Содержание

Введение

ε-алгоритм

ε-алгоритм (эпсилон – алгоритм) был предложен Питером Винном в 1956 году [1, с. 55] 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 также во многих других сферах [2, гл. 4.3.2]..

Ускорение достигается за счет преобразования (трансформации) последовательности.
Последовательность $\{\mathrm{S_n}\}$, которая расходится или сходится так медленно, что практически не применима, превращается, с помощью функции $\boldsymbol{T}$, в последовательность $\{\mathrm{T_n}\}$, которая сходится быстрее

$$\boldsymbol{T}: \{\mathrm{S_n}\} \to \{\mathrm{T_n}\}, \mathrm{n} \in \boldsymbol{N_0}$$

Считается, что функция $\boldsymbol{T}$ ускоряет сходимость, если $\{\mathrm{T_n}\}$ сходится к S быстрее, чем $\{\mathrm{S_n}\}$

$$\lim_{n\to\infty} \frac{|\mathrm{T_n} - S|}{|S_n - S|} = 0$$

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

Исторически первым методом ускорения сходимости стал алгоритм Δ^2, разработанный Эйткеном в 1926 году., который эффективен для линейно сходящихся последовательностей [3, с. 18]. ***Метод Эйткена*** не является теоретически обоснованным, но при приближенных значениях параметров позволяет увеличить скорость сходимости [1].

Метод Эйткена

Пусть

$$S_n - S \simeq C\lambda^n, C \neq 0, |\lambda| < 1, n \in \boldsymbol{N_0}, \tag{1.1}$$

где C и λ некоторые константы. Тогда:

$$S_{n-1} - S = C\lambda^{n-1}, \qquad S_n - S = C\lambda^n, \qquad S_{n+1} - S = C\lambda^{n+1}$$

Следовательно,

$$(S_{n+1} - S_n)^2 = C^2\lambda^{2n}(\lambda - 1)^2, \qquad (S_{n+1} - 2S_n + S_{n-1}) = C\lambda^{n-1}(\lambda - 1)^2$$

Откуда получаем:

$$\frac{(S_{n+1} - S_n)^2}{(S_{n+1} - 2S_n + S_{n-1})} = C\lambda^{n+1} = S_{n-1} - S$$

Стало быть:

$$S \simeq S_{n+1} - \frac{(S_{n+1} - S_n)^2}{(S_{n+1} - 2S_n + S_{n-1})}$$

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

Из этого метода и выводится алгоритм Δ^2:

$$A_1^{(n)} = S_{n+1} - \frac{(S_{n+1} - S_n)^2}{(S_{n+1} - 2S_n + S_{n-1})}, n \in \boldsymbol{N_0}$$

Обозначим операторы: $\Delta S_n = S_{n+1} - S_n$ и $\Delta^2 S_n = S_{n+1} - 2S_n + S_{n-1}$

Однако если использовать Δ^2на A_1^n для улучшения точности, то это позволит вывести рекурсивную формулу:

$$A_0^{(n)} = S_n,\ A_{k+1}^{(n)} = A_k^{(n)} - \frac{\left(\Delta A_k^{(n)}\right)^2}{\Delta^2 A_k^{(n)}},\ k, n \in \boldsymbol{N_0} \tag{1.2}$$

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

Δ^2 особенно хорошо подходит для последовательностей с линейной сходимостью (отклонение от их предела ведет себя до бесконечности, как геометрическая последовательность).

Основной недостаток данного алгоритма заключается в его численной неустойчивости.: рекомендуется вычислять последовательность S_n, а также $A_k^{(n)}$ с большим количеством значащих цифр. Некоторые записи алгоритма меньше распространяют ошибки округления, например:

$$A_1^{(n)} = S_{n+1} + \frac{1}{\frac{1}{S_{n+2} - S_{n+1}} - \frac{1}{S_{n+1} - S_n}}$$

1. Эпсилон Алгоритм

Обобщением формулы (1.1) является:

$$S_n = S + \sum_{j=0}^{k-1} C_j \left(\lambda_j\right)^n, \ |\lambda_0| > |\lambda_1| > \cdots > |\lambda_{k-1}|, \forall i \ \lambda_i \neq 1, \ \mathrm{k, n} \in \boldsymbol{N_0}. \qquad (2.1)$$

Однако, Δ^2 в обобщенной формуле (1.2) для $k > 1$ точно не дает (2.1). Вместо этого используется Преобразование Шенкса, которое определено следующим отношением определителей Ханкеля:

$$e_k(S_n) = \frac{\begin{vmatrix} S_n & S_{n+1} & \cdots & S_{n+k} \\ \Delta S_n & \Delta S_{n+1} & \cdots & \Delta S_{n+k} \\ \vdots & \vdots & \ddots & \vdots \\ \Delta S_{n+k-1} & \Delta S_{n+k} & \cdots & \Delta S_{n+2k-1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ \Delta S_n & \Delta S_{n+1} & \cdots & \Delta S_{n+k} \\ \vdots & \vdots & \ddots & \vdots \\ \Delta S_{n+k-1} & \Delta S_{n+k} & \cdots & \Delta S_{n+2k-1} \end{vmatrix}} = \frac{H_{k+1}(S_n)}{H_k(\Delta^2 S_n)}, \ \ k, n \in \boldsymbol{N_0}$$

Где $H_k(u_n)$ обозначает определитель Ханкеля:

$$H_0(u_n) = 1, \qquad H_k(u_n) = \begin{vmatrix} u_n & u_{n+1} & \cdots & u_{n+k-1} \\ u_{n+1} & u_{n+2} & \cdots & u_{n+k} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n+k-1} & u_{n+k} & \cdots & u_{n+2k-2} \end{vmatrix}, \qquad k, n \in \boldsymbol{N_0}$$

Оно полностью соответствует (2.1).

Определители Ханкеля в Преобразовании Шенкса могут быть вычислены нелинейной рекурсией:

$$H_0(u_n) = 1, \qquad H_1(u_n) = u_n, \qquad n \in \boldsymbol{N_0} \qquad (2.2)$$

$$H_{k+2}(u_n) H_k(u_{n+2}) = H_{k+1}(u_n) H_{k+1}(u_{n+2}) - [H_{k+1}(u_{n+1})]^2 \quad k, n \in \boldsymbol{N_0}$$

Подробнее о Преобразовании Шенкса можно узнать в файле ***Отчет_МОПК_Шенкс.***

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

$$\varepsilon_{-1}^{(n)} = 0, \ \varepsilon_0^{(n)} = S_n, \quad \varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + \left[\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}\right]^{-1}, \ \ k, n \in \boldsymbol{N_0} \qquad (2.3)$$

ε-алгоритм является обобщением устаревшего алгоритма Δ^2 и был крайне важным шагом в ускорении сходимости. Результатом работы алгоритма будет ε-таблица (эпсилон-таблица), которая в теории бесконечна, но при ограниченном n даст треугольник.

$$
\begin{array}{ccccccc}
 & \varepsilon_0^{(0)} = S_0 & & & & & \\
\varepsilon_{-1}^{(1)} = 0 & & \varepsilon_1^{(0)} & & & & \\
 & \varepsilon_0^{(1)} = S_1 & & \varepsilon_2^{(0)} & & & \\
\varepsilon_{-1}^{(2)} = 0 & & \varepsilon_1^{(1)} & & \varepsilon_3^{(0)} & & \\
 & \varepsilon_0^{(2)} = S_2 & & \varepsilon_2^{(1)} & & \varepsilon_4^{(0)} & \\
\varepsilon_{-1}^{(3)} = 0 & & \varepsilon_1^{(2)} & & \varepsilon_3^{(1)} & & \cdots \\
 & \varepsilon_0^{(3)} = S_3 & & \varepsilon_2^{(2)} & & \varepsilon_4^{(1)} & \\
\varepsilon_{-1}^{(4)} = 0 & & \varepsilon_1^{(3)} & & \varepsilon_3^{(2)} & & \cdots \\
 & \varepsilon_0^{(4)} = S_4 & & \varepsilon_2^{(3)} & & & \\
\vdots & & \vdots & & \vdots & &
\end{array}
$$

Винн доказал, что каждый $2k$ (четный) ряд ε-таблицы эквивалентен k преобразований Шенкса:

$$\varepsilon_{2k}^{(n)} = e_k(S_n), \qquad k, n \in \boldsymbol{N_0}$$

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

$$\varepsilon_{2k+1}^{(n)} = 1/e_k(\Delta S_n), \qquad k, n \in \boldsymbol{N_0}$$

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

ε-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 [2].

2. Проблема катастрофического сокращения точности и модификация Винна

Проблема:

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:

$$S_0 - \text{знак}, \qquad E_1 E_2 \dots E_{11} - \text{порядок}, \qquad F_{12} F_{13} \dots F_{63} - \text{мантисса}. [3, \text{стр}. 18]$$

Из – за ограниченного числа битов, для чисел с плавающей точкой имеется фундаментальное ограничение – при операциях с близкими числами возникает катастрофическая потеря значащих разрядов.

Рассмотрим пример: пусть имеются три переменные типа double: x, y, z:

```
double x = 1.000000000000001;  // Фактическое значение: 1 + 5*2^{-52}
double y = 1.000000000000002;  // Фактическое значение: 1 + 9*2^{-52}
double z = y - x;              // Теоретически:        1.0*10^{-15}
                                                 // Практически:         8.88*10^{-16} (11% ошибка)
```

По логике, z должен быть равен 1.0×10^{-15}, но на практике ответ будет равен 8.88×10^{-16}, что дает 11% относительной ошибки.

В алгоритме проблема катастрофического сокращения особенно критична при вычислении разности $\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}$. Когда эти значения очень близки:

1. Результат может стать нулевым, вызывая в конечном итоге деление на ноль,
2. Или слишком маленькое число, которое, при делении на него, приведет к экспоненциальному росту ошибки.

Для решения этой проблемы существуют следующие решения:

Учетверенная точность

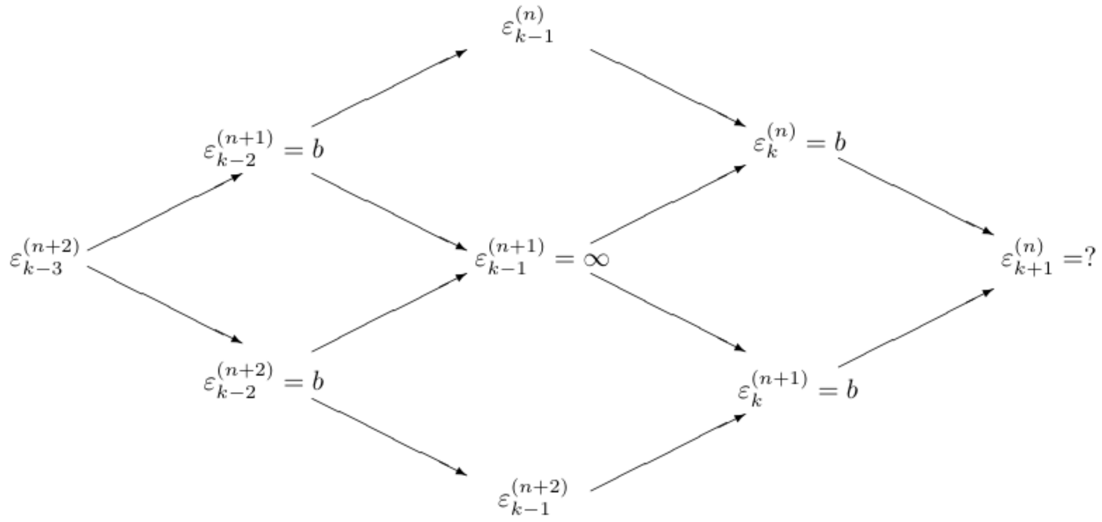
Использование форматов числа четвертой точности 128-битных чисел (float 128/Quad) [3, стр. 19] отодвигает порог возникновения проблемы до $\sim 10^{\{-34\}}$, но не устраняет её полностью.

Модификация Винна

Питер Винн предложил специальное правило для случаев, когда $\varepsilon_{k-2}^{(n+1)} = \varepsilon_{k-2}^{(n+2)} = b$, [3, стр. 20] что вызывает:

- Бесконечность в $\varepsilon_{k-1}^{(n+1)}$,
- Неопределённость в $\varepsilon_{k+1}^{(n)}$

Графическое представление проблемы:



Ромбовидная схема вычислений [3, стр. 20]

Модифицированная формула:

$$\varepsilon_{k+1}^{(n)} = \frac{a}{\left(1 + \frac{a}{\varepsilon_{k-1}^{(n+1)}}\right)}, \text{где } a = \frac{\varepsilon_{k-1}^{(n+2)}}{\left(1 - \frac{\varepsilon_{k-1}^{(n+2)}}{\varepsilon_{k-1}^{(n+1)}}\right)} + \frac{\varepsilon_{k-1}^{(n)}}{\left(1 - \frac{\varepsilon_{k-1}^{(n)}}{\varepsilon_{k-1}^{(n+1)}}\right)} - \frac{\varepsilon_{k-3}^{(n+2)}}{\left(1 - \frac{\varepsilon_{k-3}^{(n+2)}}{\varepsilon_{k-1}^{(n+3)}}\right)}, \qquad k, n \in \boldsymbol{N}_0 \tag{3.1}$$

[3, стр. 21]

Эта формула позволяет нам проигнорировать $\varepsilon_{k-2}^{(n+1)}$ и $\varepsilon_{k-2}^{(n+2)}$, однако требует хранить в памяти целый набор из 4 переменных: $\varepsilon_{k-1}^{(n+1)}$, $\varepsilon_{k-1}^{(n+2)}$, $\varepsilon_{k-1}^{(n)}$ и $\varepsilon_{k-3}^{(n+2)}$ для получения $\varepsilon_{k+1}^{(n)}$. [3, стр. 21]

Программная реализация этой формулы описана в файле: ***epsilon_algorithm_two.h.*** Для хранения столбцов ε-таблицы используется четырехмерный массив, кроме того, что бы избежать возможной ошибки в виде деления на 0, которая все равно может возникнуть при очень большом катастрофическом сокращении, применяем резервное правило (если правило 3.1 не позволило избежать ошибки):

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n)}$$

Критерий остановки алгоритма:

Можно использовать и другой метод борьбы с катастрофическим сокращением: создание константы t, которая остановит алгоритм при достижении необходимой абсолютной погрешности.

- Абсолютная погрешность: $\left|\varepsilon_k^{(0)} - S\right| < t$ (если предел S известен),
- Относительная погрешность: $\left|\varepsilon_k^{(0)} - \varepsilon_{k-2}^{(0)}\right| < t$ (если предел неизвестен).

3. Векторный ε- алгоритм

Векторная форма ε-алгоритма также была изучена П. Винном. Формулировка этого алгоритма основана на выражении (2.3) скалярного ε-алгоритма и адаптирована для работы с векторами. Векторная версия ε-алгоритма позволяет ускорять сходимость последовательностей векторов, в отличие от скалярного случая. Однако, поскольку операции деления векторов не определены, необходимо изменить формулу. Векторный ε-алгоритм, предложенный П. Винном, использует скалярные произведения для замены делений.

Пусть $(S_n)_n$ - векторная последовательность с $S_n \in \mathbb{R}^d$, для $n \in N$.

$$\varepsilon_{-1}^{(n)} = 0 \in \mathbb{R}^d, \qquad \varepsilon_0^{(n)} = S_n \in \mathbb{R}^d, \qquad n = 0, 1 \ldots$$

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + \left[\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}\right]^{-1}, \ k, n = 0, 1 \ldots$$

где для фиксированных значений n и k, $\varepsilon_k^{(n)} \in \mathbb{R}^d$

Чтобы применить этот алгоритм, мы должны определить значение, обратное вектору. $\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}$ — это вектор в $\mathbb{R}^d$. Для этого П. Винн определил обратную величину вектора **v** $= (u_i)_{1 \le i \le d} \in \mathbb{R}^d$.

$$\mathrm{v}^{-1} = \frac{\mathrm{v}}{(\mathrm{v}; \mathrm{v})}$$

с точечным произведением (v; v), определяемым

$$(\mathrm{v}; \mathrm{v}) = \sum_{i=1}^{d} v_i^2 = ||\mathrm{v}||_2^2$$

Поскольку этот алгоритм основан на формулировке скалярного ε-алгоритма, векторный ε-алгоритм представлен таким же образом. Действительно, мы можем построить такую же таблицу с двойной записью, где элементы последовательности $(S_n)_n$ расположены во втором столбце. Однако величины $\varepsilon_k^{(n)}$ будут векторами. Мы продвигаемся слева направо и сверху вниз по таблице ε. Более того, для каждого нового элемента исходной последовательности $(S_n)^n$ мы можем построить восходящую диагональ в таблице ε.

Все еще основываясь на результатах скалярного ε-алгоритма, только величины с еще более низким индексом могут быть интегрированы в новую последовательность, созданную ε-алгоритмом. Мы также можем найти правило пересечения для векторного ε-алгоритма, которое является: [3, стр. 26 − 27]

$$\varepsilon_{-2}^{(n)} = \infty \qquad \varepsilon_0^{(n)} = \boldsymbol{S}_n \qquad n = 0, 1\ldots$$

$$\frac{\varepsilon_{2k=2}^{(n-1)} - \varepsilon_{2k}^{(n)}}{||\varepsilon_{2k+2}^{(n-1)} - \varepsilon_{2k}^{(n)}||_2^2} + \frac{\varepsilon_{2k+2}^{(n+1)} - \varepsilon_{2k}^{(n)}}{||\varepsilon_{2k-2}^{(n+1)} - \varepsilon_{2k}^{(n)}||_2^2} = \frac{\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}}{||\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}||_2^2} + \frac{\varepsilon_{2k}^{(n-1)} - \varepsilon_{2k}^{(n)}}{||\varepsilon_{2k}^{(n-1)} - \varepsilon_{2k}^{(n)}||_2^2}, \qquad n, k = 0, 1\ldots$$

4. Модификация

Итерированный Δ^2 и ε-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 ρ – алгоритм (Rho – алгоритм):

$$\rho_{-1}^{(n)} = 0, \ \rho_0^{(n)} = S_n,$$

$$\rho_{k+1}^{(n)} = \rho_{k-1}^{(n+1)} + \frac{k+1}{\rho_k^{(n+1)} - \rho_k^{(n)}}, \qquad k, n \in \boldsymbol{N_0} \tag{6.1}$$

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 ϴ-алгоритм (тета – алгоритм), разработанный в 1971 году Брезински [2].

$$\upsilon_{-1}^{(n)} = 0, \qquad \upsilon_0^{(n)} = S_n,$$

$$\upsilon_{2k+1}^{(n)} = \upsilon_{2k-1}^{(n+1)} + \frac{1}{\Delta\upsilon_{2k}^{(n)}},$$

$$\upsilon_{2k+2}^{(n)} = \upsilon_{2k}^{(n+1)} + \frac{[\Delta\upsilon_{2k}^{(n+1)}][\Delta\upsilon_{2k+1}^{(n+1)}]}{\Delta^2\upsilon_{2k+1}^{(n)}}, \qquad k, n \in \boldsymbol{N_0}$$

Как и в случае с ε-алгоритмом и ρ – алгоритмом, ϴ-алгоритм дает практически применимые значения только в случаях четных $2k+2$, нечетные значения $2k+1$ являются лишь вспомогательными данными.

"ϴ-алгоритм, предложенный Брезински [5], является модификацией ε-алгоритма для случаев логарифмической сходимости. Возможны реализации через один трехмерный массив или один двумерный [5].

Анализируя модификации ε-алгоритма, а также алгоритм Левина (см. алгоритм Левина), Левина - Сиди, Ченг создал эффективный алгоритм, схожий по параметрам даже иногда превосходящий с ϴ-алгоритмом [6].

$$T_0^{(n)} = S_n, \qquad T_1^{(n)} = (T_0^{(n+1)} - T_0^{(n)}),$$

$$T_2^{(n)} = T_0^{(n+1)} - \frac{[\Delta \mathrm{T}_0^{(n)}][\Delta \mathrm{T}_0^{(n+1)}][\Delta^2 \mathrm{T}_0^{(n+1)}]}{\left[\Delta \mathrm{T}_0^{(n+2)}\right]\left[\Delta^2 \mathrm{T}_0^{(n)}\right] - [\Delta \mathrm{T}_0^{(n)}][\Delta^2 \mathrm{T}_0^{(n+1)}]},$$

$$F^{(n)} = \frac{\left[\Delta^2 \mathrm{T}_0^{(n)}\right]\left[\Delta^2 \mathrm{T}_0^{(n+1)}\right]}{\left[\Delta \mathrm{T}_0^{(n+2)}\right]\left[\Delta^2 \mathrm{T}_0^{(n)}\right] - [\Delta \mathrm{T}_0^{(n)}][\Delta^2 \mathrm{T}_0^{(n+1)}]},$$

$$T_{k+1}^{(n)} = T_{k-1}^{(n+1)} + \frac{1 - k + kF^{(n)}}{T_k^{(n+1)} - T_k^{(n)}}, \qquad k = 2,3 \dots, \qquad n \in \boldsymbol{N_0}$$

Данный алгоритм можно представить в виде одного двумерного массива и одного одномерного. Реализация находится в файле ***chang_whynn_algorithm.h.***

Заключение

Проведенный анализ позволяет утверждать, что ε-алгоритм демонстрирует высокую эффективность при ускорении сходимости числовых последовательностей, особенно в случаях линейной сходимости. Применение позволяет существенно сократить количество итераций и общее время вычислений при минимальных затратах.

Однако данный метод оказывается малоэффективным для последовательностей с уже высокой скоростью сходимости, например, квадратичной, как в методе Ньютона. Кроме того, логарифмически сходящиеся последовательности также слабо поддаются ускорению с помощью ε-алгоритма.

Результаты исследований свидетельствуют, что ε-алгоритм является эффективным методом для определенного класса задач, в частности, для последовательностей с линейной скоростью сходимости. В более сложных случаях целесообразно исследовать альтернативные подходы, такие как ρ-алгоритм или θ-алгоритм, обладающие иными свойствами и потенциально более высокой эффективностью при работе с «трудными» последовательностями.

Литература

1. Ионкин Н.: Лекции по курсу «Численные методы» (2019) 55-56 стр.

2. Brezinski, C.: Algorithmes d'Accel´ eration de la Convergence— ´ Etude Num ´ erique. ´ Editions Technip, ´ Paris (1978) Chapter 4.3.2

3. Clément V.: Acceleration of convergence for numerical sequences (2023) 18-27 стр.

4. Weniger, E.: Nonlinear sequence transformations for the acceleration of convergence and the summation of divergent series (1989) pp. 279–281

5. Xiang-Ke C.: Construction of new generalizations of Wynn's epsilon and rho algorithm by solving finite difference equations in the transformation order (2019) 25 стр.

6. Steele J.: SOME RESULTS CONCERNING THE FUNDAMENTAL NATURE OF WYNN'S VECTOR EPSILON ALGORITHM (2002) 21-23 стр.