Wojciech Ładyga - zadanie 17

Język technologia: c++

Program wykorzystujący metodę Laguerre'a pozwalający znaleźć wszystkie pierwiastki równania (nawet te zespolone).

Więc aby móc rozwiązać problem w c++ należy zainicjować klasę szablonową `complex` pozwalającą działać na liczbach zespolonych.

Oraz wykorzystać generator liczb pseudolosowych do losowania początkowego punktu, który jest bez znaczenia dla wyliczania wartości funkcji.

Kod programu:

/* * @Author: Wojciech Ladyga * @Date: 2019-01-22 * @Description: Zad 10 */ //metoda Laguerre'a #include #include #include #include #include //setprecisoion

```
using namespace std;

//http://staff.elka.pw.edu.pl/~rnowak2/zprwiki/doku.php?
id=liczby_zespolone_complex

//definicja typu danych do przechowywania liczb zespolonych a+ib = z
typedef complex<double> compNumber;
//def kontenera do przechowywania liczb zespolonych
typedef vector<complex<double>> vecCompNumbers;

//klasa reprezentująca metodę laguerre'a
class Laguerre
{
private:
    //zmienne globalne typu complex<double>
    compNumber tmp;
    compNumber wartoscLosowana;

    //metoda licząca wielomian lub 1/2 pochodną
    //control = 0 - wielomian
    //control = 1 - pierwsza pochodna
    //control = 2 - 2 pochodna
    compNumber pochodna(vecCompNumbers wartosci, int control)
    {
        switch (control)
        {
        case 0:
            tmp = wartosci[wartosci.size() - 1];
            break;
        case 1:
            tmp = wartosci[wartosci.size() - 1] * (double)(wartosci.size() - 1);
            break;
```

```
        case 2:
            tmp = wartosci[wartosci.size() - 1] * (double)(wartosci.size() - 1) *
(double)(wartosci.size() - 2);
            break;
        }

        for (double i = wartosci.size() - 2; i >= control; i--)
        {
            switch (control)
            {
            case 0:
                tmp = tmp * wartoscLosowana + wartosci[i];
                break;
            case 1:
                tmp = tmp * wartoscLosowana + wartosci[i] * i;
                break;
            case 2:
                tmp = tmp * wartoscLosowana + wartosci[i] * i * (i - 1);
                break;
            }
        }

        return tmp;
    }

    //wypisywanie wartosci
    void wypisz(complex<double> x1, complex<double> x2)
    {
        check(x1);
        check(x2);
    }

    //formatowanie wypisywania
    void check(complex<double> x)
    {
        if ((x.real() == 0.0 || x.real() == -0.0) && (x.imag() == 0.0 || x.imag()
== -0.0))
        {
            cout << fixed << "0.0" << endl;
        }
        else if ((x.real() == 0.0 || x.real() == -0.0) && (x.imag() != 0.0 ||
x.imag() != -0.0))
        {
            cout << fixed << setprecision(10) << x.imag() << "i" << endl;
        }
        else if ((x.imag() == 0.0 || x.imag() == -0.0) && (x.real() != 0.0 ||
x.real() != -0.0))
        {
            cout << fixed << setprecision(6) << x.real() << endl;
        }
        else
```

```
        {
            cout << fixed << setprecision(6) << x.real() << " + " << fixed <<
setprecision(10) << x.imag() << "i" << endl;
        }
    }

    //stosowanie meody laguerra
    /*
        G = p'/p
        H = G^2 - p''/p
        a = n/(G(+/-) ((n-1)(nH-G^2)^(1/2))
        */
    compNumber laguerr(vecCompNumbers wartosci)
    {
        compNumber sq = sqrt((double)(wartosci.size() - 2) * ((double)
(wartosci.size() - 2) * pochodna(wartosci, 1) * pochodna(wartosci, 1) - (double)
(wartosci.size() - 1) * pochodna(wartosci, 0) * pochodna(wartosci, 2)));
        compNumber firstP = pochodna(wartosci, 1);

        //wybieramy większy znak na moduł
        if (abs(firstP + sq) > abs(firstP - sq))
        {
            return wartoscLosowana - ((double)(wartosci.size() - 1) *
pochodna(wartosci, 0)) / (firstP + sq);
        }
        else
        {
            return wartoscLosowana - ((double)(wartosci.size() - 1) *
pochodna(wartosci, 0)) / (firstP - sq);
        }
    }

    //losowanie liczb z przedzedzialu 0 - 1 jako punkt startowy
    double randZeroToOne()
    {
        return rand() / (RAND_MAX + 1.);
    }

public:
    //wyliczamy rozwiązania funkcji
    void rozwiaz(vecCompNumbers wartosci)
    {

        while (wartosci.size() - 1 > 2) //aż nie osiągniemy funkcji kwadratowej
        {
            //losujemy punkt startowy
            compNumber los(randZeroToOne(), randZeroToOne());

            //wykonujemy, aż osiągniemy odpowiednią dokładnosc
            while (abs(pochodna(wartosci, 0)) > 0.00000001)
            {
```

```
                wartoscLosowana = los;
                los = laguerr(wartosci);
            }

            //wygladzanie naszej funkcji
            vecCompNumbers vecTmp = wartosci;
            wartosci.resize(wartosci.size() - 1); //zmniejszamy nasz wektor
            wartosci[wartosci.size() - 1] = vecTmp[wartosci.size()];

            for (int i = wartosci.size() - 1; i > 0; i--)
            {
                wartosci[i - 1] = (vecTmp[i] + (wartoscLosowana * wartosci[i]));
            }

            //wypisujemy wyniki
            check(los);
        }

        //jesli odpowiednio wygladzimy to możemy rozwiązać równaie kwadratowe

        //delta = b^2-4ac
        complex<double> delta = pow(wartosci[1], 2) - (4.0 * wartosci[2] *
wartosci[0]);
        //x1/x2 = (-b +/- (delta)^(1/2))/2a
        complex<double> x1 = ((-wartosci[1] + sqrt(delta)) / (2.0 * wartosci[2]));
        complex<double> x2 = ((-wartosci[1] - sqrt(delta)) / (2.0 * wartosci[2]));

        //wypisujmy wyniki
        wypisz(x1, x2);
    }
};

//funkcja wyliczajaca funkcje podane jako tablica
void wynik(Laguerre lag, vecCompNumbers wartosci, int tab[], int wielkosc, bool
complex, int t[] = NULL)
{
    compNumber a[wielkosc];

    //wypisanie funkcji
    for (int i = 0; i < wielkosc + 1; i++)
    {
        if (tab[i] != 0)
            cout << tab[i] << "z^" << wielkosc - i << " ";

        if (tab[i] == 0)
        {
            cout << t[i] << "iz^" << wielkosc - i << " ";
        }
        if (i != wielkosc)
        {
            cout << "+ ";
```

```
        }
    }
    cout << endl;

    for (int i = wielkosc; i >= 0; i--)
    {
        if (complex)
        {
            compNumber tmp(tab[i], t[i]);
            wartosci.push_back(tmp);
        }
        else
        {
            compNumber tmp(tab[i], 0);
            wartosci.push_back(tmp);
        }
    }
    lag.rozwiaz(wartosci);
    wartosci.clear();
}
int main()
{
    Laguerre lag;
    vecCompNumbers wartosci;

    int tab[] = {243, -486, 783, -990, 558, -28, -72, 16};
    wynik(lag, wartosci, tab, 7, false);

    cout << endl;

    int tab2[] = {1, 1, 3, 2, -1, -3, -11, -8, -12, -4, -4};
    wynik(lag, wartosci, tab2, 10, false);

    cout << endl;

    int tab3[] = {1, 0, -1, 0, 1};
    int tab33[] = {0, 1, 0, -1, 0};
    wynik(lag, wartosci, tab3, 4, true, tab33);

    return 0;
}
```

wyik działania programu:

```
243z^7 + -486z^6 + 783z^5 + -990z^4 + 558z^3 + -28z^2 + -72z^1 + 16z^0
0.666601 + 0.0000009982i
0.666753 + 0.0001450630i
0.666749 + -0.0001477329i
0.333333 + -0.0000000002i
0.000000 + 1.4142135624i
0.000000 + -1.4142135624i
-0.333333 + 0.0000000000i

1z^10 + 1z^9 + 3z^8 + 2z^7 + -1z^6 + -3z^5 + -11z^4 + -8z^3 + -12z^2 + -4z^1 + -4z^0
0.000002 + 0.9999966891i
-0.000008 + 1.0000115885i
0.000008 + -0.9999898757i
-0.000000 + -1.4142135612i
-0.500000 + 0.8660254036i
0.000000 + 1.4142135623i
-0.500000 + -0.8660254032i
-0.000007 + -1.0000367094i
1.414214 + -0.0000000000i
-1.414214 + 0.0000000000i

1z^4 + 1iz^3 + -1z^2 + -1iz^1 + 1z^0
0.951057 + 0.3090169944i
0.587785 + -0.8090169944i
-0.587785 + -0.8090169944i
-0.951057 + 0.3090169944i
```