Interpolacja

Jakub Tkacz

11.04.2019

Spis treści

1 Wstęp **3**
1.1 Klasa Poly . . . 3
1.2 Klasa Term . . . 3

2 Interpolacja Lagrange'a **5**
2.1 Wstęp . . . 5
2.1.1 Wzór na interpolację . . . 5
2.2 Wykonanie . . . 5
2.2.1 Kod . . . 5
2.3 Wyniki . . . 6
2.3.1 Opracowanie w R . . . 6
2.3.2 Wykres . . . 6

3 Interpolacja metodą Newtona **7**
3.1 Wstęp . . . 7
3.1.1 Wzór na interpolację . . . 7
3.1.2 Wyliczanie współczynników wielomianu . . . 7
3.2 Wykonanie . . . 7
3.3 Wyniki . . . 8

4 Porównanie metod Lagrange'a, Newton'a i implementacji GSL **9**
4.1 Wyliczenie wielomianu za pomocą GSL . . . 9
4.2 Wykres . . . 9
4.3 Czas wykoniania . . . 10

5 Funkcje sklejane **11**
5.1 Wykres . . . 11

6 Porównanie z interpolacją wielomianową **12**

7 Efekt Rungego **13**
7.1 Ilustracja efektu Rungego . . . 13

1 Wstęp

Program realizujący zadania z labolatorium 5 postanowiłem napisać w C++.
Do reprezentacji wielomianów stworzyłem własną klasę Poly, która implementowała również podstawowe operacje matematyczne takie jak (dodawanie, mnożenie, dzielenie). Dzięki takiemu podejściu kod funkcji liczącej wielomian stał się dużo czytelniejszy.
Z uwagi na prostotę rysowania wykresów w R, wszystkie dane eksportowałem do plików csv, a następnie za pomocą pakietu ggplot rysowałem wykresy.

1.1 Klasa Poly

Klasa Poly zawiera dwa pola: listę dwumianów (Term) oraz stopień wielomianu. Metody w niej zawarte to mnożenie wielomiany przez wielomian, dodawanie wielomianów, dzielenie i mnożenie przez stałą.

Listing 1: Klasa Poly

```
class Poly {
private:
    std::list<Term> terms;
    int degree;
public:
    Poly();

    Poly(const std::list<Term> &terms);

    Poly operator*(Poly const &obj);

    Poly operator+(Poly const &obj);

    Poly operator/(double const &d);

    Poly operator*(double const &d);

    double calculate(double x) {
        double ret = 0.0;
        for (auto term: terms)ret += term.calculate(x);
        return ret;
    }

    std::string toString();
};
```

1.2 Klasa Term

Dodatkową pomocniczą klasą jest Term, reprezentujący dwumian w postaci $a{\cdot}x^n$

Listing 2: Klasa Term

```
class Term {
private:
    double coefficient;
    int exponent;
```

```
public:
    Term(double coefficient, int exponent);

    Term();

    Term add(double coefficient);

    double getCoefficient();

    int getExponent();

    double calculate(double x) {
        return coefficient * pow(x, exponent);
    }

    std::string toString();
};
```

2 Interpolacja Lagrange'a

2.1 Wstęp

2.1.1 Wzór na interpolację

$$W(x) = \sum_{i=0}^{N} y_i L_i(x)$$

$$L_i(x) = \prod_{k=0,k\neq i}^{N} \frac{x - x_k}{x_i - x_k}$$

2.2 Wykonanie

Kod funkcji wykonującej interpolację Lagrange'a jest dosłowną realizacją matematycznych wzorów w kodzie programu.

2.2.1 Kod

Listing 3: Interpolacja metodą Lagrange'a

```
1  Poly lagrangeInterpolation(std::vector<Node> nodes) {
2      Poly L[nodes.size()];
3      Poly W({Term(0, 0)});
4      for (int i = 0; i < nodes.size(); i++) {
5          Poly num({Term(1, 0)});
6          double denom = 1.0;
7          for (int k = 0; k < nodes.size(); k++) {
8              if (k != i) {
9                  Poly b({Term(1, 1), Term(-nodes[k].getX(), 0)});
10                 num = num * b;
11                 denom *= nodes[i].getX() - nodes[k].getX();
12             }
13         }
14         L[i] = num / denom;
15     }
16     W = L[0] * nodes[0].getY();
17     for (int i = 1; i < nodes.size(); i++) {
18         W = W + L[i] * nodes[i].getY();
19     }
20     return W;
21 }
```

Na początku działania funkcji deklaruje wielomian wynikiowy $W(x) = 0$.
Następnie iteruje i obliczam watości wielomianów pomocniczych $L_i(x) = \prod_{k=0,k\neq i}^{N} \frac{x-x_k}{x_i-x_k}$
Pojawia się tu deklaracja dwóch zmiennych num - licznik będący wielomianem oraz mianownik który jest wartością rzeczywistą.
Pętla for z 7 linii zajmuje się obliczeniem iloczynu będącego wartością $L_i(x)$.
Zadeklarowany w 9 lini wielomian pomocniczy ma wartość $b(x) = x - x_i$ i jest pojedynczym licznikiem w iloczynie.

Ostatnim krokiem jest obliczenie wartości wielomianów L i ich zsumowanie $W(x) = \sum_{i=0}^{N} y_i L_i(x)$

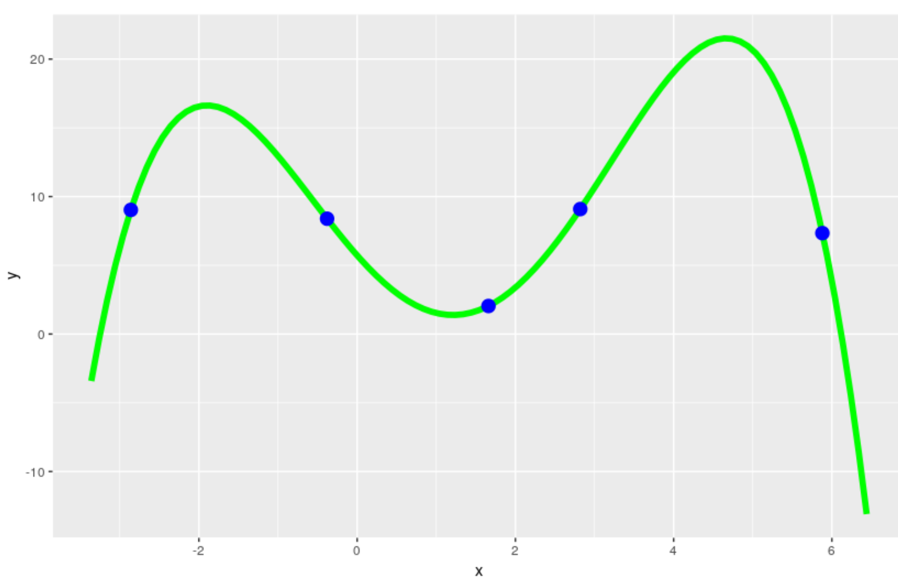
2.3 Wyniki

2.3.1 Opracowanie w R

Listing 4: Rysowanie wykresu wielomianu

```
dataLagrange = read.csv("Lagrange.csv")
points = read.csv("points.csv")
ggplot()+
  xlab("x") +
  geom_line(data=dataLagrange, aes(x, y), colour="green", size=2)+
  ylab("y") +
  geom_point(data=points, aes(x, y), colour="blue",size=4)
```

2.3.2 Wykres



Rysunek 1: Wykres interpolacji metodą Lagrangea

3 Interpolacja metodą Newtona

3.1 Wstęp

3.1.1 Wzór na interpolację

Wielomian Newtona dla $n+1$ punktów przybiera postać:

$$W(x) = a_0 + \sum_{i=1}^{n} a_i \prod_{j=0}^{i-1}(x - x_j) =$$
$$a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)\ldots(x - x_{n-1})$$

$$\begin{aligned} W(x_0) &= \quad a_0 \\ W(x_1) &= \quad a_0 + a_1(x_1 - x_0) \\ & \quad\Downarrow \\ a_1 &= \quad \tfrac{y_1 - y_0}{x_1 - x_0} \end{aligned}$$

Iloraz różnicowy dzielony

$$\begin{aligned} f[x_i] &= f(x_i) \\ f[x_i, \ldots, x_{i+j+1}] &= \tfrac{f[x_{i+1}, \ldots, x_{i+j+1}] - f[x_i, \ldots, x_{i+j}]}{x_{i+j+1} - x_i} \end{aligned}$$

3.1.2 Wyliczanie współczynników wielomianu

$a_0 = f[x_0] = y_0$
$a_1 = f[x_0, x_1]$
$\vdots$
$a_i = f[x_0, \cdots, x_i]$

3.2 Wykonanie

Tak jak w przypadu metody Lagrange'a, kod realizujący tą metodę jest odzwierciedleniem powyżej opisanych metod matematycznych. W celu optymalizacji zastosowałem programowanie dynamiczne do wyliczenia funkcji F

Listing 5: Interpolacja metodą Newtona

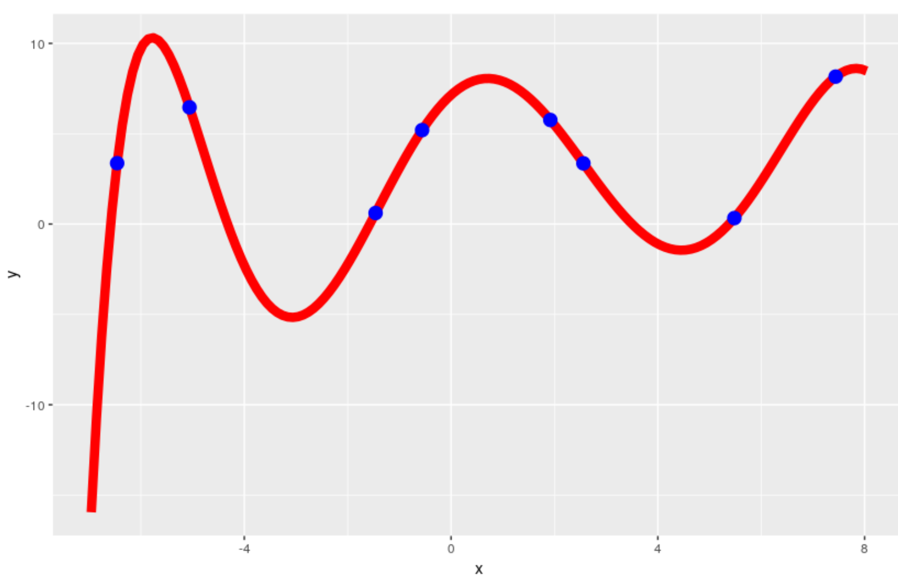
```
Poly newtonInterpolation(std::vector<Node> nodes) {
    Poly W({Term{F(0, 0, nodes), 0}});
    for (int i = 1; i < nodes.size(); i++) {
        Poly C({Term(F(0, i, nodes), 0)});
        for (int j = 0; j < i; j++) {
            C = C * Poly({Term(1, 1), Term(-nodes[j].getX(), 0)});
        }
        W = W + C;
    }
    return W;
```

```
11 }
12
13 double F(int a, int b, std::vector<Node> nodes) {
14     if (!isSet[a][b]) {
15         if (a == b) {
16             _F[a][b] = nodes[a].getY();
17         } else {
18             _F[a][b] = (F(a + 1, b, nodes) - F(a, b - 1, nodes)) /
                   (nodes[b].getX() - nodes[a].getX());
19         }
20         isSet[a][b] = true;
21     }
22     return _F[a][b];
23 }
```

Funkcja wyznaczająca wielomian metodą Newtona zaczyna swoje działanie od deklaracji wielomiany $W = F[x_0] = a_0$.
W liniach 4-7 wyliczam wartość poszczególnych mniejszych wielomianów będących częścią sumy $C(x) = a_i \prod_{j=0}^{i-1}(x - x_j)$

3.3 Wyniki

Kod w języku R do rysowania wykresu jest analogiczny jak w przypadku omawiania metody Lagrange'a.



Rysunek 2: Wykres interpolacji metodą Newtona

4 Porównanie metod Lagrange'a, Newton'a i implementacji GSL

4.1 Wyliczenie wielomianu za pomocą GSL

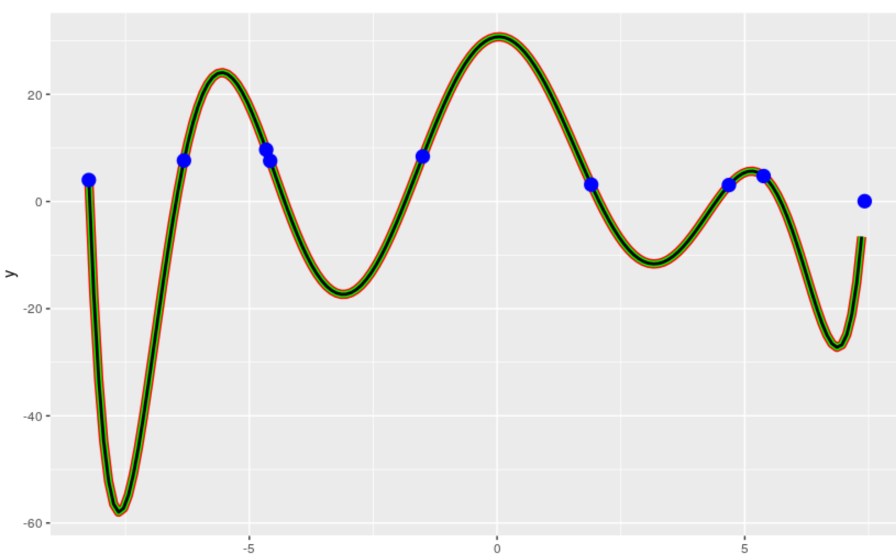
Listing 6: Interpolacja za pomocą GSL

```
1     gsl_interp *interp = gsl_interp_alloc(gsl_interp_polynomial,
          nodes.size());
2     gsl_interp_init(interp, xArr, yArr, nodes.size());
3     gsl_interp_accel *accel = gsl_interp_accel_alloc();
4     for (double x = minX; x <= maxX; x += 0.1) {
5         gsl_interp_eval(interp, xArr, yArr, x, accel);
6     }
```

Z powyższego kodu zostały usunięte: zapis do pliku oraz mierzenie czasu. Interpolacja wielomianu odbywa się w drugiej linii. W lini 6 obliczam wartości dla poszczególnych punktów. Pozostały kod to deklaracje zmiennych pomocniczych.

4.2 Wykres

Z uwagi na zastosowanie funkcji z biblioteki gsl, wykres generowałem tylko na przedziale od pierwszego do ostatniego punktu włącznie.



Rysunek 3: Wykres interpolacji (czerowny - newton, zielony - lagrange, czarny - gsl)

Analizując wykres, pierwszą rzeczą która zwraca uwagę jest pokrywanie się wykresów wyliczonych funkcji. Wynika to z twierdzenia o jednoznaczności interpolacji wielominaowej - istnieje tylko jeden wielomian interpolujący funkcję w zadnaych punktach.

4.3 Czas wykoniania

Analizę czasów wykonania przeprowadziłem w języku R.

Listing 7: Analiza czasów wykonań w R

```
data = read.csv("times.csv")
analysis = data.frame(
    newton_mean = apply(data["newton"],2,mean),
    lagrange_mean = apply(data["lagrange"],2,mean),
    gsl_mean = apply(data["gsl"],2,mean),
    newton_std = apply(data["newton"],2,sd),
    lagrange_std = apply(data["lagrange"],2,sd),
    gsl_std = apply(data["gsl"],2,sd)
)
```

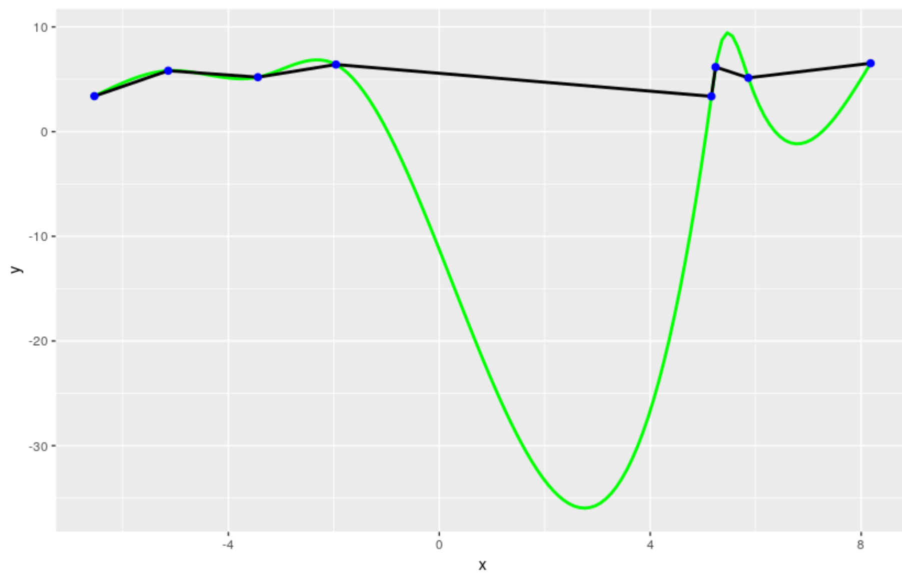
	Średnia	Odchylenie
Lagrange	0.0002128772	0.0001499118
Newton	0.0001070180	0.0001181048
GSL	3.3451e-07	5.006755e-07

Otrzymane wyniki pokazują, że fukcja interpolująca wielomian w bibliotece GSL jest o 3 rzędy wielkości szybsza niż funcje zaimplementowane prze ze mnie. Może to wynikać z zastosowanych w nim nie najwydajnieszych oraz rozbudowanych struktur danych.

5 Funkcje sklejane

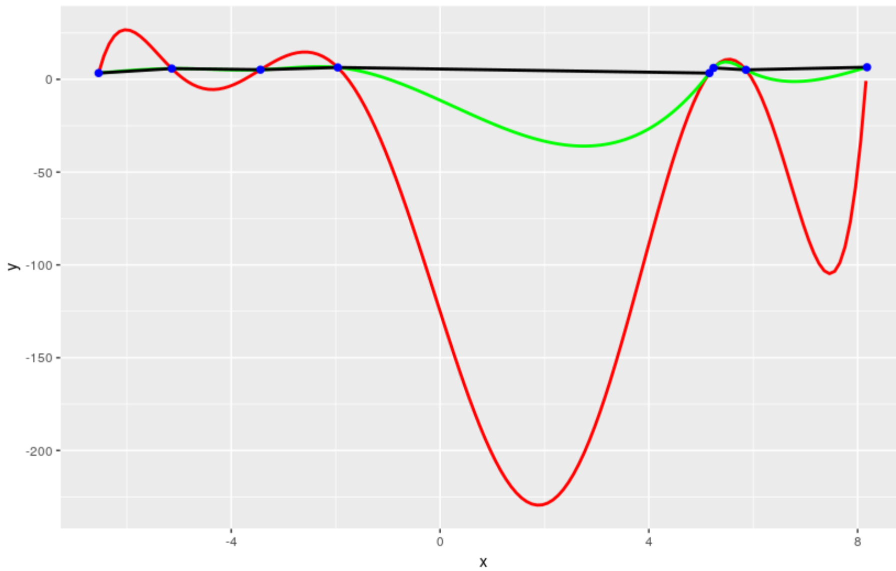
Do interpolacji funkcjami sklejanymi urzyłem modelu cubic z pakietu GSL oraz funkcji liniowej.

5.1 Wykres



Rysunek 4: Wykres funkcji sklejanych (zielony - cubic, czarny - liniowa)

6 Porównanie z interpolacją wielomianową



Rysunek 5: Porównanie funkcji sklejanych i wielomianów (czerwony - wielomian, zielony - cubic, czarny - liniowa)

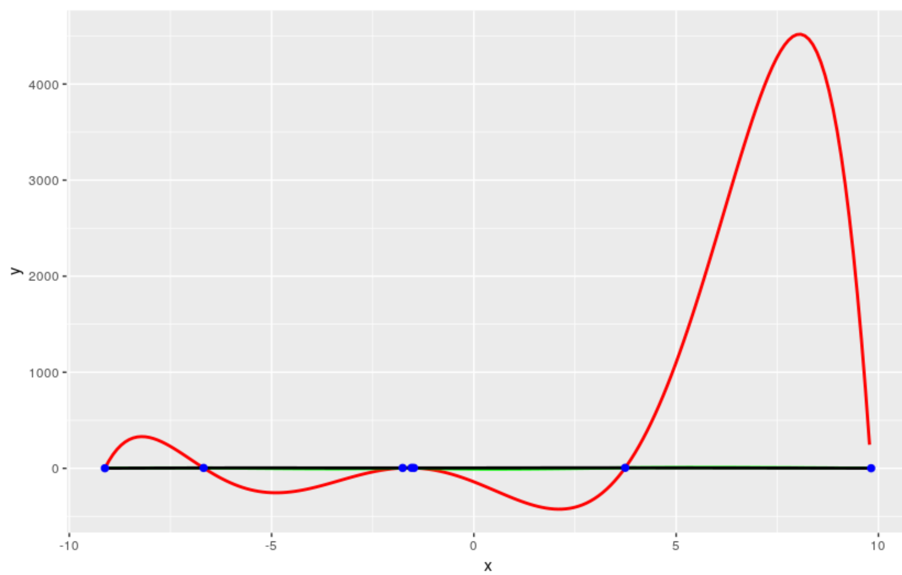
Na wykresie 5 żemy zaobserwować że przy interpolacji funkcjami sklejanymi występują dużo mniejesze ekstrema niż przy interoplacji tych samych punktów np. metodą Newtona.

7 Efekt Rungego

Efekt Rungego występuje gdy mamy zbyt dużą liczbę równo odległych punktów. Następuje wtedy zbyt duże dopasowanie wilomianu, przez co poza punktami przyjmuje duże ekstrema.

7.1 Ilustracja efektu Rungego

W celu ilustracji efektu Rungego na jednym wykresie umieściłem funkcję sklejaną oraz wykres wielomianu otrzymanego z metody Newtona



Rysunek 6: Efekt Rungego (czerwony - wielomian, czarny - funkcja sklejana)

Możemy zauważyć iż między punktami 5 i 10 wielomian przyjmuje maksimum. Patrząc na oś y widimy że jest ono ok 100 krotnie większe niż analogiczne otrzymane za pomocą funkcji sklejanych.