# Введение

-алгоритм (эпсилон – алгоритм) был предложен Питером Винном в 1956 году для вычисления преобразования Шенкса, и до сих пор является одним из самых важных алгоритмов ускорения сходимости, используемых в Численном Анализе, Методах решения уравнений, включая дифференциальные и интегральные, а так же во многих других сферах.

Ускорение достигается за счет преобразования (трансформации) последовательности.   
Последовательность , которая расходится или сходится так медленно, что практически не применима, превращается, с помощью функции ,в последовательность, которая сходится быстрее

Считается, что функция ускоряет сходимость, если сходится к S быстрее, чем

Трансформация последовательности позволяет улучшить сходимость и/или значительно уменьшить количество необходимых итераций.

Одним из первых методов ускорения сходимости является алгоритм (Дельта 2), открытый Александром Эйткеном в 1926 году. ***Метод Эйткена*** не является теоритически обоснованным, но при приближенных значениях параметров позволяет увеличить скорость сходимости [1].

***Метод Эйткена***

Пусть

где и некоторые константы. Тогда:

Следовательно,

Откуда получаем:

Стало быть:

Однако, из – за неточности в качестве следующей итерации мы должны взять значение, близкое к .

Из этого метода и выводится алгоритм :

Обозначим операторы:и

Однако если использовать на для улучшения точности, то это позволит вывести рекурсивную формулу:

(1.2)

Благодаря этой формуле, алгоритм можно имплементировать, используя один одномерный массив.

особенно хорошо подходит для последовательностей с линейной сходимостью (отклонение от их предела ведет себя до бесконечности, как геометрическая последовательность).

К сожалению, это численно нестабильный алгоритм: рекомендуется вычислять последовательность , а также с большим количеством значащих цифр. Некоторые записи алгоритма меньше распространяют ошибки округления, например:

# Эпсилон Алгоритм

Обобщением формулы (1.1) является:

(2.1)

Однако, в обобщенной формуле (1.2) для точно не дает (2.1). Вместо этого используется Преобразование Шанкса, которое определено следующим отношением определителей Ханкеля:

Где обозначает определитель Ханкеля:

Оно полностью соответствует (2.1).

Определители Ханкеля в Преобразовании Шанкса могут быть вычислены нелинейной рекурсией:

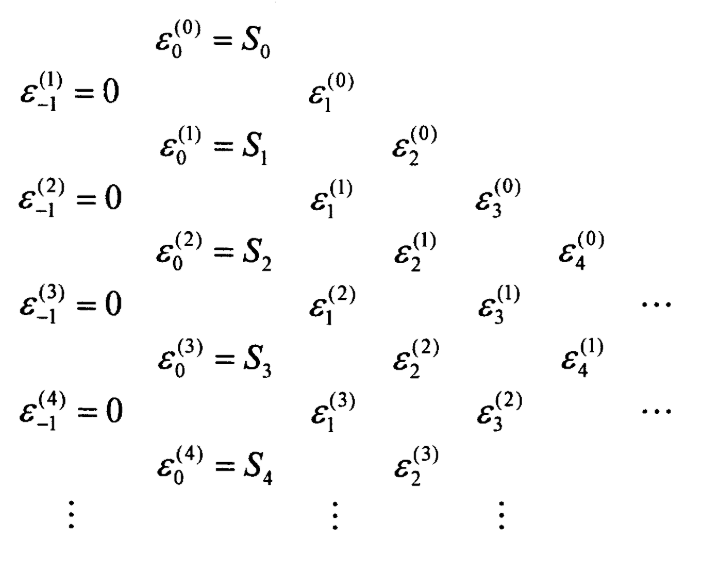
(2.2)

Подробнее о Преобразовании Шанкса можно узнать в файле ***Проект\_ПОПК.pdf,*** а реализация находится в файле ***shanks\_transformation.h.***

Рекурсивная схема (2.2) довольно сложна, и -алгоритм Винна существенно упрощает ее, убирая необходимость в вычислении определение Ханкеля:

(2.3)

-алгоритм является обобщением устаревшего алгоритма и был крайне важным шагом в ускорении сходимости. Результатом работы алгоритма будет -таблица (эпсилон-таблица), которая в теории бесконечна, но при ограниченном даст треугольник.



В формулу

Не забудь!

Винн доказал, что каждый (четный) ряд -таблицы эквивалентен преобразований Шанкса:

Нечетные же значения нужны лишь для промежуточных вычислений и не несут практической ценности:

Из формулы (2.3) очевидно, что -алгоритм связывает величины, расположенные в четырех вершинах ромба. И самым эффективным решением будет вычисление диагоналей таблицы, постепенно считая новые значения за счет увеличения .

-алгоритм можно представить в виде двух одномерных массивов. Реализация находится в файле ***epsilon\_algorithm.h.*** Однако,В книге за авторством Брезински описан вариант реализации через одномерный массив и несколько дополнительных переменных [3].

# Катастрофическое сокращение

Стандарт IEEE754 – широко используемый формат представления чисел с плавающей точкой. Он использует только ограниченное количество битов. Например, представление с двойной точностью использует 64 бита. 1 бит – знак, 11 битов на порядок и 52 бита на мантиссу:

Из – за ограниченного числа битов, операции над числами с плавающей точкой могут приводить к ошибкам. Если из одного числа вычесть другое, почти похожее на него, получившееся число может оказаться 0 или, что еще хуже, результат может содержать мало значащих битов.

Пусть имеются три переменные типа double: x, y, z:

double x = 1.000000000000001; *// Округляется до: 1 + 5\*2^{-52}*

double y = 1.000000000000002; *// Округляется до: 1 + 9\*2^{-52}*

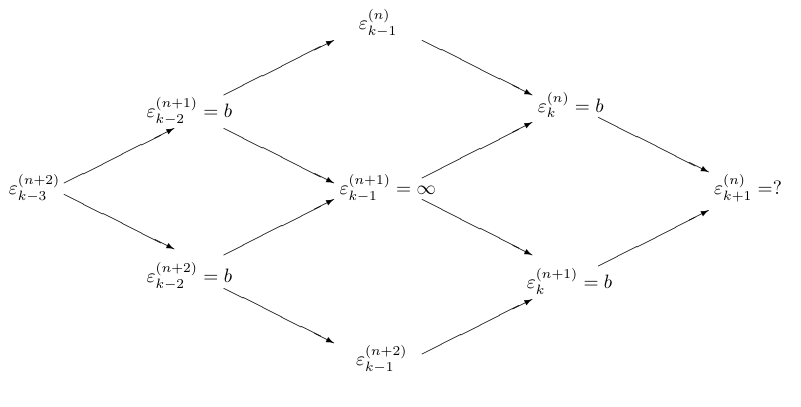
double z = y - x; *// Разница равна: 4\*2^{-52}*

По логике, z должен быть равен , но ответ будет равен , что дает 11% относительной ошибки.

Катастрофическое сокращение появляется при вычислении . Когда эти значения очень близки, получившееся число может либо сломать алгоритм, выдав 0, на который невозможно делить, либо слишком маленькое число, которое, при делении на него, создаст большую ошибку в вычислениях.

Для решения этой проблемы необходимо использовать форматы числа четвертой точности: float 128 или Quad [4]. Однако, избежать катастрофического сокращения можно с помощью изменения -алгоритма, которое описал сам Питер Винн:

Пусть и оба оказались равны . Тогда становится равен бесконечности, и вновь равны b. получается неопределенным и имеет минимум практического смысла:



В формулу

Не забудь!

Что бы обойти эту проблему, Винн установил следующее правило:

(3.1)

Эта формула позволит нам проигнорировать и , однако вынуждает хранить в памяти целый набор переменных: и что бы получить .

Программная реализация этой формулы описана в файле: ***epsilon\_algorithm\_two.h.***   
Для хранения столбцов -таблицы используется четырехмерный массив, кроме того, что бы избежать возможного деления на 0, которое все равно может возникнуть при очень большом катастрофическом сокращении, , если правило (3.1) не позволило избежать ошибки.

Можно использовать и другой метод борьбы с катастрофическим сокращением: создание константы , которая остановит алгоритм при достижении необходимой абсолютной погрешности. Если известен придел или не известен, то вычисление идет, пока не будет выполнено

# Модификации

Итерированный и -алгоритм прекрасно подходят для ускорения линейно сходящихся последовательностей, а так же многих расходящихся. Однако, и те и другие не эффективны в случае логарифмической сходимости. Для решения этой проблемы Винн создал – алгоритм (ро – алгоритм):

(4.1)

Алгоритм эффективен для ускорения в случае логарифмической сходимости, но абсолютно не подходит для линейной и, тем более, для расходящихся рядов.

Попыткой получить преимущества обоих версий алгоритмов был -алгоритм (тета – алгоритм), разработанный в 1971 году Брезински.

Как и в случае с -алгоритмом и – алгоритмом, -алгоритм дает практически применимые значения только в случаях четных , нечетные значения являются лишь вспомогательными данными.

Тета алгоритм оказался удачным экспериментом, и он позволяет получить стабильно хорошие результаты для большого количества различных рядов. Возможны реализации через один трехмерный массив или один двумерный [5].

(Тут бы еще об эйткине и прочем – прочем. Векторный трогать больно… Надо думать. Либо тащить инфу у 4куров?)

Анализируя модификации -алгоритма, а так же алгоритм Левина, Левина - Сиди (тут будет о файле о левине и л-с), Ченг создал эффективный алгоритм, схожий по параметрам даже иногда превосходящий с -алгоритмом [6].

Данный алгоритм можно представить в виде одного двумерного массива и одного одномерного. Реализация находится в файле ***chang\_whynn\_algorithm.h.***

# Выводы

# Литература

[1] Ионкин Н.: [Лекции по курсу «Численные методы»](http://cmcstuff.esyr.org/vmkbotva-r15/3%20%D0%BA%D1%83%D1%80%D1%81/6%20%D0%A1%D0%B5%D0%BC%D0%B5%D1%81%D1%82%D1%80/%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5%20%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B/%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D0%B8.pdf) (2019) 55-56 стр.

[2] Steele J.: [SOME RESULTS CONCERNING THE FUNDAMENTAL NATURE OF WYNN'S VECTOR EPSILON ALGORITHM](https://www.researchgate.net/publication/344177013_SOME_RESULTS_CONCERNING_THE_FUNDAMENTAL_NATURE_OF_WYNN'S_VECTOR_EPSILON_ALGORITHM) (2002) 21-23 стр.

[3] Brezinski, C.: [Algorithmes d’Accel´ eration de la Convergence— ´ Etude Num ´ erique. ´ Editions Technip, ´ Paris](https://books.google.ru/books?id=TxDghaunVjkC&printsec=frontcover&hl=ru#v=onepage&q&f=false) (1978) Chapter 4.3.2

[4]Clément V.: [Acceleration of convergence for numerical sequences](https://hal.science/hal-04207550/document) (2023) 18-19 стр.

[5] Weniger, E.: [Nonlinear sequence transformations for the acceleration of convergence and the summation of divergent series](https://arxiv.org/abs/math-ph/0306302) (1989) pp. 279–281

[6]Xiang-Ke C.: [Construction of new generalizations of Wynn’s epsilon and rho algorithm by solving finite difference equations in the transformation order](https://link.springer.com/article/10.1007/s11075-019-00695-w) (2019) 25 стр.