

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 wykonania	9
5	Funkcje sklejane	11
5.1	Wykres	11
6	Porównanie z interpolacją wielomianową	12
7	Efekt Rungego	13
7.1	Ilustracja efektu Rungego	13
8	Grafika komputerowa	13
8.1	Interpolacja najbliższego sąsiada	14
8.2	Interpolacja liniowa	15
8.3	Interpolacja kubiczna	16

1 Wstęp

Program realizujący zadania z laboratorium 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

```
1 class Poly {
2 private:
3     std::list<Term> terms;
4     int degree;
5 public:
6     Poly();
7
8     Poly(const std::list<Term> &terms);
9
10    Poly operator*(Poly const &obj);
11
12    Poly operator+(Poly const &obj);
13
14    Poly operator/(double const &d);
15
16    Poly operator*(double const &d);
17
18    double calculate(double x) {
19        double ret = 0.0;
20        for (auto term: terms) ret += term.calculate(x);
21        return ret;
22    }
23
24    std::string toString();
25 };
```

1.2 Klasa Term

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

Listing 2: Klasa Term

```
1 class Term {
2 private:
3     double coefficient;
4     int exponent;
```

```

5  public:
6      Term(double coefficient, int exponent);
7
8      Term();
9
10     Term add(double coefficient);
11
12     double getCoefficient();
13
14     int getExponent();
15
16     double calculate(double x) {
17         return coefficient * pow(x, exponent);
18     }
19
20     std::string toString();
21 };

```

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

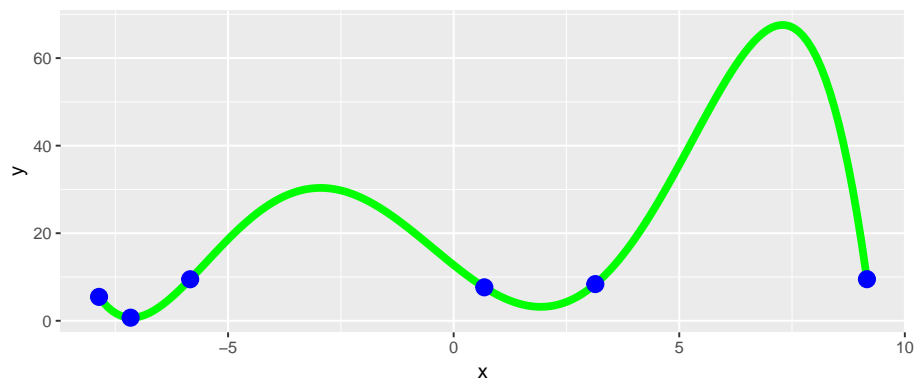
2.3 Wyniki

2.3.1 Opracowanie w R

Listing 4: Rysowanie wykresu wielomianu

```
1 dataLagrange = read.csv("Lagrange.csv")
2 points = read.csv("points.csv")
3 ggplot()+
4   xlab("x") +
5   geom_line(data=dataLagrange, aes(x, y), colour="green", size=2)+
6   ylab("y") +
7   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:

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

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

Iloraz różnicowy dzielony

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

3.1.2 Wyliczanie współczynników wielomianu

$$\begin{aligned} a_0 &= f[x_0] = y_0 \\ a_1 &= f[x_0, x_1] \\ &\vdots \\ a_i &= f[x_0, \dots, x_i] \end{aligned}$$

3.2 Wykonanie

Tak jak w przypadku 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

```
1 Poly newtonInterpolation(std::vector<Node> nodes) {
2     Poly W({Term{F(0, 0, nodes), 0}});
3     for (int i = 1; i < nodes.size(); i++) {
4         Poly C({Term{F(0, i, nodes), 0}});
5         for (int j = 0; j < i; j++) {
6             C = C * Poly({Term(1, 1), Term(-nodes[j].getX(), 0)});
7         }
8         W = W + C;
9     }
10    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)) /
19                         (nodes[b].getX() - nodes[a].getX());
19         }
20         isSet[a][b] = true;
21     }
22     return _F[a][b];
23 }

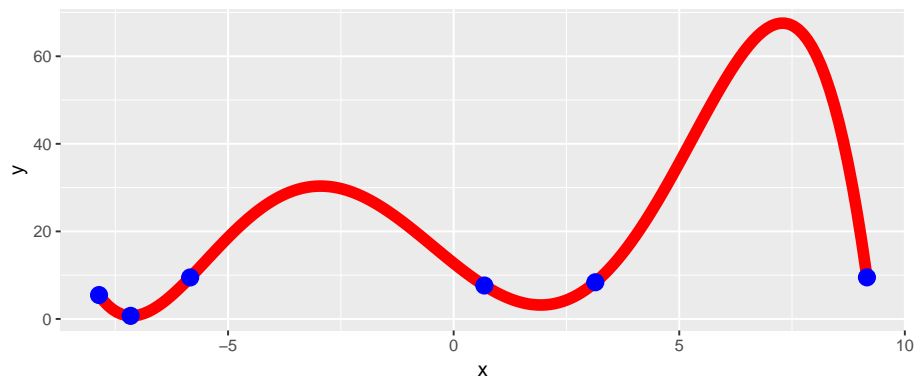
```

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

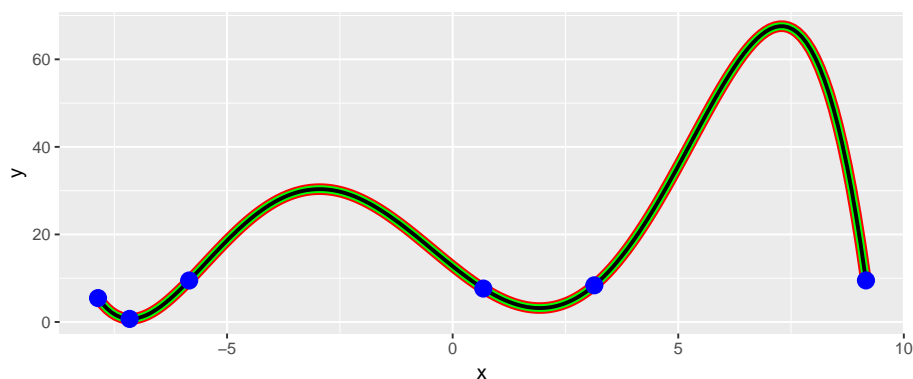
Listing 6: Interpolacja za pomocą GSL

```
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 (czerwony - 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 wielomianowej - istnieje tylko jeden wielomian interpolujący funkcję w zadanych punktach.

4.3 Czas wykonania

Analizę czasów wykonania przeprowadziłem w języku R.

Listing 7: Analiza czasów wykonania w R

```

1 data = read.csv("times.csv")
2 analysis = data.frame(
3   newton_mean = apply(data["newton"], 2, mean),
4   lagrange_mean = apply(data["lagrange"], 2, mean),
5   gsl_mean = apply(data["gsl"], 2, mean),
6   newton_std = apply(data["newton"], 2, sd),
7   lagrange_std = apply(data["lagrange"], 2, sd),
8   gsl_std = apply(data["gsl"], 2, sd)
9 )

```

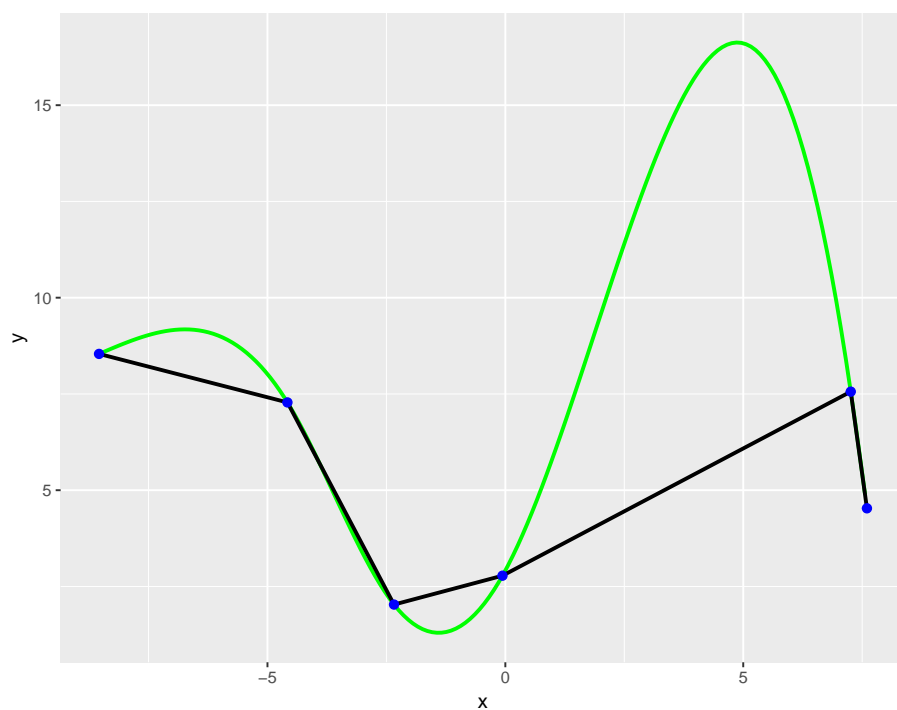
	Średnia	Odchylenie
Lagrange	0.0002128772	0.0001499118
Newton	0.0001070180	0.0001181048
GSL	3.3451e-07	5.006755e-07

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

5 Funkcje sklejane

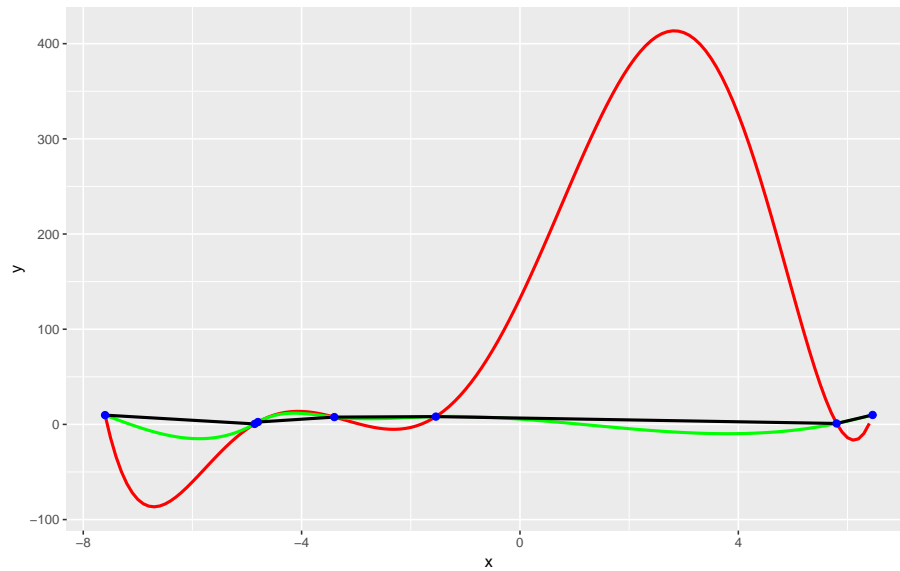
Do interpolacji funkcjami sklejanyymi użyłem modelu cubic z pakietu GSL oraz funkcji liniowej.

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)

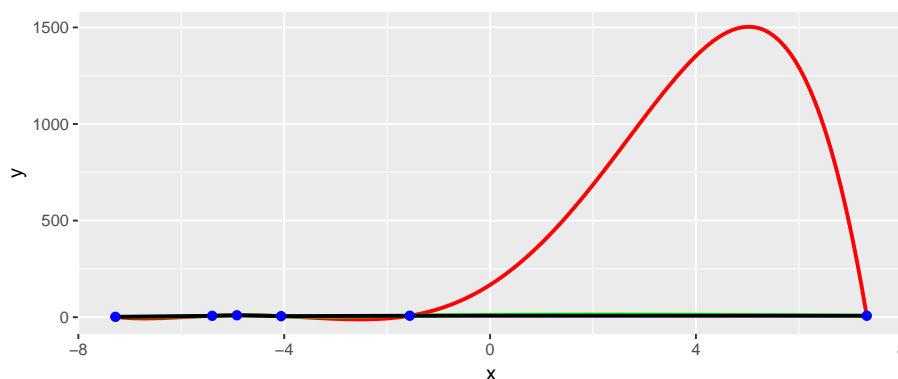
Na wykresie 5 żemy zaobserwować że przy interpolacji funkcjami sklejanyimi występują dużo mniejsze ekstremum niż przy interpolacji tych samych punktów np. metodą Newtona.

7 Efekt Rungego

Efekt Rungego występuje gdy mamy zbyt dużą liczbę równo odległych punktów. Następuje wtedy zbyt duże dopasowanie wielomianu, 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 widzimy że jest ono ok 100 krotnie większe niż analogiczne otrzymane za pomocą funkcji sklejanych.

8 Grafika komputerowa

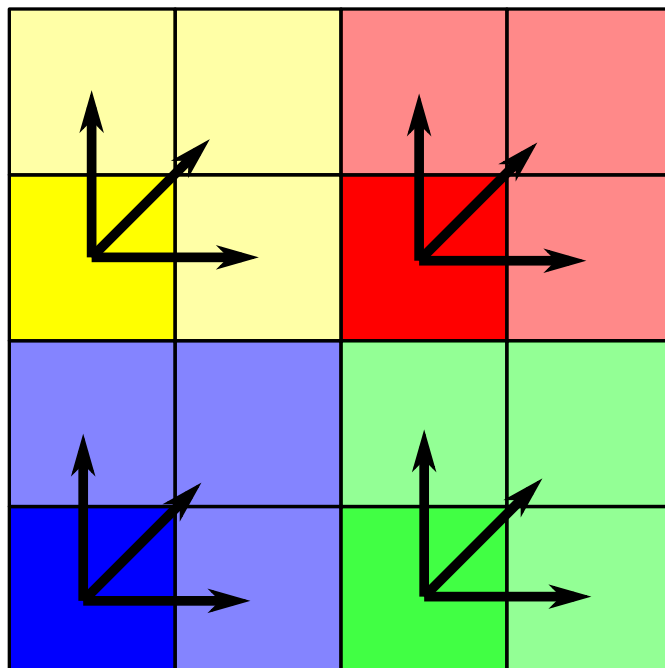
Interpolacja jest często wykorzystywana przy skalowaniu obrazów. Przy ilustracji działania każdej z metod skalowałem obraz lwa.



Rysunek 7: Obraz bazowy (100x90)

8.1 Interpolacja najbliższego sąsiada

Metoda najbliższego sąsiada polega na kopiowaniu wartości pixela z pixela sąsiedniego



Rysunek 8: Obrazowanie działania metody najbliższego sąsiada



Rysunek 9: Powiększenie metody najbliższego sąsiada

8.2 Interpolacja liniowa

Wartość piksela jest wyliczana z funkcji interpolującej dwa sąsiednie piksele



Rysunek 10: Powiększenie metody interpolacji liniowej

8.3 Interpolacja kubiczna

Działa analogicznie jak liniowa, tylko że wykorzystuje funkcje sześciennne. Obraz otrzymany tą metodą jest bardziej rozmaryny lecz wydaje się być bardziej naturalny.



Rysunek 11: Powiększenie metody interpolacji kubicznej