Содержание

1 Введение

Метод Лагерра принадлежит к общим методам, сходящимся к любым типам корней: действительным, комплексным, одиночным или кратным. В данном отчёте рассмотрены его свойства и указаны сильные и слабые стороны.

Немного информации:
- метод гарантированно сходится для любых многочленов с полностью действительным набором корней, доказательство ниже;
- к простым корням метод сходится кубически, к кратным корням сходится линейно;
- для комлексных корней нельзя уверенно заявить о сходимости метода для любого случая, хотя экспериментально подтверждено, что случаи несходимости очень редки.

2 Теоретическое обоснование

2.1 Обозначения и вывод схемы

Для начала введём обозначения:
Пусть есть многочлен

$$P_n(x) = (x - x_0)(x - x_1)\dots(x - x_{n-1})$$

Взяв натуральный логарифм его модуля, получим:

$$ln|P_n(x)| = ln|x - x_0| + ln|x - x_1| + \dots + ln|x - x_{n-1}|$$

Тогда его первая производная будет равна:

$$\frac{dln|P_n(x)|}{dx} = \frac{1}{x - x_0} + \dots + \frac{1}{x - x_{n-1}} = \frac{P_n^{'}(x)}{P_n(x)} = G \tag{1.1}$$

Тогда:

$$-\frac{d^2ln|P_n(x)|}{dx^2} = \frac{1}{(x - x_0)^2} + \dots + \frac{1}{(x - x_{n-1})^2} = (\frac{P_n^{'}(x)}{P_n(x)})^2 - \frac{P_n^{''}(x)}{P_n(x)} = H \tag{1.2}$$

Введём обозначения:
x - текущее предполагаемое значение корня,
x_0 - искомое значение корня, к которому сходится метод,
x_i - остальные корни многочлена
$x - x_0 = a$ - расстояние от текущего предполагаемого значения до искомого корня
Сделаем предположение относительно расположения корней многочлена:
Пусть $\forall i\ x - x_i = b$ - расстояние от текущего предполагаемого значения до остальных корней, то есть все корни многочлена, кроме того, к которому мы сходимся, находятся на некотором примерно одинаковом удалении от искомого. Тогда:

$$\frac{1}{a} + \frac{n-1}{b} = G \tag{1.3}$$

$$\frac{1}{a^2} + \frac{n-1}{b^2} = H \tag{1.4}$$

Из этого получаем предполагаемое расстояние до искомого корня:

$$a = \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}} \tag{2}$$

Знак в знаменателе выбирается таким образом, чтобы знаменатель был наибольший, тем самым, уменьшая a. Это делается для того, чтобы обеспечить более точную сходимость.
Исходя из того, что подкоренное выражение может быть отрицательное, a может быть комплексным числом. Таким образом, обеспечивается возможность метода сойтись и к комплексным корням даже при действительном начальном приближении.

В исследуемом методе выбрано условие сходимости: $a < \epsilon$, по аналогии с методом Ньютона.

2.2 Критерий сходимости метода

Теорема 1. Пусть $p(z)$ - нормализованный многочлен со степенью $m \geq 4$ и пусть $P := \{\rho_1, \ldots, \rho_l\}$ - множество его уникальных корней. Тогда для части комплексной плоскости $u_0 \in \mathbb{C} - P$ с условием $p''(u_0) \neq 0$ и $p'(u_0) \neq 0$ последовательность Лаггера определяется как:

$$u_{n+1} = u_n - \frac{m}{q(u_n) + s(u_n)r(u_n)}, n \in \mathbb{N}, q(z) = \frac{p'(z)}{p(z)} \tag{3.1}$$

где

$$r(z) := \sqrt{(m-1)(mt(z) - q^2(z)}, \tag{3.2}$$

$$t(z) := q^2(z) - \frac{p''(z)}{p(z)}, \text{и} \tag{3.3}$$

$$s(z) := \begin{cases} 1, (Req(z))(Rer(z)) + (Imq(z))(Imr(z)) > 0, \\ -1, \text{в ином случае} \end{cases} \tag{3.4}$$

пока $q(u_n) + s(u_n)r(u_n) \neq 0$.
Если есть простой корень $\rho \in P$ такой, что выполняется условие

$$|u_0 - \rho| \leq \frac{1}{2m-1} min\{|\sigma - \rho| \sigma \in P \backslash \{\rho\}\} \tag{3.5}$$

Тогда последовательность Лаггера $L_p(u_0)$ сходится к пределу ρ, выполняется

$$|u_n - \rho| < \lambda^n |u_0 - \rho|, \text{ где } \lambda = \frac{15}{16} \ \forall n \in \mathbf{N} \tag{3.6}$$

Таким образом, обеспечивается, по крайней мере, линейная сходимость на всех дисках простых корней.

3 Описание работы алгоритма

Algorithm 1: Метод Лагерра для вычисления корней многочлена

Используемые константы:
MAXIT - максимально допустимое количество итераций, отводящееся на поиск одного корня
EPS - верхняя оценка ошибки округления
Итерации продолжаются, пока алгоритм не сойдётся либо не наткнётся на цикл выше определённого количества итераций
for $iter = 1$ to $MAXIT$ **do**
    b = a[m]
    err = |b|
    Вычислить пошагово многочлен в точке x и его две производные:
    for $j = m - 1$ to 0 **do**
        f=x*f+d $\#p''(x_k)$
        d=x*d+b $\#p'(x_k)$
        b=x*b+a[j] $\#p(x_k)$
        err=|b|+abx*err # Обновить значение ошибки при вычислении значения многочлен
    end for
    err=err*EPS # Применить ошибку округления
    Если многочлен равен 0 в выбранной точке:
    if $|b| \leq err$ **then**
        return x
    end if
    По формуле Лаггера:
    $G = \frac{d}{b}$
    $H = G^2 - 2 * \frac{f}{b}$
    $sq = \sqrt{(m-1)(mH - G^2)}$
    if $|G + sq| > |G - sq|$ **then**
        $den = G + sq$
    else
        $den = G - sq$
    end if
    $a = \frac{m}{den}$
    $x_1 = x - a$
    if $x_1 == x$ **then**
        return x
    end if
end for
Если превышено максимальное число итераций, выйти с ошибкой:
return 1

4 Имплементация на C++

4.1 ExtendedFunctions.h

Описание:

Набор необходимых функций для реализации алгоритмов нахождения корней многочленов.
Функции:

- **anynotfinite (bool)** : проверка, является ли хотя бы одно число конечным;
  Аргументы функции:
  - **T && ... t** : множество чисел (любое количество).

```
1 inline bool anynotfinite(T && ... t);
```

- **complexnotfinite (bool)** : проверка, содержит ли комплексное число значения NaN или Inf;
  Аргументы функции:
  - **a (complex<T>)** : комплексное число;
  - **big (T)** : максимальное значение для типа T.

```
1 bool complexnotfinite(complex<T> a, T big);
```

- **anycomplex (bool)** : проверка, что хотя бы одно комплесное число содержит мнимую часть;
  Аргументы функции:
  - **T && ... t** : множество чисел (любое количество).

```
1 inline bool anycomplex(T && ... t);
```

- **anycomplex (bool)** : проверка, что хотя бы одно комплесное число в векторе содержит мнимую часть;
  Аргументы функции:
  - **vec (vector<complex<T»)** : вектор комплексных чисел.

```
1 inline bool anycomplex(vector<complex<T>> vec);
```

- **sign (int)** : определение знака числа
  Аргументы функции:
  - **val (number)** : заданное значение;

  Возвращаемое значение: если заданное значение положительно, функция возвращает 1, если отрицательно, возвращает -1, если значение равно 0, возвращает 0.

```
1 inline int sign(number val);
```

- **fms (number)** : операция "fused multiply-subtract"(FMS), которая вычисляет разность произведения первых двух чисел и других двух чисел;
  Аргументы функции:

- **a (number)** : первое число;
- **b (number)** : второе число;
- **c (number)** : третье число;
- **d (number)** : четвертое число;

Возвращаемое значение: число $a \cdot b - d \cdot c$.

```
1 inline number fms(number a, number b, number c, number d);
```

- **fms (complex<number>)** : операция "fused multiply-subtract"(FMS) для комплесных чисел, которая вычисляет разность произведения первых двух чисел и других двух чисел;
  Аргументы функции:
  - **a (std::complex<number>)** : первое комплесное число;
  - **b (std::complex<number>)** : второе комплесное число;
  - **c (std::complex<number>)** : третье комплесное число;
  - **d (std::complex<number>)** : четвертое комплесное число;

  Возвращаемое значение: комплесное число $a \cdot b - d \cdot c$.

```
1 inline complex<number> fms(std::complex<number> a,
2                            std::complex<number> b,
3                            std::complex<number> c,
4                            std::complex<number> d);
```

- **fma (complex<number>)** : операция "fused multiply-add"(FMA) для комплесных чисел, которая вычисляет разность произведения двух чисел и третьего числа;
  Аргументы функции:
  - **a (std::complex<number>)** : первое комплесное число;
  - **b (std::complex<number>)** : второе комплесное число;
  - **c (std::complex<number>)** : третье комплесное число;

  Возвращаемое значение: комплесное число $a \cdot b - c$.

```
1 inline complex<number> fma(std::complex<number> a,
2                            std::complex<number> b,
3                            std::complex<number> c);
```

- **printVec (void)** : вывод вектора чисел в консоль;
  Аргументы функции:
  - **vec (vector<number>)** : вектор чисел;

```
1 inline void printVec(vector<number> vec);
```

- **castVec (vector<number>)** : преобразование вектора с типом T в вектор с типом number;
  Аргументы функции:
  - **vec (vector<T>)** : вектор чисел;

Возвращаемое значение: вектор чисел типа number.

```
1 inline vector<number> castVec(vector<T> vec);
```

4.2 Класс BaseSolver

Описание класса:
Абстрактный базовый класс для нахождения корней многочленов.
Методы класса:

- **operator() (void)** : нахождение корней многочлена;

  Аргументы метода:

  - **coeff (std::vector<T>&)** : вектор, содержащий коэффициенты многочлена;
  - **roots (std::vector<std::complex<T>&)** : вектор для хранения корней многочлена;
  - **conv (std::vector<int>&)** : вектор для хранения статуса сходимости каждого корня;
  - **itmax (int)** : максимально допустимое количество итераций.

```
1 virtual void operator()(std::vector<T>& coeff,
2                          std::vector<std::complex<T>>& roots,
3                          std::vector<int>& conv,
4                          int itmax) = 0;
```

4.3 Класс Original

Описание класса:
Класс, реализующий обычный алгоритм Лагерра для поиска корней многочлена.
Атрибуты класса:

- **eps (T)** : машинная точность для типа T;

Методы класса:

- **Original()** : конструктор класса **Original**;
- **operator() (void)** : нахождение корней многочлена с использованием базового метода Лагерра;
  Аргументы метода:

  - **poly (const std::vector<T>&)** : вектор, содержащий коэффициенты многочлена;
  - **roots (std::vector<std::complex<T>&)** : вектор для хранения корней многочлена;
  - **conv (std::vector<int>&)** : вектор для хранения статуса сходимости каждого корня;
  - **itmax (int)** : максимально допустимое количество итераций.

```
1 void operator()(std::vector<T>& poly,
2                  std::vector<std::complex<T>>& roots,
3                  std::vector<int>& conv, int itmax=80);
```

- **laguer (void)** : функция, реализующая метод Лагерра
  Аргументы метода:
  - **a (std::vector<std::complex<T»&)** : вектор, содержащий коэффициенты многочлена;
  - **x (std::complex<T>&)** : вектор для хранения корней многочлена;
  - **converged (int&)** : вектор для хранения статуса сходимости каждого корня;
  - **itmax (int)** : максимальное допустимое количество итераций;

```
1 inline void laguer(
2     const std::vector<std::complex<T>>& a,
3     std::complex<T>& x,
4     int& converged,
5     int itmax);
```

5 Тестирование

Во время тестирования для каждой степени полинома случайным образом генерировалось 10000 экспериментальных полиномов. Для данных типа float рассматривались только полиномы 5-ой степени и ниже, так как при более высоких степенях происходила потеря точности коэффициентов и, как следствие, нахождение неверных корней.

5.1 Результаты тестов для обычных корней (float)

Степень полинома	**Худшая абсолютная погрешность**	**Худшая относительная погрешность**
3	0.00156981	0.00176066
4	0.00212818	0.0037673
5	0.00109243	0.00163109

5.2 Результаты тестов для обычных корней (double)

Степень полинома	**Худшая абсолютная погрешность**	**Худшая относительная погрешность**	**Диапазон корней**
5	0.0000005	0.0000193436	[-1, 1]
10	0.0000005	0.0000317785	[-1, 1]
20	0.0001404608	0.0000008770	[-1, 1]
50	0.0402049456	0.0208416372	[-1, 1]
100	0.480197997	0.312220484	[-1, 1]
200	0.520666119	0.431904215	[-1, 1]
500	6.4353402647	0.9128599126	[-10, 10]
1000	89.7019046293	1.53283211	[-100, 100]
2000	110.1241004113	1.2356889	[-100, 100]
5000	154.391927846	1.96834675	[-200, 200]
10000	246.758487075	3.28456	[-200, 200]

5.3 Результаты тестов для кластеризованных корней (float)

Степень полинома	**Худшая абсолютная погрешность**	**Худшая относительная погрешность**	**Диапазон корней**	**Максимальная разница между корнями**
3	0.000445783	0.00487917	[-1, 1]	1e-5
4	0.00134838	0.0105308	[-1, 1]	1e-5
5	0.00236577	0.00780734	[-1, 1]	1e-5

5.4 Результаты тестов для кластеризованных корней (double)

Степень полинома	**Худшая абсолютная погрешность**	**Худшая относительная погрешность**	**Диапазон корней**	**Максимальная разница между корнями**
5	0.0000244001	0.0030831141	[-1, 1]	1e-5
10	0.0001024728	0.0033105529	[-1, 1]	1e-5
20	0.0007387185	0.0341325907	[-1, 1]	1e-5
50	0.2320763243	0.1342833161	[-1, 1]	1e-5
100	0.8339492304	1.001194048	[-2, 2]	1e-5
200	59.6137909595	0.4630768206	[-50, 50]	0.1
500	246.2037015651	1.9970928925	[-100, 100]	0.1
1000	411.2658618496	2.9911	[-500, 500]	0.1
2000	510.1241004113	3.2356889	[-500, 500]	0.1
5000	556.1241004113	3.56834675	[-500, 500]	0.1
10000	642.758487075	4.48456	[-500, 500]	0.1

6 Заключение

Плюсы метода:

1. Простота реализации;
2. Высокая скорость сходимости (кубическая для простых и линейная для кратных корней);
3. Редкость ситуаций, когда метод не сходится.

Минусы метода:

1. Отсутствие полной гарантии сходимости - при попадании на цикл или на точку сингулярности (редкое явление, но может случиться) сходимости нет;
2. Использование сложной арифметики, даже для вычисления действительных корней - в методе используется операция квадратного корня, также ветвление;

7 Список литературы

1. William H. P., Saul A. T., William T. V., Brian P. F.: Numerical Recipes in C The Art of Scientific Computing Second Edition - pp 463-473. CAMBRIDGE UNIVERSITY PRESS, Cambridge (2007).

2. Moeller, H.: The Laguerre-and-Sums-of-Powers Algorithm for the Efficient and Reliable Approximatio of All Polynomial Roots. Problems of Information Transmission, 2015, Vol. 51, No. 4, pp. 361–370 (2015).

3. Petkovic, M.S., Ilic, S., Trickovic, S.: The guaranteed convergence of Laguerre-like method. Computers and Mathematics with Applications 46 2003, pp 239-251. (2003).

4. Thomas R. Cameron : An effective implementation of a modified Laguerre method for the roots of a polynomial. Numer. Algor. 82, 1065-1084 (2018)