EcoLab1

Вычисление кубического корня числа (функция cbrt)

Алгоритмы:

В ходе данного исследования оказалось, что cамыми популярными и используемыми на практике способами вычисления кубического корня являются метод Ньютона и метод Половинного деления.

Проект состоит из компонента, содержащего по три метода для каждого алгоритма (double, float, long double)

**Метод Ньютона(метод касательных):**

Начиная с неоторого начального приближения *x*0 уточняет приближение к корню.

Формула итерации:



*Псевдокод для Метода Ньютона:*

Функция cubeRoot\_newton(x, epsilon) :

x0 = x

Пока True :

x1 = (2 \* x0 + x / (x0 \* x0)) / 3

Если | x1 - x0 | < epsilon, то

Прервать цикл

x0 = x1

Вернуть x1

Сложность *O*(log(1/*ϵ*​)), где *ϵ*​ - требуемая точность.

**Метод половинного деления:**

Работает на промежутке, на котором функция изменяет знак.Выбирает середину промежутка и определяет новый промежуток, в котором корень находится: процесс повторяется до достижения требуемой точности.

Метод гарантированно сходится, но его скорость сходимости медленнее, чем у метода Ньютона.

*Псевдокод для Метода Половинного деления:*

Функция CbrtHalfDivision(x, epsilon) :

low = 0.0

high = x

guess = (low + high) / 2.0

Пока | guess \* guess \* guess - x| >= epsilon :

Если guess \* guess \* guess > x, то

high = guess

Иначе :

low = guess

guess = (low + high) / 2.0

Вернуть guess

Сложность *O*(log(*n*​)), где n количество итераций, необходимых для достижения заданной точности(зависит от разницы high – low).

Оба алгоритма имеют похожие алгоритмические сложности, которые предстоит проверить опытным путем, чтобы понять где и какой алгоритм лучше всего использовать.

Тесты:

Из псевдокода можно догадаться, что оба алгоритма зависят от двух переменных x- числа, которое мы будем извлекать и epsilon – точность вычислений. Для того, чтобы убедиться в этом, давайте сравним их работу при условии x=const и постепенно уменьшающийся epsilon.

График Double

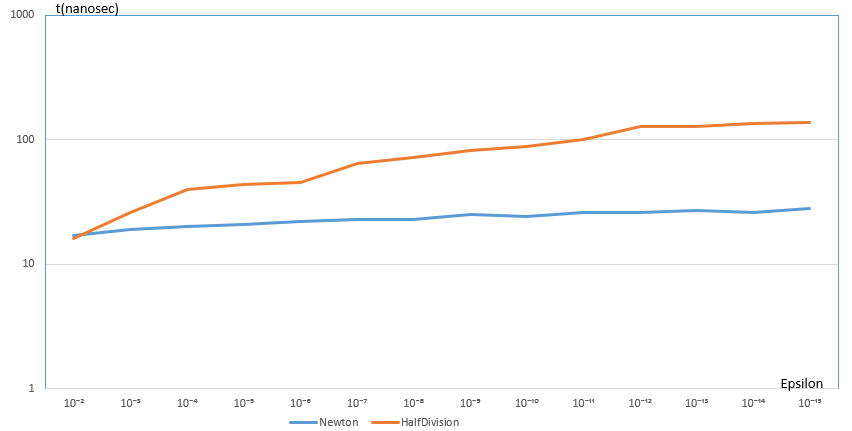


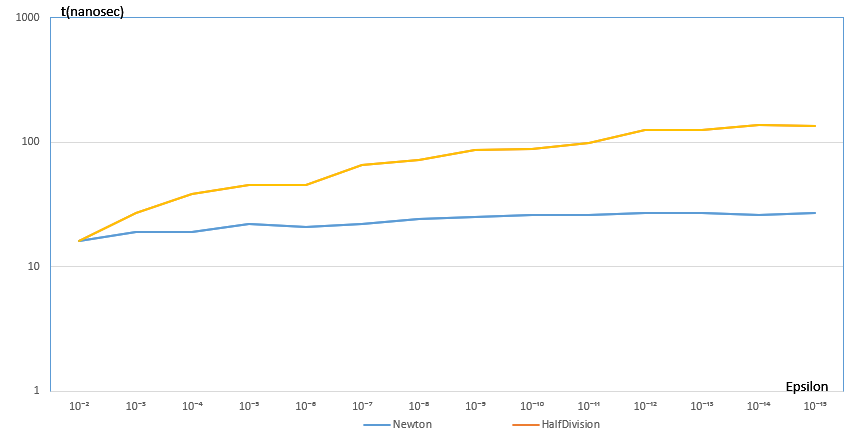
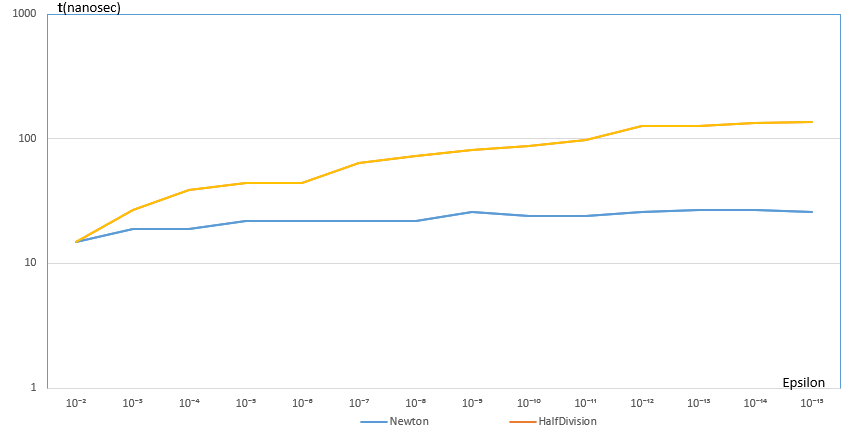
График Float

График Long Double

Графы для всех случаев ведут себя примерно одинаково. С уменьшением epsilon уменьшается и скорость работы алгоритмов. Но это и не удивительно, ведь увеличивается точность знаков при извлечении кубических корней.

Еще на графе можно заметить, что при низкой точности метод Половинного деления работает быстрее, однако, чем точнее нужен результат, тем больше времени тратится на обработку (зато алгоритм дает высокую точность).

Теперь попробуем, наоборот, epsilon=const чтобы проверить как размер числа влияет на скорость работы. Для данного теста я также использовал стандартную библиотеку для извлечения кубических корней (cbrt и её аналоги - cbrtf, cbrtl)

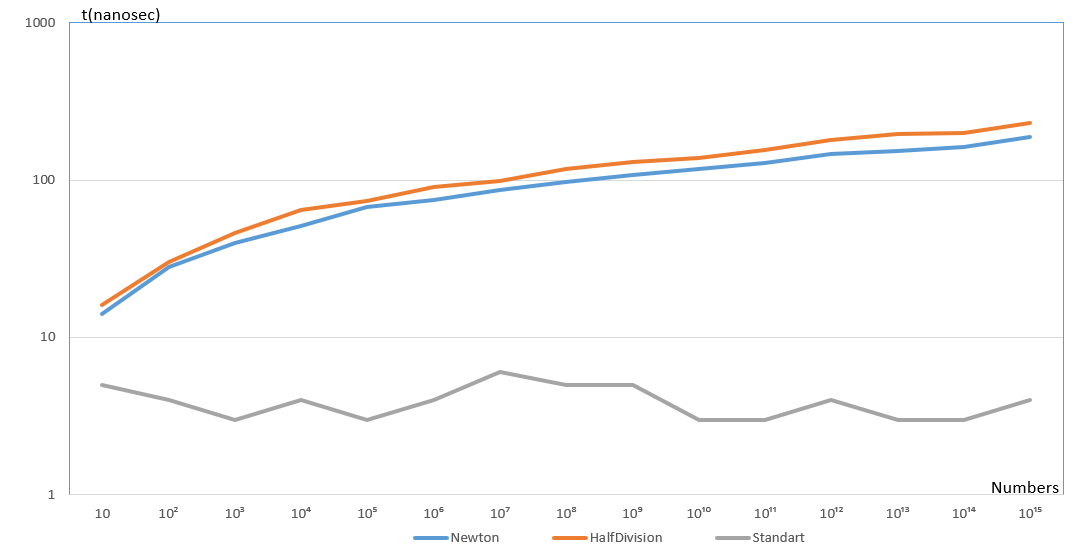
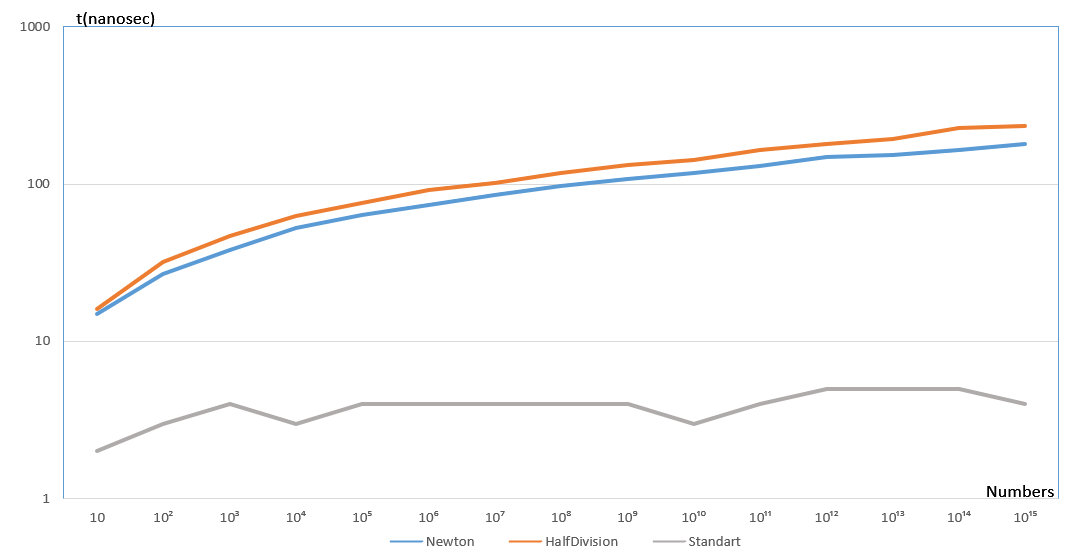
График Double

График Float



Из графиков видим, что алгоритмы действительно зависят от размера числа: чем больше введенный параметр, тем больше времени требуется на обработку.

Графики напоминают график логарифмической функции, что только подтверждает логарифмическую зависимость в сложности алгоритмов

Если сравнивать метод Ньютона и метод Касательных, то на всем промежутке метод Ньютона показывает себя лучше, однако так может быть не для всех чисел (на графиках мы даже видим колебания для определенных чисел, в местах плохих сходимостей алгоритмы ведут себя примерно одинаково), поскольку метод половинного деления может быть более стабильным и предсказуемым, особенно если требуется обеспечить гарантированную сходимость.

Вывод:

**Стандартный метод**: Этот метод может быть наиболее простым, но не всегда самым эффективным, особенно при работе с большими числами. Он зависит от реализации математических операций на конкретной платформе.

**Метод Ньютона**: Метод Ньютона часто сходится быстрее, особенно когда начальное приближение к корню хорошее. Он зависит от точности начального приближения и количества итераций, которое требуется для достижения точности.

**Метод половинного деления**: Этот метод гарантирует сходимость, но обычно медленнее, особенно если количество итераций фиксировано и не зависит от величины корня.

Визуализация

Для визуализации компонента Ecolab1 реализовано несколько обратных вызовов:

1) int16\_t (ECOCALLMETHOD \*OnPrintType)(/\* in \*/ struct IEcoLab1Events\* me, /\* in \*/ int16\_t num)

OnPrintType - отображает тип, используемый для извлечения кубического корня(Newton || Binary)

2) int16\_t (ECOCALLMETHOD \*OnPrintDataBefore)(/\* in \*/ struct IEcoLab1Events\* me, /\* in \*/ double\_t number, /\* in \*/ double\_t epsilon)

OnPrintDataBefore - показывает исходное состояние введенных данных(number && epsilon)

3) int16\_t (ECOCALLMETHOD \*CurrentState)(/\* in \*/ struct IEcoLab1Events\* me, /\* in \*/ int16\_t index,double\_t x,double\_t epsilon);

CurrentState - пошаговая работа алгоритма,включающая в себя Номер шага(Step) и Текущий результат(result)

4) int16\_t (ECOCALLMETHOD \*OnPrintDataAfter)(/\* in \*/ struct IEcoLab1Events\* me, /\* in \*/ double\_t number)

OnPrintDataAfter - отображает итоговые данные в