Metody numeryczne

Wojciech Chrobak

2 stycznia 2018

Zadanie 11

Algorytm siecznych

Punktem wyjścia są dowolne dwa punkty, dla których $f(x_1) \neq f(x_2)$. Prowadzimy sieczną przez te punkty (bez względnu na znak $f(x_1) \cdot f(x_2)$), i jako x_3 bierzemy miejsce zerowe tej siecznej, dane wzorem:

$$x_3 = \frac{f(x_1)x_2 - f(x_2)x_1}{f(x_1) - f(x_2)}$$

W kolejnych krokach bierzemy zawsze dwa ostatni punkty, bez względu na to, czy funkcja zmienia znak.

Interpolacja odwrotna

Interpolacja odwrotna jest algorytmem poszukiwania rozwiązań nieliniowych równań algebraicznych. Idea algorytmu jest prosta. Skroro znamy funkcję to stwórzmy tabelę zawierającą kilka węzłów oraz odpowiadające im wartości w n miejscach:

x_1	x_2	...	x_n
$f(x_1)$	$f(x_2)$	...	$f(x_n)$

Jeśli stabelaryzowane wartości są ściśle monotoniczne oznacza to dla nas, że funkcja jest odwracalna, czyli możemy zamienić miejscami wartości oraz węzły.

$f(x_1)$	$f(x_2)$	...	$f(x_n)$
x_1	x_2	...	x_n

Interpolujmy więc funkcję odwrotną, znajdźmy jej wartość w 0, a wartość ta będzie szukanym przez nas miejscem zerowym.

Kod

```
#include <iostream>
#include <math.h>
#include <iomanip>
#include <vector>

using namespace std;

double f(double x) {
    return (x * x - 1) * pow(sinh(x), 3);
}

```

```
double lagrange(vector<double> x, vector<double> y, double val) {
    double t;
    double res = 0.0;

    for (int k = 0; k < x.size(); k++) {
        t = 1.0;
        for (int j = 0; j < x.size(); j++) {
            if (j != k) {
                t = t * ((val - x[j]) / (x[k] - x[j]));
            }
        }
        res += t * y[k];
    }
    return res;
}

vector<double> interpolacjaOdwrotna(double x1, double x2, double x3) {
    double x0 = x1;
    vector<double> x, y;
    x.resize(3);
    y.resize(3);

    vector<double> res;

    int counter = 1;
    while (true) {
        x[0] = f(x1);
        x[1] = f(x2);
        x[2] = f(x3);

        y[0] = x1;
        y[1] = x2;
        y[2] = x3;
        x0 = lagrange(x, y, 0.0);

        x1 = x2;
        x2 = x3;
        x3 = x0;

        if(isnan(x0)){
            cout << "[INTERPOLACJA ODWROTNA] Dzielenie przez 0" << endl;
            break;
        }
        if (abs(x0) < 1e-8)
            break;

        counter++;
    }


    res.push_back(x0);
    res.push_back((double) counter);

    return res;
}

vector<double> sieczne(double x1, double x2) {
    double x0;
    vector<double> res;

    if (abs(f(x1) - f(x2)) < 1e-8) {
        cout << "Wartosci w punktach startowych sa takie same! Blad.\n";
        return res;
    }
```

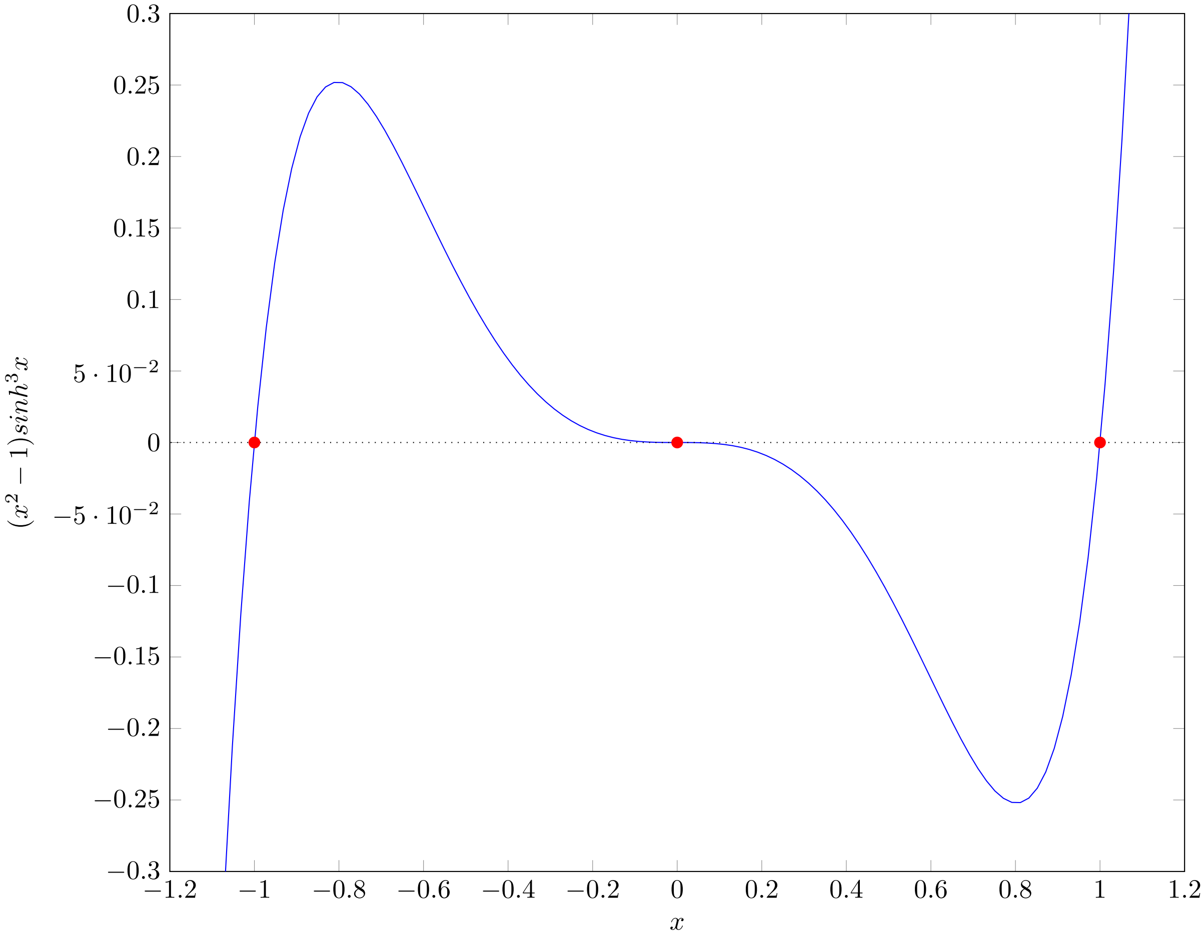
```
    double xm;
    int counter = 1;
    do {
        x0 = (x1 * f(x2) - x2 * f(x1)) / (f(x2) - f(x1));

        double c = f(x1) * f(x0);

        x1 = x2;
        x2 = x0;

        if (c == 0) {
            cout << "[SIECZNE]                Dzielenie przez 0" << endl;
            return res;
        }

        xm = (x1 * f(x2) - x2 * f(x1)) / (f(x2) - f(x1));

        counter++;

    } while (abs(xm - x0) >= 1e-8);


    res.push_back(x0);
    res.push_back((double) counter);

    return res;
}

int main() {
    cout.setf(ios::fixed, ios::floatfield);
    cout.precision(16);
    srand(time(NULL));

    double x1, x2, x3;
    vector<double> temp;
    vector<vector<double>> punkty;
    for (int j = 0; j < 10; ++j) {
        temp.clear();

        x1 = (rand() % 100) / 100.0;
        do {
            x2 = (rand() % 100) / 100.0;
        } while(x2 == x1);

        do {
            x3 = (rand() % 100) / 100.0;
        } while(x3 == x2);


        temp.push_back(x1);
        temp.push_back(x2);
        temp.push_back(x3);

        punkty.push_back(temp);
    }

    for (int i = 0; i < 10; ++i) {
        double t = sieczne(punkty[i][0], punkty[i][1])[0];
        double k = sieczne(punkty[i][0], punkty[i][1])[1];
        cout << setprecision(2) << punkty[i][0] << " & " << punkty[i][1] << " & " << setprecision(16)<< t << " = "  << setprecision(2) << t << setprecision(0) << " & " << k << " \\\\" << endl;
    }
```

```cpp

    cout << endl << endl;
    for (int i = 0; i < 10; ++i) {
        double t = interpolacjaOdwrotna(punkty[i][0], punkty[i][1], punkty[i][2])[0];
        double k = interpolacjaOdwrotna(punkty[i][0], punkty[i][1], punkty[i][2])[1];
        cout << setprecision(2) << punkty[i][0] << " & " << punkty[i][1] << " & " << punkty[i][2] << " & " << setprecision(16) << t << " = " << setprecision(2) << t << setprecision(0) << " & " << k << " \\\\" << endl;

    }

    return 0;
}
```

Wynik

Równanie

$$(x^2 - 1)sinh^3x = 0$$



Algorytm siecznych

x_1	x_2	Miejsce zerowe	k
0.92	0.15	0.0000000323984951 = 0.00	56
0.60	0.02	0.0000000331693375 = 0.00	49
0.05	0.53	0.0000000351214288 = 0.00	53
0.11	0.03	0.0000000362566443 = 0.00	50
0.15	0.51	0.0000000406728323 = 0.00	56
0.85	0.94	1.0000000000034168 = 1.00	8
0.62	0.38	0.0000000399183367 = 0.00	58
0.70	0.25	0.0000000392922814 = 0.00	57
0.90	0.60	0.0000000349235987 = 0.00	50
0.51	0.22	0.0000000346125516 = 0.00	57

Interpolacja odwrotna

x_1	x_2	x_3	Miejsce zerowe	k
0.92	0.15	0.03	0.0000000073775620 = 0.00	47
0.60	0.02	0.76	0.0000000074993314 = 0.00	46
0.05	0.53	0.49	0.0000000088082513 = 0.00	49
0.11	0.03	0.58	0.0000000078072529 = 0.00	47
0.15	0.51	0.80	0.0000000087939196 = 0.00	52
0.85	0.94	0.54	0.0000000087213359 = 0.00	58
0.62	0.38	0.67	0.0000000076300398 = 0.00	54
0.70	0.25	0.13	0.0000000077037246 = 0.00	51
0.90	0.60	0.01	0.0000000093412881 = 0.00	43
0.51	0.22	0.89	0.0000000074500751 = 0.00	53

Obie metody zbiegają w mniej wiecej tym samym czasie (ponieważ interpolacje odwrotną startujemy z 3 punktow), ale algorytm siecznych nie zawsze zbiega do miejsca zerowego. Często występuje w nim błąd dzielenia przez 0 przez co algorytm kończy działanie i daje wynik, który nie jest miejscem zerowym.

Intepolacja odwrotna jest opłacalna, jeśli stosujemy ją na małych ilościach węzłów. Często wykorzystywana jest jako punkt startowy dla innych metod.