Metody numeryczne

Wojciech Chrobak

9 stycznia 2018

Zadanie 6 - obowiązkowe

Metoda Laguerre'a

Metoda Laguerre'a jest metodą numerycznego poszukiwania **miejsc zerowych wielomianów**.
W wypadku ogólnym:

- metoda jest zbieżna sześciennie do wszystkich pojedynczych miejsc zerowych (rzeczywistych i zespolonych)
- metoda jest zbieżna liniowo do wielokrotnych miejsc zerowych
- przypadki braku zbieżności są bardzo rzadkie; w nielicznych przypadkach, w których metodzie grozi stagnacja, można ją przerwać wykonując jeden-dwa kroki metodą Newtona, a potem powrócić do metody Laguerre'a
- metoda jest podobna do metody opartej o rozwinięcie w szereg Taylora do drugiego rzędu ale jest lepsza, gdyż uwzględnia stopień wielomianu
- metoda wymaga obliczania drugiej pochodnej, ale w wypadku wielomianów jest to bardzo proste
- metoda nawet dla rzeczywistych punktów początkowych może prowadzić do zespolonych miejsc zerowych. Jednak z uwagi na specyfikę wielomianów, nie ma sensu upieranie się przy operowaniu na liczbach rzeczywistych

Algorytm

Metoda Laguerre'a znalezienia jednego pierwiastka wielomianu $P(x)$ stopnia n jest następująca:

- wybierz dowolny zespolony punkt startowy x_0
- dla $k = 0, 1, 2, ...$
  - jeśli $P(x_k)$ bardzo małe, przerwij pętle
  - wylicz
  $$G = \frac{P'(x_k)}{P(x_k)}$$
  - wylicz
  $$H = G^2 - \frac{P''(x_k)}{P(x_k)}$$

  - wylicz

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

    znak wybierany jest w ten sposób aby mianownik miał większy moduł dla uniknięcia utraty cyfr znaczących
  - przypisz $x_{k+1} = x_k - a$

- powtarzaj dotąd aż a stanie się dostatecznie małe lub będzie osiągnięta maksymalna ilość iteracji

Strategia postępowania

W ten sposób dostaliśmy **przybliżenie** miejsca zerowego $\overline{z}_{k+1}$ wielomianu P_{n-k} W celu **wygładzenia** tego miejsca zerowego, liczby $\overline{z}_{k+1}$ używamy jako warunku początkowego dla metody Laguerre'a zastosowanej do pełnego wielomianu $P_n(z)$. Spodziewamy się, że w ciągu kilku iteracji znajdziemy miejsce zerowe z_{k+1}. **Faktoryzujemy** wielomian $P_{n-k}(z)$, to znaczy obliczamy $P_{n-k}(z) = (z - z_{k+1})P_{n-k-1}(z)$. Postępujemy tak dopóki nie dojdziemy do wielomianu stopnia 2, któego pierwiastki obliczamy według znanego wzoru.

Kod

```
#include <vector>
#include <iostream>
#include <iomanip>
#include <complex>

using namespace std;

typedef complex<double> zesp;

const double eps = 1e-8;

zesp horner(const vector<zesp> &P, zesp x) {
    int n = P.size() - 1;
    zesp res = P[n];
    for (int i = n - 1; i >= 0; i--) {
        res = res * x + P[i];
    }

    return res;
}

vector<zesp> pochodna(const vector<zesp> &P) {
    vector<zesp> result;
    result.resize(P.size() - 1);

    int n = P.size();
    for (double i = 1; i < n; ++i) {
        zesp temp = i;
        result[i - 1] = temp * P[i];
    }

    return result;
}

vector<zesp> deflacja(const vector<zesp> &P, zesp z0) {
    vector<zesp> result;
    result.resize(P.size() - 1);
    result[P.size() - 2] = P[P.size() - 1];
    for (int i = P.size() - 3; i >= 0; i--) {
```

```cpp
        result[i] = P[i + 1] + result[i + 1] * z0;
    }

    return result;
}

zesp Laguerre(const vector<zesp> &P, zesp z0) {
    int n = P.size() - 1;
    zesp nz = zesp((double) n);
    vector<zesp> pochodna1 = pochodna(P);
    vector<zesp> pochodna2 = pochodna(pochodna1);

    for (int i = 0; i < 500000; ++i) {
        zesp wartosc = horner(P, z0);

        if (abs(wartosc) < eps)
            break;

        zesp wartosc_pochodna1 = horner(pochodna1, z0);
        zesp wartosc_pochodna2 = horner(pochodna2, z0);

        zesp G = wartosc_pochodna1 / wartosc;
        zesp H = G * G - wartosc_pochodna2 / wartosc;
        zesp R = sqrt((nz - 1.0) * (H * nz - G * G));

        zesp D1 = G + R;
        zesp D2 = G - R;

        zesp M;
        if (abs(D1) > abs(D2))
            M = D1;
        else
            M = D2;

        zesp a = nz / M;

        z0 -= a;


        if (abs(a) < eps)
            break;
    }

    return z0;
}

vector<zesp> szukaj(const vector<zesp> &P) {
    vector<zesp> res;
    vector<zesp> P_def = P;
    while (P_def.size() > 2) {
        zesp z = (rand() / double(RAND_MAX), rand() / double(RAND_MAX));
        z = Laguerre(P_def, z);
        z = Laguerre(P, z); // wygladzanie
        P_def = deflacja(P_def, z); // faktoryzacja
        res.push_back(z);
    }
    res.push_back(-P_def[0] / P_def[1]);
    return res;
}

void wypisz(vector<zesp> roots) {
    for (auto x : roots) {
        if (abs(x.imag()) < 1e-8)
            x.imag(0);
```

```
        if (abs(x.real()) < 1e-8)
            x.real(0);
        cout << x << endl;
    }
    cout << endl;
}

int main() {
    cout.setf(ios::fixed, ios::floatfield);
    cout.precision(8);

    vector<zesp> P1{16.0, -72.0, -28.0, 558.0, -990.0, 783.0, -486.0, 243.0};
    vector<zesp> P2{-4.0, -4.0, -12.0, -8.0, -11.0, -3.0, -1.0, 2.0, 3.0, 1.0, 1.0};
    vector<zesp> P3{1.0, 0.0, -1.0, 0.0, 1.0};
    P3[1].imag(-1.0);
    P3[3].imag(1.0);

    vector<zesp> roots1 = szukaj(P1);
    wypisz(roots1);

    vector<zesp> roots2 = szukaj(P2);
    wypisz(roots2);

    vector<zesp> roots3 = szukaj(P3);
    wypisz(roots3);

    return 0;
}
```

Wynik

$$243z^7 - 486z^6 + 783z^5 - 990z^4 + 558z^3 - 28z^2 - 72z + 16 = 0$$

Miejsca zerowe: $(\Re, \Im)$

$$(0.66599495, 0.00000000)$$
$$(0.66700240, -0.00058032)$$
$$(0.33333334, 0.00000000)$$
$$(0.66700265, 0.00058032)$$
$$(-0.33333333, 0.00000000)$$
$$(0.00000000, 1.41421356)$$
$$(0.00000000, -1.41421356)$$

$$z^{10} + z^9 + 3z^8 + 2z^7 - z^6 - 3z^5 - 11z^4 - 8z^3 - 12z^2 - 4z - 4 = 0$$

Miejsca zerowe: $(\Re, \Im)$

$$(0.00001593, -0.99997605)$$
$$(0.00001105, 0.99997767)$$
$$(-0.00001104, 1.00002233)$$
$$(-0.50000000, -0.86602540)$$
$$(0.00000000, -1.41421356)$$
$$(-0.50000000, 0.86602540)$$

$$(-0.00001593, -1.00002395)$$
$$(1.41421356, 0.00000000)$$
$$(0.00000000, 1.41421356)$$
$$(-1.41421356, 0.00000000)$$

$$z^4 + iz^3 - z^2 - iz + 1 = 0$$

Miejsca zerowe: $(\Re, \Im)$

$$(0.95105652, 0.30901699)$$
$$(0.58778525, -0.80901699)$$
$$(-0.95105652, 0.30901699)$$
$$(-0.58778525, -0.80901699)$$