УДК 004.942

А.Я. Аноприенко (д-р техн.наук, проф.),
В.А. Гранковский (студент), **Иваница** (аспирант)
Донецкий национальный технический университет
anoprien@cs.dgtu.donetsk.ua

ПРИМЕР РУМПА В КОНТЕКСТЕ ТРАДИЦИОННЫХ, ИНТЕРВАЛЬНЫХ И ПОСТБИНАРНЫХ ВЫЧИСЛЕНИЙ

На базе примера, предложенного профессором З. Румпом еще в 1988 году, проведен детальный анализ условий формирования неверных результатов в современных системах компьютерной математики. Проанализирована возможность использования интервальных методов для минимизации отрицательных последствий получения ошибочных результатов вычислений. Предложены варианты постбинарного представления численной информации и методы работы с ней, исключающие появление ситуаций, подобных тем, которые имеют место в случае с примером Румпа.

пример Румпа, интервальные вычисления, постбинарные вычисления.

Введение

Лавинообразное нарастание объемов вычислений в процессе исследования, моделирования и проектирования сложных систем и динамических процессов как никогда ранее актуализирует контроль достоверности и точности вычислительных процессов. Обоснованно предполагается, например, что многие техногенные катастрофы последних десятилетий, причину которых традиционно объяснили преимущественно «человеческим фактором», были в первую очередь обусловлены разного рода вычислительными ошибками [1]. В большинстве же случаев ошибки в вычислениях просто остаются незамеченными, существенно искажая полученные результаты.

Одним из простейших путей минимизации ошибок, связанных с округлением и потерей точности ввиду ограничений, обусловленных особенностью представления чисел в современных цифровых компьютерах, является дальнейший рост разрядности вычислений. В августе 2008 года в этом направлении был сделан важный шаг, связанный с публикацией стандарта IEEE 754-2008, который заменил ранее действовавший стандарт вычислений с плавающей запятой IEEE 754-1985. Стандарт 1985 года предусматривал 2 типа чисел с плавающей запятой: одинарной (32-разрядные) и двойной (64-разрядные) точности. Одной из первых его аппаратных реализаций стал арифметический сопроцессор Intel 8087, в котором дополнительно в качестве внутреннего формата использовался «расширенный» 80-разрядный формат с 64-разрядной мантиссой и 16-разрядным порядком. Основным нововведением в стандарте 2008 года стал 128-разрядный формат «квадро», т.е. формат «учетверенной точности»,

который по идее должен обеспечить соответствующие потребности по части точности вычислений еще на ближайшие 20 лет.

Но один довольно простой пример показывает, насколько эти надежды могут оказаться иллюзорными. Речь идет о так называемом примере Румпа, впервые опубликованном еще в 1988 году [2].

Пример Румпа

Профессор Зигфрид Румп, являясь в 1988 году сотрудником немецкого отделения фирмы ИБМ и исследуя алгоритмы вычислений с гарантированными границами интервалов, заведомо включающими правильный результат, получил полином, который при определенном сочетании значений переменных давал заведомо неправильный результат при всех стандартных значениях точности вычислений с плавающей запятой:

$$f = 333.75\, b^6 + a^2\,(11a^2\, b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + a/(2b), \qquad (1)$$

где a = 77617, b = 33096.

При различной разрядности вычислений с плавающей запятой на практически стандартной для того времени большой вычислительной системе ИБМ 370 для данного полинома были получены примерно одинаковые результаты:

32-bit: f = 1.172604;
64-bit: f = 1.1726039400531786;
128-bit: f = 1.172603940053178631858834904520183 8.

Но во всех случаях этот результат весьма существенно (даже своим знаком!) отличался от правильного (полученного с помощью специального алгоритма):

f = - 0.827396059946821368141165095479816…

В дальнейшем данный пример стал классической иллюстрацией тех проблем, которые присущи современным вычислениям с плавающей запятой. В частности, в 2001 году были опубликованы результаты исследования примера Румпа с применением различных современных вычислительных платформ и инструментов [3]. Вычисление полинома Румпа на компьютерах с процессорами Intel Pentium и Sun Sparc дало различный результат в зависимости от того, какая система компьютерной математики использовалась. При этом, например, использование системы Matlab давало классический результат 1.1726… практически независимо от разрядности. В системе Maple V:

f = -1.18059… 10^{21}.

В системе Mathematica v.4.0.2:

f = 1.1726… или -1.18059… 10^{21} в зависимости от различных условий. При этом делается вывод, что **для получения правильного результата необходимо минимум 122 двоичных разряда, т.е. 36 десятичных разрядов** [3]. Стандартное же представление чисел с плавающей запятой эквивалентно фактически **8-ми десятичным разрядам при одинарной точности, 17-ти – при двойной, 34-м – при учетверенной**.

В 2002 году этот пример исследовался сотрудниками фирмы Sun Microsystems, которые показали, что в некоторых случаях современные стандартные средства вычислений дают еще более непредсказуемые результаты. Например, соответствующие вычисления полинома Румпа на базе программы, полученной с помощью компилятора Fortran 95 фирмы Sun Microsystems Inc., дали следующие результаты [4]:

32-bit: f = −6. 338253E + 29,
64-bit: f = −1. 1805916207174113E + 21,
128-bit: f = +1. 172603940053178631858834904520 1838.

Позднее З. Румп показал [5], что аналогичные проблемы при использовании приведенных выше значений a и b возникают и в случае вычисления более простого полинома (в дальнейшем «полином 2») вида

$$f = 21\,b^2 - 2\,a^2 + 55\,b^4 - 10\,a^2b^2 + a/(2b). \qquad (2)$$

Особую обеспокоенность такого рода результаты вызывают в связи с широким распространением в последнее время вычислений на базе графических процессоров (GPU), которые для достижения максимальной производительности чаще всего оперируют с числами только одинарной точности, что резко повышает вероятность получения неверных результатов [6]. Это не столь критично при расчетах, связанных исключительно с визуализацией, но весьма опасно при использовании GPU для высокопроизводительных вычислений общего назначения.

При этом самым тревожным фактом является то, что при проверке примера Румпа практически во всех современных математических пакетах не только результат оказывается неверным, но и **отсутствуют какие-либо признаки того, что при вычислениях возникли проблемы**! А это означает, что такого рода незамечаемых ошибок в современных вычислениях и вычислительном моделировании может быть непредсказуемое множество.

Исследование примера Румпа в современных условиях

Все вышеизложенное побудило провести более детальное исследование примера Румпа с использованием последних версий современного вычислительного программного обеспечения типа электронных таблиц и систем компьютерной математики (СКМ) Wolfram

Mathematica, Mathsoft MathCAD и SciLab [7]. Как и следовало ожидать в стандартном режиме все СКМ вычислили полином неверно.

На рис 1, в частности, приведен пример получения традиционного неверного результата в довольно популярной свободно распространяемой СКМ SciLab. Аналогичный результат получен и СКМ MathCAD.

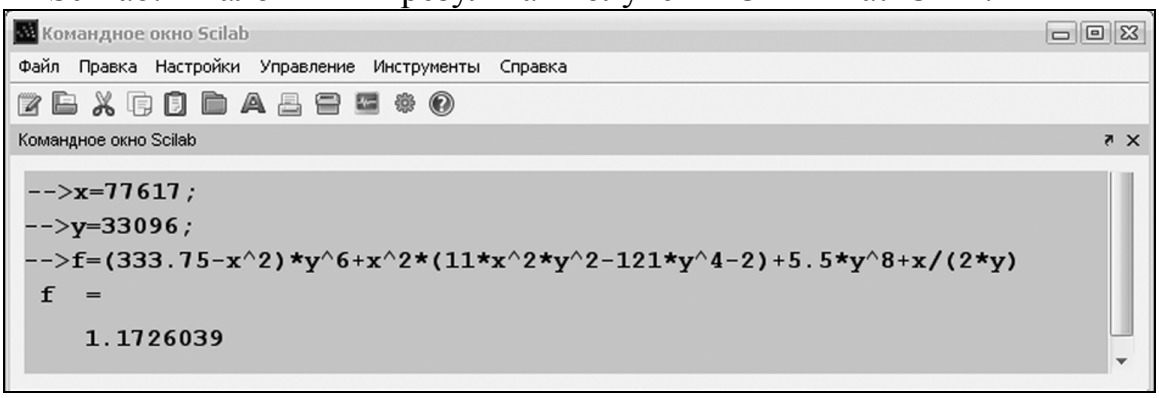

Рисунок 1 – Пример получения неверного результата в СКМ SciLab

На рис. 2 и 3 показаны результаты поэтапного вычисления полиномов 1 и 2 в MS Excel 2007. При этом не только не получен правильный результат, но наглядно продемонстрировано, что для полинома 1 разброс неверных значений существенно зависит от последовательности выполняемых вычислений, но при этом даже порядок результата весьма значительно отличается от правильного.

В современной версии СКМ Mathematica 7 попытка вычисления полинома стандартным способом также дает те же результаты, которые были получены для версии 4 практически 10 лет назад и представлены в работе [3].

Таким образом, проблема в общем случае остается нерешенной.

a	**b**	
77617	33096	
	Полином 1	
	$333.75\ b^6$	4,38605750846393000000000000000E+29
$a^2\ (11a^2\ b^2 - b^6\ - 121b^4 - 2)$		-7,91711177927471000000000000000E+36
	$5.5b^8$	7,91711134066896000000000000000E+36
	a/(2b)	1,17260394005318000000000000000E+00
	∑	**-1,18059162071741000000000000000E+21**
	Полином 2	
	$21\ b^2$	2,30022495360000000000000000000E+10
	$-\ 2\ a^2$	-1,20487973780000000000000000000E+10
	$+\ 55\ b^4$	6,59879628217748000000000000000E+19
	$-\ 10\ a^2b^2$	-6,59879628327282000000000000000E+19
	+ a/(2b)	1,17260394005318000000000000000E+00
	∑	**1,17260394005318000000000000000E+00**

Рис. 2 Результаты вычисления классического (1) и сокращенного (2) полиномов Румпа в MS Excel (для представления чисел в экспоненциальном формате установлено максимально возможное значение количества десятичных знаков, равное 30-ти)

a	b	
77617	33096	
	Полином 1	
	$333.75\ b^6$	438 605 750 846 393 000 000 000 000 000,000000000000000000
	$x = 11a^2 b^2$	72 586 759 116 001 000 000,000000000000000000
$y = - b^6 - 121b^4 - 2$		-1 314 174 679 544 730 000 000 000 000,000000000000000000
	$a^2 (x+y)$	-7 917 111 779 274 710 000 000 000 000 000 000,000000000000000000
	$5.5b^8$	7 917 111 340 668 960 000 000 000 000 000 000,000000000000000000
	a/(2b)	1,17260394005318000000
	∑	**-1 314 175 562 601 590 000 000 000 000,000000000000000000**
	Полином 2	
	$21\ b^2$	23 002 249 536,000000000000000000
	$-\ 2\ a^2$	-12 048 797 378,000000000000000000
	$+\ 55\ b^4$	65 987 962 821 774 800 000,000000000000000000
	$-\ 10\ a^2b^2$	-65 987 962 832 728 200 000,000000000000000000
	+ a/(2b)	1,17260394005318000000
	∑	**1,17260394005318000000**

Рис. 3 Результаты вычисления классического (1) и сокращенного (2) полиномов Румпа в MS Excel (для наглядности представления чисел выбран числовой формат с количеством десятичных знаков 20): изменение порядка операций существенно меняет значение результата при вычислении полинома 1.

Способы получения правильного результата для примера Румпа

Одним из способов получения правильного результата для примера Румпа является использование символьных вычислений. В данном контексте это означает, что в таком режиме СКМ при вычислении полинома представляет вещественные числа не в традиционном формате с плавающей точкой, а в виде рациональных дробей.

В СКМ MathCAD данное преобразование необходимо выполнять вручную, т. е. непосредственно при вводе формулы полинома с клавиатуры. В этом случае полином Румпа принимает следующий вид:

$$f := \frac{33375}{100} y^6 + x^2 \left(11x^2 y^2 - y^6 - 121y^4 - 2\right) + \frac{55}{10} y^8 + \frac{x}{2y}.$$

Для получения правильного результата, вычисления необходимо производить, используя символьный процессор MathCAD [8, с.139-143], обеспечивающий необходимую точность вычислений. Полученный результат в виде обыкновенной дроби, при ее численном вычислении дает, наконец, искомое правильное значение

$$f \to \frac{-54767}{66192} = -0.827396059946821,$$

В СКМ Mathematica можно осуществить ввод рациональных дробей не только ручным способом, но и воспользоваться функцией Rationalize[], которая возвращает аргумент, преобразованный в рациональную дробь. В результате вычисления примера Румпа со всеми числами с плавающей точкой, преобразованными в обыкновенные дроби, СКМ Mathematica также позволяет получить правильное значение в виде дроби:

$$F = -\frac{54767}{66192}.$$

Однако символьные вычисления не всегда позволяют решить вычислительную проблему типа полинома Румпа. Например, если бы в выражение Румпа входила функция, возвращающая иррациональный результат (синус, логарифм, радикал и др.), то ее преобразование к дробному виду, гарантированно привело бы к погрешности, и такая нефиксируемая потеря точности стала бы причиной получения неверного результата.

Еще одним способом получения правильного результата являются использование при вычислениях разрядности, существенно превышающей стандартную. Для примера Румпа это составляет **минимум 122 двоичных разряда, т.е. 36 десятичных разрядов**.

Высокой эффективности вычислений при этом можно добиться, используя встроенные средства относительно низкоуровневого языка программирования. Традиционно для этих целей используется либо программирование на уровне ассемблера, отличающееся чрезвычайной трудоемкостью и жесткой зависимостью от конкретной аппаратной платформы, либо принимается более компромиссное решение, заключающееся в использовании специальных библиотек для такого широко распространенного языка системного программирования как C++ [9]. Для платформы UNIX/Linux наиболее популярной является библиотека GMP (GNU Multiple-Precision Library) [10]. Для платформы MS Windows наиболее типичным примером является свободно распространяемый класс cBigNumber, который реализует целые числа неограниченной разрядности для C++ [11]. В нем предусмотрены все стандартные операции языка C++, включая арифметические, логические и битовые операции, а также операции сравнения, сдвиги и др. Класс оптимизирован для работы с числами от 500 до 20 000 двоичных разрядов. Испытания проведены для чисел, содержащих до 12 000 000 двоичных разрядов [11].

В отличие от C++ в большинстве более современных языков программирования для обеспечения вычислений повышенной точности стандартно поддерживаются целые числа произвольной длины. Например, в C# это обеспечивается использованием типа System.Numerics.BigInteger, включенного в базовую библиотеку классов начиная с версии 4.0. В Java это обеспечивается с помощью классов BigInteger и BigDecimal. Еще одним типичным примером такого рода является Python (Питон) — объектно-ориентированный язык сверхвысокого уровня. Целое в Питоне соответствует типу long. Длинное целое — это целое бесконечной длины. С помощью таких

чисел можно производить вычисления неограниченной разрядности, этот тип эмулируется библиотекой, встроенной в интерпретатор.

Однако, следует отметить, что использование всех перечисленных программных средств оправдано лишь тогда, когда заведомо требуется оперировать с числами повышенной разрядности. В случае примера Румпа эта необходимость не столь очевидна И, более того, целесообразность более всестороннего исследования такого рода проблем требует использования более гибкого инструмента, который, обладая меньшей вычислительной производительностью, обеспечивал бы существенное повышение суммарной производительности исследований. С этой точки зрения на сегодня практически оптимальным является выбор СКМ Mathematica.

Исследование примера Румпа в СКМ Mathematica

В СКМ Mathematica начиная с версии 2.0 (в нынешнем виде – начиная с версии 3.0) используется два типа внутреннего представления вещественных чисел – числа с машинной точностью (mashine-precision numbers) и числа с произвольной точностью (abitrary-precision numbers). Числа с машинной точностью содержат фиксированное количество десятичных знаков, в типичном случае – 16, так что значение Precision для них всегда равно 16. В отличие от них, числа с произвольной точностью могут содержать произвольное, практически сколь угодно большое число знаков, и значение Precision для них может быть любым. Операции над числами с машинной точностью выполняются быстрее, чем для чисел с произвольной точностью, однако использование чисел с произвольной точностью позволяет получить практически сколь угодно точный результат. В общем случае числа с произвольной точностью представляются в следующем формате [12]:

$$\overbrace{\underbrace{x_1\,x_2\cdots x_s}_{\textbf{Scale}}.\underbrace{x_{s+1}\,x_{s+2}\cdots x_{s+a}}_{\textbf{Accuracy}}}^{\textbf{Precision}}\underbrace{z_1\,z_2\cdots z_n}_{\textbf{Guard digits}}$$

Где **Precision** (общая точность) – общее количество десятичных знаков в числе, **Accuracy** (точность дробной части) – количество цифр после десятичной точки, **Scale** – количество цифр перед десятичной точки, **Guard digits** - вспомогательные цифры для сохранения точности вычислений.

Для того чтобы обеспечить заданную точность вычислений необходима фиксация соответствующих параметров $MaxPrecision и $MinPrecision с помощью функции Block[] на время вычисления выражения. Для удобства можно написать собственную функцию, осуществляющую вычисление таким образом:

```
SetAttributes[FixedPrecisionEvaluate,HoldFirst];
FixedPrecisionEvaluate[input_,digits_]:=
Block[{$MaxPrecision=digits,$MinPrecision=digits},input];
```

Аргумент input — это вычисляемое выражение; digits — точность, которую необходимо зафиксировать (количество десятичных значащих цифр, которые может иметь число). Необходимо также, чтобы все числа, входящие в выражение, имели ту же фиксированную точность digits.

Для исследования полинома Румпа его необходимо представить в форме, позволяющей задавать необходимую точность:

$$\begin{aligned} f3[x\_,y\_,digits\_]:= {} & \mathrm{SetPrecision}[333.75,digits]\, y^6 \\ & + x^2 * \left(11 * x^2 y^2 - y^6 - 121 * y^4 - 2\right) \\ & + \mathrm{SetPrecision}[5.5, digits] * y^8 + \mathrm{SetPrecision}[0.5, digits] \frac{x}{y} \end{aligned}$$

Для вычисления полинома Румпа с различными значениями точности в диапазоне, например, от 1 до 25, необходимо сформировать следующее выражение:

```
Table[FixedPrecisionEvaluate[f3[SetPrecision[77617,i],SetPrecision[33096,i],i],i],{i,25}]
```

Полученный при этом результат будет представлен в виде следующего пакета данных:

```
{-1.,-1.0,-1.00,-1.000,-1.0000,-1.00000,-1.000000,-1.0000000,-1.00000000,-1.000000000,-
1.0000000000,-1.00000000000,-1.000000000000,-1.0000000000000,-1.00000000000000,-
1.000000000000000,-1.0000000000000000,-0.827396059946821368,-0.8273960599468213682,-
0.82739605994682136818,-0.827396059946821368176,-0.8273960599468213681761,-
0.82739605994682136817613,-0.827396059946821368176126,-0.8273960599468213681761258}
```

Проведенные исследования показали, что, начиная с определенного значения общей точности (в данном случае начиная уже с 18-ти десятичных цифр, т.е. при незначительном превышении двойной точности), мы получаем практически правильный результат, который на каждом последующем шаге лишь несколько уточняется.

К сожалению, у данного метода тот же недостаток, что и у метода символьных вычислений: заранее не известна погрешность результата. В общем случае совершенно не ясно, до каких пор стоит увеличивать точность, чтобы получить правильный результат с необходимой точностью.

Зависимость времени вычисления полинома Румпа от задаваемой точности в пакете Wolfram Mathematica 7 (в виде измеренных точечных значений для компьютера Apple MacBook с процессором класса Intel Core 2 Duo 2 GHz), а также ее линейная аппроксимация приведены на рис. 4. Из рисунка видно, что исследуемая зависимость является практически линейной.

В процессе исследований было также установлено, что резкое нарастание ошибки вычислений при недостаточной заданной разрядности имеет место не только для значений a=77617 и b=33096, но и для практически неограниченного количества сочетаний значений исходных параметров в случае, если они соотносятся как a/b = 77617/33096 =2.345.. (рис. 5).

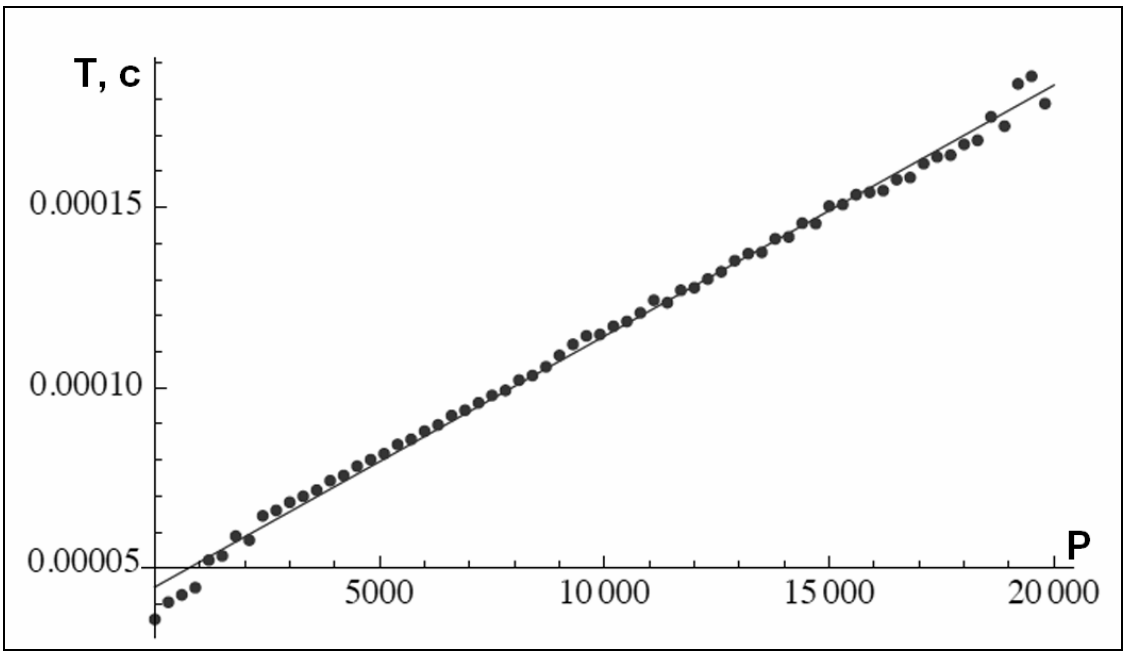

Рисунок 4 — Зависимость времени вычисления функции Румпа (T, секунд) от задаваемой фиксированной точности (P – Precision, количество десятичных разрядов).

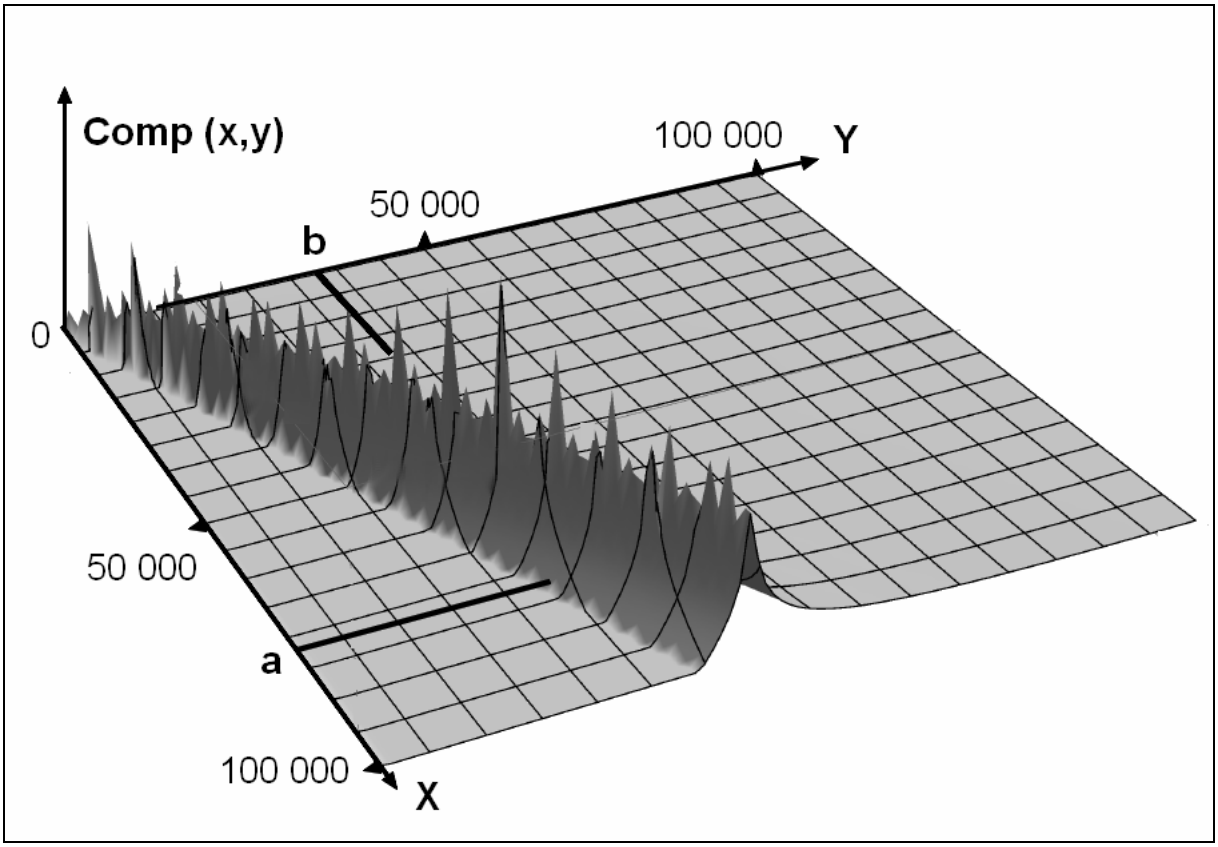

Рисунок 5 – Резкое нарастание ошибки вычисления полинома Румпа (Comp(x,y)) при значениях исходных данных, приближающихся к соотношению a/b = 77617/33096 =2.345..

Пример Румпа в контексте интервальных вычислений

Целым рядом авторов в контексте проблемы, связанной с примером Румпа, в качестве своеобразной панацеи предлагается использование интервальной арифметики [6]. В этом случае для расчетов вместо «точечных» чисел используются так называемые интервалы — диапазоны числовых значений, определенных левой/нижней и правой/верхней границами, что позволяет, как правило, решить целый ряд проблем, связанных с неточностью представления исходных данных и пр. [13].

В СКМ Mathematica 7 поддержка интервальной арифметики реализована достаточно эффективно и просто. Для того, чтобы ею пользоваться, достаточно при расчетах задавать исходные значения в виде интервалов. При этом ответ также выдается в виде интервала, в котором должно находиться искомое правильное значение. Например, выражение Sin[Interval[{0, 0.1}]] дает в результате Interval[{0, 0.0998334}].

Вычисление примера Румпа с использованием интервальной арифметики осуществляется при помощи интервалов, имеющих ширину, обеспечивающую необходимую точность (такие числа представляются в виде набора Interval[SetPrecision[a, digits]]). Чтобы вычислить значение полинома Румпа с использованием интервальной арифметики с точностью, например, в 20 цифр можно воспользоваться командой:

$$f[\mathrm{Interval}[\mathrm{SetPrecision}[77617, 20]], \mathrm{Interval}[\mathrm{SetPrecision}[33096, 20]]]$$

где f — полином Румпа.

С помощью интервальной арифметики можно с заданной точностью получить интервал, в котором заведомо находится искомый точный результат, и определить максимальную возможную погрешность вычислений (определяется из ширины интервала). Так, в частности, при заданной точности в 60 цифр, был получен достаточно узкий интервал, локализующий правильный результат вычисления полинома Румпа с точностью до 20-го десятичного знака после запятой:

```
f[Interval[SetPrecision[a, 60]], Interval[SetPrecision[b, 60]]]

Interval[{-0.827396059946821368142, -0.827396059946821368140}]
```

Зависимость ширины получаемого интервала от задаваемой точности вычислений представлена на рис. 6, где по оси ординат представлена ширина интервала, а по оси абсцисс — задаваемая точность (количество десятичных цифр). Ось ординат имеет логарифмический масштаб.

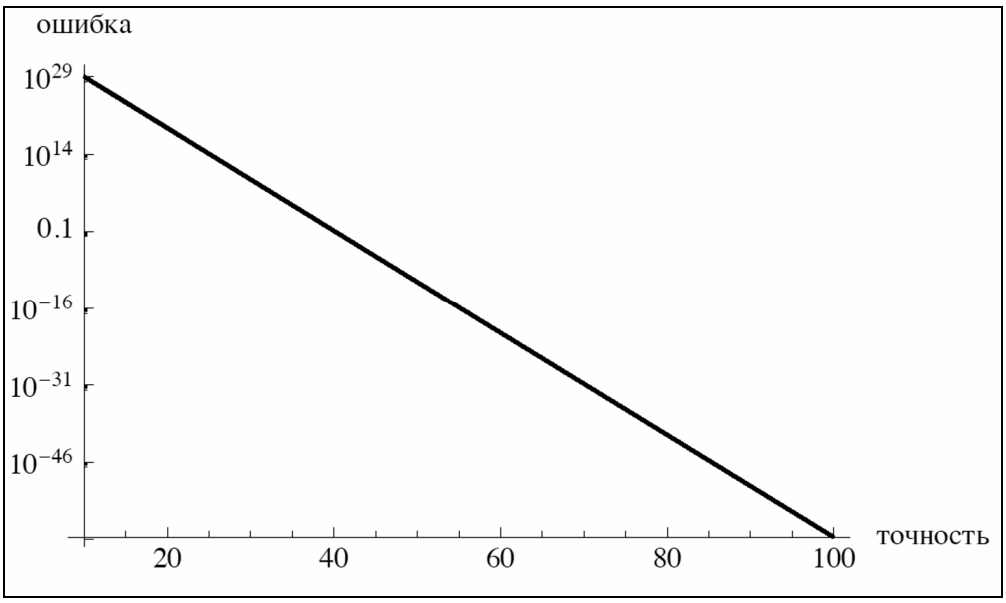

Рисунок 6 — Зависимость ошибки (итоговая ширина интервала) от точности (значение Precision, разрядов) при использовании интервальной арифметики для вычисления полинома Румпа

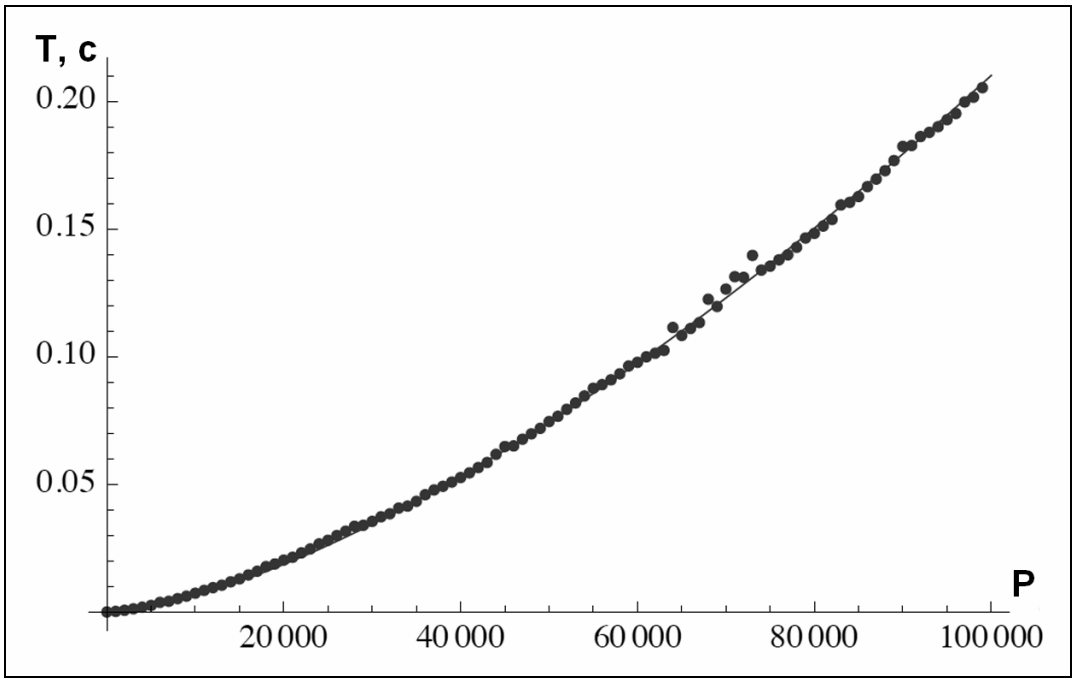

Рисунок 7 — Зависимость времени вычисления полинома Румпа (T, секунд) от задаваемой фиксированной точности (P – Precision, количество десятичных разрядов) при использовании интервальной арифметики.

В ходе исследований была также оценена зависимость времени вычислений с использованием интервальной арифметики от точности в СКМ Wolfram Mathematica 7. Эта зависимость приведена на рис. 7 в виде измеренных точечных значений (компьютер Apple MacBook с процессором класса Intel Core 2 Duo 2 GHz) и аппроксимации результатов функцией

$$y = ax^{1.5}.$$

Таким образом, использование традиционных интервальных вычислений также позволяет получить правильный результат в случае вычисления полинома Румпа, но при этом требуется существенно повышенная разрядность вычислений (примерно в 2 раза), так и значительно увеличенные затраты времени (более чем на порядок).

Преодоление проблем, связанных с ограничением разрядности

Полином Румпа является все-лищь моделью, демонстрирующей тот печальный факт, что современные вычисления весьма уязвимы и практически полностью не защищены от появления такого рода грубейших ошибок, рискующих остаться в большинстве случаев просто незамеченными вплоть до момента катастрофического проявления некорректных результатов вычислений. Другие примеры такого рода приведены, например, в работах [14-16]. Самое тревожное во всем этом то, что вероятность таких ошибок пока еще растет практически прямо пропорционально росту вычислительных мощностей современных компьютерных систем.

Как показали исследования, корректный результат в такого рода ситуациях может быть получен, как минимум, одним из 3-х следующих способов:

- увеличением разрядности до требуемого для корректных вычислений значения (для полинома Румпа в зависимости от способа вычислений это должна быть или учетверенная или превышающая ее точность).
- вычислением отдельно числителя и знаменателя с получением результата их деления только на самом последнем шаге;
- использованием интервальных вычислений.

Однако, вычислительная производительность при использовании перечисленных способов на современных компьютерных системах в среднем в разы ниже, чем при выполнении обычных вычислений ввиду преимущественно программной их реализации. Поэтому требуется поиск кардинально новых решений, позволяющих без существенного падения производительности добиться достаточно надежного и эффективного контроля за корректностью вычислений. Наиболее перспективным направлением при этом представляется разработка и реализация постбинарных методов вычислений, основанных на постбинарном представлении количественных значений [17, 18].

Следует отметить, что и в рамках традиционных вычислений уже делались достаточно успешные попытки преодоления проблем, связанных с ограничением разрядности традиционных компьютерных вычислений. Например, разрабатывавшиеся под руководством академика В.М. Глушкова в 1961-81 гг. в Институте кибернетики НАН Украины ЭВМ серии МИР (серийный выпуск с 1965 г., всего выпущено свыше 3-х тысяч), обеспечивали

достаточно эффективную реализацию работы с вещественными числами произвольной разрядности и целыми числами неограниченной разрядности. Кроме этого, была реализована поддержка точных операций над дробными рациональными числами и др. Компьютеры серии МИР, не имевшие аналогов в мире, патентно чистые и защищенные многочисленными авторскими свидетельствами СССР и других стран, были отмечены в 1968 году Государственной премией СССР. Это был первый случай в стране, когда такого рода наградой была отмечена работа в области вычислительной техники. Но существенного продолжения в дальнейшем эти разработки не получили. Одной из причин было то, что такого рода проекты существенно опережали свое время и плохо вписывались в парадигму традиционных бинарных вычислений.

В настоящее время уже созрели все предпосылки для существенной модификации всей системы компьютерных вычислений с целью повышения ее надежности и адекватности современным требованиям.

Постбинарные числа и вычисления

В современных условиях речь может идти как о расширении как логической, так и вычислительной составляющей современного компьютинга. Особо актуальной представляется при этом задача такой модификации вычислений с плавающей запятой, которая бы позволила исключить рассмотренные выше ситуации. Наиболее перспективным вариантом модификации представляется расширение существующих стандартных форматов представления чисел с плавающей запятой и вычислений с ними путем введения следующих изменений:

1. Для обеспечения совместного гибкого использования различных форматов чисел с плавающей запятой кроме стандарных полей мантиссы и порядка со знаками вводится поле идентификатора формата, для которого в стандартных форматах могут быть задействованы часть младших разрядов мантиссы (рис. 8) и которое в общем случае состоит из кода и модификатора формата (таблицы 1 и 2).
2. Идентификаторы формата определяют разрядность и форму представления чисел с плавающей запятой в диапазоне от 16-ти двоичных разрядов (binary16) до неопределенно больших значений. При этом вплоть до разрядности 128 (binary128) обеспечивается совместимость с существующими форматами (таблица 1), а начиная с разрядности binary256 обеспечивается последовательное удвоение разрядности формата по мере необходимости с соблюдением следующих правил роста разрядности отдельных элементов формата (при каждом удвоении): код и модификатор формата увеличиваются в совокупности на 2 разряда, порядок (экспонента) – на 4 разряда, под мантиссу используются все оставшиеся разряды формата, что в совокупности обеспечит постепенное увеличение

удельной разрядности мантиссы по мере роста совокупной разрядности представления чисел.

3. Кроме традиционного бинарного (точечного) представления численных значений дополнительно вводятся «парные форматы», в которых путем удвоения каждого из разрядов в виде единого «постбинарного значения» представляются пары традиционных точечных значений, интерпретируемые либо как числитель и знаменатель обыкновенной дроби, либо как граничные значения интервалов (для поддержки интервальных вычислений).
4. В качестве основного постбинарного формата рассматривается тетракод [17], на базе которого может быть реализовано представление «нормированных интервалов» (равных весовым коэффициентам двоичных разрядов) и своеобразное маскирование реально значимых разрядов (за счет использования значения «неопределенности» для разрядов не заданных и не полученных в явном виде, а появившихся, например, в результате дополнения разрядности числа до стандартной или при выравнивании порядков).
5. В алгоритмы выполнения арифметических операций вносятся такие изменения, которые позволят автоматически наращивать (и сокращать) разрядность по мере необходимости, что позволит реализовать арифметику «без округлений», а, следовательно, – и без потери точности.
6. Основаниями для перехода к увеличенной разрядности могут быть как возрастание требуемой разрядности мантиссы (например, при умножениях и возведениях в степень), так и возрастание требуемой разрядности порядка, что позволит исключить такие характерные для вычислений с плавающей запятой явления как исчезновение и переполнение порядка.

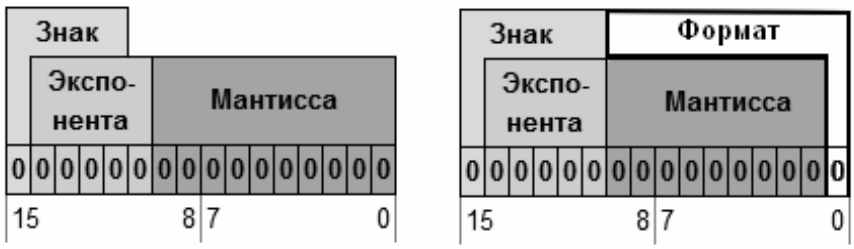

Рисунок 8 — Предлагаемая модификация формата чисел с плавающей запятой (на примере числа половинной точности): слева – традиционное представление, справа – формат с поддержкой постбинарного кодирования, содержащий на месте младших разрядов мантиссы код формата

Таблица 1 - Предлагаемая последовательность форматов чисел с плавающей запятой: жирным шрифтом выделены форматы стандарта IEEE 754-2008, в столбце «Разрядность мантиссы» вторые строки представляют модифицированную разрядность мантиссы с учетом бит, выделяемых для модификатора и кода формата

Формат **Код формата**	**Название**	**Разрядность мантиссы со знаком (модифицированная мантисса + код и модификатор формата)**	**Разрядность порядка (со знаком)**	**Макс. значение порядка**
binary16 **0**	**Half** **(половинная точность)**	**11** **(10+1)**	**5**	**+16**
binary32 01	Single **(одинарная точность)**	24 **(20+2+2)**	8	+127
binary64 011	Double **(двойная точность)**	53 **(47+3+3)**	11	+1023
binary128 0111	**Quadruple** **(квадро точность)**	113 **(105+4+4)**	15	+16383
binary256 **01111**	**Octuple** **(октоточность)**	**236** **(226+5+5)**	**20**	**+524287**

Таблица 2 – Предлагаемые модификации формата **binary32**

Модификатор	Формат	Модификация формата
00	**binary32**	**Single** стандарта IEEE 754-2008
01	binary32/16f	Дробное половинной точности
10	binary32/16i	Интервальное половинной точности
11	binary32/16p	Постбинарное половинной точности

Таблица 3 – Предлагаемые модификации формата **binary64**

Модиф-тор	Формат	Модификация формата
000	**binary64**	**Double** стандарта IEEE 754-2008
001	binary64/32f	Дробное одинарной точности
010	binary64/32i	Интервальное одинарной точности
011	binary64/32p	Постбинарное одинарной точности
100	**binary64/32**	Сдвоенный **Single** стандарта IEEE 754-2008
101	binary64/16fp	Постбинарное дробное Half
110	binary64/16ip	Постбинарное интервальное Half
111	binary64/16p	Постбинарное одинарной точности

Заключение

Детальный анализ основных проблем, связанных с традиционными вычислениями с плавающей запятой, приведен в фундаментальном издании «Искусство программирования» Дональда Кнута [19]. В частности, он отмечает следующее: «Вычисления над числами в формате с плавающей точкой неточны по самой своей природе, и программисту нетрудно столь неудачно организовать их выполнение, что полученные результаты будут почти полностью состоять из «шума». Одна из главных проблем численного анализа состоит в анализе точности результатов тех или иных численных методов; сюда же относится и проблема «степени доверия»: мы не знаем, насколько правильны результаты вычислений на компьютере… Многие из серьезных математиков пытались строго проанализировать последовательность операций с плавающей точкой, но, обнаружив, что задача слишком сложна, удовлетворялись правдоподобными рассуждениями» [19, с.265]. Слова эти, впервые написанные достаточно давно, актуальны и в настоящее время. Проведенное в современных условиях исследование примера Румпа весьма наглядное тому подтверждение.

С другой стороны, возможности формального доказательства корректности получаемых результатов также имеют тенденцию к резкому сужению, о чем профессор факультета математики Лондонского королевского колледжа Брайан Дэвис в своей довольно резонансной статье «Куда идет математика?» [20] пишет следующее: «Будущее чистой математики должно разительно отличаться от ее прошлого. В 1875 году любой грамотный математик мог полностью усвоить доказательства всех существовавших на тот период теорем за несколько месяцев. В 1975 году, за год до того, как была доказана теорема о четырех цветах, об этом уже не могло быть и речи, однако отдельные математики еще могли теоретически разобраться с доказательством любой известной теоремы. К 2075 году многие области чистой математики будут построены на использовании теорем, доказательства которых не сможет полностью понять ни один из живущих на Земле математиков — ни в одиночку, ни коллективными

усилиями» (цитируется по работе [21]). А это фактически означает, что в условиях широкого распространения сугубо компьютерных методов не только для сложных и сверхсложных вычислений, но и для математических доказательств различного уровня сложности, требования у уровню достоверности вычислительных процессов существенно возрастают.

В.М. Юровицкий в своих выводах еще более категоричен: «Цивилизационное развитие уже упирается в барьер нынешнего понятия числа (рационального числа). Задачей номер 1 является создание новой концепции числа и методов его обработки. Это будет величайший историко-цивилизационный поворот, переход в **новую числовую эпоху**. Такой переход будет не менее значим, чем переход от римской системы счисления к арабской, создавшей базу промышленной революции, чем переход от ручного счета к компьютерному» [22]. Но концепция так называемых метрологических чисел, предлагаемая В.М. Юровицким, являясь шагом в правильном направлении, в современных условиях не может рассматриваться в качестве достаточного и окончательного решения.

Речь должна идти о переходе к постбинараным вычислениям в самом широком контексте: начиная от постбинарной компьютерной логики [17] и заканчивая существенным расширением форм и форматов компьютерного представления чисел, на базе чего должен в конечном итоге сформироваться **постбинарный компьютинг**, включающий существующие сегодня формы компьютинга лишь в качестве частного случая.

Список литературы

1. Петров Ю.П. Обеспечение надежности и достоверности компьютерных расчетов. – СПб: БХВ-Петербург, 2008. – 160 с.
2. Rump S. M. Algorithm for verified inclusions: Theory and practice // Reliability in Computing, R. E. Moore ed., Academic press, San Diego, 1988, p. 109-126.
3. Cuyt A., Verdonk B., Becuve S., Kuterna P. A Remarkable Example of Catastrophic Cancellation Unraveled // Computing, 66, 2001, p. 309-320.
4. Loh E., Walster G. Rump's Example Revisited // Reliable Computing 8: 2002, p. 245–248.
5. Rump S. M. Rigorous results using floating-point arithmetic // Acta Numerica, 19, 2010, p. 287–449.
6. Farber R. Numerical Precision: How Much is Enough? As we approach ever-larger and more complex problems, scientists will need to consider this question. 2009. http://www.scientificcomputing.com/article-hpc-Numerical-Precision-How-Much-is-Enough-063009.aspx.
7. Алексеев Е. Р., Чеснокова Е. А., Рудченко Е. А. Решение инженерных и математических задач в пакете Scilab. — М.: ALT Linux, 2008. — 257 с.
8. Кирьянов Д.В. Самоучитель MathCAD 2001 /Д.В. Кирьянов — СПб.: БХВ-Петербург, 2002. — 544 с.
9. Arbitrary-precision arithmetic // From Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic.
10. The GNU Multiple Precision Arithmetic Library, http://gmplib.org/

11. Шакиров Р.Н. Класс cBigNumber для целочисленной арифметики неограниченной разрядности в языке C++ // Программные продукты и системы. 2009. 1. С. 7-11.

12. Sofroniou M., Spaletta G. Precise Numerical Computation, 2000 // www-sop.inria.fr/lemme/AOC/workshop/mark_sofroniou.ps.gz.

13. Аноприенко А.Я., Иваница С.В. Интервальные вычисления и перспективы их развития в контексте кодо-логической эволюции // Научные труды Донецкого национального технического университета. Серия «Проблемы моделирования и автоматизации проектирования динамических систем» (МАП-2010). Выпуск 8 (168): Донецк: ДонНТУ, 2010. С. 150-160.

14. Яшкардин В. IEEE 754 – стандарт двоичной арифметики с плавающей точкой, 2010, http://www.softelectro.ru/ieee754.html.

15. Юровицкий В.М. IEEE754-тика угрожает человечеству, http://www.yur.ru/science/computer/IEEE754.htm

16. Пехотин Е.В. Методы выполнения операций высокой точности над многоразрядными числами в формате с плавающей точкой // Портал магистров ДонНТУ, 2010, http://masters.donntu.edu.ua/2010/fknt/pehotin/diss/index.htm.

17. Аноприенко А.Я. Обобщенный кодо-логический базис в вычислительном моделировании и представлении знаний: эволюция идеи и перспективы развития // Научные труды Донецкого национального технического университета. Серия «Информатика, кибернетика и вычислительная техника» (ИКВТ-2005) выпуск 93: – Донецк: ДонНТУ, 2005. С. 289-316.

18. Аноприенко А.Я., Иваница С.В. Особенности постбинарного кодирования на примере интервального представления результатов вычислений по формуле Бэйли-Боруэйна-Плаффа // Научные труды Донецкого национального технического университета. Серия: «Информатика, кибернетика и вычислительная техника» (ИКВТ-2010). Выпуск 11 (164). – Донецк: ДонНТУ, 2010. С. 19-23.

19. Кнут Д.Э. Искусство программирования, том 2. Получисленные алгоритмы. – М.: Издательский дом «Вильямс», 2000. – 832 с.

20. Davies E.B. Whither Mathematics? // Notices of the American Mathematical Society. December 2005. Volume 52, Number 11. – P. 1350-1356.

21. Дэвис Б. Куда идет математика? // Элементы большой науки, 14.11.2005, http://elementy.ru/news/164970.

22. Юровицкий В.М. Общая теория чисел и числовых эпох. Мир на пороге новой числовой эпохи // Материалы международной конференции «Диалог-2008», Кипр 12-15 мая 2008 г., http://www.yur.ru/Conference/Cipros/Theory-of-number.htm.

Надійшла до редакції 24.12.2010 р. *Рецензент: д.т.н., проф..Башков Є.О.*

О.Я. Анопрієнко, В.А. Гранковський, С.В. Іваниця
Донецький національній технічний університет

Приклад Румпа в контексті традиційних, інтервальних і постбінарних обчислень. На базі прикладу, запропонованого професором З. Румпа ще в 1988 році, проведений детальний аналіз умов формування невірних результатів у сучасних системах комп'ютерної математики. Проаналізовано можливість використання інтервальних методів для мінімізації негативних наслідків отримання помилкових результатів обчислень. Запропоновано варіанти постбінарного подання чисельної інформації та методи роботи з нею, що виключають появу ситуацій, подібних тим, які мають місце у випадку з прикладом Румпа.

Приклад Румпа, інтервальні обчислення, постбінарні обчислення.

A. Anoprienko, V. Grankovsky, S. Ivanitsa
Donetsk National Technical University

Rump example in the context of traditional, interval and postbinary computing. On the basis of the example offered by professor H. Rump in 1988, a detailed analysis of the formation conditions of incorrect results in modern systems of computer mathematics is described. The possibility of using interval and postbinary methods to minimize adverse impacts of obtaining erroneous results of calculations is analyzed.
Rump example, interval computing, postbinary computing.