УДК 004.921 **DOI: 10.31799/2077-5687-2023-4-64-77**

ВКЛАД В АЛГОРИТМ ОБРАТНОГО КВАДРАТНОГО КОРНЯ QUAKE

Г.Ф.Э. Док Цихон
Санкт-Петербургский государственный университет аэрокосмического приборостроения

В статье используется методология Булирша/Стоера для численной математики на основе известного алгоритма извлечения обратного квадратного корня, который стал широко известен благодаря его использованию в компьютерной игре Quake. Приводится математическое описание, как относительная погрешность увеличивается на каждой итерации Ньютона и как должно быть построено начальное значение, чтобы минимизировать относительную погрешность после произвольного числа итераций. Этот алгоритм подходит для обучения, потому что он короткий, затрагивает многие важные аспекты арифметики с плавающей запятой и имеет интересные применения.

Ключевые слова: компьютерная графика, Quake, обратный квадратный корень.

Для цитирования:
Док Цихон, Г. Ф. Э. Вклад в алгоритм обратного квадратного корня quake / Г. Ф. Э. Док Цихон // Системный анализ и логистика. – 2023. – № 4(38). – с. 64 – 77. DOI: 10.31799/2077-5687-2023-4-64-77.

CONTRIBUTIONS ON QUAKE INVERSESQUARE ROOT

G. Cichon
St. Petersburg State University of Aerospace Instrumentation

This paper uses the methodology of Bulirsch/Stoer for numerical mathematics on the well-known inverse square root algorithm that became well-known with its use in the Quake computer game. We are going to describe mathematically, how the relative error propgates through each Newton iteration, and how the initial value has to be constructed, as to minimize the relative error after an arbitrary number of iterations. This algorithm is suitable for teaching because it is short, touches many important aspects of floating point arithmetics, and it has interesting applications.

Keywords: computer graphics, Quake, inverse square root.

For citation:
Cichon, G. Contributions on quake inversesquare root / G. Cichon // System analysis and logistics. – 2023. – № 4(38). – p. 64 – 77. DOI: 10.31799/2077-5687-2023-4-64-77.

Введение

В [1] упоминается трюк с плавающей запятой для быстрого вычисления обратного квадратного корня. Этот алгоритм стал широко известен, когда его использовал Джон Кармак при реализации компьютерной игры Quake. До сих пор этот простой алгоритм ненамного хуже, чем оптимизированные аппаратные реализации [2].

В данной статье применим методологию численной математики, представленную в [3].

Алгоритм очень прост и его легко реализовать аппаратно: требуется всего одно вычитание целого числа из «магической» константы. А затем - три умножения и одно сложение на каждую итерацию Ньютона.

Об этом алгоритме много написано в сети Интернет. Алгоритм привлекателен своей простотой и применимостью в компьютерных играх.

В компьютерной игре Quake, использует одно специфичное значение для определения начального значения величины. В предыдущей работе была проанализирована эта константа и показано, что она не приводит к минимальной ошибке начального значения. Было описано, что наилучшая относительная погрешность при начальном значении не приводит к наилучшей относительной погрешности после одной итерации Ньютона, и как это используется алгоритмом Quake. Кроме того, описан метод уточнения магической константы.

В этой статье применим методологию численной математики, описанную в [3], чтобы обосновать значение магической константы. В частности, будет математически описано, как при каждой итерации Ньютона увеличивается относительная погрешность, и как должно быть

построено начальное значение, чтобы минимизировать относительную погрешность после произвольного числа итераций.

Этот алгоритм подходит для изучения студентами, потому что он небольшой и затрагивает многие важные аспекты арифметики с плавающей точкой и имеет интересные применения.

Ошибка

[3] дает представление о представлении чисел в цифровых системах. Такое представление относится к 1950-м годам и разработано для работы в очень ограниченных аппаратных ресурсах первых компьютеров.

В качестве математической переменной обозначим правильное значение как x. Можно не знать *фактического значения* x. Но даже если его знаем, то, возможно, нельзя представить его с помощью небольшого или даже конечного числа бит. Здесь x просто означает математическое понятие.

На компьютере с конечным числом бит, при некоторых ограниченных ресурсах мы можем показать только *представленное значение* x'. В случае применения встроенных систем ресурсы могут быть сильно ограничены.

Абсолютная ошибка Δx определяется как разница между фактическим и представленным значением: $\Delta x = |x' - x|$. Это позволяет определить абсолютное значение разницы величин. Если нас интересует знак этой разницы, то мы можем не учитывать абсолютное значение этой величины. В этом случае мы вычитаем *фактическое* значение из *представленного* значения.

Относительная ошибка ε_x учитывает, что для большего числа допустимой является большая ошибка. Она определяется как абсолютная погрешность, деленная на фактическое значение $\varepsilon_x = \frac{\Delta x}{|x|} = \left|\frac{x\prime - x}{x}\right|$. Пожалуйста, обратите внимание, что относительная погрешность относится к возможному неизвестному или непредставимому фактическому значению, а не к представленному значению. Это важно, если фактическое значение очень мало, т.е. близко к нулю.

Вещественные значения в стандартах IEEE

В компьютерах вещественные числа могут быть представлены либо в виде числа с фиксированной точкой, либо в виде числа с плавающей точкой. Эти представления обладают очень разными свойствами в отношении абсолютной и относительной погрешности, а также в накоплении ошибок при выполнении арифметических операций, таких как сложение, вычитание, умножение и деление. Более подробно это описано в [4] и [3].

В данной статье мы кратко рассмотрим представление чисел с плавающей точкой, в частности, представление чисел с плавающей точкой одинарной точности. Числа с плавающей точкой двойной точности очень на них похожи, но с большим количеством битов и разными константами, и, таким образом, обеспечивают большую точность.

Одинарная точность

Число с одинарной точностью, согласно стандарту IEEE, представлено в виде 32 бит. Наиболее значимым битом является знаковый бит.

Затем следует 8 бит показателя степени (экспоненты), который является числом со знаком, но он не хранится в обычном представлении в дополнительном коде, как другие числа (например, байты). Вместо этого в 8 битах всегда хранится положительное число без знака, а к представленному показателю степени добавляется так называемый показатель *смещения* степени. 8 бит могут представлять числа от 0 до 255. Показатель степени 0 зарезервирован для очень малых (денормализованных) чисел и нуля. А показатель степени 255 зарезервирован для бесконечности и недопустимого числа (NaN, не число). Детали этих особых случаев в данной статье не рассматриваются.

Смещение показателя степени составляет не 128, как можно было бы ожидать для 8-битного числа со знаком, а 127. Это тщательно разработано и реализовано, так что переполнения и опустошения идеально сбалансированы, а диапазон значения показателя степени используется наилучшим образом. Детали этого балансирования также не рассматриваются в данной статье.

И, наконец, младшие 23 бита являются показателем степени. Показатель степени имеет точность 24 бита. Однако, показатель степени нормализован, и старший бит не хранится.

1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0										
S	E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M

Таким образом, значение, заданное вещественным числом с плавающей точкой, может быть задано по следующей формуле (1):

$$\begin{aligned} x = (-1)^S \cdot (1.0 + M_{22} \cdot 2^{-1} + M_{21} \cdot 2^{-2} + \cdots + M_1 \cdot 2^{-22} + M_0 \cdot 2^{-23}) \cdot 2^{E-127} \\ = (-1)^S \cdot (1.0 + M \cdot 2^{-23}) \cdot 2^{E-127} \\ = (-1)^S \cdot (2^{23} + M) \cdot 2^{-23} \cdot 2^{E-127} \\ = (-1)^S \cdot (2^{23} + M) \cdot 2^{E-150} \end{aligned} \quad (1)$$

Здесь используются заглавные буквы для величин, которые представлены в виде отдельных битов. Существует знаковый бит S, смещенный показатель степени E, состоящий из 8 отдельных битов от E_7 до E_0, и мантисса M, без ведущего бита, состоящая из 23 отдельных битов M_{22} до M_0. И можно положить, что 24-й бит $M_{23} = 1$, но он не хранится в представлении числа по стандарту IEEE.

Представление показателя степени в виде смещенного беззнакового целого числа вместо представления в дополнительном коде, обладает еще одним свойством, на которое в значительной степени опирается алгоритм Quake обратного квадратного корня, который является темой этой статьи: любые положительные числа с плавающей точкой можно сравнивать друг с другом, поскольку они представляются целыми числами. Изначально это было разработано для того, чтобы сделать операцию сравнения чисел с плавающей точкой очень простой операцией. Гораздо проще, чем сложение чисел плавающей точкой, которое требует операции полной нормализации в конце вычисления. Это достигается за счет того, что числа с плавающей точкой становятся непрерывными, если рассматривать их как целые числа. Как будет показано далее, алгоритм Quake обратного квадратного корня в значительной степени зависит от этого свойства.

Теперь, после того как задано представление чисел с плавающей точкой, давайте обратим внимание на то, как можно вычислять нетривиальные функции.

Итерация Ньютона

Итерация Ньютона – это метод определения нуля нелинейной функции. Он может быть применен для вычисления всех типов значений при условии, что может быть задана непрерывная дифференцируемая функция $f(x)$, такая, что функция имеет ноль $f(a) = 0$ при значении a, которое надо вычислить. Затем значение a может быть последовательно аппроксимировано следующей формулой (2):

$$x_{n+1} = x_n - \frac{f(x_n)}{f\prime(x_n)} \quad (2)$$

Итерацию Ньютона можно использовать для вычисления деления, квадратного корня, обратного квадратного корня, и многих других.

Итерация Ньютона обладает свойством квадратичной сходимости. Это означает, что количество допустимых цифр удваивается с каждой итерацией.

Это хорошо, если значение *x* уже содержит любое количество допустимых цифр. С другой стороны, поиск хорошего начального значения x_0 является сложной задачей для итерации Ньютона. Начальное значение часто определяется с помощью справочной таблицы.

Алгоритм Quake можно рассматривать как таблицу поиска всего с одной записью. Но мы увидим, что на самом деле это не так. Это скорее линейная интерполяция, содержащая три различных линейных сегмента (см. рис. 1).

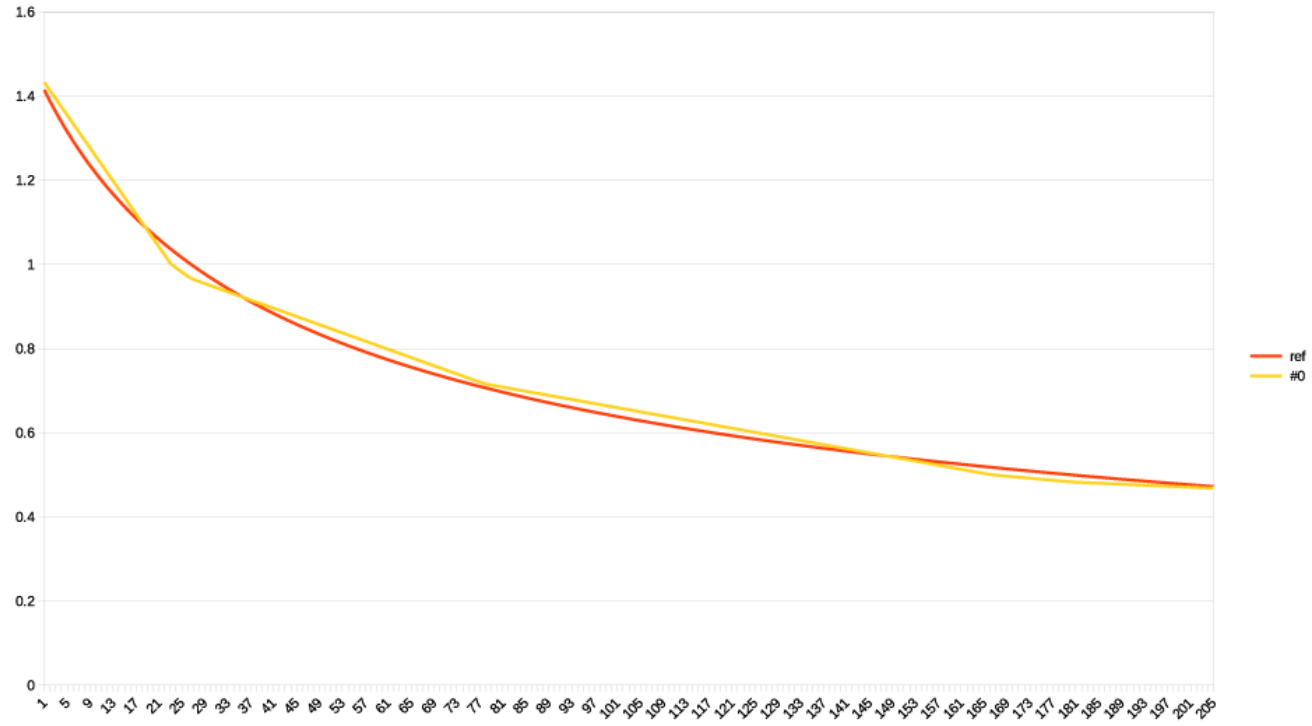

Рис. 1. Интерполяция линейными сегментами

Давайте сначала рассмотрим три наиболее распространенных применения итерации Ньютона в арифметике с плавающей запятой:

Пример: Деление

Наиболее элементарной неатомарной арифметической операцией является деление. Деление может быть реализовано напрямую, с использованием восстанавливающего или невосстанавливающего деления. Либо это может быть реализовано с использованием итерации Ньютона.

Для вычисления деления можно представить нашу функцию *f* как

$$f(x) = \frac{1}{x} - a \tag{3}$$

Эта функция имеет нулевое значение при $f(\frac{1}{a}) = 0$

Первая производная от *f* равна:

$$f'(x) = -\frac{1}{x^2} \tag{4}$$

Это приводит к следующему итерационному алгоритму:

$$x_{n+1} = x_n - \frac{\frac{1}{x_n} - a}{-\frac{1}{x_n^2}} = x_n \cdot (2 - a \cdot x_n) \quad (5)$$

Этот алгоритм требует 2 умножения и одно сложение за итерацию. Это довольно много для простого деления. Рассмотрим, например, итерацию Ньютона для деления, которая создает начальное значение с использованием поисковой таблицы с точностью до 6 бит. Затем выполняем первую итерацию, что дает точность в 12 бит. И далее – вторая итерация, дающая 24 бита точности. Это означает, что в общей сложности используется 4 умножения и 2 сложения для операции деления с плавающей точкой с одинарной точностью.

Пример: квадратный корень

Следующим применением итерации Ньютона является вычисление квадратного корня. Для этого вычисления можно задать функцию *f* в виде:

$$f(x) = 1 - \frac{a}{x^2} \quad (6)$$

Эта функция имеет нулевое значение в $f(\sqrt{a}) = 0$.

Первая производная этой функции имеет вид:

$$f'(x) = \frac{2a}{x^3} \quad (7)$$

Это приводит к следующему итерационному алгоритму:

$$x_{n+1} = x_n - \frac{1 - \frac{a}{x_n^2}}{\frac{2a}{x_n^3}} = \frac{1}{2} x_n \left(3 - \frac{1}{a} \cdot x_n^2\right) \quad (8)$$

Этот алгоритм требует 3 умножения и одно сложение за итерацию. Кроме того, обратите внимание, что алгоритм требует обратного значения величины *a*. Это означает, что для того, чтобы использовать этот алгоритм, сначала требуется вычислить значение, обратное *a*, возможно, с помощью приведенной выше итерации Ньютона (5), прежде чем можно будет вычислить квадратный корень.

Пример: Обратный квадратный корень

В качестве третьего применения итерации Ньютона рассмотрим обратный квадратный корень. В качестве функции *f* можно выбрать следующую функцию:

$$f(x) = \frac{1}{x^2} - a \quad (9)$$

Эта функция имеет нулевое значение $f(\frac{1}{\sqrt{a}}) = 0$

Первая производная *f* равна:

$$f'(x) = -\frac{2}{x^3} \quad (10)$$

Это дает следующий итерационный алгоритм:

$$x_{n+1} = x_n - \frac{\frac{1}{x_n^2} - a}{-\frac{2}{x_n^3}} = \frac{1}{2} x_n (3 - a \cdot x_n^2) \quad (11)$$

Как и алгоритм извлечения квадратного корня, описанный в предыдущем параграфе, обратный квадратный корень также требует 3 умножения и 1 сложения за итерацию. В отличие от алгоритма извлечения квадратного корня, он не требует наличия обратного значения величины a. Поэтому, для этой итерации не требуется никакого деления.

Обратный квадратный корень применяется в компьютерных играх для нормализации векторов нормалей (они представляют собой нормали поверхностей). Именно так этот алгоритм применяется в компьютерной игре Quake.

Кроме того, обратный квадратный корень также можно использовать для вычисления обычного квадратного корня: просто умножьте результат $\frac{1}{\sqrt{a}}$ на величину a и получите $\sqrt{a}$. Таким образом можно избежать операции деления.

И, наконец, деление может быть вычислено с помощью обратного квадратного корня путем возведения результата в квадрат $\frac{1}{\sqrt{a}} \cdot \frac{1}{\sqrt{a}} = \frac{1}{a}$. Вот лишь несколько интересных применений алгоритма вычисления обратного квадратного корня.

Квадратичная сходимость

Как уже упоминалось, положительным свойством итерации Ньютона является квадратичная сходимость. Давайте математически рассмотрим это свойство.

Функция f имеет нулевое значение при искомом значении a. Согласно теореме о среднем значении, это значение функции может быть преобразовано в ряд Тейлора второго порядка, с центром в текущем значении итерации x, с неизвестным средним значением ξ, которое находится где-то между a и x. Имеем:

$$0 = f(a) = f(x) + f'(x)(a - x) + \frac{1}{2} f''(\xi)(a - x)^2, \xi \in [a, x] \tag{12}$$

Мы можем использовать это определение, чтобы выразить абсолютную погрешность x. Как мы видим, новая абсолютная погрешность после очередной итерации зависит от квадрата текущего значения абсолютной погрешности, а также некоторых значений первой и второй производной функции f.

$$\begin{aligned} x - a &= \frac{f(x)}{f\prime(x)} + \frac{f\prime\prime(\xi)}{2f\prime(x)} \cdot (x - a)^2 \\ \underbrace{x - \frac{f(x)}{f\prime(x)}}_{x_{n+1}} - a &= \frac{f\prime\prime(\xi)}{2f\prime(x)} \cdot (x - a)^2 \end{aligned} \tag{13}$$

Поскольку значения ξ не известно, и обычно необходимо учитывать точность алгоритма в определенном диапазоне величины a, то можно задать диапазон погрешности, указав граничные значения f' и f'' в диапазоне параметра a. В частности, нам нужен максимум второй производной и минимум первой производной.

$$\begin{aligned} M_2 &= \max|f''(\xi)| \\ m_1 &= \min|f'(\xi)| \end{aligned} \tag{14}$$

Используя эту информацию, можно предусмотреть некоторое число обусловленности C, которое зависит только от математических свойств функции f и диапазона ее параметров, но не зависит от фактической реализации, такой как точность представления промежуточных значений или точность арифметических операций.

$$\begin{aligned} x_{n+1} - a &\leq \frac{M_2}{2m_1} \cdot (x_n - a)^2 \\ \Delta_{n+1} &\leq C \cdot \Delta_n^2 \\ \varepsilon_{n+1} &\leq C \cdot a \cdot \varepsilon_n^2 \end{aligned} \tag{15}$$

До сих пор рассматривалась абсолютная погрешность. Для получения относительной погрешности надо разделить ее на фактическое значение. Чтобы бороться с квадратичной операцией при квадратичной сходимости, необходимо умножить относительную погрешность на фактическое значение *a* в формуле распределения относительной погрешности.

Это означает, что относительная погрешность увеличивается на величину значения *a*. Это свойство станет важным позже, при сравнении относительной погрешности исходного значения и относительной погрешности после одной итерации в алгоритме обратного квадратного корня Quake.

Начальное значение

Давайте теперь рассмотрим вычисление обратного квадратного корня. Начинаем со значения *a*, из которого надо извлечь обратный квадратный корень, который представлен в виде вещественного числа с плавающей точкой одинарной точности. Используем *v* (16) для представления значения *a*. Для удобства записи будем использовать строчные буквы *e* для несмещенного показателя степени, и строчные буквы *m* для записи нормализованной мантиссы следующим образом (16):

$$\begin{aligned} v &= 2^e \cdot m = 2^{E-127} \cdot (2^{23} + M) \cdot 2^{-23} \\ e &= E - 127 \\ E &= e + 127 \\ m &= (2^{23} + M) \cdot 2^{-23} \\ M &= (m - 1.0) \cdot 2^{23} \end{aligned} \tag{16}$$

Теперь вычислим обратный квадратный корень из величины *v* (16). Для показателя степени вычисление обратного квадратного корня несложно: надо взять отрицательный показатель степени для обратной величины и разделить показатель степени на два для получения квадратного корня.

Деление на два может быть выполнено путем сдвига числа вправо на один бит.

Кроме того, у нас нет показателя степени в виде числа в дополнительном коде, как это бывает обычно. Поэтому, придется вычесть смещение, чтобы получить представление со знаком, затем разделить на два, а затем снова добавить смещение. Этот процесс можно сократить, если просто вычесть сдвинутый показатель степени из константы, которую далее будем называть *H*.

И этот сдвиг - вычитание из константы - является ядром алгоритма Quake обратного квадратного корня. Его можно выполнить как простую целочисленную операцию, в отличие от операций с плавающей точкой, которые обычно выполняются над числами с плавающей точкой. Однако нам приходится иметь дело с вещественными числами с плавающей точкой стандарта IEEE.

Знаковый бит должен быть равен нулю, потому что можно извлечь квадратный корень только из положительного числа. А сдвинутый знаковый бит создаст нулевой бит в значащем бите показателя степени, как это и должно быть.

В мантиссе ситуация намного сложнее: сдвигаем младший значащий бит показателя степени в старший значащий бит мантиссы. Это потенциально может привести к очень плохому начальному значению, в котором уже второй бит неверен. Но, к счастью, разложение

в ряд Тейлора обратного квадратного корня из 1 дает результат равный $-\frac{1}{2}$ в коэффициенте первого порядка, поскольку производная равна $f'(1) = -\frac{1}{2}$. Таким образом, сдвиг на единицу и отрицание также приводят к правильному результату для мантиссы.

Единственное, что должно волновать, это неявный бит, который не сдвигается и не вычитается, случайный бит показателя степени, который внезапно появляется в мантиссе. Все это работает просто идеально: поскольку смещение показателя степени является нечетным числом в спецификации IEEE, оно просто смещается на ноль или единицу, как и требуется для хорошего начального значения.

Наконец, можно использовать младшие биты константы, из которых мы вычитаем все, чтобы увеличить или уменьшить результирующее начальное значение. Эти биты в дальнейшем будем называть Q.

Когда выполняем вычитание из мантиссы, то можно получить или не получить заимствование в битах мантиссы, в зависимости от того, будет ли результат вычитания в мантиссе отрицательным или положительным. В дальнейшем будем называть этот потенциально отрицательный результат вычитания – b. А знаковый бит этого результата обозначим как B – это будет флаг заимствования из мантиссы в показатель степени.

В итоге получим три линейных сегмента отрезка: один от *1* до *2*, второй от *2* до $2q$ и третий от $2q$ до *4*. Наклон этих сегментов определяется операцией сдвига и полярностью смещения показателя степени.

И затем можно перемещать эту аппроксимационную кривую, состоящую из трех отрезков, вверх и вниз, используя константу q, чтобы получить максимально возможную точность начального значения.

Еще одним удачным совпадением является то, что представление смещения показателя степени и свойство целочисленного сравнения чисел с плавающей точкой одинарной точности стандарта IEEE, как описано выше, гарантирует, что в функции не будет резких скачков при выполнении сдвига и вычитания, а просто перемещение младших разрядов показателя степени в старшие разряды мантиссы. Это является следствием оригинального представления формата чисел с плавающей точкой, разработанного IEEE.

Итак, обозначим:

b - заимствование мантиссы в показатель степени
E_0 - последний бит показателя степени переводится в мантиссу
H - константа для показателя степени
Q - константа для мантиссы

Используя эти обозначения, запишем расчет начального значения следующим образом (17):

$$\begin{aligned} E' &= H - E \gg 1 - B \\ M' &= B \cdot 2^{23} + Q - E_0 \cdot 2^{22} - M \gg 1 \\ b &= Q - E_0 \cdot 2^{22} - M \gg 1 \\ B &= b < 0 \\ q &= Q \cdot 2^{-23} \end{aligned} \tag{17}$$

Для заимствования мантиссы в показатель степени рассмотрим следующие неравенства:

$$\begin{aligned} Q - E_0 \cdot 2^{22} - M \gg 1 &< 0 \\ Q &< E_0 \cdot 2^{22} + M \gg 1 \end{aligned} \tag{18}$$

Заимствование меняется только один раз, в зависимости от Q_{22}. В условиях алгоритма Quake $Q_{22} = 0$. Поэтому, флаг заимствования меняется только в случае $E_0 = 0$ при $M \gg 1 = Q$. Если бы $Q_{22} = 1$, то, как правило, флаг заимствования сменился бы на $E_0 = 1$ при $M \gg 1 = Q - 2^{22}$.

Для показателя степени можно установить следующую связь между результатом сдвига и вычитания из H, и искомым результатом для обратного квадратного корня:

$$\begin{aligned} e' &= E' - 127 \\ e' &= H - E \gg 1 - B - 127 \\ e' &= H - \tfrac{e+127-E_0}{2} - B - 127 \\ e' &= H - \tfrac{e}{2} - \tfrac{127}{2} + \tfrac{E_0}{2} - B - 127 \\ e' &= -\tfrac{e}{2} + H - \tfrac{3}{2} \cdot 127 + \tfrac{E_0}{2} - B \\ &\quad \underbrace{\phantom{-\tfrac{e}{2} + H - \tfrac{3}{2} \cdot 127 + \tfrac{E_0}{2}}}_{\approx 0} \end{aligned} \tag{19}$$

Для мантиссы получим следующее соотношение:

$$\begin{aligned} m' &= 1.0 + M' \cdot 2^{-23} \\ m' &= 1.0 + (B \cdot 2^{23} + Q - E_0 \cdot 2^{22} - M \gg 1) \cdot 2^{-23} \\ m' &= 1.0 + B - \tfrac{E_0}{2} + \left(Q - \tfrac{M - M_0}{2}\right) \cdot 2^{-23} \\ m' &= 1.0 + B - \tfrac{E_0}{2} + \left(Q - (m - 1.0) \cdot 2^{22} + \tfrac{M_0}{2}\right) \cdot 2^{-23} \\ m' &= \left(1.5 + B - \tfrac{E_0}{2} + q\right) - \tfrac{1}{2} \cdot m - M_0 \cdot 2^{-24} \end{aligned} \tag{20}$$

Собрав всё вместе, получим следующую формулу для начального значения:

$$\begin{aligned} v' &= 2^{e'} \cdot m' \\ v' &= 2^{-\frac{e}{2} + H - \frac{3}{2} \cdot 127 + \frac{E_0}{2} - B} \cdot \left(1.5 + B - \tfrac{E_0}{2} + q - \tfrac{1}{2} \cdot m - M_0 \cdot 2^{-24}\right) \\ v' &= 2^{-\frac{e}{2}} \cdot 2^{H - \frac{3}{2} \cdot 127 + \frac{E_0}{2} - B} \cdot \left(\left(1.5 + B - \tfrac{E_0}{2} + q\right) - \tfrac{1}{2} \cdot m - M_0 \cdot 2^{-24}\right) \end{aligned} \tag{21}$$

Вычисление начального значения, а также выбор значений H и Q подробно были описаны в предыдущей работе.

Минимизация относительной ошибки в первоначальной оценке

Также в предыдущей работе была выполнена оценка и минимизация ошибки в первоначальной оценке (см. рис. 2). Весьма примечательно, что благодаря всем этим удачным совпадениям, как описано ранее, максимальная относительная погрешность исходного значения составляет всего около 3,5%, что уже составляет почти 5 бит точности. Как видно, абсолютная ошибка значительно выше при низких значениях a. Но при меньших значениях a результирующее фактическое значение x выше, поэтому относительная ошибка, соответственно, ниже.

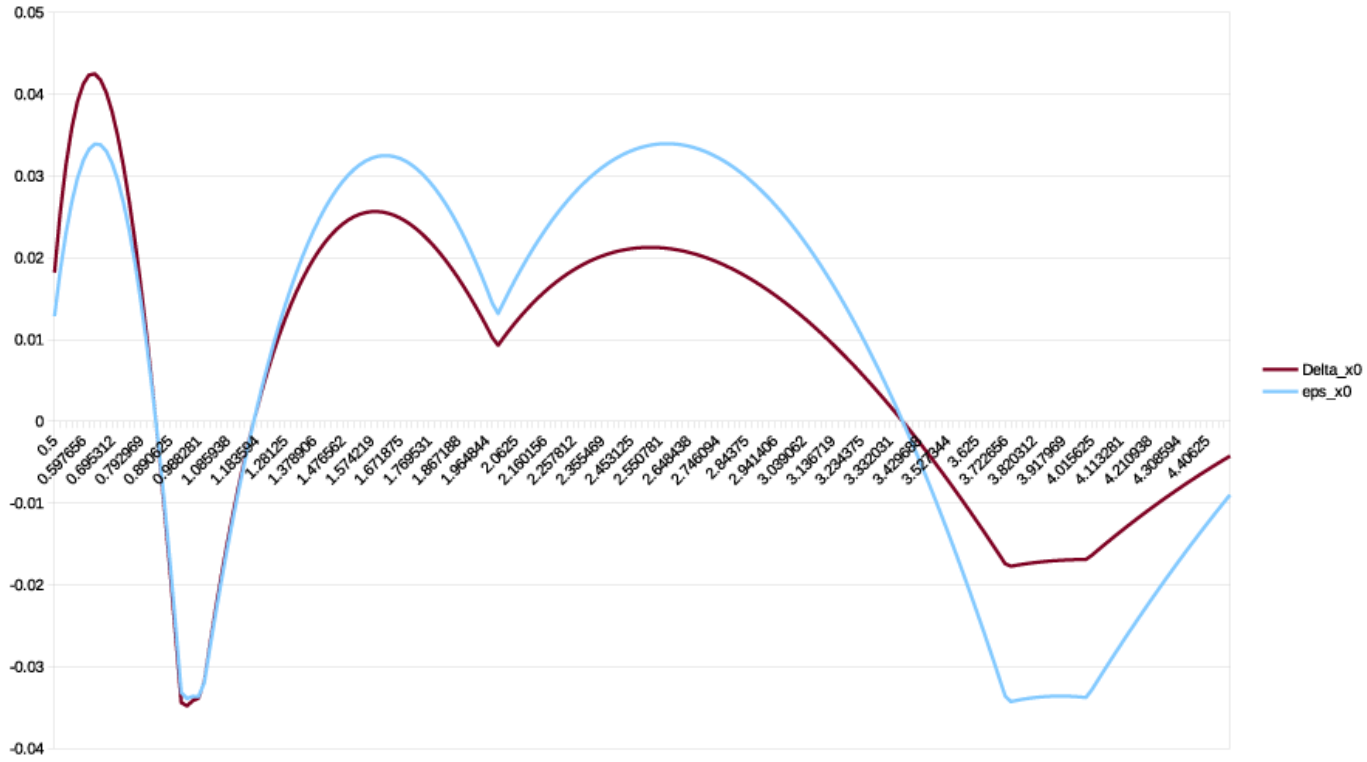

Рис. 2. Оценка и минимизация ошибки в первоначальной оценке

И, как отмечалось в предыдущей работе, константа q не была выбрана так, чтобы обеспечить минимальную относительную ошибку в целом. Видно, что результирующая относительная ошибка немного выше для больших значений a. Будет показано, как происходит это выравнивание после первой итерации Ньютона.

Разложим функцию методом кусочно-линейной аппроксимации. Каждая линейная функция определена на отрезке $[a, b]$, таком что $v' = s \cdot v + t$.

В нашем случае константы определяются так (22):

$$s = -\frac{1}{2} \cdot 2^{-\frac{e}{2}+H-\frac{3}{2}\cdot 127+\frac{E_0}{2}-B}$$

$$t = 2^{-\frac{e}{2}+H-\frac{3}{2}\cdot 127+\frac{E_0}{2}-B} \cdot \left(1.5 + B - \frac{E_0}{2} + q - M_0 \cdot 2^{-24}\right) \quad (22)$$

Обратите внимание, что q – это свободный параметр, который необходимо оптимизировать.

Относительная ошибка ε определяется системой (23) как:

$$\begin{aligned} \varepsilon_0 &= \left|\frac{x_0 - x}{x}\right| \\ \varepsilon_0 &= \left|\frac{s\cdot v + t - \frac{1}{\sqrt{v}}}{\frac{1}{\sqrt{v}}}\right| \\ \varepsilon_0 &= \left|s \cdot v^{\frac{3}{2}} + t \cdot \sqrt{v} - 1\right| \\ \frac{\partial \varepsilon_0}{\partial v} &= \frac{3}{2} s v^{\frac{1}{2}} + \frac{1}{2} t v^{-\frac{1}{2}} \\ \frac{\partial \varepsilon_0}{\partial v} &= \frac{3sv + t}{2\sqrt{v}} \end{aligned} \quad (23)$$

Максимум и минимум этой функции могут находиться в границах a или b, или это может быть локальный экстремум при $\frac{\partial \varepsilon_0}{\partial v} = 0$.

$$\begin{aligned} \frac{\partial \varepsilon_0}{\partial v} &= 0 \\ \frac{3sv_{\updownarrow}+t}{2\sqrt{v_{\updownarrow}}} &= 0 \\ v_{\updownarrow} &= \frac{-t}{3s} \\ \varepsilon_{0,\updownarrow} &= \left|\frac{2}{3\sqrt{-3}} t^{\frac{3}{2}} s^{-\frac{1}{2}}\right| \\ \varepsilon_{0,\leftarrow} &= \left|s \cdot a^{\frac{3}{2}} + t \cdot \sqrt{a} - 1\right| \\ \varepsilon_{0,\rightarrow} &= \left|s \cdot b^{\frac{3}{2}} + t \cdot \sqrt{b} - 1\right| \end{aligned} \tag{24}$$

Это было подробно описано в предыдущей работе.

Обсуждение кривой начального значения

Пусть e – некоторое четное число, например 0. Тогда, соответственно, $E = e + 127$ будет нечетным числом, а значит, бит $E_0 = 1$.

Без необходимости обобщения, положим $H = 190$, как это сделано в константах Quake.

И также положим $Q_{22} = 0$, как в константах Quake. При такой расстановке значений не будет изменений ни в бите E_0, ни в B от e до $e + 1$. Поэтому, положим первый линейный сегмент как $[e, e + 1]$. Это означает, что $a_1 = e$ и $b_1 = e + 1$. И из $E_0 = 1$ и $Q_{22} = 0$ следует, что $B = 1$.

$$\begin{aligned} v' &= 2^{-\frac{e}{2}+H-\frac{3}{2}\cdot 127+\frac{E_0}{2}-B} \cdot \left(1.5 + B - \frac{E_0}{2} + q - \frac{1}{2}\cdot m - M_0 \cdot 2^{-24}\right) \\ v' &= 2^{-\frac{e}{2}+190-\frac{3}{2}\cdot 127+\frac{1}{2}-1} \cdot \left(1.5 + 1 - \frac{1}{2} + q - \frac{1}{2}\cdot m - M_0 \cdot 2^{-24}\right) \\ v' &= 2^{-\frac{e}{2}} \cdot 2^{-1} \cdot \left(2 + q - \frac{1}{2}\cdot m - M_0 \cdot 2^{-24}\right) \\ v' &= 2^{-\frac{e}{2}} \cdot \left(1 + \frac{1}{2}q - \frac{1}{4}m - M_0 \cdot 2^{-25}\right) \end{aligned} \tag{25}$$

Если умножить входное значение на 4, то есть добавим 2 к показателю степени, то в результате берем на единицу больше половины показателя степени, и это означает, что выходное значение делится пополам. Это оставляет относительную ошибку неизменной. Итак, без потерь в обобщении, мы продолжаем с $e = 0$.

$$v' = \left(1 + \frac{1}{2}q - \frac{1}{4}m - M_0 \cdot 2^{-25}\right) \cdot 2^{-\frac{e}{2}} \tag{26}$$

В качестве примера, выполним эти операции для первого из трех сегментов кусочно-линейной аппроксимации, следующим образом. Подберем следующие константы (27):

$$\begin{aligned} a &= 1 \cdot 2^{-\frac{e}{2}} \\ b &= 2 \cdot 2^{-\frac{e}{2}} \\ s_1 &= -\frac{1}{4} \cdot 2^{-\frac{e}{2}} \\ t_1 &= \left(1 + \frac{1}{2}q - M_0 \cdot 2^{-25}\right) \cdot 2^{-\frac{e}{2}} \end{aligned} \tag{27}$$

Это даст следующих кандидатов на точки экстремума:

$$\begin{aligned} v_{1,\updownarrow} &= \frac{-t_1}{3s_1} = \frac{1+\frac{1}{2}q-M_0\cdot 2^{-25}}{3\frac{1}{4}} = \frac{4}{3}+\frac{2}{3}q-\frac{1}{3}M_0\cdot 2^{-23} \\ \varepsilon_{0,\updownarrow} &= \left|\frac{2}{3\sqrt{-3}}t^{\frac{3}{2}}s^{-\frac{1}{2}}\right|\cdot 2^{-\frac{e}{2}} \\ \varepsilon_{0,\leftarrow} &= \left|s\cdot a^{\frac{3}{2}}+t\cdot\sqrt{a}-1\right|\cdot 2^{-\frac{e}{2}} \\ \varepsilon_{0,\rightarrow} &= \left|s\cdot b^{\frac{3}{2}}+t\cdot\sqrt{b}-1\right|\cdot 2^{-\frac{e}{2}} \end{aligned} \tag{28}$$

Как видно из (28), значения не могут быть выражены в закрытой арифметической форме, и предыдущая работа была посвящена их численной обработке и поиску минимальных значений с использованием метода деления пополам [5], [6], [7].

В численных методах деления пополам было отмечено, что постоянная Q с минимальной относительной ошибкой для начальной оценки не дает минимальной относительной ошибки вычислений после первой итерации, как это выполняется в алгоритме Quake.

И был поднят вопрос, почему это так и как вычислить наилучшее возможное значение константы. В предыдущей работе было рассмотрено множество численный методов поиска с применением метода деления пополам и предложено несколько неубедительных альтернатив, все они отличаются от постоянной Quake. В этой статье ситуация анализируется математически, применяя методологию численной математики. В этом и заключается новизна данной статьи.

Передача ошибки после первой итерации Ньютона

Было отмечено, что после первой итерации будут существовать другие константы q, которые дают лучшую минимальную ошибку, чем те, которые приведены в расчете только для начального значения.

Это может быть вызвано различными характеристиками сходимости итерации Ньютона. В этом случае мы можем указать точное условие остатка (29):

$$\begin{aligned} f(x) &= \frac{1}{x^2}-a \\ x &= \frac{1}{\sqrt{a}} \\ f(\tfrac{1}{\sqrt{a}}) &= 0 \\ x_{n+1} &= \tfrac{1}{2}x_n(3-ax_n^2) \\ x_n &= x+\Delta_n \\ x+\Delta_{n+1} &= \tfrac{1}{2}(x+\Delta_n)(3-a(x+\Delta_n)^2) \\ \Delta_{n+1} &= \tfrac{1}{2}(x+\Delta_n)(3-ax^2-2ax\Delta_n-\Delta_n^2)-x \\ &= \tfrac{1}{2}(3x-ax^3-2ax^2\Delta_n-ax\Delta_n^2+ \\ &\quad 3\Delta_n-ax^2\Delta_n-2ax\Delta_n^2-a\Delta_n^3-2x) \\ &= \tfrac{1}{2}(3x-ax^3-2x)+\tfrac{1}{2}(3-2ax^2-ax^2)\Delta_n \\ &\quad -\tfrac{1}{2}(ax+2ax)\Delta_n^2-\tfrac{1}{2}a\Delta_n^3 \\ &= \tfrac{1}{2}(x-ax^3)+\tfrac{1}{2}\Delta_n(3-3ax^2)-\tfrac{1}{2}\Delta_n^{23}ax-\tfrac{1}{2}\Delta_n^3a \\ &= \underbrace{\tfrac{1}{2}x(1-ax^2)}_{=0}+\tfrac{3}{2}\Delta_n\underbrace{(1-ax^2)}_{=0}-\tfrac{3}{2}\Delta_n^2ax-\tfrac{1}{2}\Delta_n^3a \\ &= -\tfrac{3}{2}\sqrt{a}\Delta_n^2-\tfrac{1}{2}a\Delta_n^3 \\ &= -\tfrac{1}{2}\sqrt{a}\Delta_n^2(3+\sqrt{a}\Delta_n) \end{aligned} \tag{29}$$

Обратите внимание, что если Δ_n мала, то вторым членом можно пренебречь. И данную последовательность можно аппроксимировать выражением $\Delta_{n+1} \approx -\frac{3}{2}\sqrt{a}\Delta_n^2$. Обратите внимание, что после первого приближения, значение Δ всегда отрицательно. Итерация Ньютона функцию обратного квадратного корня всегда аппроксимирует снизу.

На рисунке 3 показаны абсолютная и относительная ошибки после одной итерации Ньютона. Как описано ранее, относительная ошибка исходного значения уменьшается еще больше при больших значениях a. Таким образом, максимумы относительной погрешности теперь сравнялись. Максимальная относительная ошибка составляет около 0,17%, что соответствует примерно 9 битам точности. Для плавности световых эффектов на нормальных векторах в игре Quake этой точности достаточно. Чтобы получить 16 бит точности, потребуется еще одна итерация Ньютона, а для полной одинарной точности вещественных чисел стандарта IEEE - потребуется еще одна итерация. (Обратите внимание, что для чисел двойной точности понадобится всего лишь еще одна, четвертая итерация Ньютона)

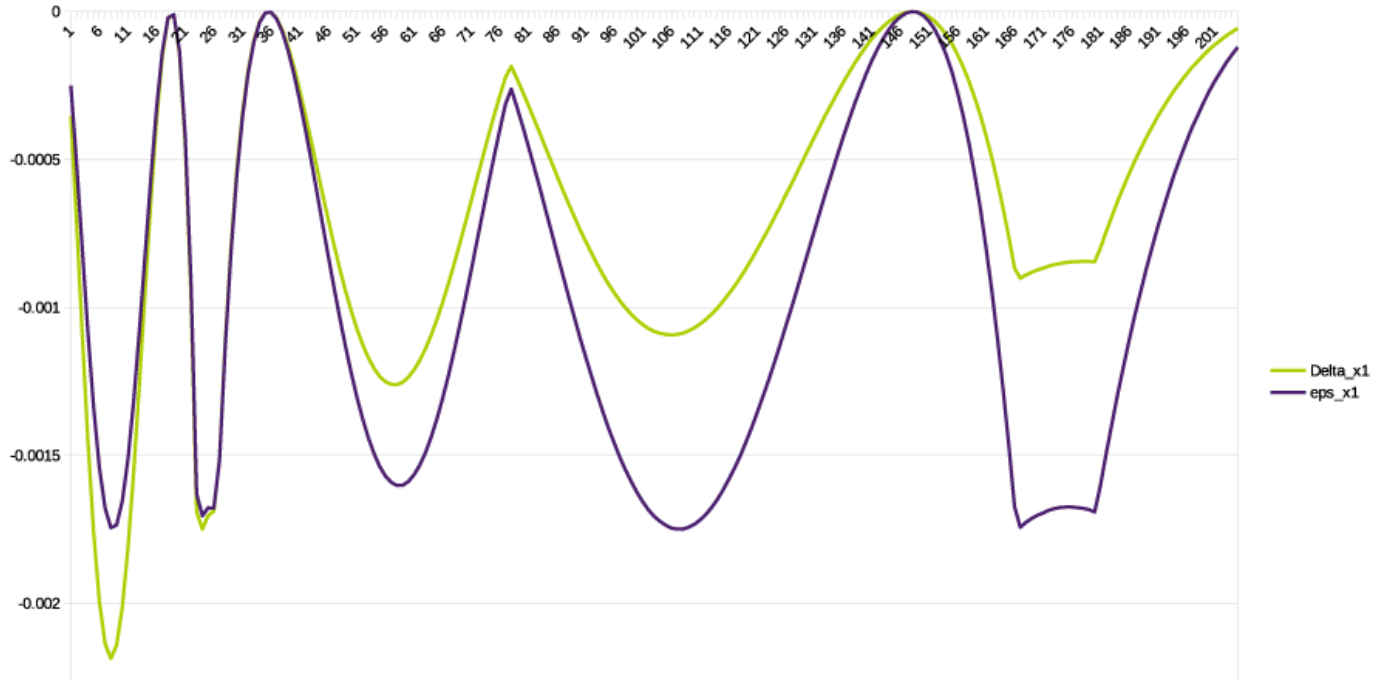

Рис. 3. Абсолютная и относительная ошибка после одной итерации Ньютона

Чтобы получить еще один бит точности, что означает сокращение Δ пополам, вы можете добавить половину Δ к результату вычисления. Таким образом, Δ также может стать положительным, и абсолютное значение Δ выше и ниже истинного значения может быть уменьшено вдвое. Однако следует отметить, что эта мера снижает точность предыдущего значения, которое уже является более точным. Тогда невозможно будет быстрее добиться большей точности.

Заключение

Применена методология численной математики, приведенная в [3], к магической постоянной алгоритма Quake. Методически обоснованы эмпирические результаты.

В дальнейшем проведем более тщательное исследование вычислений, используя программное обеспечение для символьной алгебры и применяя программное обеспечение для автоматизированного доказательства теорем.

Также планируется исследовать передачу ошибок для второй и последующих итераций Ньютона.

СПИСОК ЛИТЕРАТУРЫ

1. Blinn J. Floating-point tricks / J. Blinn // IEEE Computer Graphics and Applications. – 1997. – 17(4). – P. 80–84.
2. E. Ruskin. Timing square root [Электронный ресурс] // assemblyrequired.crashworks.org. – 2009. – URL: http://assemblyrequired. crashworks.org/timing-square-root/ (дата обращения 08.12.2023).
3. Stoer J. Introduction to Numerical Analysis. / J. Stoer and R. Bulirsch. // Texts in Applied Mathematics. Springer. – New York, 2002. – 672 p.
4. Goldberg D. What every computer scientist should know about floating-point arithmetic / D. Goldberg // ACM Comput. Surv. – 1991. – 23(1). – P. 5–48.
5. Lomont C. Fast inverse square root [Электронный ресурс] // Tech-315 nical Report. 2003. Vol. 32. – URL: http://www.matrix67.com/data/InvSqrt.pdf, Feb 2003. (дата обращения 08.12.2023).
6. C. McEniry. The mathematics behind the fast quare root function code [Электронный ресурс] // MathematicsBehind. 2007. – URL: https://0x5f37642f.com/documents/ McEniryMathematicsBehind.pdf, Aug 2007 (дата обращения 08.12.2023).
7. B. Self. Efficiently computing the inverse square root using integer operations. [Электронный ресурс] // sites.math.washington.edu – 2012. – URL: https://sites.math.washington.edu/˜morrow/336_12/papers/ben.pdf. (дата обращения 08.12.2023).

ИНФОРМАЦИЯ ОБ АВТОРЕ

Док Цихон Гордон Франц Эмануэль –
Доцент кафедры аэрокосмических компьютерных и программных систем, кандидат технических наук
Санкт-Петербургский государственный университет аэрокосмического приборостроения
190000, Санкт-Петербург, ул. Большая Морская, д. 67, лит. А
E-mail: gordon.cichon@guap.ru

INFORMATION ABOUT THE AUTHOR

Gordon Cichon –
PhD, Associate Professor of the Department of Aerospace Computer and Program Systems
Saint-Petersburg State University of Aerospace Instrumentation
67, Bolshaya Morskaia str., Saint-Petersburg, 190000, Russia
E-mail: gordon.cichon@guap.ru