Тестирование современных библиотек тригонометрических функций

Е. С. Чернов, В. В. Кулямин
{ches, kuliamin}@ispras.ru

Аннотация. Данная статья посвящена созданию тестового набора по методу, описанному в работе [1], для четырех тригонометрических функций (cos, tan, arccos, arctan) на основе существующих стандартов и дополнительных требований, возникающих из математических особенностей этих функций и специфических свойств представления чисел с плавающей точкой. Здесь также представлены результаты тестирования различных реализаций этих функций в математических библиотеках на разных платформах и при различных режимах округления.

1. Введение

Одной из областей человеческой деятельности, опирающейся на использование сложного программного обеспечения (ПО), является математическое моделирование сложных явлений. Это, например, моделирование развития вселенной в целом, галактик и звезд, физических процессов в экстремальных условиях, биохимических, климатических и социальных процессов. Во многих случаях компьютерное моделирование дает важную информацию о таких явлениях, но очень нелегко получить независимую оценку правильности получаемых результатов, что позволило бы проверить корректность самого используемого ПО. Это порождает серьезные проблемы при разработке надежных систем математического моделирования.

Тем не менее, повысить надежность и правильность работы таких систем можно за счет формальной проверки корректности библиотечных компонентов, на которые оно во многом опирается, в частности, реализаций математических функций. Уверенность в надежности фундамента, на котором построены эти системы, даст возможность разрабатывать их более качественно и с меньшими усилиями, сосредоточившись на поиске и исправлении ошибок в других компонентах. Правильность работы библиотек обычно определяется стандартами. Но большинство имеющихся стандартов, которые должны определять требования к реализациям математических функций, почти не содержат таких требований, ограничиваясь общими указаниями и апеллируя к широкой известности свойств этих функций. При попытке детализировать подразумеваемые требования оказывается, что

свойства реализаций функций существенно отличаются от свойств самих функций из-за использования чисел с плавающей точкой. В связи с этим актуальной является работа по определению четких требований к работе реализаций математических функций на основе чисел с плавающей точкой, а также разработка тестовых наборов для проверки соответствия этим требованиям. Этим двум вопросам и посвящена данная статья.

Во втором разделе статьи производится обзор существующих стандартов для реализаций математических функций и существующих тестовых наборов. В третьем разделе проводится исследование свойств 4-х тригонометрических функций и на его основе строится тестовый набор для них. В четвертом разделе представлены результаты тестирования нескольких различных реализаций выбранных функций с помощью созданного набора.

2. Обзор стандартов и тестовых наборов

Правильность работы реализаций математических функций определяется стандартами. Рассмотрим несколько наиболее распространенных стандартов для математических функций и требования, которые они предъявляют к работе их реализаций.

2.1. Существующие стандарты для математических функций

Общепринятые базовые стандарты на вычисления с числами с плавающей точкой — это IEEE 754 [2] и IEEE 854 [3]. Они определяют представление чисел с плавающей точкой и основные операции над ними. Помимо обычных чисел, числа с плавающей точкой могут представлять положительную и отрицательную бесконечности ($\pm\infty$), а также NaN («not-a-number» — «не число», приписываемое как результат действиям, у которых нельзя определить корректно ни конечный, ни бесконечный результат) и -0 (отрицательный ноль, используемый иногда в значении «очень маленькое по абсолютной величине отрицательное число»). Кроме того, обычные числа могут быть *нормализованными* (если их абсолютная величина достаточно велика, например, при двойной точности — не меньше 2^{-1022}) или *денормализованными* (если их абсолютная величина меньше некоторого, своего для каждой точности, значения).

Для получения представимых результатов при выполнении вычислений с числами с плавающей точкой эти стандарты предписывают использовать один из четырех *режимов округления*: к ближайшему числу, вверх, вниз и к нулю. Они также предписывают возвращать правильно округленный точный результат при выполнении арифметический действий (сложения, вычитания, умножения, деления), преобразовании типов, взятии остатка по модулю и извлечении квадратного корня. Про другие функции в этих стандартах ничего не сказано.

Стандарты библиотек языка C. В стандарте языка C ISO/IEC 9899 [4] и в стандарте переносимого интерфейса операционной системы IEEE 1003.1 [5] (известном как POSIX), описывающих библиотеку математических функций языка C, так же мало говорится о других функциях.

Стандарт языка C ссылается на стандарт IEEE 754, добавляя лишь требования на поведение ряда функций в некоторых особых точках. Например, $exp(0) = 1$, а $sin(0) = 0$. Стандарт POSIX, в свою очередь, ссылается на требования стандарта языка C, добавляя лишь требования на выставление флагов в случае возникновения переполнения, слишком маленьких или слишком больших аргументах, а также на поведение функции при денормализованных аргументах, в нуле, в бесконечности и в NaN. К примеру, для функции синус POSIX требует, чтобы $sin(x) = x$ при $x = 0, -0$, а также при денормализованных значениях x. Значение синуса для NaN, $-\infty$ и $+\infty$ должно быть NaN. Кроме того, указаны случаи выставления флага выхода за нормализованные значения (range error) и флага недопустимого аргумента (domain error). Других ограничений на синус не накладывается.

Стандарты ISO/IEC 10967. В последние 5-10 лет появились предложения стандартизовать необходимые для аккуратного моделирования требования к реализациям математических функций [6,7]. Многие инициаторы этой деятельности работают в проекте Arenaire [8], совместно проводимом во Франции INRIA, CNRS и Высшей Нормальной школой Лиона. В результате появился набор стандартов ICO/IEC 10967 [9-11], формулирующий естественные ограничения на работу реализаций элементарных функций — корней, экспонент и логарифмов с различными основаниями, гиперболических и тригонометрических функций, а также обратных к ним. Эти ограничения касаются нескольких аспектов: точности вычислений, интервалов сохранения знака и монотонности, а также поведения функции в окрестности некоторых особых для данной функции точек (например, реализация экспоненты для значений аргументов, достаточно близких к 0, должна возвращать в точности 1).

Эти стандарты на сегодняшний день являются наиболее полными и строгими, однако, к сожалению, они не используются на практике, поскольку не поддерживаются никем из основных производителей программных или аппаратных реализаций математических функций. Кроме того, набор стандартов ICO/IEC 10967 ничего не говорит о неэлементарных функциях, к которым относятся некоторые часто используемые библиотечные функции, например, гамма-функция и функции Бесселя.

2.2. Работы по тестированию реализаций математических функций

Существует достаточно большое количество различных наборов тестов для тестирования функций над числами с плавающей точкой (см. например, [12]),

но подавляющее большинство таких тестов крайне несистематично и проверяет какой-то один аспект вычислений.

Несмотря на то, что стандартизация вычислений над числами с плавающей точкой началась более 20 лет назад, проблема проверки соответствия реализаций стандартам до сих пор актуальна и тесты на правильность поведения реализаций математических функций по-прежнему необходимы.

Среди наиболее систематичных работ по тестированию вычислений с плавающей точкой можно назвать следующие.

- Тестовый набор ELEFUNT [13] содержит тесты для нескольких математических функций в виде программ на C и Java. В этих тестах проверяется корректность возвращаемых значений для специальных значений аргументов (0, 1, $+\infty$, $-\infty$, NaN), а также для ряда генерируемых случайно значений аргументов проверяется выполнение некоторых тождеств, связанных с данной функцией (например, $exp(x)\cdot exp(-x) = 1$).
- Тестовый набор UCBTEST [14] предназначен для тестирования базовых арифметических действий и достаточно широкого набора математических функций (тестируется больше функций, чем в ELEFUNT). Он оформлен как набор тестовых программ на Fortran и C и предопределенных входных данных для разных функций. В качестве тестовых данных, по-видимому, выбраны специальные значения аргумента и несколько случайным образом полученных значений. В целом, однако, в этом наборе используется больше данных и проверяется больше функций, чем в ELEFUNT.
- Использование в качестве тестовых данных специальных значений, границ интервалов, определяемых часто используемыми алгоритмами вычисления данной функции, а также чисел, построенных по некоторым шаблонам и случайных значений, применялось для построения более объемных тестовых наборов, например, набора Беркли [15].

Работы, выполнявшиеся в проекте Arenaire. Отдельно стоит отметить работы в рамках проекта Arenaire [16] по определению чисел, для которых корректное вычисление элементарных функций с заданной точностью наиболее трудоемко. На основе их результатов был разработан инструмент MPCheck [17] для тестирования правильности реализаций элементарных функций с точки зрения сохранения монотонности и корректности округления.

В целом, имеющиеся в открытом доступе исследования, посвященные выработке требований к реализациям математических функций и тестированию на соответствие этим требованиям, не содержат систематического подхода к этим вопросам, объединяющего рассмотрение всех указанных проблем. Нигде, кроме работ группы Arenaire, не

рассматривается дилемма составителя таблиц (необходимость использовать вычисления повышенной точности для корректного округления) и значения аргументов, для которых вычисление корректных результатов функций наиболее трудоемко. В то же время, в работах группы Arenaire никак не фигурируют тестовые данных, основанные на границах интервалов, в которых заданная функция ведет себя однородным образом.

3. Анализ требований к функциям и создание тестового набора

3.1. Методика определения требований

Используемый метод определения требований к реализациям математических функций описан в статье В. Кулямина [1] и заимствует б*о*льшую часть идей из стандарта ISO 10967 [9-11] и работ [6,7], посвященных разработке стандартов с повышенными требованиями к корректности вычисления математических функций.

Требования к результатам реализации математической функции могут быть разделены на несколько аспектов.

- **Область определения функции и особые точки функции.**
  - *Область определения*. Во всех точках области определения, где значения функции лежат в интервале чисел с плавающей точкой, ее реализация должна возвращать соответствующим образом округленные значения.
  - *Предельные значения.* В точках, где функция не определена, но имеет предел в диапазоне чисел с плавающей точкой, реализация должна возвращать округленное значение этого предела.
  - *Односторонние пределы в 0.* Значения функции в точках 0, -0 определяются как односторонние пределы. Например, *ctg*(0) = +∞ и *ctg*(-0) = -∞.
  - *Полная неопределенность.* В точках, где функции нельзя осмысленно приписать ни конечного, ни бесконечного значения, ее значением считается NaN. Примеры таких ситуаций: *sqrt*(-1.0) = *ln*(-1.0) = NaN, *sin*(+∞) = NaN.
  - *Окрестности полюсов.* Необходимо точно определить окрестности полюсов, в которых значение функции выходит за интервал чисел с плавающей точкой.
  - *Окрестности бесконечностей при бесконечных пределах.* Для функций, стремящихся к бесконечности при $x \to +\infty$ или $x \to -\infty$, должны быть точно определены границы, за пределами которых их значения не представимы.

- **Специальные значения, значения в 0, касательные и асимптоты.**
  - *Бесконечности и -0.* Нужно наиболее естественным образом определить значения функции для особых значений аргумента: -0, +∞, -∞.
  - *Значение в* NaN. Значение функции для аргумента NaN должно быть равно NaN.
  - Точные значения. В ряде точек значения функций должны вычисляться точно. Например, *exp*(0) = *cos*(0) = *ch*(0) = 1, *sin*(0) = *sh*(0) = *tg*(0) = *arcsin*(0) = 0, *ln*(1) = 0 и т.п.
  - *Окрестности экстремумов, являющихся точными значениями.* Если в такой точке производная функции равна 0, то для любого аргумента из некоторой ее окрестности реализация должна возвращать то же самое значение.
  - *Окрестность 0.* Функции, имеющие ненулевое значение в 0, тоже должны в некоторой окрестности 0 возвращать это же значение.
  - *Горизонтальные асимптоты.* В тех случаях, когда функция имеет горизонтальные асимптоты, необходимо аккуратно определить границы, после которых ее значение должно стать постоянным при определенном режиме округления.
  - *Асимптотики.* На интервалах, где основные асимптотики функций при правильном округлении превращаются в тождества, следует требовать соблюдения этих соотношений. Например, $exp(x) \sim 1+x$ при $x \sim 0$ или $sin(x) \sim x$ при $x \sim 0$.

- **Область значений функции.**
  - *Выбор ветви.* Если математическая функция в строгом смысле является многозначной, должна быть четко определена ее ветвь, которая будет соответствовать реализации.
  - *Сохранение ограничений области значений.* Ограничения сверху или снизу на значения функции нужно соблюдать и в ее реализации, в противном случае возможны ложные нарушения непрерывности поведения при моделировании сложных систем.
  - *Денормализованные значения.* Необходимо аккуратно определить интервалы, на которых значения функции должны быть денормализованными.

- **Монотонность и сохранение знака.**
  - *Сохранение монотонности.* На всех интервалах, где математическая функция монотонна, ее реализация должна иметь тот же вид монотонности.

- *Сохранение знака.* Знак значения реализации функции для некоторого аргумента должен совпадать со знаком значения самой функции.

- **Симметрии и периодичность.**
  - *Четность и нечетность.* Если математическая функция является четной или нечетной, этим же свойством должна обладать ее реализация.
  - *Другие симметрии.* Помимо четности и нечетности все другие симметрии функции должны быть проанализированы на предмет возможности их соблюдения для чисел с плавающей точкой. Например, соотношение $\Gamma(1+x) = x\cdot\Gamma(x)$ для гамма-функции может быть выполнено точно только для представимых целых чисел, для них оно и должно быть выполнено в любой реализации. Соотношения типа периодичности тригонометрических функций, или, например, $sin(\pi-x) = sin(x)$, могут выполняться на числах с плавающей точкой только приближенно, поэтому не имеет смысла накладывать соответствующие ограничения.

- **Корректное округление.**
  Помимо всех перечисленных ограничений, нужно требовать, чтобы результат, возвращаемый реализацией, получался из точного результата функции для данного аргумента при помощи округления в соответствии с текущим режимом.

3.2. Анализ требований для выбранных функций

Для двух тригонометрических функций и двух обратных к ним — тангенса, косинуса, арктангенса и арккосинуса — анализ требований для дальнейшего построения тестового набора был проведен по описанной выше методике. При этом пришлось иметь дело со следующими их особенностями.

- **Нули, полюса и прообразы специальных значений.**
  Для определения интервалов сохранения знака важно вычислить нули и полюса функции. Для определения интервалов монотонности ту же роль играют максимумы и минимумы.
  Нули косинуса совпадают с полюсами тангенса и являются числами вида $\pi/2+\pi\cdot n$. Максимумы и минимумы косинуса имеют вид $2\pi\cdot n$ и $(2n+1)\cdot\pi$. Кроме того, для проверки корректности вычисления тригонометрических функций можно использовать широко известные соотношения вида $cos(\pm\pi/3+2\pi\cdot n) = 1/2$, $cos(\pm5\pi/3+2\pi\cdot n) = -1/2$, $tg(\pi/4+\pi\cdot n) = 1$ и $tg(-\pi/4+\pi\cdot n) = -1$.

  Во всех этих случаях фигурируют целые кратные иррациональных чисел $\pi/3$ или $\pi/4$. Чтобы определить накладываемые этими соотношениями ограничения на реализации тригонометрических

функций на базе чисел с плавающей точкой, нужно вычислить числа с плавающей точкой, приближающие подобные числа наилучшим образом. Это можно сделать с помощью одной и той же техники (упоминаемой, например, в [18,19]). Любое число с плавающей точкой, большее 1, может быть представлено как $m\cdot2^k$, где m — натуральное число, не превосходящее $2^{53}-1$, а k — натуральное число, не превосходящее 971. Если такое число расположено очень близко к $n\cdot\alpha$, где α — иррациональное число, то $\alpha \approx m\cdot2^k/n$. Таким образом, можно искать эти числа с помощью наилучших рациональных приближений к α. Для этого числа вида $\alpha\cdot2^k$ при k от 52 до -971 раскладываются в непрерывные дроби, при обрыве которых в каком-либо месте получаются подходящие дроби, наилучшим образом приближающие исходные числа среди всех рациональных чисел, имеющих меньшие или такие же знаменатели [20].

Такие вычисления дают около 1000 кратных для каждого из чисел $\pi/4$, $\pi/3$, $\pi/2$ и π, которые приближаются числами с плавающей точкой двойной точности, с абсолютной погрешностью не больше, чем 2^{-52}. Эти числа с плавающей точкой являются ближайшими к нулям, полюсам, максимумам, минимумам или к прообразам значений ±1 и $\pm1/2$ для косинуса и тангенса. Поэтому они являются хорошими тестовыми значениями.

Например, так можно вычислить число двойной точности, ближайшее к нечетному кратному $\pi/2$ — $1.0110101011000101101100100110001011001010000111111111_2\cdot2^{849}$. Поскольку нечетные кратные $\pi/2$ являются полюсами тангенса, вычислив его значение в этой точке, получаем максимальное по абсолютной величине его значение на числах двойной точности — $2.133485385753703936_{10}\cdot10^{18}$. Так как кратные $\pi/2$ являются нулями косинуса, его значение в этой точке дает минимальное по абсолютной величине значение косинуса на числах с плавающей точкой двойной точности вне окрестности нуля — $4.687165924254626691_{10}\cdot10^{-19}$.

- **Асимптотики.**
  Тангенс и арктангенс имеют хорошо известные асимптотики в нуле: $tg(x) \sim x$ и $arctg(x) \sim x$ при $x \sim 0$. Арктангенс имеет горизонтальные асимптоты на бесконечности: $arctg(x) \sim \pi/2$ при $x \sim +\infty$ и $arctg(x) \sim -\pi/2$ при $x \sim -\infty$.
  Каждая из таких асимптотик определяет некоторые интервалы чисел с плавающей точкой, в которых она должна быть выполнена точно при режиме округления к ближайшему и либо точно, либо же с отступом на одно значение при других режимах («точное» равенство $\pi/2$ означает равенство числу с плавающей точкой, к которому $\pi/2$ округляется при заданном режиме).

Кроме того, поскольку в окрестности нуля числа с плавающей точкой расположены плотнее, чем в окрестностях других точек, любая функция, имеющая ненулевое значение в 0, должна принимать его же и в некоторой окрестности нуля. Это верно для косинуса и арккосинуса — первый в нуле имеет значение 1, второй — π/2.

Границы соответствующих интервалов можно вычислить с помощью метода последовательных приближений. Например, для тангенса имеем

$tg(1.00100101000010111111111000011011000010000010111101000_2 \cdot 2^{-26} = x_1) =$
$1.00100101000010111111111000011011000010000010111101000\ 01^{50}01\ldots_2 \cdot 2^{-26}$;
$tg(1.00100101000010111111111000011011000010000010111101010_2 \cdot 2^{-26}) =$
$1.00100101000010111111111000011011000010000010111101010\ 10^{52}10\ldots_2 \cdot 2^{-26}$;
$tg(1.01110001001101110100010010010001001000111110111101010_2 \cdot 2^{-26} = x_2) =$
$1.01110001001101110100010010010001001000111110111101010\ 1^{50}011\ldots_2 \cdot 2^{-26}$;
$tg(1.01110001001101110100010010010001001000111110111101110_2 \cdot 2^{-26}) =$
$1.01110001001101110100010010010001001000111110111101111\ 0^{55}100\ldots_2 \cdot 2^{-26}$.

Следующие из этих соотношений требования к значениям тангенса в окрестности 0 для всех режимов округления сформулированы в Таблице 1. В этой таблице функция *next*(*x*) для числа с плавающей точкой *x* обозначает непосредственно следующее за *x* число с плавающей точкой, а функция *previous(x)* — число, предшествующее x.

Округление	**к -∞**	**к 0**	**к ближайшему**	**к +∞**
Интервал/точка				
$x = -next(x_2)$	*Previous* (*previous*(*x*))	*previous*(*x*)		
$-next(x_1) \ge x \ge -x_2$	*previous*(*x*)	*x*	*previous*(*x*)	*x*
$0 > x \ge -x_1$	*previous* (*x*)	*x*		
$x = 0$	0			
$x_1 \ge x > 0$	*x*			*next(x)*
$x_2 \ge x \ge next(x_1)$	*x*	*next(x)*		
$x = next(x_2)$	*next*(*x*)			*Next* (*next*(*x*))

Таблица 1. Требования к поведению тангенса в окрестности 0.

Аналогичные требования определяются и для других функций в тех интервалах, где их асимптотики превращаются в точные или почти точные равенства.

- **Симметрии.**
  Тангенс и арктангенс являются нечетными функциями, а косинус — четной. Эти свойства учитываются при выборе тестовых значений и проверяются в тестах. Другие симметрии, в частности, периодичность, выполнены для чисел с плавающей точкой только приближенно и не накладывают строгих требований на реализации выбранных функций.

3.3. Выбор тестовых значений

В качестве тестовых значений для всех функций выбирались специфические числа с плавающей точкой: 0, -0, +∞, -∞, NaN, минимальное и максимальное положительные денормализованные и нормализованные числа, противоположные им, а также вычисленные границы интервалов действия асимптотик для различных режимов округления.

Были выбраны периоды тангенса и косинуса, содержащие хотя бы одну из упоминавшихся выше точек, близких к кратным π (включая 0), π/2, π/3 (для косинуса) и π/4 (для тангенса). Для каждого из выбранных периодов вычислены его левый и правый концы, а также точки, ближайшие на этом интервале к кратным π/2, π/3 (для косинуса) и π/4 (для тангенса). Эти точки разбивают каждый период на несколько интервалов.

Всего таких точек меньше, чем 3000, поэтому получающийся набор тестовых данных будет одновременно достаточно представительным и обозримым.

Полученные таким образом интервалы можно разбить далее, пользуясь техникой, описанной в статье [1] в разделе 4.2. А именно, разбивать каждый интервал на N более мелких интервалов, при этом получая (N+1) точку. Далее брать все числа, лежащие в рассматриваемом интервале и отстоящие не более чем на K чисел с плавающей точкой.

Кроме того, к тестовым значениям добавлены числа, для которых вычисление корректно округленного значения функций наиболее трудоемко. Вот пример такого числа из [21] для косинуса.

$cos(1.10000000000000000000000000000000000000000000001001_2 \cdot 2^{-23}) =$
$1.11111111111111111111111111111111111111111111101110000\ 0^{89}110\ldots_2 \cdot 2^{-01}$

Для четных или нечетных функций для каждого полученного уже тестового значения в тестовый набор было добавлено противоположное значение. Значение косинуса для него должно быть таким же, а значение тангенса и котангенса вычисляются по-разному в зависимости от режима округления. При округлении к 0 или к ближайшему достаточно просто поменять знак значения нечетной функции, чтобы получить ее значение на противоположном аргументе. При режимах округления к ±∞ нужно менять и знак, и режим — значение нечетной функции при режиме округления к +∞ на противоположном аргументе получается сменой знака из ее значения на исходном аргументе при режиме округления к -∞, и наоборот.

4. Результаты тестирования

4.1. Тестовые платформы

Тестирование проводилось в рамках проекта OLVER с использованием технологии UniTESK. Были протестированы реализации 4-х тригонометрических функций *(tan, cos, arctg, arcos*) при четырех режимах округления (“к ближайшему”, “к +∞”, “к -∞” и “к нулю”) на следующих пяти платформах:

1. Glibc 2.3.4 (Red Hat 4.64 AMD64);
2. Glibc 2.3.6 (Debian 4 IA32);
3. Glibc 2.4 (SUSE 10.0 IA32);
4. Glibc 2.4 (SUSE 10.0 Itanium);
5. MS Visual C++ RunTime Library (Windows XP IA32).

4.2. Классификация ошибок

Для анализа ошибок числа с плавающей точкой были разделены на 5 классов: NaN, ±∞, ±0, нормализованные числа и денормализованные числа.

При сборе статистики использовалась следующая классификация ошибок.

- “*Грубые ошибки*” фиксировались, если полученный и правильный результаты лежали в разных классах по приведенной классификации.
- “*Нарушение знака*” — полученный результат имеет неправильный знак.
- “*Очень неточный результат*” — все 53 бита полученного результата отличаются от соответствующих бит в правильном результате.
- “*Не очень точный результат*” — количество неправильных бит в полученном результате меньше 53.

Если полученный результат совпадал с правильным (совпадал знак, экспонента и все биты мантиссы), то никакой ошибки не фиксировалось, и результат считался “*абсолютно точным*”.

4.3. Анализ результатов тестирования

В Таблице 2 приведена общая статистика ошибок, обнаруженных при выполнении построенных тестов при режиме округления к ближайшему.

Функция/ Платформа	Грубые ошибки	Нару-шение знака	Очень неточный результат	Не очень точный результат	Абсолютно точный результат
Тангенс					
Glibc 2.3.6 (IA32)	0	9305	5772	7324	599
Glibc 2.4 (IA32)	0	9305	5772	7324	599
MSVCRT(Windows)	0	9337	5708	7356	599
Glibc 2.3.4 (AMD64)	0	5010	3015	1829	13146
Glibc 2.4 (Itanium)	0	5010	3015	1832	13143
Косинус					
Glibc 2.3.6 (IA32)	0	10128	2691	11336	1006
Glibc 2.4 (IA32)	0	10128	2691	11336	1006
MSVCRT(Windows)	0	10115	2700	11340	1006
Glibc 2.3.4 (AMD64)	0	5359	1445	3926	14431
Glibc 2.4 (Itanium)	0	5359	1445	3934	14423
Арктангенс					
Glibc 2.3.6 (IA32)	0	3914	1792	6324	4010
Glibc 2.4 (IA32)	0	3914	1792	6324	4010
MSVCRT(Windows)	0	0	0	5	16035
Glibc 2.3.4 (AMD64)	0	3968	1752	6310	4010
Glibc 2.4 (Itanium)	0	3968	1752	6310	4010
Арккосинус					
Glibc 2.3.6 (IA32)	0	0	0	0	1116
Glibc 2.4 (IA32)	0	0	0	0	1116
MSVCRT(Windows)	0	0	0	0	1116
Glibc 2.3.4 (AMD64)	0	0	0	0	1116
Glibc 2.4 (Itanium)	0	0	0	0	1116

Таблица 2. Общая статистика обнаруженных ошибок.

Грубых ошибок ни на одной из платформ зафиксировано не было. Арккосинус во всех протестированных реализациях вычисляется достаточно точно, поскольку наиболее прост для вычисления из рассматриваемых функций.

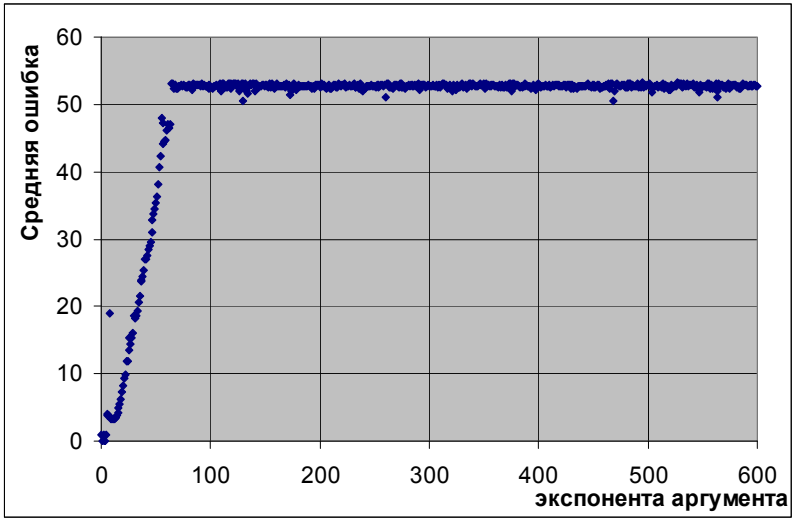

Рис. 1. Средняя величина ошибки (в количестве неправильных знаков результата) при вычислении тангенса в зависимости от экспоненты аргумента (IA32).

Тангенс и косинус вычисляются с заметным количеством ошибок, в основном связанных с корректной редукцией аргумента — аккуратным приведением аргумента в интервал от 0 до $\pi/2$ [19]. Дело в том, что для правильного приведения в этот интервал больших по абсолютной величине значений, необходимо хранить много дополнительных знаков числа π, что, по-видимому, не делается ни в одной из проверенных реализаций. Величина ошибок начинает возрастать при некотором удалении от нуля (там, где используемого представления π становится недостаточно для корректной редукции аргумента), причем на 64-разрядных платформах это происходит гораздо позже, чем на 32-разрядных (см. Рис. 1 и 2).

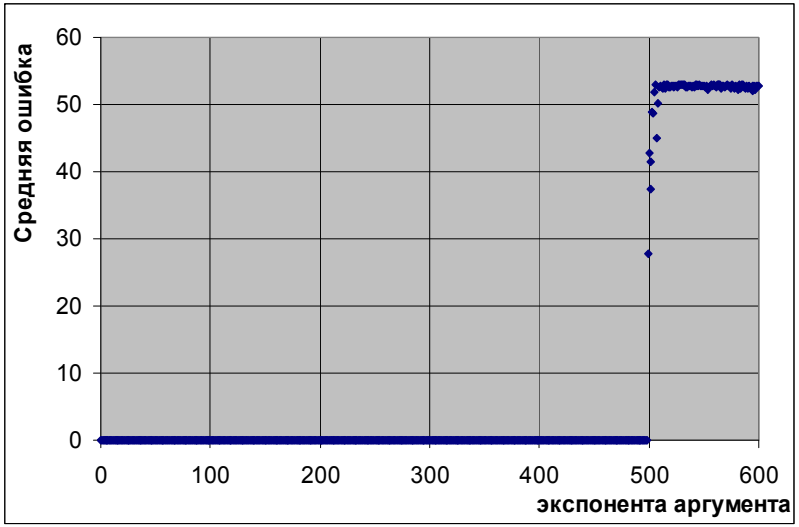

Рис. 2. Средняя величина ошибки (в количестве неправильных знаков результата) при вычислении тангенса в зависимости от экспоненты аргумента (AMD64).

На 32-битных платформах тангенс вычисляется неправильно для чисел, больших 2^{64}, на 64-битных платформах соответствующая граница расположена около 2^{500}.

Для тангенса картина, показанная на Рис. 1 и 2, примерно сохраняется при всех режимах округления. Для косинуса она такая же при режиме округления к ближайшему, но изменяется при других режимах (см. Ри). Здесь видно влияние других факторов, не связанных с редукцией аргумента.

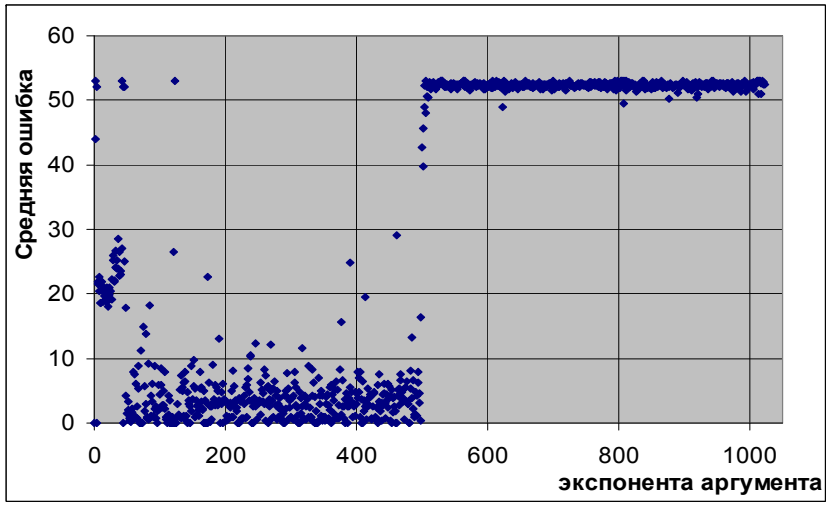

Рис. 3. Средняя величина ошибки (в количестве неправильных знаков результата) при вычислении косинуса в зависимости от экспоненты аргумента (AMD64, режим округления к 0).

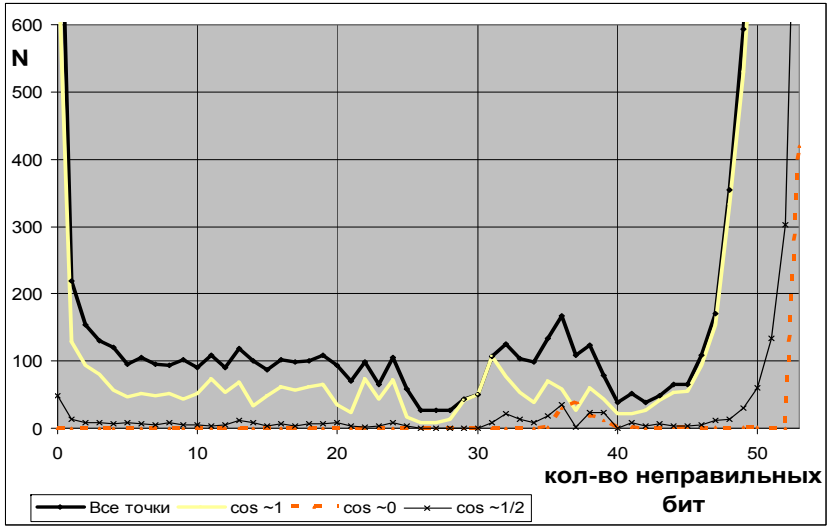

Рис. 4. Распределение величины ошибки для косинуса в точках разных типов (IA32).

Другая особенность ошибок, обнаруженных при тестировании реализаций тангенса и косинуса на 32-битных платформах, связана с некорректным

вычислением значений тангенса в точках, близких к его полюсам или нулям, отличным от 0, а также косинуса в точках, близких к его нулям. В обоих случаях ошибки практически для всех таких точек превышают 35-40 бит (т.е. результаты имеют столько неправильных знаков) — см. Рис. 4. Это также, скорее всего, связано с проблемой корректной редукции аргумента, поскольку для правильного вычисления функций в окрестностях их нулей и полюсов редукция аргумента должна быть более точной.

Однако для 64-битных платформ этой особенности нет — во всех таких точках тангенс и косинус на них вычисляются правильно.

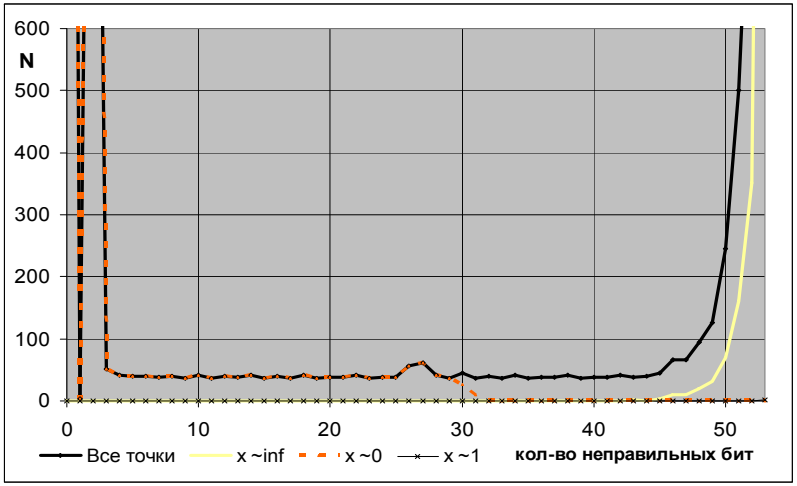

Рис. 5. Распределение величины ошибки для арктангенса (Glibc).

Самая большая неожиданность — значительные ошибки при вычислении арктангенса в реализации библиотеки Glibc на всех платформах. Реализация этой функции от Microsoft работает значительно более корректно — обнаружено лишь 5 небольших ошибок. Полное недоумение вызывают как ошибки на больших по абсолютной величине аргументах (превосходящих 2^{30}), где график арктангенса превращается в горизонтальные прямые, так и ошибки, связанные с нарушением знака результата (знак значения арктангенса всегда должен совпадать со знаком аргумента). Рис. 5 показывает распределение величины ошибки вычисления арктангенса в библиотеке Glibc в зависимости от типа тестовых значений.

5. Заключение

В данной статье представлены результаты работы по созданию тестового набора на основе метода, предложенного в [1] для тестирования реализаций 4-х математических функций — тангенса, арктангенса, косинуса и арккосинуса.

Использованный метод продемонстрировал хорошие результаты, позволив при помощи небольшого числа тестов обнаружить серьезные ошибки в широко используемых реализациях тригонометрических функций.

Большинство найденных ошибок связано с проблемой корректной редукции аргумента тригонометрических функций [19]. Обнаруженные в реализации арктангенса в библиотеке Glibc серьезные ошибки имеют непонятную природу, поскольку возникают в окрестности нуля или на очень больших числах, где арктангенс имеет достаточно хорошие асимптотики, позволяющие вычислять его точно без больших усилий.

Вместе с тем для более серьезной проверки корректности вычисления математических функций необходимо расширить набор тестовых точек, где корректное округление результата функции требует повышенной точности вычислений. В работах группы Arienaire [16,21] для каждой функции приводится лишь несколько таких точек, вычисленных с помощью оптимизированных переборных алгоритмов. Для пополнения тестов такими значениями на систематической основе необходимы техники, которые позволят вычислять их гораздо эффективнее.

Литература

[1] В. Кулямин. Формальные подходы к тестированию математических функций. Труды ИСП РАН, 10:69–114, 2006.
[2] IEEE 754-1985. IEEE Standard for Binary Floating-Point Arithmetic. NY: IEEE, 1985.
[3] IEEE 854-1987. IEEE Standard for Radix-Independent Floating-Point Arithmetic. NY: IEEE, 1987.
[4] ISO/IEC 9899:1999. Programming Languages — C. Geneve: ISO, 1999.
[5] IEEE 1003.1-2004. Information Technology — Portable Operating System Interface (POSIX). NY: IEEE, 2004.
[6] G. Hanrot, V. Lefevre, J.-M. Muller, N. Revol, and P. Zimmermann. Some Notes for a Proposal for Elementary Function Implementation in Floating-Point Arithmetic. Proc. of Workshop IEEE 754R and Arithmetic Standardization, in ARITH-15, June 2001.
[7] D. Defour, G. Hanrot, V. Lefevre, J.-M. Muller, N. Revol, and P. Zimmermann. Proposal for a standardization of mathematical function implementation in floating-point arithmetic. Numerical Algorithms, 37(1–4):367–375, December 2004.
[8] http://www.inria.fr/recherche/equipes/arenaire.en.html
[9] ISO/IEC 10967-1:1994. Information Technology — Language Independent Arithmetic — Part 1: Integer and Floating Point Arithmetic. Geneve: ISO, 1994.
[10] ISO/IEC 10967-2:2002. Information Technology — Language Independent Arithmetic — Part 2: Elementary Numerical Functions. Geneve: ISO, 2002.
[11] ISO/IEC 10967-3. Information Technology — Language Independent Arithmetic — Part 3: Complex Integer and Floating Arithmetic and Complex Elementary Numerical Functions. Draft. Geneve: ISO, 2002.
[12] http://www.math.utah.edu/~beebe/software/ieee/
[13] http://www.math.utah.edu/pub/elefunt/
[14] http://www.netlib.org/fp/ucbtest.tgz

[15] Z. A. Liu. Berkeley Elementary Function Test Suite. M.S. thesis, Computer Science Division, Dept. of Electrical Engineering and Computer Science, University of California at Berkeley, December 1987.
[16] D. Stehle, V. Lefevre, P. Zimmermann. Searching Worst Cases of a One-Variable Function Using Lattice Reduction. IEEE Transactions on Computers, 54(3):340–346, March 2005.
[17] http://www.loria.fr/~zimmerma/mpcheck/
[18] W. Kahan. Minimizing q*m–n. 1983. Неопубликованные заметки, доступны по http://http.cs.berkeley.edu/~wkahan/testpi/nearpi.c.
[19] K. C. Ng. Arguments Reduction for Huge Arguments: Good to the Last Bit. 1992. Доступна как http://www.validlab.com/arg.pdf.
[20] А. Я. Хинчин. Цепные дроби. М: Наука, 1978.
[21] V. Lefevre, J.-M. Muller. Worst Cases for Correct Rounding of the Elementary Functions in Double Precision. Proc. of 15-th IEEE Symposium on Computer Arithmetic, Vail, Colorado, USA, June 2001.