Содержание

1 Введение

В данной статье мы рассмотрим модифицированный метод Лагерра, который подходит для нахождения кратных корней многочлена, поиск которых часто осложняет работу аналогичных методов.
Во втором разделе мы описываем вывод итерационной формулы для модфикации Эйлера-Коши и оцениеваем его погрешность.
В третьем разделе объясняется основная схема работы алгоритма.
Четвертый раздел посвящен описанию созданной имплементации на языке C++.
В пятом разделе представлены результаты работы созданной программы с различными многочленами.

2 Теоретическое обоснование

2.1 Вывод метода

Большинство алгоритмов для нахождения корней многочлена работают лишь для простых корней (корней кратности 1) нелинейного уравнения $f(x) = 0$, то есть для корня α, имеем:

$$\begin{aligned} f(\alpha) &= 0 \\ f'(\alpha) &\neq 0 \end{aligned} \tag{1}$$

Мы же рассмотрим случай, когда α - корень кратности $m > 1$. Существует очень мало методов для нахождения кратных корней, когда кратность известна. Одним из таких методов является метод Лагерра. Он основан на том, что вместо рассмотрения функции $f(x)$, рассматривается функция $G(x) = \sqrt[m]{f(x)}$, которая также имеет простой корень α, в то же время являющийся корнем кратности m для фунции $f(x)$.

В этом случае итерационная формула для метода Лаггера принимает слудующий вид:

$$x_{n+1} = x_n - \frac{\lambda \frac{f(x_n)}{f'(x_n)}}{1 + sgn(\lambda - m)\sqrt{(\frac{\lambda - m}{m})[(\lambda - 1) - \lambda \frac{f(x_n)f''(x_n)}{f'(x_n)^2}]}} \tag{2}$$

где λ - вещественный параметр. Если $f(x)$ - многочлен степени n, этот метод с $\lambda = n$ принимает вид стандартного метода Лагерра для кратных корней. Этот метод сходится кубически.

Модицификация Эйлера-Коши для $\lambda = 2m$:

$$x_{n+1} = x_n - \frac{2m \frac{f(x_n)}{f'(x_n)}}{1 + \sqrt{(2m - 1) - 2m \frac{f(x_n)f''(x_n)}{f'(x_n)^2}}} \tag{3}$$

2.2 Оценка погрешности метода

Доказано, что погрешность для метода Лагерра определяется выражением:

$$e_{n-1} = K_3(m, \lambda)e_n^3 + O(e_n^4) \tag{6}$$

где асимптотическая константа ошибки $K_3(m, \lambda)$ определяется следующим образом:

$$K_3(m, \lambda) = A_1(m, \lambda)\Big(\frac{f^{(m+1)}(\alpha)}{f^{(m)}(\alpha)}\Big)^2 - A_2(m)\frac{f^{(m+2)}(\alpha)}{f^{(m)}(\alpha)} \tag{7}$$

$$A_1(m, \lambda) = \frac{1}{2m(m+1)^2}\Big(1 - \frac{1}{\lambda - m}\Big)$$

$$A_2(m) = \frac{1}{m(m+1)(m+2)}$$

Для модификации Эйлера-Коши асимптотическая константа ошибки равна:

$$K_3(m, 2m) = \frac{m - 1}{2m^2(m+1)^2}\Big(\frac{f^{(m+1)}(\alpha)}{f^{(m)}(\alpha)}\Big)^2 - A_2(m)\frac{f^{(m+2)}(\alpha)}{f^{(m)}(\alpha)} \tag{8}$$

3 Описание работы алгоритма

Пусть $p(z)$ - полином следующего вида:

$$p(z) = (z - \xi_1)^m (z - \xi_2)^m ... (z - \xi_k)^m$$

или, если раскрыть скобки:

$$p(z) = a_0 + a_1 z + \cdots + a_n z^n$$

Мы стремимся вычислить корни $p(z)$ и для этого необходимо выполнить следующие пункты:

Algorithm 1: Метод Лаггера для вычисления кратных корней многочлена

Находим наибольший общий делитель для многочлена и его производной
с помощью алгоритма Евклида
$h(z) = \text{НОД}(p(z), p'(z))$
$g(z) = \frac{f(z)}{h(z)}$
Степень полученного многочлена является количеством различных корней
исходного полинома
$deg(g(z)) = k$
$m = \frac{n}{k}$
$\lambda = 2m$
for $i = 1$ to k **do**
    Задаем начальное приближение
    $x_i = 0$
    while $p'(x_i) \neq 0$ **do**

$$x_i = x_i - \frac{2m \frac{p(x_i)}{p'(x_i)}}{1 + \sqrt{(2m-1) - 2m \frac{p(x_i) p''(x_i)}{p'(x_i)^2}}}$$

    end while
    $p(x) = \dfrac{p(x)}{(x - x_i)^m}$
end for

---

4 Имплементация на C++

4.1 ExtendedFunctions.h

Описание:
Набор необходимых функций для реализации алгоритмов нахождения корней полиномов.
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 полиномов.

Методы класса:

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

  Аргументы метода:

  - **coeff (std::vector<T>&)** : вектор, содержащий коэффициенты полинома;
  - **roots (std::vector<std::complex<T>&)** : вектор для хранения корней полинома;
  - **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 ModifiedLaguerre13

Описание класса:

Класс, реализующий модификацию Эйлера-Коши для алгоритма Лагерра для поиска корней полиномов.

Атрибуты класса:

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

Методы класса:

- **ModifiedLaguerre13()** : конструктор класса **ModifiedLaguerre13**;
- **operator() (void)** : нахождение корней полинома с использованием модифицированного метода Лагерра;

  Аргументы метода:

  - **poly (std::vector<T>&)** : вектор, содержащий коэффициенты полинома;
  - **roots (std::vector<std::complex<T>&)** : вектор для хранения корней полинома;
  - **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);
```

- **laguer13 (void)** : функция, реализующая модификацию Эйлера-Коши для алгоритма Лагерра.
  Аргументы метода:
  - **a const std::vector<std::complex<T> >** : вектор, содержащий коэффициенты многочлена;
  - **x std::complex<T>** : вектор для хранения корней многочлена;
  - **lam (int)** : параметр λ, равный удвоенной кратности корня

```
1 inline void laguer13(const std::vector<std::complex<T>>& a,
2                       std::complex<T>& x,
3                       int& lam);
```

5 Тестирование

Во время тестирования для каждой степени полинома случайным образом генерировалось 10000 экспериментальных полиномов. В качестве кластеризованных корней были выбраны корни от -1 до 1 с максимальным расстоянием между ними 1e-5.

5.1 Результаты тестов для кратных корней (double)

Степень полинома	**Худшая абсолютная погрешность**	**Худшая относительная погрешность**	**Диапазон корней**
5	0.0000004998	0.0002146611	[-1, 1]
10	0.0000004999	0.0003525780	[-1, 1]
20	0.0000004998	0.0002308061	[-1, 1]
50	0.0000004995	0.0001064237	[-1, 1]
100	0.0195306148	0.01066753	[-1, 1]
200	0.2202940669	0.203467	[-1, 1]
500	0.4531183	0.4682544	[-1, 1]
1000	1.5634323	1.4542908	[-1, 1]

6 Список литературы

1. Osada, N., Asymptotic error constants for cubically convergent zero finding methods, J. Comput. Appl. Math., 196, 347-357, (2006)

2. Vrscay, E. R., Gilbert, W. J., Extraneous fixed points, basin boundaries and chaotic dynamics for Schroder and Konig rational iteration functions, Numer. Math., 52, 1-16, (1988)

3. Neta, Beny, On a family of Laguerre methods to find multiple roots of nonlinear equations, United States Code, 17, 101, (2013)