Metody obliczeniowe w nauce i technice

Sprawozdanie z zadania 5
Interpolacja

Bartosz Pietrzyk

Spis treści

1 Interpolacja wielomianem interpolacyjnym Lagrange'a **2**
1.1 **Wzory** . . . 2
1.2 **Przykładowa implementacja w języku Scala** . . . 2
1.3 **Wylosowane punkty przeznaczone do przykładowych interpolacji** . . . 3
1.4 **Interpolacja wygenerowana wielomianem interpolacyjnym Lagrange'a dla punktów z rysunku 1** . . . 4

2 Interpolacja wielomianem interpolacyjnym Newtona **5**
2.1 **Wzory** . . . 5
2.2 **Przykładowa implementacja w języku Scala** . . . 5
2.3 **Interpolacja wygenerowana wielomianem interpolacyjny Lagrange'a dla punktów z rysunku 1** . . . 6

3 Interpolacja wielomianowa z pakietu Polynomials z języka Julia **7**
3.1 **Przykładowe użycie** . . . 7
3.2 **Interpolacja wygenerowana przy pomocy pakietu Polynomials dla punktów z rysunku 1.** . . . 7

4 Porównanie interpolacji Newtona, Lagrange'a i z pakietu Polynomials **8**
4.1 **Wszystkie trzy interpolacje na jednym wykresie** . . . 8

5 Porównanie czasu obliczania interpolacji dla róznych algorytmów **9**
5.1 **Wyniki pomiarów** . . . 9

6 Interpolacja funkcją sklejaną pierwszego stopnia **10**
6.1 **Wykres** . . . 10

7 Efekt Runge'go **11**

1 Interpolacja wielomianem interpolacyjnym Lagrange'a

1.1 Wzory

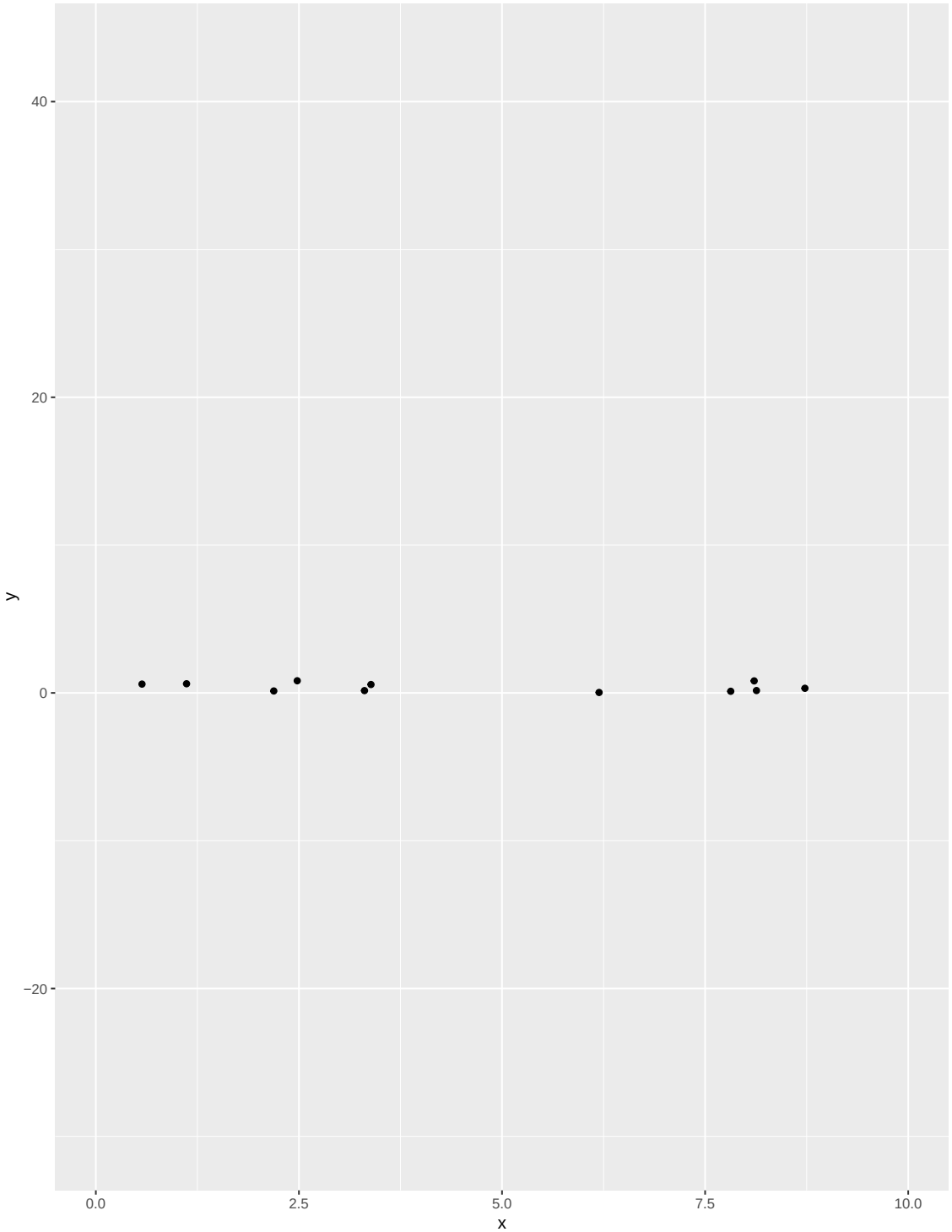
$$L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i} \qquad P_n(x) = \sum_{k=0}^{n} y_k L_k(x)$$

1.2 Przykładowa implementacja w języku Scala

```
object Interpolation {
 type Factor = List[Double => Double]
 type Interpolation = List[Factor]

 def lagrange(listX: List[Double], listY: List[Double]):
    Interpolation = {
  val n = listX.size
  def computeLi(i: Int, j: Int, first: Boolean): Factor = {
   (i, j) match {
    case _ if i == j =>
     computeLi(i, j + 1, first)
    case (_, a) if a == n =>
     List()
    case _ if first => ((x: Double) =>
     listY(i) * (x - listX(j)) / (listX(i) - listX(j)))
      :: computeLi(i, j + 1, !first)
    case _ => ((x: Double) =>
     (x - listX(j)) / (listX(i) - listX(j))) ::
      computeLi(i, j + 1, first)
   }
  }

  def recLagrange(i: Int): Interpolation = {
   i match {
    case _ if i == n => List()
    case _ => computeLi(i, 0, true) :: recLagrange(i + 1)
   }
  }

  recLagrange(0)
 }
}
```

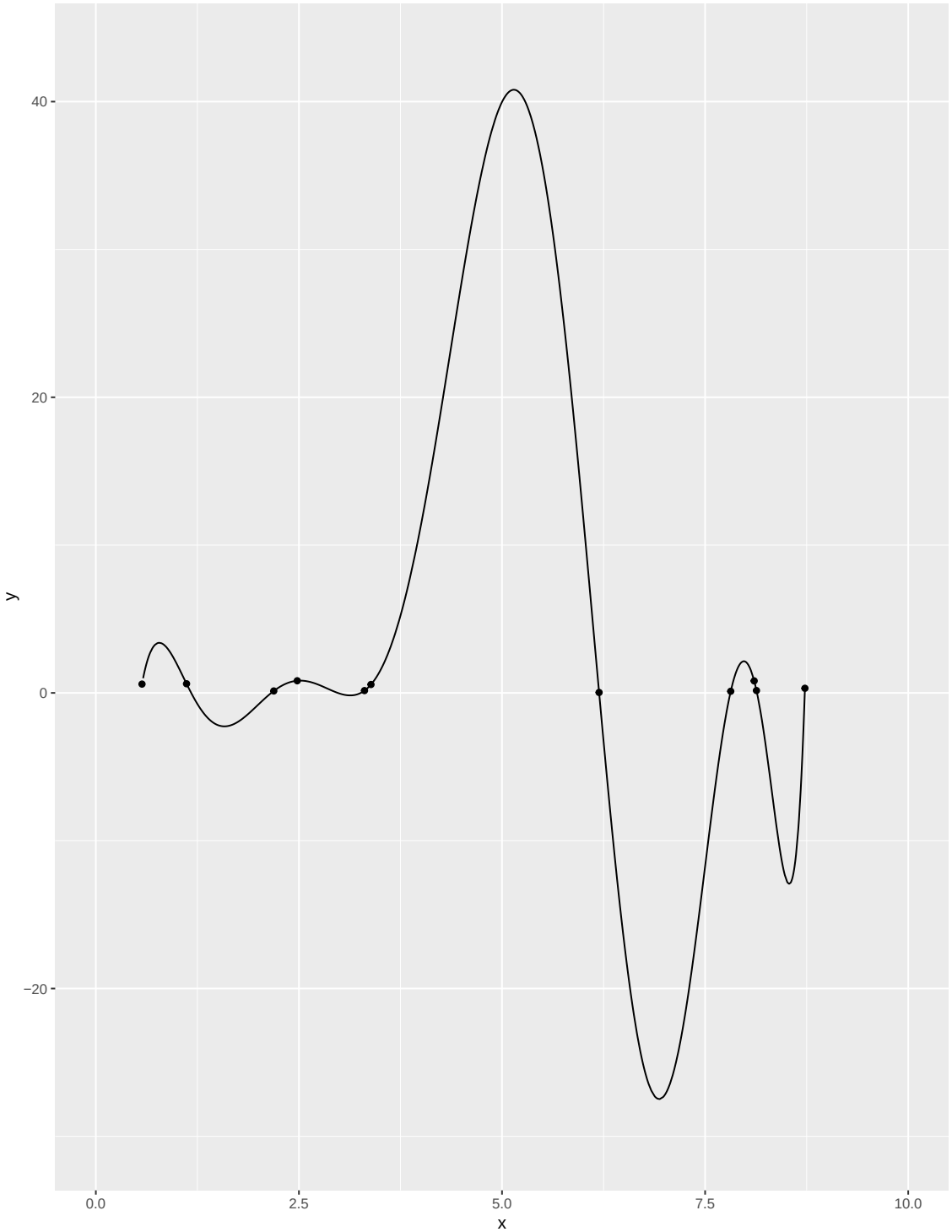
```
 def evalInterpolation(interpolation: Interpolation, x: Double):
    Double = {
  interpolation.map(yIlI => yIlI.map(subFun => subFun(x))
   .reduce(_ * _)).sum
 }
```

1.3 Wylosowane punkty przeznaczone do przykładowych interpolacji



Rysunek 1: Wylosowane punkty

1.4 Interpolacja wygenerowana wielomianem interpolacyjnym Lagrange'a dla punktów z rysunku 1



Rysunek 2: Interpolacja wielomianem Lagrange'a

2 Interpolacja wielomianem interpolacyjnym Newtona

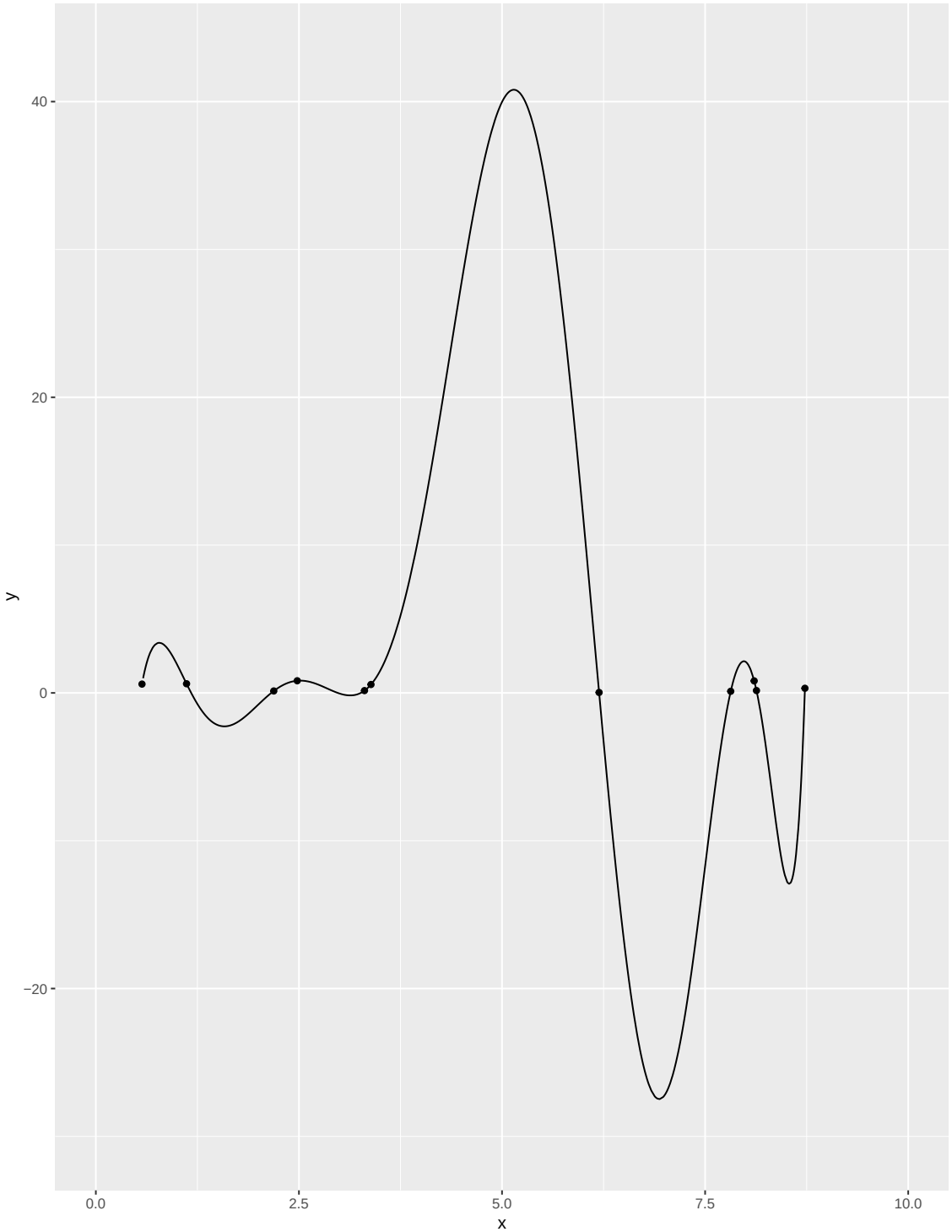
2.1 Wzory

$$P_k(x) = P_{k-1}(x) + c_k(x - x_0)(x - x_1)...(x - x_{k-1}) \qquad P_0(x) = c_0 = y_0$$

2.2 Przykładowa implementacja w języku Scala

```
object Interpolation {
 type Factor = List[Double => Double]
 type Interpolation = List[Factor]

 def evalInterpolation(interpolation: Interpolation, x: Double):
    Double = {
  interpolation.map(yIlI => yIlI.map(subFun => subFun(x))
.reduce(_ * _)).sum
 }

 def newton(listX: List[Double], listY: List[Double]):
    Interpolation = {
  val n = listX.size
  val p0: Interpolation = List(List((_: Double) => listY.head))

  def computePk(k: Int, cK: Double): Factor = {
    (listX.take(1).map(x => ((x: Double) => cK * x)
compose ((k: Double) => k - x))
     ++ listX.slice(1,k).map(x => (l: Double) => l - x))
  }

  @tailrec
  def recNewton(k: Int, acc: Interpolation): Interpolation = {
    if (k == n) acc
    else {
      val denominator: Double = listX.take(k)
    .map(x => listX(k) - x).reduce(_ * _)

val cK: Double =
     (listY(k) - evalInterpolation(acc, listX(k))) / denominator
      recNewton(k+1, computePk(k,cK) :: acc)
    }
  }
  recNewton(1, p0)
 }
}
```

2.3 Interpolacja wygenerowana wielomianem interpolacyjny Lagrange'a dla punktów z rysunku 1



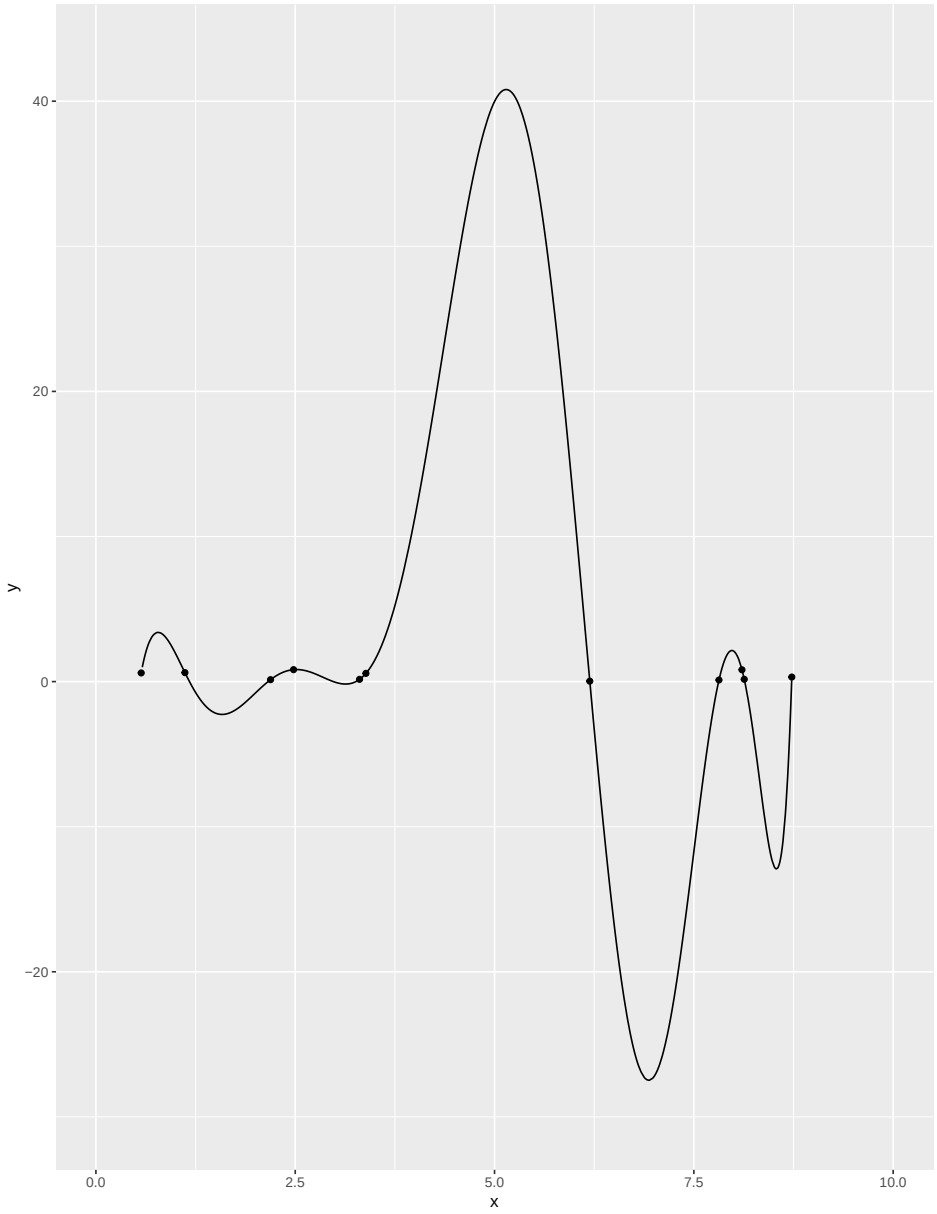
Rysunek 3: Interpolacja metodą Newtona

3 Interpolacja wielomianowa z pakietu Polynomials z języka Julia

3.1 Przykładowe użycie

```
using DataFrames
using CSV
df1=CSV.read("example_points.csv", delim=",")
xs = df1[:x]
A = df1[:y]
xsf=minimum(xs):0.01:maximum(xs)
using Polynomials
fit1=polyfit(xs, A)
B=[fit1(x) for x in xsf]
```

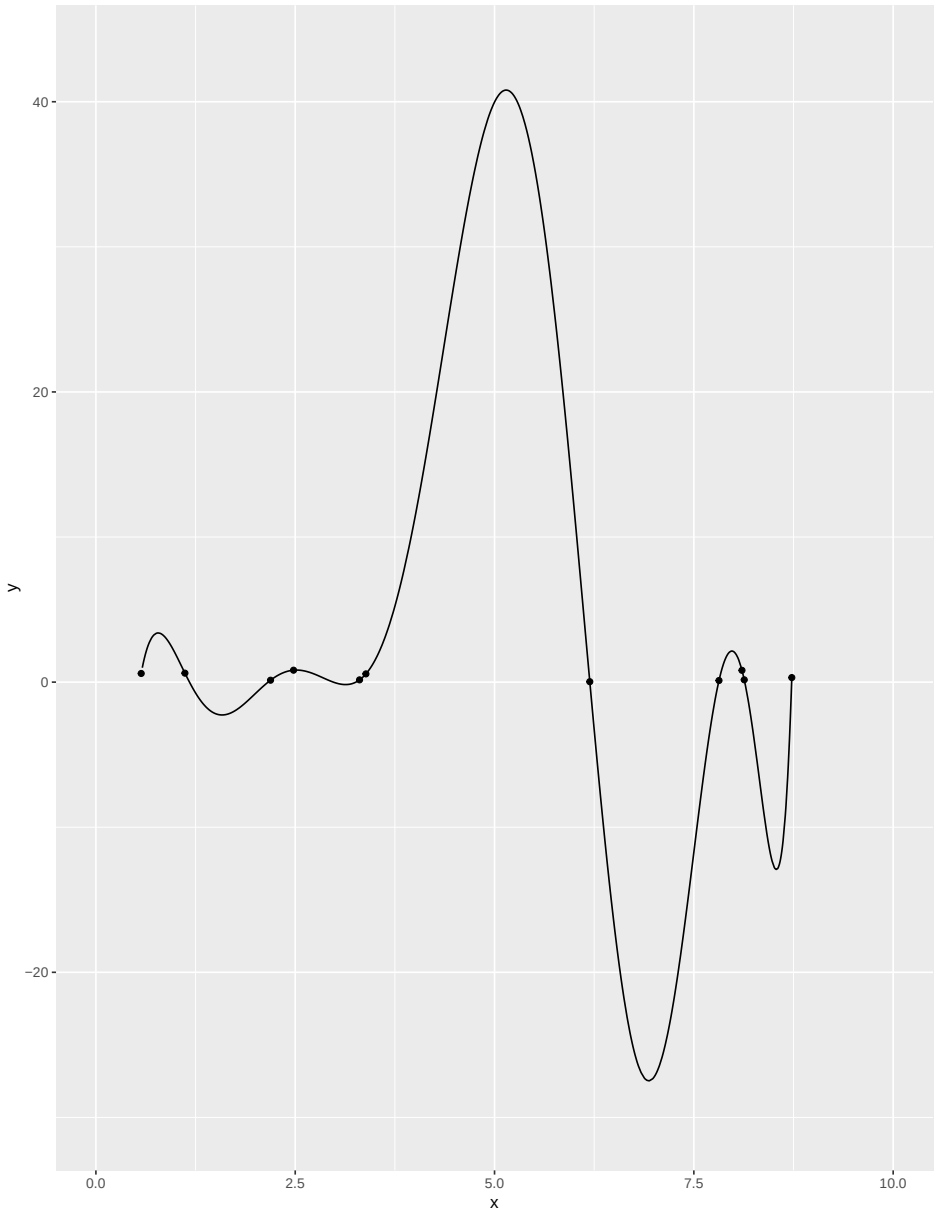
3.2 Interpolacja wygenerowana przy pomocy pakietu Polynomials dla punktów z rysunku 1.



Rysunek 4: Interpolacja z pakietu Polynomials

4 Porównanie interpolacji Newtona, Lagrange'a i z pakietu Polynomials

4.1 Wszystkie trzy interpolacje na jednym wykresie

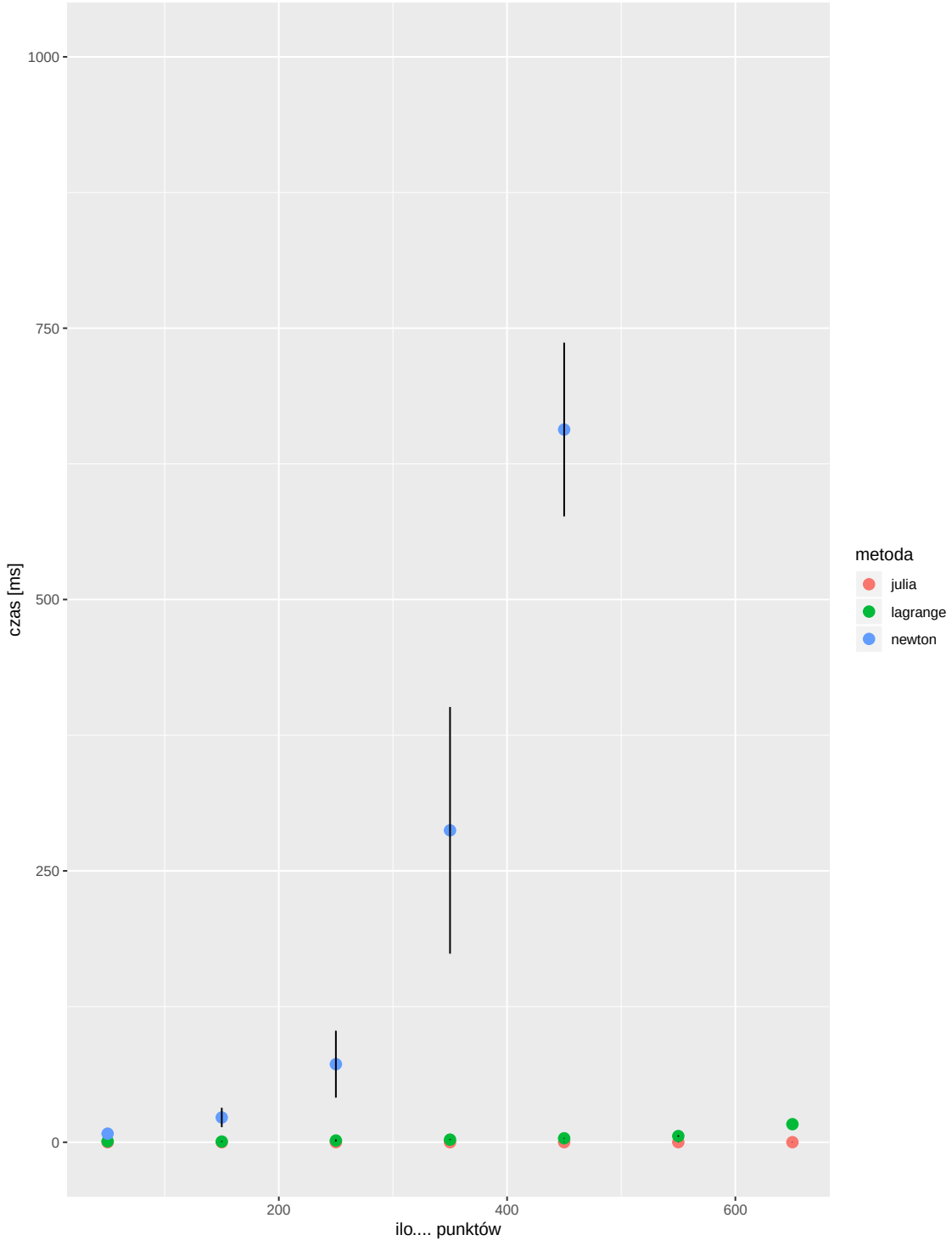


Rysunek 5: Interpolacja Lagrange'a, Newtona i z pakietu Polynomials na jednym wykresie

Jak można zauważyć wszystkie trzy interpolacje wygenerowały ten sam wielomian interpolujący dla punktów z rysunku 1. Taki wynik wyjaśnia *Jednoznaczność interpolacji wielomianowej*. Twierdzenie to dowodzi, że przez wybrane punkty można przeprowadzić tylko jeden wielomian interpolujący n-tego stopnia.

5 Porównanie czasu obliczania interpolacji dla róznych algorytmów

5.1 Wyniki pomiarów

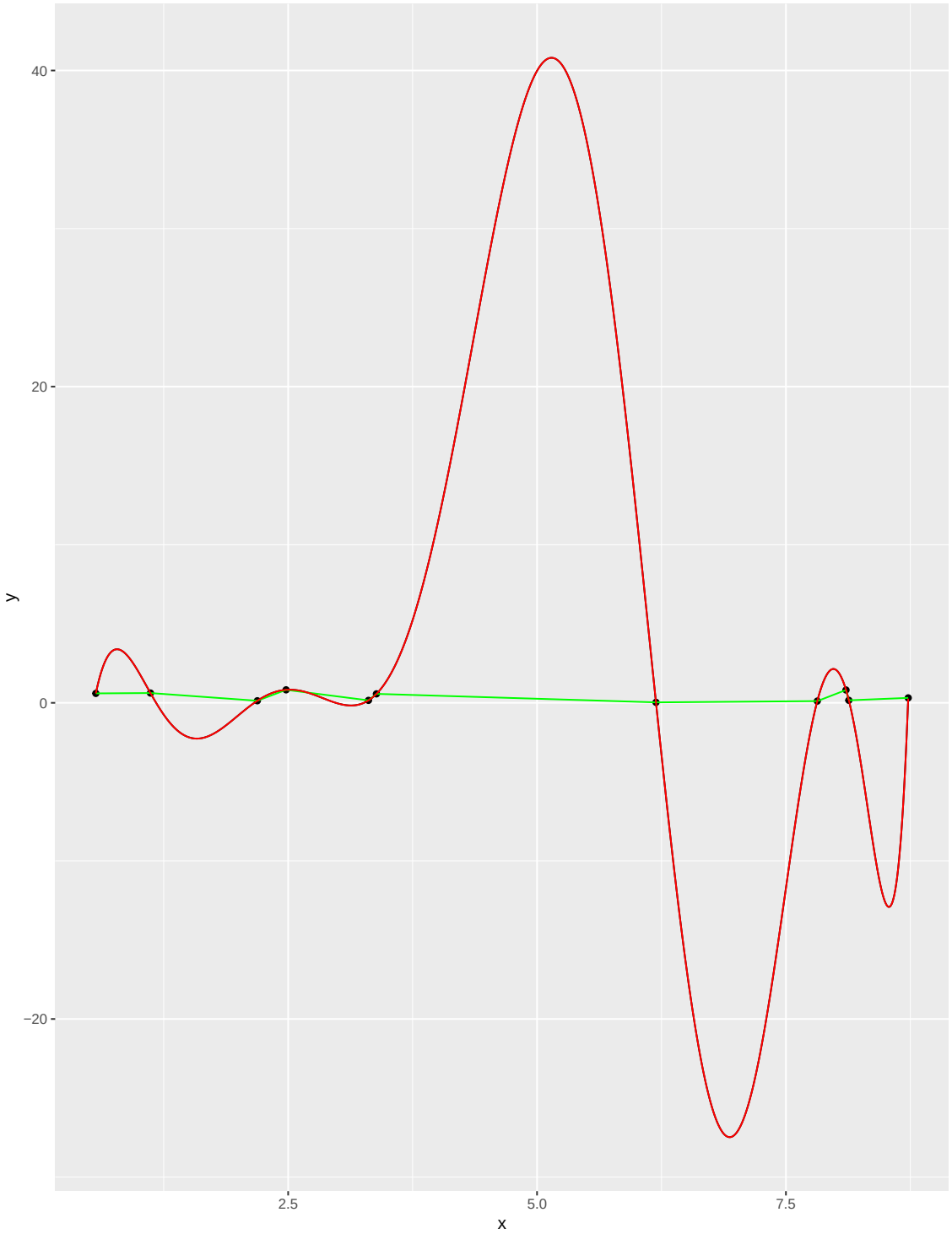


Rysunek 6: Porównanie metod

Wyniki ukazane na rysunku 6 jednoznacznie pokazują przewage algorytmu Lagrange'a i funkcji z pakietu Polynomials nad algorytmem Newtona. Interpolacja Newtona była nawet 700-krotnie wolniejsza od pozostałych interpolacji.

6 Interpolacja funkcją sklejaną pierwszego stopnia

6.1 Wykres



Rysunek 7: Interpolacja linear spline

Można zauważyć ze linear spline wygenerowal wykres znacznie mniej zmienny niż intepolacja z pakietu Polynomials.

7 Efekt Runge'go

Pogorszenie jakości interpolacji wielomianowej, mimo zwiększenia liczby jej węzłów. Początkowo ze wzrostem liczby węzłów n przybliżenie poprawia się, jednak po dalszym wzroście n, zaczyna się pogarszać. Widoczne na przykład na rysunku 2 na środku wykresu