ISSN 2224-087X. Електроніка та інформаційні технології. 2019. Випуск 11. С. 81–98
Electronics and information technologies. 2019. Issue 11. P. 81–98

УДК 519.6 DOI: https://doi.org/10.30970/eli.11.8

ДОВГА АРИФМЕТИКА В EXCEL. I. VBA-РЕАЛІЗАЦІЯ

В. Фурман, М. Хом'як, Я. Марко

Львівський національний університет імені Івана Франка,
вул. Грушевського, 4, 79005, Львів, Україна
fourman@i.ua

Проаналізовано програмні та інтернет-ресурси для точних обчислень. Ми пропонуємо використовувати популярні електронні таблиці Excel як зручний і достатньо потужний символьний калькулятор. За допомогою VBA реалізовано базові операції: порівняння, арифметичні дії та знаходження НСД двох цілих чисел довільної розрядності. Обґрунтовано алгоритм скороченого ділення десяткових чисел, який не потребує циклу віднімання для корекції розряду-результату. Ці алгоритми є необхідними для роботи з раціональними числами та складніших обчислень на їхній основі (наприклад, для розв'язування систем лінійних алгебраїчних рівнянь). У подальшому дана версія може бути прототипом для ефективнішого перепроектування, особливо, з використанням двійкових алгоритмів, реалізованих на сучасних процесорах.

Ключові слова: електронні таблиці, надбудова Excel, символьні обчислення, алгоритми ділення.

Часто ми потребуємо достатньо простих, зручних і знайомих програмних інструментів для вирішення наукових, навчальних або прикладних завдань, зокрема, й для нескладних арифметичних розрахунків чи тестування окремих частин складного програмного коду. Електронні таблиці Excel достатньо популярні, інтуїтивно зрозумілі та легко освоюються, тому багато користувачів використовують їх, принаймні, як сховище кількісних даних. Разом з тим, електронні таблиці – це й потужний калькулятор [1], а середовище Visual Basic for Application (VBA) [2] дає змогу розширювати функціональні можливості, не заглиблюючись у тонкощі сучасних IT-технологій і програмування. Якщо питання ефективності, зокрема швидкодії, гостро не стоїть, то код-прототип у VBA – цілком виправдане рішення для апробації нових ідей та алгоритмів або отримання конкретних числових результатів.

На жаль, символьні обчислення або арифметика чисел з довільною розрядністю на даний час в Excel не підтримується, що спонукає користувачів самостійно розв'язувати цю проблему [3,4,5,6]. У VBA наявний універсальний тип Variant, який фактично інкапсулює один з базових типів даних, зокрема, для цілих чисел – це Long (32 біти, до 10 десяткових цифр), або для чисел з плаваючою крапкою – Double (64 біти або 15 значущих десяткових цифр). Дещо окремо стоять специфічні для Excel типи Decimal та Currency [2]. Тому арифметичні операції з цілими числами без похибок заокруглення на базі вказаних типів можна реалізувати тільки для обмеженої розрядності чисел: максимальна кількість значущих десяткових цифр, загалом, не перевищує 29. Зазвичай цього достатньо, однак таке обмеження не вирішує вказану проблему довільної

розрядності, принаймні в "розумних" межах (наприклад, хотілося оперувати хоча б 100–1000 десятковими цифрами). З іншого боку, для базових типів виконання операцій оптимізовано, часто на рівні команд процесора, тому бажано використати швидкодію цих типів для реалізації операцій довгої арифметики, наприклад, на основі цілих чисел. Ідея найпростішої з реалізацій, яку ми тут пропонуємо: зберігати кожну з десяткових цифр цілого числа як BCD-представлення [7] в масиві типу Byte. Хоч і неефективний щодо швидкодії, такий підхід використовує відомі зі школи, а тому інтуїтивно зрозумілі правила додавання, віднімання, множення й ділення "в стовпчик". Навпаки, двійкове представлення з використанням, наприклад, 32-бітного типу Long було б ефективнішим як щодо пам'яті, так і підтримки операцій сучасними процесорами, однак значно складнішим для реалізації "з нуля". Однак відмінності і мистецтво реалізації довгої арифметики не матиме жодного значення для користувача, як тільки він має змогу отримати потрібний результат, зокрема, за прийнятний бюджет часу, сумірний з іншим етапами його завдань, наприклад, підготовкою та діалоговим вводом даних.

Відмітимо, що на даний час символьні обчислення підтримуються багатьма іншими програмами та бібліотеками. Найперше треба згадати такі потужні комерційні програми комп'ютерної алгебри як Matlab, Mathematica та Mapple (за наявності вони можуть бути включені в Excel як надбудови [8,9,10]) або програми з відкритим кодом PARI/GP [111], SageMath [12] та багато інших. Про них можна дізнатися через пошук в інтернеті або, для початку, завітавши на сайт Wikipedia [13].

У різних мовах програмування та бібліотеках є підтримка великих цілих, іноді й раціональних чисел на основі їхнього представлення "чисельник/знаменник" як цілих чисел [14,15,16]. З включенням у NET Framework 4.0 класу BigInteger [17] появляється можливість досить легко і ефективно реалізувати аналогічну функціональність (зокрема, дій над раціональними числами) в електронних таблицях як надбудову з використанням динамічних бібліотек XLL (DLL) на C++ або C#.

Якщо, незалежно з яких причин (прагнення до самостійності, нові ідеї та застосування, з навчальною метою тощо), відмовитися від використання інших програм, то програмування в інтегрованому середовищі VBA дає можливість доповнити функціональність Excel. Наша мета – реалізувати арифметичні операції над цілими числами довільної розрядності, що будуть основою для класу раціональних чисел, а також складніших обчислень та алгоритмів, викладених у наступних частинах статті.

Розглянемо далі такі базові операції: порівняння, додавання, віднімання, множення та ділення двох довгих цілих чисел a і b з довжинами n_1 і n_2 відповідно. Ці VBA-алгоритми представлені в Додатку до електронної версії статті (Лістинги 1–7), а також у файлі надбудови RatioMatr.xla, що доступний як електронний ресурс за таким посиланням: http://old.geology.lnu.edu.ua/phis_geo/nauka/metod/metod.htm.

Розрізнятимемо два рівні: 1) зовнішній – для чисел у символьному (текстовому) форматі; 2) внутрішній – для реалізації на основі BCD-представлення [7] (або іншого, ефективнішого, з двійковою основою числення – як перспектива). Текстовий формат підтримується в клітинках Excel, а BCD-представлення працює у макросах VBA. Ми використовуватимемо динамічний масив, що містить кожну десяткову цифру {0,…,9} як беззнакове число типу Byte, починаючи від молодших розрядів (тобто у зворотному порядку щодо звичного запису числа зліва направо). Хоч межі масивів у VBA можна динамічно змінювати, ми дотримуватимемося концепції "одне число – одне виділення пам'яті", а для результату деякої дії треба створювати новий об'єкт у пам'яті. Тому

ISSN 2224-087X. Електроніка та інформаційні технології. 2019. Випуск 11

виділення пам'яті потрібно робити "про запас", на основі апріорних оцінок. Для цього зберігатимемо окремо кількість значущих цифр, крім нулів у старших розрядах. Отже внутрішнє представлення десяткового числа включатиме такі атрибути: 1) масив цифр типу Byte, де індексація починається від нуля і відповідає степеню 10 для ціни розряду; 2) довжина – кількість значущих цифр (фактичний розмір заповнення виділеної для масиву пам'яті) – тип Integer або Long; 3) знак числа. Для його створення потрібно задати десяткове число у символьному (текстовому) форматі, можливо, зі знаком (Лістинг 1**)**, а протилежна операція формує текстову стрічку з масиву цифр – обидві дії достатньо тривіальні.

Функція порівняння двох довгих цілих чисел A і B важлива як сама по собі, так і для реалізації інших операцій та алгоритмів. Результатом є одне з чисел –1, 0 або +1, якщо $A < B$, $A = B$ або $A > B$ відповідно. Алгоритм порівняння спочатку має враховувати знаки та кількості цифр двох чисел, а тоді вже – порозрядно порівнювати цифри, що є елементами масивів, якщо знаки і кількості цифр однакові (Лістинг 2). Передбачено також можливість порозрядного порівняння зі зміщенням початкового індексу першого, довшого числа, що використовується для реалізації операції ділення.

Додавання і віднімання. Аргументами цих операцій на зовнішньому рівні є два числа зі знаками, а на внутрішньому рівні достатньо реалізувати порозрядне додавання та віднімання, причому результат записується на місце першого аргументу (з накопиченням) (Лістинг 3). Довжину результату додавання $(A+B)$ апріорно оцінюємо як $n+1$, де $n = \max(n_1, n_2)$ – довжина більшого з двох аргументів A і B, а довжину результату віднімання $(A-B)$ – як довжину першого аргументу n_1, припускаючи $A \geq B$. Також передбачимо можливість зміщення початкового індексу першого, довшого числа. Додавання зі зміщенням використовується для реалізації порозрядного множення, а віднімання зі зміщенням – для ділення.

Відмітимо, що за необхідності довжину результату треба скоректувати в менший бік, якщо старша значуща цифра є нулем. Це справедливо й для інших операцій.

Множення. Результатом множення двох чисел A і B буде число, довжина якого не перевищує $n = n_1 + n_2 - 1$. На внутрішньому рівні реалізовано циклічне множення першого аргументу (беззнакове число як масив цифр) на наступний розряд другого аргументу (теж масив цифр); добуток додаємо (зі зміщенням) до результату. Вхідні аргументи не змінюються. На зовнішньому рівні знак результату має враховувати знаки аргументів згідно із загальноприйнятими правилами. Також ефективніше множити довше число A на коротше B (менше операцій додавання): $n_1 \geq n_2$. Ймовірно, додаткові перевірки, наприклад, чи містить розряд 0 або 1, також є корисними, оскільки зменшують фактичну кількість операцій множення для процесора (Лістинг 4).

Ділення (з остачею). Ця арифметична операція є найскладнішою, нетривіальною для реалізації [18,19]. Загальновідомо, що алгоритм ділення десяткових чисел "у стовпчик" полягає в послідовному знаходженні цифр результату (від старшого розряду до молодшого). Якщо цифра q підібрана вдало, то множимо дільник B на цю цифру та віднімаємо отриманий добуток від старших розрядів діленого A, причому кількості цих розрядів дорівнює одна одній або кількість розрядів діленого на 1 більше, ніж дільника. Для простоти тут припускаємо $A > B$. Інакше матимемо результат 0 (і залишок, що дорівнює діленому), якщо $A < B$; або 1 (і нульовий залишок), якщо $A = B$. Складність

полягає в "правильному" підборі цифри результату q: якщо взято "замало", то потрібно збільшити, наприклад, на одиницю, якщо ж "забагато" – то зменшити. Це вимагає додаткових повторних операцій множення або віднімання та циклічної структури алгоритму загалом. Вірною буде така цифра q_c, для якої результат віднімання (залишок) дорівнює нулю або менший дільника. Тоді переходимо до наступної, молодшої цифри або зупиняємо ділення з остачею, якщо вичерпано всі розряди.

На двійковому рівні й у командах процесора реалізовано ефективні алгоритми ділення, зокрема, SRT-алгоритми [20,21,22], що реалізують ідею *скороченого* ділення, коли для підбору наступної цифри результату аналізуються тільки декілька старших цифр діленого та дільника, а решту цифр ніби ігнорують. Ефективність алгоритму забезпечують наперед обчислена індексована таблиця (a look-up table), що місить "правильний" коефіцієнт для заданих аргументів-індексів. Аналогічний підхід можна реалізувати й для десяткових чисел. Як правило, беруть дві цифри діленого й одну цифру дільника: алгоритм класу "2:1". Дещо складніші алгоритми класів "3:2" чи інші, зокрема, комбіновані.

Зробимо деякі оцінки щодо результату *скороченого* ділення. Нехай задано два числа A і B, $0 < A < \beta^{n+1}$, $\beta^n / 2 \le B < \beta^n$, n – кількість розрядів (цифр) дільника, причому $B < A < \beta B$, де $\beta = 10$ – основа системи числення. Точним (коректним) результатом ділення є цифра $q_c = [A/B]$, $1 \le q_c \le \beta - 1$, де позначено $[x]$ – ціла частина числа x. За означенням ділення з остачею,

$$0 \le A - q_c B = r_c < B. \tag{1}$$

У числах A і B виділимо s старших розрядів (наприклад, для алгоритму класу "2:1" $s = 2$): $A = (a + m_a)\beta^n$, $B = (b + m_b)\beta^n$, причому $1 \le a < \beta^s$, $1 \le b < \beta^{s-1}$, $b \le a < \beta b$, $0 \le m_a, m_b < 1$ – "мантиси" (однакової довжини), що ігноруються в алгоритмі *скороченого* ділення. Враховуючи, що числа внутрішньо зберігаються як масиви цифр, запишемо символічно $A = a.m_a = a.([0..9]...[0..9])$, $B = b.m_b = b.([0..9]...[0..9])$, відділивши крапкою цифри, що ігноруємо для оцінки q_c (можна вважати їх дробовою частиною чисел, а декілька початкових, "лідерських" цифр – цілою частиною). Позначення $([0..9]...[0..9])$ трактуємо як масив довільних десяткових цифр, однакової довжини для A і B. Як пробне для ділення візьмемо значення-цифру $q = [a/b]$, $1 \le q \le \beta - 1$.

Використовуючи очевидну нерівність для натуральних чисел a і b,

$$\frac{a}{b+1} < \frac{a}{b} < \frac{a+1}{b},\ a \ge b > 0, \tag{2}$$

отримаємо нижню та верхню оцінки:

$$\left[\frac{a}{b+1}\right] = \left[\frac{a.[0]}{(b+1).[0]}\right] < \left[\frac{a.([0]...[0])}{b.([9]...[9])}\right] \le \left[\frac{A}{B}\right] = \left[\frac{a.m_a}{b.m_b}\right] = \left[\frac{a.([0..9]...[0..9])}{b.([0..9]...[0..9])}\right] \le$$

$$\le \left[\frac{a.([9]...[9])}{b.([0]...[0])}\right] < \left[\frac{(a+1).[0]}{b.[0]}\right] = \left[\frac{a+1}{b}\right]$$

або

ISSN 2224-087X. Електроніка та інформаційні технології. 2019. Випуск 11

$$\left\lfloor \frac{a}{b+1} \right\rfloor < q_c = \left\lfloor \frac{A}{B} \right\rfloor < \left\lfloor \frac{a+1}{b} \right\rfloor. \quad (3)$$

Функція цілої частини $[x]$ – неспадна, тому, з урахуванням (1.2), виконується

$$\left\lfloor \frac{a}{b+1} \right\rfloor \le q = \left\lfloor \frac{a}{b} \right\rfloor \le \left\lfloor \frac{a+1}{b} \right\rfloor, \quad (4)$$

тобто інтервали довіри для q_c і q співпадають, причому для q – інтервал замкнутий.

Легко показати, що $q_c \le q$ [20]. Справді, $q = \left\lfloor \frac{a}{b} \right\rfloor > \frac{a}{b} - 1$, а звідси для цілочисельних нерівностей $qb \ge a - b + 1$. Тоді $A - qB \le A - a + b - 1 = b + (m_a - 1) < b \le B$, тобто виконується нерівність, аналогічна до (1). Припустимо супротивне: $q_c > q$ або $q_c \ge q + 1$. Тоді $A - q_c B \le A - (q+1)B = A - qB - B < B - B < 0$, що суперечить означенню (1). Отже, $q = \lfloor a / b \rfloor$ є верхньою оцінкою для q_c замість $\lfloor (a+1)/b \rfloor$ в формулі (4).

Якщо виконується умова $\beta^n / 2 \le B < \beta^n$, то для "2:1"–алгоритму справедлива оцінка [20]: $q_c \le q \le q_c + 2$, тобто циклічний процес підбору q_c можна (і ефективніше) замінити максимум двома порівняннями та відніманнями, щораз зменшуючи на 1 значення коефіцієнта-цифри q. Цю умову, або їй еквівалентну $5 \le b < 10$, задовольняють, не змінюючи результату, помноживши ділене A і дільник B на те саме мале число-цифру, як правило, степінь 2, що можна реалізувати на рівні процесора через швидкі операції побітового зсуву. Однак, у десятковій системі числення такий підхід вимагає множення довгих чисел A і B, а тому тут не використовується. Як наслідок, базовий алгоритм ділення класу "2:1" має циклічну структуру (Лістинги 5 і 6). Наприклад, [189/20]=[18/2]=9 – вірно, а [180/29]=6<9 – з різницею $q - q_c = 3$.

Як альтернативу, розглянемо алгоритм *скороченого* ділення класу "3:2". Оцінимо різницю між краями інтервалу (2), враховуючи, що $b \le a < 10b$, або $a + 1 \le 10b$:

$$\frac{a+1}{b} - \frac{a}{b+1} = \frac{a+b+1}{b(b+1)} \le \frac{11b}{b(b+1)} = \frac{11}{b+1} \le 1, \text{ якщо } b \ge 10. \quad (5)$$

Згідно із властивістю функції $\lfloor x \rfloor$, якщо зміщення між аргументами $c \le 1$, то

$$\lfloor x + c \rfloor - \lfloor x \rfloor \le 1. \quad (6)$$

Отже, якщо $b \ge 10$, то гарантовано матимемо:

$$q - q_c \le \left\lfloor \frac{a}{b} \right\rfloor - \left\lfloor \frac{a}{b+1} \right\rfloor \le \left\lfloor \frac{a+1}{b} \right\rfloor - \left\lfloor \frac{a}{b+1} \right\rfloor \le 1, \quad (7)$$

тобто $q = q_c$ або $q = q_c + 1$, у силу дискретності значень q. Зменшити цю різницю і гарантувати $q = q_c$ не вдасться, якщо навіть врахувати більшу розрядність чисел a і b. Контраргумент: нехай $a = 2b$, і якщо для "відкинутих" цифр виконується нерівність $m_a < 2m_b$, наприклад, $m_a = 0$, $m_b \ne 0$, то оцінка q буде на 1 перевищувати коректне значення q_c.

Фактично, ми обґрунтували лінійну структуру алгоритму порозрядного ділення з остачею класу "3:2", з єдиною перевіркою і, можливо, корекцією оціночного коефіцієнта

q в меншу сторону на 1, якщо $r \geq b$ (тоді також зменшуємо й остачу, $r := r - b$). Для цього використовуємо функцію ділення націло $\mathrm{Int}(a / b)$ три- або двоцифрового числа a на двоцифрове число b (тобто $10 \leq b < 100$, $10 < a < 1000$, $a > b$) або наперед сформовану аналогічну індексну таблицю результатів, що потребує додатково пам'яті, але має збільшити, ймовірно, швидкодію. Залишок переписує відповідні розряди діленого, внаслідок чого появляться нулі в старших розрядах.

Зауважимо, що дана реалізація показала найвищу швидкодію в порівнянні з циклічними алгоритмами класу "2:1", зокрема, як з використанням функції $\mathrm{Int}(a / b)$, так і Табл. 1 (див. Додаток до електронної версії). Також ми порівнювали з комбінованим підходом, коли скорочене ділення класу "2:1" апріорі задовольняє оцінку (4). Для цього використовувалася Табл. 2 (там же), причому тільки 11,5% від усіх входів цієї таблиці не задовольняють умову (4), Табл. 3 (там же). Мабуть, деякий виграш часу, що дає успішне застосування алгоритму класу "2:1", коли умова (4) виконана, нівелюється необхідністю повторно переключатися на алгоритм класу "3:2" в разі невдачі. Наш числовий експеримент полягав у послідовному переборі діленого та дільника з деякого заданого діапазону чисел (наприклад, від 1 до 10000 – для діленого, та від 1 до 1000 – для дільника), а сумарний час виконання такого подвійного циклу порівнювався для різних реалізацій Do_Div1 (Лістинг 6). Для алгоритму ділення класу "2:1" ми також спробували реалізувати індексний доступ до наперед обчисленої Табл. 1, однак в цьому тесті виграшу в часі, порівняно із застосуванням функції $\mathrm{Int}(a / b)$ для округлення до меншого цілого, в рамках VBA-реалізації не отримали.

Далі для реалізації раціональних чисел нам потрібен алгоритм Евкліда [18,19,23] для знаходження найбільшого спільного дільника (НСД) двох чисел (Лістинг 7). Він широко використовуватиметься, якщо потрібно мати нескорочуваний дріб. Цей алгоритм теж не найефективніший, а отже, може бути замінений іншими, наприклад, двійковим алгоритмом Евкліда [24].

Запропоновані ідеї та програмна реалізація орієнтовані на практичне використання електронних таблиць у наукових дослідженнях, але не обмежуються ними. В Excel можна зручно використовувати розроблені програми на VBA для свого кола задач (навчальних, практичних, дослідницьких) як функції користувача або макроси, скопіювавши єдиний ".xla*"-файл (http://old.geology.lnu.edu.ua/phis_geo/nauka/metod/metod.htm). У перспективі ця надбудова може бути перепроектована і ефективніше реалізована з використанням динамічних бібліотек (DLL), написаних на інших мовах програмування.

Список використаної літератури

1. Use Excel as your calculator [Electronic resource]. – Mode of access: https://support.office.com/en-us/article/use-Excel-as-your-calculator-a1abc057-ed11-443a-a635-68216555ad0a.
2. Excel VBA Tutorial for Beginners: Learn in 3 Days [Electronic resource]. – Mode of access: https://www.guru99.com/vba-tutorial.html.
3. Calculation of very large numbers in Excel – Part 5 – Add-In [Electronic resource]. – Mode of access: http://www.Excel-ticker.com/calculation-of-very-large-numbers-in-Excel-part-5-add-in/.

4. Daily Dose of Excel – Large Number Arithmetic [Electronic resource]. – Mode of access: http://dailydoseofExcel.com/archives/2009/01/12/large-number-arithmetic/
5. Full Precision Calculator [Electronic resource]. – Mode of access: https://www.mathsisfun.com/calculator-precision.html.
6. EtherCalc is a web spreadsheet [Electronic resource]. – Mode of access: https://ethercalc.net.
7. Binary-coded decimal [Electronic resource]. – Mode of access: https://en.wikipedia.org/wiki/Binary-coded_decimal.
8. Mapple and Excel [Electronic resource]. – Mode of access: https://www.maplesoft.com/support/help/maple/view.aspx?path=Excel.
9. Bringing the Powep of Mathematica to Excel [Electronic resource]. – Mode of access: https://www.wolfram.com/products/applications/Excel_link/.
10. Using MATLAB with Excel [Electronic resource]. – Mode of access: https://www.mathworks.com/discovery/matlab-Excel.html.
11. PARI/GP [Electronic resource]. – Mode of access: https://en.wikipedia.org/wiki/PARI/GP.
12. SageMath [Electronic resource]. – Mode of access: https://en.wikipedia.org/wiki/SageMath.
13. List of arbitrary-precision arithmetic software [Electronic resource]. – Mode of access: https://en.wikipedia.org/w/index.php?title=List_of_arbitrary-precision_arithmetic_software&oldid=870548337.
14. Boost C++ Libraries [Electronic resource]. – Mode of access: http://www.boost.org.
15. The GNU Multiple Precision Arithmetic Library [Electronic resource]. – Mode of access: https://gmplib.org.
16. SciLib [Electronic resource]. – Mode of access: https://www.scilab.org/
17. BigInteger Struct [Electronic resource]. – Mode of access: https://docs.microsoft.com/ru-ru/dotnet/api/system.numerics.biginteger?view=netframework-4.7.2.
18. *Cormen T.H.* Introduction to Algorithms, 3rd Ed. / T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. – MIT Press, 2009. – 1312 p.
19. *Knuth D.E.* The Art of Computer Programming. 2. Seminumerical Algorithms, 3rd Ed. – Addison-Wesley Professional. – 1998. – 762 p.
20. *Hasselström K.* Fast Division of Large Integers. A Comparison of Algorithms [Electronic resource]. – 2003. – Mode of access: http://bioinfo.ict.ac.cn/~dbu/AlgorithmCourses/Lectures/Lec5-Fast-Division-Hasselstrom2003.pdf.
21. *Gantumur T.* Basic computer arithmetic [Electronic resource].– 2018. – Mode of access: www.math.mcgill.ca/gantumur/math387w18/float.pdf.
22. *Harris D.L.* SRT Division: Architectures, Models, and Implementations / D.L. Harris, S.F. Oberman, M.A. Horowitz. – Stanford, 1998. [Electronic resource].– Mode of access: https://pdfs.semanticscholar.org/b98b/430ae9d1ef83cb09f3d52e9ecf71d44b44a7.pdf.
23. Euclidean algorithm [Electronic resource]. – Mode of access: https://en.wikipedia.org/wiki/Euclidean_algorithm.
24. Binary GCD algorithm [Electronic resource]. – Mode of access: https://en.wikipedia.org/wiki/Binary_GCD_algorithm.

LONG ARITHMETIC IN EXCEL. I. VBA-IMPLEMENTATION

V. Fourman, M. Khomyak, Ya. Marko

Ivan Franko National University of Lviv,
Hrushevskogo Str. 4, UA–79005 Lviv, Ukraine,
fourman@i.ua

Approaches that increase the accuracy of computations occupy a special place in computational methods. The use of integer arithmetic of arbitrary precision requires a software implementation of operations based on 32- or 64-bit types supported by modern processors. The software and Internet resources for full precision computations representing the commercial sector and free-code programs are analyzed.

We propose to use wide-popular Excel spreadsheets as a convenient and sufficiently powerful symbolic calculator as an alternative to well-known powerful but high-priced packages. The basic operations are implemented in VBA: comparison, arithmetic actions and finding of the greatest common divisor for two integers with arbitrary digits saved in cells as text. At internal level an integer is represented by a BCD-array of digits with variable dimension, and its sign is stored separately. So we have two levels of implementation: through text strings and their internal representation. Operations "add", "subtract", "multiply" correspond to schoolbook algorithms but "divide" is more complex. The algorithm of short dividing for decimal numbers is justified which do not require the subtraction loop to correction the result digit. Numerical experiments have confirmed the effectiveness of the algorithm named as "3:2"-class: in this case, we take into account 3 digits of the dividend and 2 digits of the divisor to obtain one digit of the result.

These algorithms are necessary to work with rational numbers and for more complex calculations on their basis (for example, to solve systems of linear algebraic equations). In future, this version may be a prototype for more efficient redesign, especially with use of binary algorithms implemented in modern processors.

Key words: electronic spreadsheet, Excel Add-In, symbolic computations, dividing algorithms.

ДОДАТОК 1

Лістинг 1. Перетворення в BCD-формат і обернене – до стрічки символів.

```
Public Function ToDigits(s As String, ByRef lens As Integer, _
                         ByRef sign As Integer) As Byte()
  Dim i As Integer, j As Integer, ch As String, arr() As Byte
  j = Len(s): lens = j – 1: sign = IIf(Mid(s, 1, 1) = "-", -1, 1)
  If sign = -1 Then lens = lens - 1
  ReDim arr(0 To lens)
  For i = 0 To lens
    ch = Mid(s, j, 1):   arr(i) = Val(ch):   j = j - 1
  Next
  Do While lens >= 0   'Cut leading zeros
    If arr(lens) <> 0 Then Exit Do
    lens = lens - 1
  Loop
  If lens < 0 Then sign = 0: ToDigits = arr()
End Function
```

```
Public Function Digits2Str(sign As Integer, arr() As Byte, _
                           n As Integer) As String
   If sign = 0 Or n < 0 Then
      Digits2Str = "0": Exit Function
   End If
   Dim i As Integer, s As String     ' s = ""
   If sign = -1 Then s = "-"
   For i = n To 0 Step -1
      s = s + LTrim(Str(arr(i)))
   Next
   Digits2Str = s
End Function

Public Sub Do_CutZeros(arr() As Byte, ByRef n As Integer)
' Assume: n >=0.
Do While n >= 0
   If arr(n) <> 0 Then Exit Do
   n = n - 1 ' can be -1!
Loop
End Sub
```

Лістинг 2. Символьне порівняння: функція SCompare та її реалізація Do_Compare.

```
Public Function SCompare(s1 As String, s2 As String) As Integer
' Symbolically compare two signed integer values in the string format.
' Assume: strings s1 and s2 contain digits only, may be with leading "-" or "0" .
Dim v1() As Byte, n1 As Integer, v2() As Byte, n2 As Integer
Dim sign1 As Integer, sign2 As Integer, signRes As Integer  ' signRes = 0
   v1() = ToDigits(s1, n1, sign1)
   v2() = ToDigits(s2, n2, sign2)
If sign1 = sign2 Then
   If sign1 <> 0 Then
      signRes = Do_Compare(v1(), n1, v2(), n2)
      If sign1 < 0 Then signRes = -signRes
   End If
Else    ' sign1 <> sign2
   signRes = IIf(sign1 > sign2, 1, -1)
End If
SCompare = signRes
End Function

Public Function Do_Compare(v1() As Byte, n1 As Integer, _
         v2() As Byte, n2 As Integer, Optional iBeg As Integer=0) As Integer
' Symbolically compare two unsigned integers: v1(iBeg..n1) and v2(0..n2).
Dim iRes As Integer, i As Integer, j As Integer
   n1 = n1 – iBeg    ' re-define n1
   iRes = IIf(n1 > n2, 1, -1)   ' init. for n1 <> n2
   If n1 = n2 Then
      j = iBeg :    ' iRes = 0    'as default
```

```
      For i = n2 To 0 Step -1  ' compare from left to right (from high to low digits)
         If v1(j) <> v2(i) Then
            iRes = IIf(v1(j) > v2(i), 1, -1): Exit For
         End If
         j = j + 1
      Next
   End If
Do_Compare = iRes
End Function
```

Лістинг 3. Символьні арифметичні операції додавання (SAdd) та віднімання (SSubtract), а також їхні реалізації.

```
' Symbolically add or subtract two signed integer values in the string format.
' Assume: strings s1 and s2 contain digits only, may be with leading "0" or "-".
' Absolute values are saved in digit indices arrays: v1(0..n1), v2(0..n2).
' Order of digits: [0]- lowest digit,..., [n1] - highest digit.
Public Function SAdd(s1 As String, s2 As String) As String
Dim v1() As Byte, n1 As Integer, v2() As Byte, n2 As Integer, _
v() As Byte, n As Integer
Dim sign1 As Integer, sign2 As Integer, signRes As Integer
Dim bSwap As Boolean
   v1() = ToDigits(s1, n1, sign1): v2() = ToDigits(s2, n2, sign2)
If sign1 = 0 Or sign2 = 0 Then  ' simple cases
   v() = v1(): signRes = sign1: n = n1
   If sign1 = 0 Then
      signRes = sign2: v() = v2(): n = n2
   End If
Else
   If sign1 = sign2 Then   ' sign1 <> 0 and  sign2 <> 0 !!!
      bSwap = (n1 < n2):  signRes = sign1   ' 1 or -1
   Else ' sign1 <> sign2
      signRes = Do_Compare(v1(), n1, v2(), n2)
      If signRes = 0 Then
         SAdd = "0":  Exit Function
      End If
      bSwap = (signRes = -1)
      signRes = IIf(bSwap, sign2, sign1) ' swap to v1() > v2()
   End If
   CopyAndAddOrSub v1(), n1, v2(), n2, v(), n, (sign1 = sign2), bSwap
End If
  SAdd = Digits2Str(signRes, v(), n)
End Function

Public Function SSubtract(s1 As String, s2 As String) As String
  SSubtract = SAdd(s1, IIf(Mid(s2, 1, 1) = "-", Mid(s2, 2) , "-" + s2))
End Function

Public Sub CopyAndAddOrSub(v1() As Byte, n1 As Integer, _
         v2() As Byte, n2 As Integer, v() As Byte, ByRef n As Integer, _
```

```
            bAdd As Boolean, bSwap As Boolean)
' Implementation of a := a + b (bAdd=True) or a := a - b (bAdd=False).
' Assume: n1 >=n2 !
Dim v1_() As Byte, n1_ As Integer, v2_() As Byte, n2_ As Integer    ' temp
   If bSwap Then
      v1_() = v2(): n1_ = n2: v2_() = v1(): n2_ = n1
   Else
      v1_() = v1(): n1_ = n1: v2_() = v2(): n2_ = n2
   End If    ' |v1_|>|v2_| Now!
   n = n1_ : ReDim v(0 To n + 1)  ' with one reserved byte for op."add"
   CopyBytes v1_(), 0, n1_, v()    ' v1_(0..n1_) -> v(0..n1_)
   If bAdd Then
      Do_Add v(), n, v2_(), n2_    ' incremental add: "+=", correct n1_ if need
   Else
      Do_Sub v(), n, v2_(), n2_   ' decremental subtract: "-="
   End If
End Sub

Public Sub Do_Add(v1() As Byte, ByRef n1 As Integer,  _
                        v2() As Byte, n2 As Integer)
' v1(0..n1) += v2(0..n2).
' Assume:  n1>n2, UBound(v1) > n1.
Dim i As Integer, b As Byte, bOver As Byte
          ' bOver = 0   ' as default
   For i = 0 To n1 ' from right to left digits
      If i > n2 Then Exit For
      b = v1(i) + v2(i) + bOver: bOver = 0
      If b >= 10 Then
         b = b - 10: bOver = 1
      End If
      v1(i) = b
   Next
   If bOver > 0 Then                  ' bOver = 1 or 0.
      v1(i) = v1(i) + bOver           ' i = n1+1.
      n1 = n1 + 1            ' correct n1 if need.
   End If
End Sub

Public Sub Do_Sub(v1() As Byte, ByRef n1 As Integer, _
                     v2() As Byte, n2 As Integer)
' v1(0..n1) -= v2(0..n2). Assume: v1 >= v2 > 0, n1 >= n2.
Dim i As Integer, b1 As Byte, b2  As Byte, bOver As Boolean
   ' bOver = False   ' as default
   For i = 0 To n1  ' from right to left digits
      b1 = v1(i)
      If bOver Then
         bOver = (b1 = 0): b1 = IIf(bOver, 9, b1 – 1)
      End If
      If i > n2 Then Exit For
```

```
        b2 = v2(i)
        If b1 < b2 Then
           b1 = b1 + 10: bOver = True
        End If
        b1 = b1 - b2
      End If
      v1(i) = b1
   Next
   CutZeros v1(), n1   ' correct n1 if need
End Sub
```

Лістинг 4. Символьне множення – функція SMult та її реалізація.

```
Public Function SMult(s1 As String, s2 As String, _
          Optional shift As Integer = 0) As String
' Symbolically multiplicate two signed  integer values in the string format. shift >=0.
' Assume: strings s1 and s2 contain digits only, may be with leading "0" or "-".
Dim sRes As String
Dim v1() As Byte, n1 As Integer, v2() As Byte, n2 As Integer, _
v() As Byte, n As Integer
Dim sign1 As Integer, sign2 As Integer, signRes As Integer
If s1 = "0" Or s2 = "0" Then
  sRes = "0"
ElseIf s1 = "1" Then
      sRes = s2
ElseIf s2 = "1" Then
      sRes = s1
Else      ' general case:
   v1() = ToDigits(s1, n1, sign1)
   v2() = ToDigits(s2, n2, sign2)
   Do_Mult v1(), n1, v2(), n2, v(), n, shift
   signRes = IIf(sign1 = sign2, 1, -1): sRes = Digits2Str(signRes, v(), n)
End If
 SMult = sRes
End Function

Public Sub Do_Mult(v1() As Byte, n1 As Integer, v2() As Byte, n2 As Integer, _
   dRes() As Byte, ByRef n As Integer, Optional shift2 As Integer = 0)
' Multiply two values saved in digits arrays v1(0..n1) and v2(0..n2).
   n = n1 + n2 + shift2 + 1
   ReDim dRes(0 To n) ' init zeros
Dim i As Integer
For i = 0 To n2
   If v2(i) > 0 Then Do_Mult1 v1(), n1, v2(i), dRes(), n, 0, shift2 + i
Next
   Do_CutZeros dRes(), n
End Sub

Public Sub Do_Mult1(v1() As Byte, n1 As Integer, _
   v2 As Byte, digitsRes() As Byte, ByRef n As Integer, _
```

```
    Optional n0 As Integer = 0, Optional j As Integer = 0)
' Symbolically multiply v1(0..n1) and a digit v2.
' The result is saved in the array digitsRes(j..n) like as in the operator "*=".
Dim i As Integer, j As Integer, b1 As Byte, b As Byte, bOver As Byte
    For i = n0 To n1 ' from right to left (from low to high digits)
        b = digitsRes(j) + v1(i) * v2 + bOver
        b1 = b Mod 10   ' 0..9
        bOver = (b - b1) / 10 : digitsRes(j) = b1: j = j + 1
    Next
    If bOver > 0 Then digitsRes(j) = digitsRes(j) + bOver
End Sub
```

Лістинг 5. Символьне ділення десяткових чисел у стрічковому форматі.

```
Public Function SDivRem(s1 As String, s2 As String, sReminder As String) _
        As String
' Symbolically divide two signed integer values in the string format.
' Return integer part of (s1/s2) and the remainder by ref.
' Assume: strings s1 and s2 contain digits only, may be with leading "0" or "-".
Dim sRes As String, v1() As Byte, v2() As Byte, v() As Byte, reminder() As Byte
Dim n1 As Integer, n2 As Integer, n3 As Integer, n4 As Integer
Dim sign1 As Integer, sign2 As Integer, signRes As Integer, sg As Integer
v1() = ToDigits(s1, n1, sign1): v2() = ToDigits(s2, n2, sign2)
If sign2 = 0 Then  err.Raise vbObjectError + 100, "SDivRem", "Divide by 0"
  signRes = IIf(sign1 = sign2, 1, -1): sRes = "0": sReminder = "0" ' as default
If sign1 <> 0 Then
    sg = Do_Compare(v1(), n1, v2(), n2)       ' compare as 2 abs. values
    If sg < 1 Then  '=0 or -1
        If sg = 0 Then        ' s1 = s2:
          sRes = IIf(signRes = -1, "-1","1")     ' sReminder = "0"
        Else          ' v1 < v2
          sReminder = s1                         ' sRes = "0"
        End If
    Else              ' |v1| > |v2|
        Do_DivRem v1(), n1, v2(), n2, v(), n3, reminder(), n4
        sRes = Digits2Str(signRes, v(), n3)
        If n4 >= 0 Then sReminder = Digits2Str(sign1, reminder(), n4)
                  ' the reminder is "0" or has a same sign as dividend
    End If
End If
SDivRem = sRes  ' s1 = s2*sRes + sReminder, sReminder < s2
End Function

Public Sub Do_DivRem(v1() As Byte, n1 As Integer, v2() As Byte, _
          n2 As Integer, vFraction() As Byte, ByRef n3 As Integer, _
          vReminder() As Byte, ByRef n4 As Integer)
' symbolically divide two values saved in digit indices arrays v1(0..n1) and v2(0..n2).
' Assume: n1 >= n2, v1 >= v2. Return [v1 / v2]  >0  and vReminder() by ref.
' If n4 = -1 then reminder = 0 but the array vReminder() is UNDEFINED .
Dim d() As Byte, n0 As Integer, n As Integer, sg As Integer, btFr As Byte
```

```
   n3 = n1 - n2: n4 = n2: n = n1    ' n - last significant digit in v1()
   n0 = n3          ' = n1 - n2
ReDim vFraction(0 To n3) ' init zeros
ReDim d(0 To n1):  CopyBytes v1(), 0, n1, d() ' d – worked array, init as copy of v1()
Do
   sg = Do_Compare(d(), n, v2(), n2, n0)   ' compare d(n0..n1) and v2(0..n2)
   If sg > 0 Then
         vFraction(n0) = Do_Div1_32(d(), n, v2(), n2, n0)  ' "3:2" approach or...
         ' vFraction(n0) = Do_Div1_21(d(), n, v2(), n2, n0)  ' "2:1" approach
   ElseIf sg = 0 Then
      n = n - n2 – 1:  vFraction(n0) = 1       ' d(n0..n) = 0, without subtraction!
   Else    ' set 0 if d() < v2()
    '   vFraction(n0) = 0   ' as default
      If n >= 0 Then If d(n) = 0 Then n = n - 1 ' cut a leading zero if need
   End If
   n0 = n0 - 1
Loop While n0 >= 0
Do_CutZeros vFraction(), n3
n4 = n: Do_CutZeros d(), n4    ' correct rest of division if need. n4>=0 or -1
If n4 >= 0 Then
   ReDim vReminder(0 To n4): CopyBytes d(), 0, n4, vReminder()
End If
End Sub
```

Лістинг 6. Процедури реалізації скороченого ділення з використанням алгоритмів класу "3:2" або "2:1".

```
Function Do_Div1_32(v1() As Byte, ByRef n1 As Integer, v2() As Byte, _
          n2 As Integer, n0 As Integer) As Byte
' Divide A=v1(n0..n1) by B=v2(0..n2); A > B, n1-n0+1 = n2 or n1-n0 = n2.
' "3:2"-approach, upper estimation. Return a digit 1..9 and reminder in v1(n0..n1).
Dim sg As Integer, a As Integer, b As Integer, d As Integer, btRes As Byte
   'd = 0   ' as default
   If n2 = 0 Then                    ' "short" case "2:1"
      b = v2(n2): a = v1(n1)         ' 1 digit
      If n1 > 0 Then d = 10 * a + v1(n1 - 1)
   Else    ' n2 >= 1. Assume A >= B, n1 >= n2 >= 1:
      b = 10 * v2(n2) + v2(n2 - 1):
      a = 10 * v1(n1) + v1(n1 - 1)  ' 2 digits
      If n1 > 1 And (a < b Or n1 - n2 > n0) Then _
                  d = 10 * a + v1(n1 - 2)     ' use 2 or 3 digits for <A>
   End If
   If d > 0 And d < 10 * (b + 1) Then a = d          ' re-init for both "3:2" and "2:1" cases
   btRes = Int(a / b): If btRes > 9 Then btRes = 9  '1..9, upper estimation now!
   If btRes > 1 Then
         Dim nT As Integer, T() As Byte   ' worked array, for product of bFr*v2
         nT = n2 + 1:  ReDim T(0 To nT)   ' init with 0 and reserve one (?) byte
      Do_Mult1 v2(), n2, btRes, T(), nT       ' in fact, Do_Mult1 work as "+="
      If T(nT) = 0 Then nT = nT – 1                  ' cut leading zero if need
      'Do     ' without LOOP:
```

```
        sg = Do_Compare(v1(), n1, T(), nT, n0)
        If sg < 0 Then
          btRes = btRes - 1
          If btRes > 1 Then Do_Subtract T(), nT, v2(), n2 ' ...and correct nT if need
        End If
      'Loop While sg < 0
    End If
    If btRes = 1 Then       ' assume: v1() > v2() and return reminder in v1():
      Do_Subtract v1(), n1, v2(), n2, n0
    ElseIf sg = 0 Then      ' subtract two equival numbers, v1(n0..n1) = T(0..nT)
      v1(n0) = 0: n1 = n0 - 1         ' can be n1 = -1
    Else
      Do_Subtract v1(), n1, T(), nT, n0
    End If
Do_Div1_32 = btRes        ' 1..9
End Function

Public Function Do_Div1_21(v1() As Byte, ByRef n1 As Integer, _
        v2() As Byte, n2 As Integer, n0 As Integer)
'  See comments for  Do_Div1_32,  but we use  "2:1"- approach here.
Dim sg As Integer, btRes As Byte, a As Byte, d As Byte ' a,d <100.
    If v1(n1) > v2(n2) Or n1 = 0 Then
      a = v1(n1)                                          ' 1 digits
    Else    'If v1(n1) <= v2(n2) And n1 > 0 Then
      a = 10 * v1(n1) + v1(n1 - 1)                              ' 2 digits
      If a >= 10 * (v2(n2) + 1) Then a = v1(n1)      ' 1 digits
    End If
    btRes = Int(a / v2(n2)): If btRes > 9 Then btRes = 9      '1..9, upper estimation!
    If btRes > 1 Then
        Dim T() As Byte, nT As Integer ' T() - worked arr., for product of bFr*v2
        nT = n2 + 1: ReDim T(0 To nT)   ' init with 0 (!), with one reserved (?) byte
        Do_Mult1 v2(), n2, btRes, T(), nT     ' Do_Mult1 work as "+="
        If T(nT) = 0 Then nT = nT - 1         ' cut leading zeros if need
        Do
            sg = Do_Compare(v1(), n1, T(), nT, n0)
            If sg < 0 Then
                    btRes = btRes - 1
                    If btRes = 1 Then Exit Do
                    Do_Subtract T(), nT, v2(), n2  ' correct nT if need
            End If
         Loop While sg < 0
    End If
    If btRes = 1 Then   ' assume: v1() > v2() and iDiffUL = 0 (btRes>=1)!
      Do_Subtract v1(), n1, v2(), n2, n0  ' reminder in v1()
    ElseIf sg = 0 Then  ' Subtract two equival numbers v1(n0..n1) and T(0..nT)
      v1(n0) = 0: n1 = n0 - 1  ' n1 can be -1
    Else
      Do_Subtract v1(), n1, T(), nT, n0  ' reminder in v1()
    End If
```

```
Do_Div1_21 = btRes        ' 1..9
End Function
```

Лістинг 7. Алгоритм Евкліда пошуку найбільшого спільного дільника (НСД).

```
Public Sub Do_GCD(vA() As Byte, nA As Integer, vB() As Byte, nB As Integer, _
   vGCD() As Byte, ByRef nGCD As Integer)
'Return Greatest Common Divider of two POSITIVE numbers
' in the digits array vGCD(0.. nGCD). vA and vB not chanched.
Dim v1() As Byte, n1 As Integer, v2() As Byte, n2 As Integer
Dim v3() As Byte, n3 As Integer, v() As Byte, n As Integer
Dim sg As Long
sg = Do_Compare(vA(), nA, vB(), nB)
If sg = 0 Then   ' if vA=vB
   nGCD = nA: ReDim vGCD(0 To nGCD)
   CopyBytes vA(), 0, nGCD, vGCD()   ' same as vA
   Exit Sub
End If
n1 = nA: ReDim v1(0 To n1): CopyBytes vA(), 0, n1, v1()
n2 = nB: ReDim v2(0 To n2): CopyBytes vB(), 0, n2, v2()
If sg = -1 Then ' swap v1 <-> v2
   v() = v1(): v1() = v2(): v2() = v() ' temp: v(), n
   n = n1: n1 = n2: n2 = n
End If
Do  '  assume: v1 > v2
   Do_DivRem v1(), n1, v2(), n2, v3(), n3, v(), n  ' v3(), n3 not used, in fact.
   ' assume: 0 <= v() < v2() now.
   If n >= 0 Then ' if reminder <> 0
      v1() = v2(): n1 = n2
      v2() = v(): n2 = n
   End If
Loop While n >= 0
   nGCD = n2: ReDim vGCD(0 To nGCD)  ' assume: vGCD() >= 1
   CopyBytes v2(), 0, nGCD, vGCD()
End Sub
```

ISSN 2224-087X. Електроніка та інформаційні технології. 2019. Випуск 11

Додаток 2

Таблиця 1.

Верхня оцінка для скороченого ділення $[a/b]$, $1 \le b < 10$, $1 < a < 100$, $a \ge b$ (a – затінені стовпці, b – верхній рядок). Стовпці, що не використовуються, опущено.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**		**4**	**5**	**6**	**7**	**8**	**9**		**6**	**7**	**8**	**9**		**8**	**9**
1	1									**21**	9	7	5	4	3	3	2	2	**41**	9	8	6	5	5	4	**61**	9	8	7	6	**81**	9	9
2	2	1								**22**	9	7	5	4	3	3	2	2	**42**	9	8	7	6	5	4	**62**	9	8	7	6	**82**	9	9
3	3	1	1							**23**	9	7	5	4	3	3	2	2	**43**	9	8	7	6	5	4	**63**	9	9	7	7	**83**	9	9
4	4	2	1	1						**24**	9	8	6	4	4	3	3	2	**44**	9	8	7	6	5	4	**64**	9	9	8	7	**84**	9	9
5	5	2	1	1	1					**25**	9	8	6	5	4	3	3	2	**45**	9	9	7	6	5	5	**65**	9	9	8	7	**85**	9	9
6	6	3	2	1	1	1				**26**	9	8	6	5	4	3	3	2	**46**	9	9	7	6	5	5	**66**	9	9	8	7	**86**	9	9
7	7	3	2	1	1	1	1			**27**	9	9	6	5	4	3	3	3	**47**	9	9	7	6	5	5	**67**	9	9	8	7	**87**	9	9
8	8	4	2	2	1	1	1	1		**28**	9	9	7	5	4	4	3	3	**48**	9	9	8	6	6	5	**68**	9	9	8	7	**88**	9	9
9	9	4	3	2	1	1	1	1	1	**29**	9	9	7	5	4	4	3	3	**49**	9	9	8	7	6	5	**69**	9	9	8	7	**89**	9	9
10	9	5	3	2	2	1	1	1	1	**30**		9	7	6	5	4	3	3	**50**		9	8	7	6	5	**70**		9	8	7	**90**		9
11	9	5	3	2	2	1	1	1	1	**31**		9	7	6	5	4	3	3	**51**		9	8	7	6	5	**71**		9	8	7	**91**		9
12	9	6	4	3	2	2	1	1	1	**32**		9	8	6	5	4	4	3	**52**		9	8	7	6	5	**72**		9	9	8	**92**		9
13	9	6	4	3	2	2	1	1	1	**33**		9	8	6	5	4	4	3	**53**		9	8	7	6	5	**73**		9	9	8	**93**		9
14	9	7	4	3	2	2	2	1	1	**34**		9	8	6	5	4	4	3	**54**		9	9	7	6	6	**74**		9	9	8	**94**		9
15	9	7	5	3	3	2	2	1	1	**35**		9	8	7	5	5	4	3	**55**		9	9	7	6	6	**75**		9	9	8	**95**		9
16	9	8	5	4	3	2	2	2	1	**36**		9	9	7	6	5	4	4	**56**		9	9	8	7	6	**76**		9	9	8	**96**		9
17	9	8	5	4	3	2	2	2	1	**37**		9	9	7	6	5	4	4	**57**		9	9	8	7	6	**77**		9	9	8	**97**		9
18	9	9	6	4	3	3	2	2	2	**38**		9	9	7	6	5	4	4	**58**		9	9	8	7	6	**78**		9	9	8	**98**		9
19	9	9	6	4	3	3	2	2	2	**39**		9	9	7	6	5	4	4	**59**		9	9	8	7	6	**79**		9	9	8	**99**		9
20		9	6	5	4	3	2	2	2	**40**			9	8	6	5	5	4	**60**			9	8	7	6	**80**			9	8			

Таблиця 2.

Різниця між верхньою та нижньою оцінкою для $[a/b]$ (позначення аналогічні, як у Табл. 1).

	1	2	3	4	5	6	7	8	9		2	3	4	5	6	7	8	9		4	5	6	7	8	9		6	7	8	9		8	9
1	0									**21**	2	2	1	1	0	1	0	0	**41**	1	2	1	0	1	0	**61**	1	1	1	0	**81**	0	1
2	1	0								**22**	2	2	1	1	0	1	0	0	**42**	1	1	1	1	1	0	**62**	1	1	1	0	**82**	0	1
3	2	0	0							**23**	2	2	1	1	0	1	0	0	**43**	1	1	1	1	1	0	**63**	0	2	0	1	**83**	0	1
4	2	1	0	0						**24**	1	2	2	0	1	0	1	0	**44**	1	1	1	1	1	0	**64**	0	1	1	1	**84**	0	1
5	3	1	0	0	0					**25**	1	2	1	1	1	0	1	0	**45**	0	2	1	1	0	1	**65**	0	1	1	1	**85**	0	1
6	3	1	1	0	0	0				**26**	1	2	1	1	1	0	1	0	**46**	0	2	1	1	0	1	**66**	0	1	1	1	**86**	0	1
7	4	1	1	0	0	0	0			**27**	0	3	1	1	1	0	0	1	**47**	0	2	1	1	0	1	**67**	0	1	1	1	**87**	0	1
8	4	2	0	1	0	0	0	0		**28**	0	2	2	1	0	1	0	1	**48**	0	1	2	0	1	1	**68**	0	1	1	1	**88**	0	1
9	5	1	1	1	0	0	0	0	0	**29**	0	2	2	1	0	1	0	1	**49**	0	1	1	1	1	1	**69**	0	1	1	1	**89**	0	1
10	4	2	1	0	1	0	0	0	0	**30**		2	1	1	1	1	0	0	**50**		1	1	1	1	0	**70**		1	1	0	**90**		0
11	4	2	1	0	1	0	0	0	0	**31**		2	1	1	1	1	0	0	**51**		1	1	1	1	0	**71**		1	1	0	**91**		0
12	3	2	1	1	0	1	0	0	0	**32**		1	2	1	1	0	1	0	**52**		1	1	1	1	0	**72**		0	1	1	**92**		0
13	3	2	1	1	0	1	0	0	0	**33**		1	2	1	1	0	1	0	**53**		1	1	1	1	0	**73**		0	1	1	**93**		0
14	2	3	1	1	0	0	1	0	0	**34**		1	2	1	1	0	1	0	**54**		0	2	1	0	1	**74**		0	1	1	**94**		0
15	2	2	2	0	1	0	1	0	0	**35**		1	1	2	0	1	1	0	**55**		0	2	1	0	1	**75**		0	1	1	**95**		0
16	1	3	1	1	1	0	0	1	0	**36**		0	2	1	1	1	0	1	**56**		0	1	1	1	1	**76**		0	1	1	**96**		0
17	1	3	1	1	1	0	0	1	0	**37**		0	2	1	1	1	0	1	**57**		0	1	1	1	1	**77**		0	1	1	**97**		0
18	0	3	2	1	0	1	0	0	1	**38**		0	2	1	1	1	0	1	**58**		0	1	1	1	1	**78**		0	1	1	**98**		0
19	0	3	2	1	0	1	0	0	1	**39**		0	2	1	1	1	0	1	**59**		0	1	1	1	1	**79**		0	1	1	**99**		0
20		3	1	1	1	1	0	0	0	**40**			1	2	1	0	1	0	**60**			1	1	1	0	**80**			1	0			

Таблиця 3.

Частотна таблиця якості передбачення результату ділення $q - q_c$ для алгоритму класу "2:1" (на основі Табл. 2).

$q - q_c$	Кількість	%
0	183	36,97%
1	250	50,51%
2	46	9,29%
3	11	2,22%
4	4	0,81%
5	1	0,20%
Всього	495	

Стаття: надійшла до редакції 25.02.2019,
доопрацьована 11.02.2019,
прийнята до друку 12.02.2019