<i>fileSize</i>	1MC, 3×, 1FMA	1×,1+	–	1MC, 4×, 1FMA, 1+	+	+ –
<i>rsqrt3w_xored</i> (2/0)	<i>fileSize/2</i>	1MC, 10×, 3+	1×,1+	2XOR	1/2MC, 11/2×, 2+, XOR	+ –	+ –
<i>rsqrt2dc81_everyBit</i> (3/0)	<i>fileSize/3</i>	1MC, 5×, 2FMA, 1+	1×,1+	–	1/3MC, 2×, 2/3FMA, 2/3+	–	+ +
<i>x_xored</i> (2/1)	<i>fileSize/2</i>	–	1×,1+	2XOR	1/2×, 1/2+, XOR	–	+ + +
<i>rsqrt3dc_xored</i> (2/0)	<i>fileSize/2</i>	1MC, 7×, 4FMA	1×,1+	2XOR	1/2MC, 4×, 2FMA, 1/2+, XOR	+	+ –

Додаток И

Список публікацій здобувача за темою дисертації та відомості про апробацію результатів дисертації

Наукові праці, в яких опубліковані основні наукові результати дисертації

1. Horyachyy O., Moroz L., Otenko V. Simple effective fast inverse square root algorithm with two magic constants. *International Journal of Computing*. 2019. Vol. 18, No 4. P. 461–470. (Scopus, Q3)
2. Максимович В., Шабатура М., Горячий О., Лужецька Н. Генератори псевдовипадкових бітових послідовностей на основі чисельних методів в арифметиці з рухомою комою. *Сучасна спеціальна техніка*. 2021. № 1(64). С. 81–92.
3. Hrynchyshyn A., Horyachyy O., Tymoshenko O., Moroz L. An efficient algorithm for fast inverse square root. *Processing, transmission and security of information* : monograph. Bielsko-Biała, Poland, 2018. Vol. 2. P. 105–114.
4. Moroz L. V., Samoty V. V., Horyachyy O. Y. Modified fast inverse square root and square root approximation algorithms: the method of switching magic constants. *Computation*. 2021. Vol. 9, Issue 2, 21. 23 p. (Scopus, Web of Science, Q2)
5. Горячий О., Максимович В., Шабатура М. Дослідження множини початкових значень генераторів псевдовипадкових чисел на основі арифметики з рухомою комою. *Сучасний захист інформації*. 2024. № 2(58). С. 91–102.

Наукові праці, які засвідчують апробацію матеріалів дисертації

1. Moroz L., Samoty V., Horyachyy O. An effective floating-point reciprocal. *Wireless systems* : proc. 2018 4th IEEE int. symp. within the int. conf. on intelligent data acquisition and advanced computing systems (IDAACS-SWS 2018), Lviv, September 20-21, 2018. P. 137–141 (форма участі: усна доповідь). (Scopus, Web of Science)
2. Moroz L., Samoty V., Horyachyy O., Dzelendzyak U. Algorithms for calculating the square root and inverse square root based on the second-order Householder's method. *Intelligent data acquisition and advanced computing systems: technology*

- and applications* : proc. 2019 10th IEEE int. conf. (IDAACS), Metz, France, September 18-21, 2019. Vol. 1. P. 436–442 (форма участі: усна доповідь). (Scopus, Web of Science)
3. Horyachyy O. Improving the efficiency of methods for calculating elementary functions in floating-point arithmetic. *Informatics education and educational software engineering research* : int. doc. cons., Druskininkai, Lithuania, December 2-6, 2019. P. 2–7 (форма участі: стендова доповідь).
 4. Горячий О., Журавель І. Дослідження ефективності простих стеганографічних методів обробки цифрових зображень із використанням генераторів псевдовипадкових послідовностей. *Інформаційна безпека та інформаційні технології* : зб. тез доп. V міжнар. наук.-практ. конф. (ІБІТ-2024), Львів, 27 листопада, 2024. С. 225–230 (форма участі: усна доповідь).
 5. Горячий О. Я., Журавель І. М. Порівняння простих методів стеганографічного захисту інформації на основі аналізу цифрових зображень та використання псевдовипадкових послідовностей. *Автоматизація, електроніка, інформаційно-вимірювальні технології: освіта, наука, практика* : матер. V міжнар. наук.-техн. конф., Харків, 28–29 листопада, 2024. С. 27–28 (форма участі: усна онлайн доповідь).

Наукові праці, які додатково відображають наукові результати дисертації

1. Horyachyy O., Boretskyu T., Horiachyi I. Improving the methods of protection and hacking of modern Wi-Fi networks. *Information protection and information systems security* : materials 7th int. scient. techn. conf., Lviv, Ukraine, May 30-31, 2019. P. 60–61 (форма участі: усна доповідь співавтора).
2. Мороз Л. В., Гринчишин А. І., Горячий О. Я. Проста модифікація алгоритму швидкого обчислення зворотного квадратного кореня для чисел з рухомою комою одинарної точності. *Автоматика, вимірювання та керування*. 2019. Вип. 1, № 1. С. 39–45.
3. Horyachyy I., Horyachyy O., Boretskyu T. Information security management of a banking institution based on outsourcing. *Information protection and information*

- systems security* : materials 8th int. scient. techn. conf., Lviv, Ukraine, November 11-12, 2021. – P. 11–12 (*форма участі: без доповіді*).
4. Shyshko D., Horyachyy O. Overview of main cybersecurity challenges of smart cities and methods of their protection. *Information protection and information systems security* : materials 8th int. scient. techn. conf., Lviv, Ukraine, November 11-12, 2021. – P. 31–32 (*форма участі: усна доповідь співавтора*).
5. Горячий О., Яремчук З. Аналіз ефективності генераторів псевдовипадкових чисел на основі квадратного кореня простого числа. *Інформаційна безпека та інформаційні технології* : зб. тез доп. V міжнар. наук.-практ. конф. (ІБІТ-2024), Львів, 27 листопада, 2024. С. 234–237. (*форма участі: усна доповідь співавтора*).