Jakub Gwóźdź

29.11.2018

Sprawozdanie z laboratorium 5

Metody obliczeniowe w nauce i technice

1. Interpolacja Lagrange'a

$$L(x) := \sum_{j=0}^{k} y_j \ell_j(x)$$

$$\ell_j(x) := \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)},$$

```
function lagrangeInterpolation(X, Y)
    interpolation =
    function(x)
        y=0.0
        for k in 1:1:length(X)
            t = 1.0
            for j in 1:1:length(X)
                if j != k
                    t = t*((x-X[j])/(X[k]-X[j]))
                end
            end
            y += t*Y[k]
        end
        y
    end
    return interpolation
end
```

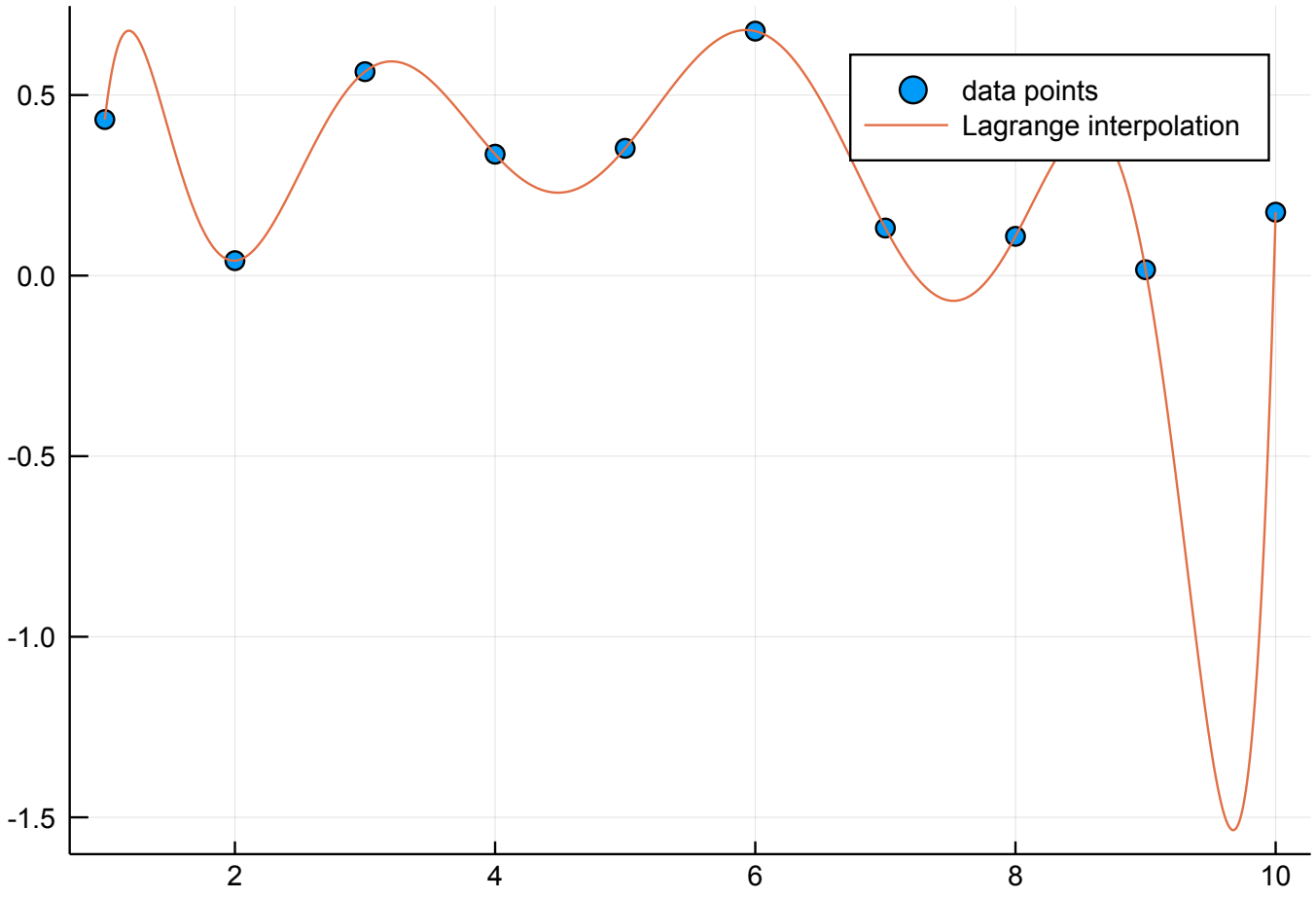
lis. 1

Wykres:

```
X = 1:1:10
Y = [rand() for x in X]

scatter(X,Y, label="data points")
# gęste punkty do rysowania wykresów funkcji interpolujących
Xs = 1:0.01:10
fit1 = lagrangeInterpolation(X,Y)
Ys1 = [fit1(x) for x in Xs]
plot!(Xs,Ys1, label="Lagrange interpolation")
```

lis. 2



wyk. 1

2. Interpolacja Newton'a

$$N(x) = [y_0] + [y_0, y_1](x - x_0) + \cdots + [y_0, \ldots, y_k](x - x_0)(x - x_1)\cdots(x - x_{k-1}) .$$

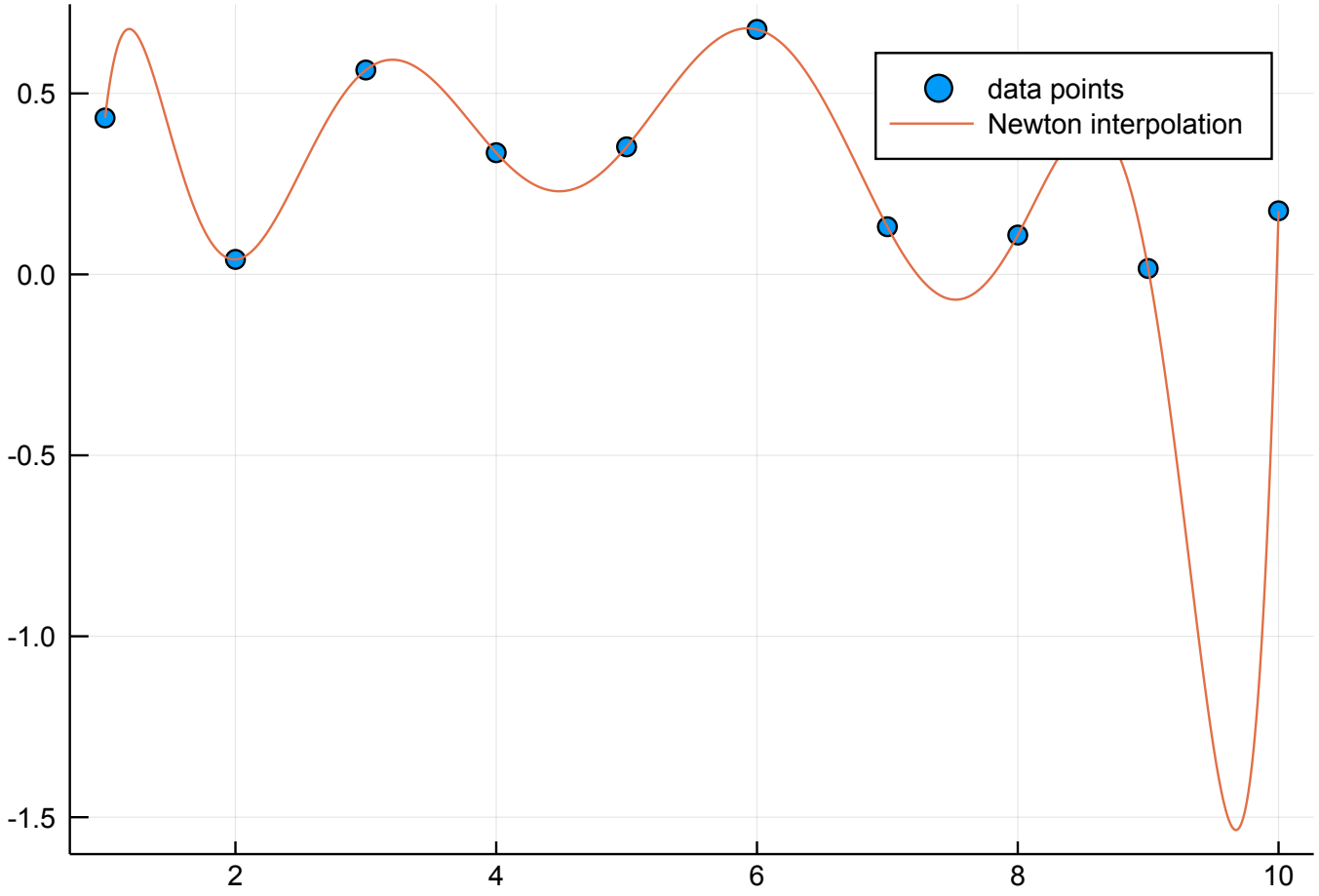
```
function newtonInterpolation(X,Y)
    P = Function[]
    push!(P,(l -> Y[1]))
   for k in 2:1:(length(X))
        Ck = Y[k] - P[k-1](X[k])
        for i in 1:1:(k-1)
            Ck = Ck/(X[k]-X[i])
        end
        pk = function(x)
            r = Ck
            for i in 1:1:(k-1)
                r = r*(x-X[i])
            end
            return r
        end
        fun = function(a) pk(a) + P[k-1](a) end
        push!(P,fun)
    end
    return P[length(X)]
end
```

lis. 3

Wykres:

```
scatter(X,Y, label="data points")
fit2 = newtonInterpolation(X,Y)
Ys2 = [fit2(x) for x in Xs]
scatter(X,Y, label="data points")
plot!(Xs,Ys2, label="Newton interpolation”)
```

lis.4

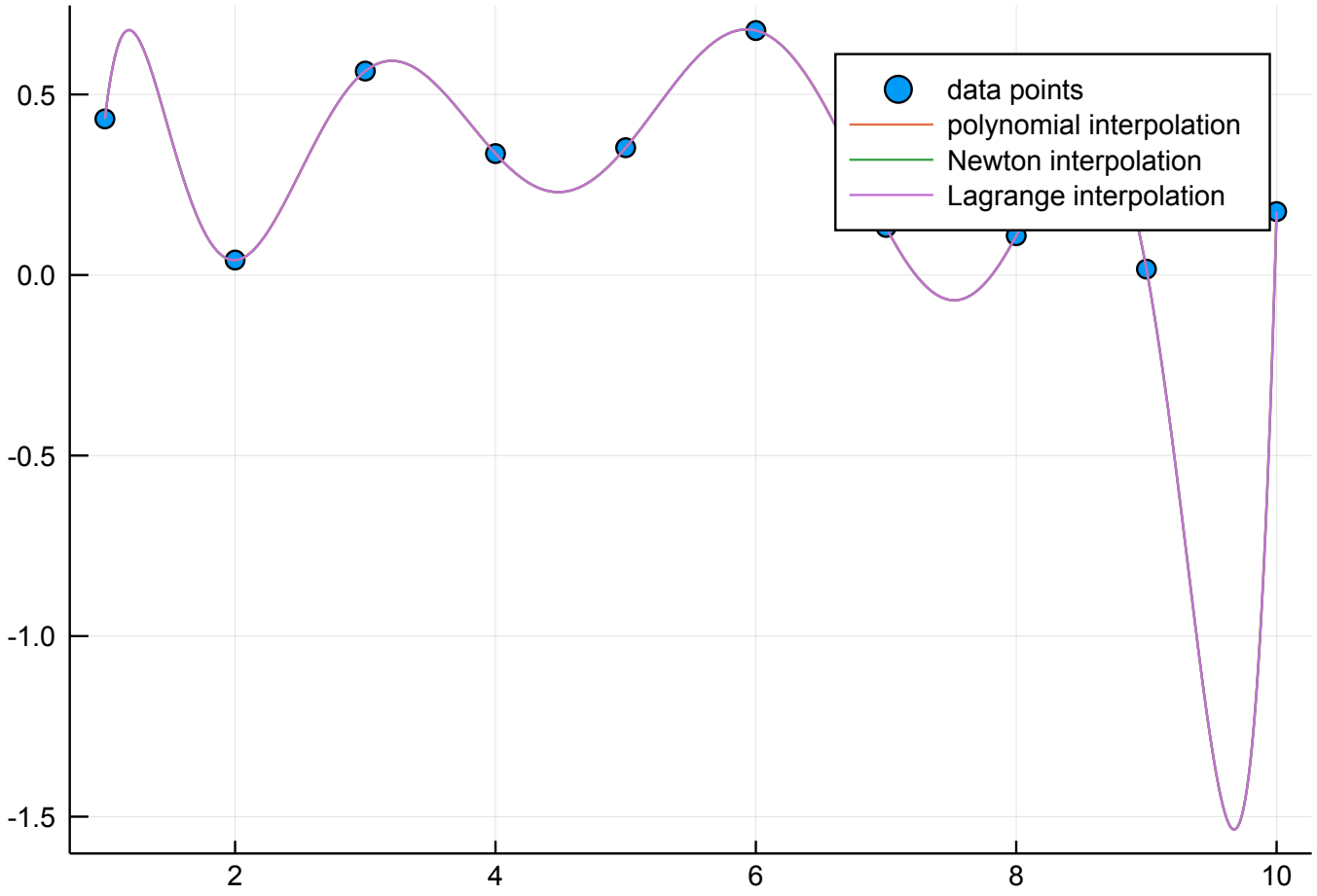


wyk. 2

Porównanie interpolacji:

```
fit3 = polyfit(X,Y)
Y3 = [fit3(x) for x in Xs]
scatter(X,Y, label="data points")
plot!(Xs,Y3, label="polynomial interpolation")
plot!(Xs,Ys2, label="Newton interpolation")
plot!(Xs,Ys1, label="Lagrange interpolation”)
```

lis. 5



wyk. 3

Wszystkie wykresy pokrywają się.

3. Porównanie wydajności interpolacji:

```
times =
DataFrame(size=[],Newton=[],Lagrange=[],Polymnomials=[])
mn = 1000000
for size in 1:4:100
    for attempt in 1:1:10
        xTest = 1:1:size
        yTest = [rand() for x in xTest]
        push!(times,
[(size),mn*@elapsed(newtonInterpolation(xTest,yTest)),
mn*@elapsed(lagrangeInterpolation(xTest,yTest)),
mn*@elapsed(polyfit(xTest,yTest))])
    end
end
```

lis. 6

	size	Newton	Lagrange	Polymnomials
	Any	Any	Any	Any
1	1.0	6.584	2.553	6.826
2	1.0	1.251	0.705	1.725
3	1.0	0.662	0.383	1.365
4	1.0	0.615	0.381	1.796
5	1.0	0.772	0.351	1.533
6	1.0	0.541	0.352	1.099
7	1.0	0.6	0.354	1.237
8	1.0	0.521	0.359	1.286
9	1.0	0.602	0.368	1.265
10	1.0	0.567	0.382	1.316

tab. 1

Grupowanie:

```
LagrangeMean = by(times,:size,df->mean(df[:Lagrange]))
LagrangeSD = by(times,:size,df->std(df[:Lagrange]))
Lagrange = DataFrame(Size = unique(times[:size]), Mean =
LagrangeMean[2], STD = LagrangeSD[2])
NewtonMean = by(times,:size,df->mean(df[:Newton]))
NewtonSD = by(times,:size,df->std(df[:Newton]))
Newton = DataFrame(Size = unique(times[:size]), Mean =
NewtonMean[2], STD = NewtonSD[2])
PolymnomialsMean = by(times,:size,df->mean(df[:Polymnomials]))
PolymnomialsSD = by(times,:size,df->std(df[:Polymnomials]))
Polymnomials = DataFrame(Size = unique(times[:size]), Mean =
PolymnomialsMean[2], STD = PolymnomialsSD[2])
```

lis. 7

Poszczególne średnie i odchylenia:

Polymnomials

	Size	Mean	STD
	Any	Float64	Float64
1	1.0	1.9448	1.72917
2	5.0	3.2773	2.53651
3	9.0	3.8529	0.663142
4	13.0	5.7107	0.835346
5	17.0	8.09	1.17775
6	21.0	20.1212	9.43944
7	25.0	21.7245	3.22144
8	29.0	23.2704	3.68346
9	33.0	29.3564	5.7905
10	37.0	39.8874	15.904
11	41.0	32.3342	2.82544
12	45.0	54.2396	15.3593
13	49.0	86.6034	63.2968
14	53.0	56.1358	9.51539
15	57.0	64.6783	25.0203
16	61.0	84.6064	20.1337
17	65.0	66.9353	7.20038
18	69.0	87.1499	9.44443
19	73.0	96.1965	22.8796
20	77.0	101.241	19.1837

tab. 2

Newton

	Size	Mean	STD
	Any	Float64	Float64
1	1.0	1.2715	1.87875
2	5.0	8.8157	4.07622
3	9.0	29.9123	0.675001
4	13.0	78.1489	1.47934
5	17.0	161.593	6.41011
6	21.0	287.797	15.7863
7	25.0	448.079	11.713
8	29.0	643.078	49.6812
9	33.0	843.918	26.3392
10	37.0	2375.58	3869.84
11	41.0	1523.55	21.1928
12	45.0	3638.71	4339.1
13	49.0	3223.39	796.087
14	53.0	4318.9	3571.45
15	57.0	6439.19	5353.64
16	61.0	6649.56	3171.62
17	65.0	7500.77	3911.13
18	69.0	9549.19	4570.88
19	73.0	10855.6	4720.33
20	77.0	12460.9	5200.41

tab. 3

Lagrange

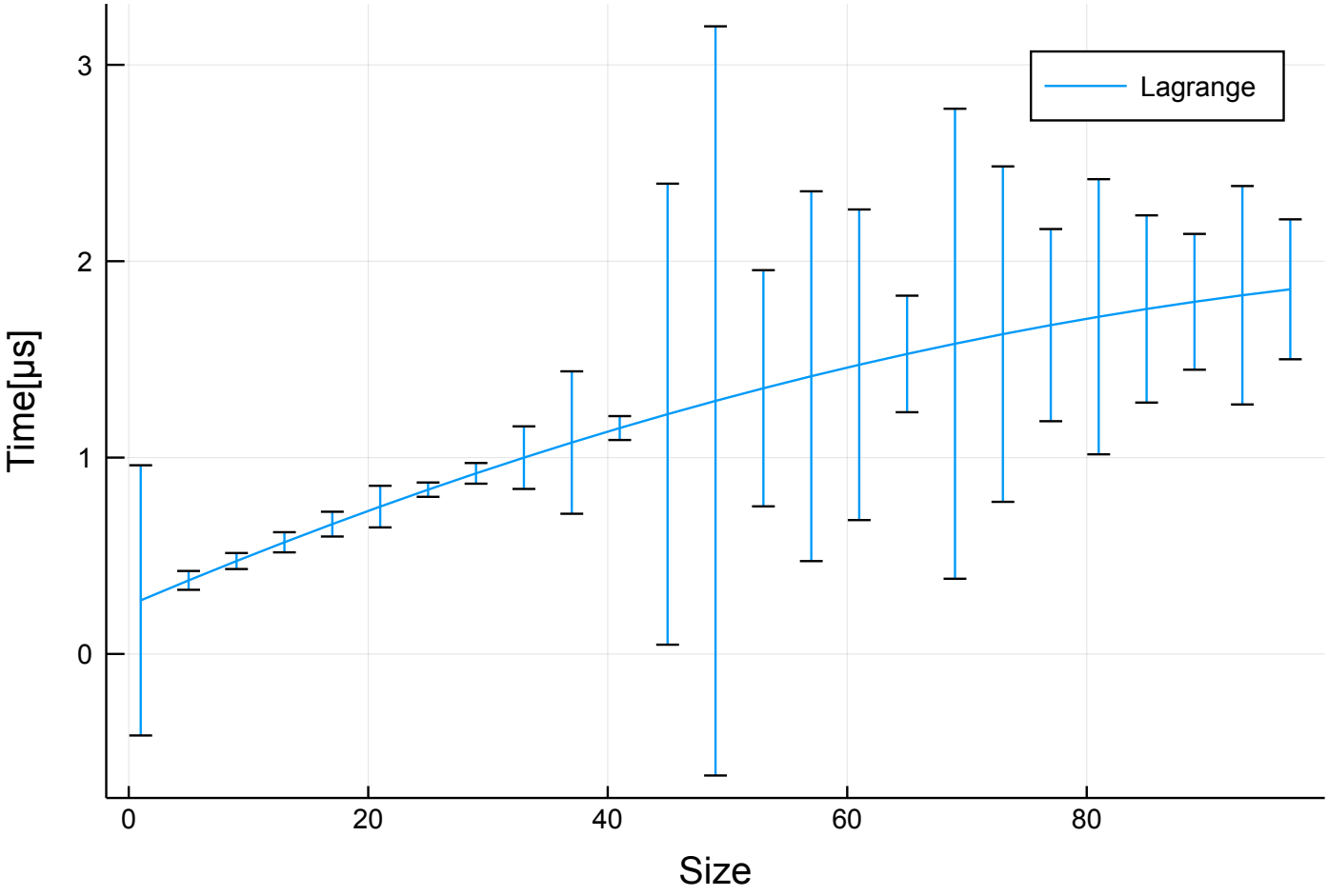
	Size	Mean	STD
	Any	Float64	Float64
1	1.0	0.6188	0.688008
2	5.0	0.4066	0.0480074
3	9.0	0.4284	0.0407764
4	13.0	0.4729	0.0512867
5	17.0	0.5303	0.0631067
6	21.0	0.7593	0.106027
7	25.0	0.7405	0.0364181
8	29.0	0.6988	0.0526029
9	33.0	0.7092	0.159352
10	37.0	0.7938	0.362607
11	41.0	0.6343	0.0608314
12	45.0	1.6302	1.17379
13	49.0	2.7996	1.90748
14	53.0	0.9662	0.601323
15	57.0	0.9739	0.94154
16	61.0	2.2435	0.790884
17	65.0	0.8984	0.296761
18	69.0	1.9982	1.19672
19	73.0	1.5015	0.854089
20	77.0	1.548	0.48928

tab. 4

Lagrange:

```
polyFit1 = polyfit(Lagrange[:Size],Lagrange[:Mean],2)
plot(Lagrange[:Size],polyval(polyFit1,Lagrange[:Size]),
label = "Lagrange",xlabel="Size",
ylabel=„Time[μs]",yerror=Lagrange[:STD])
```

lis. 8

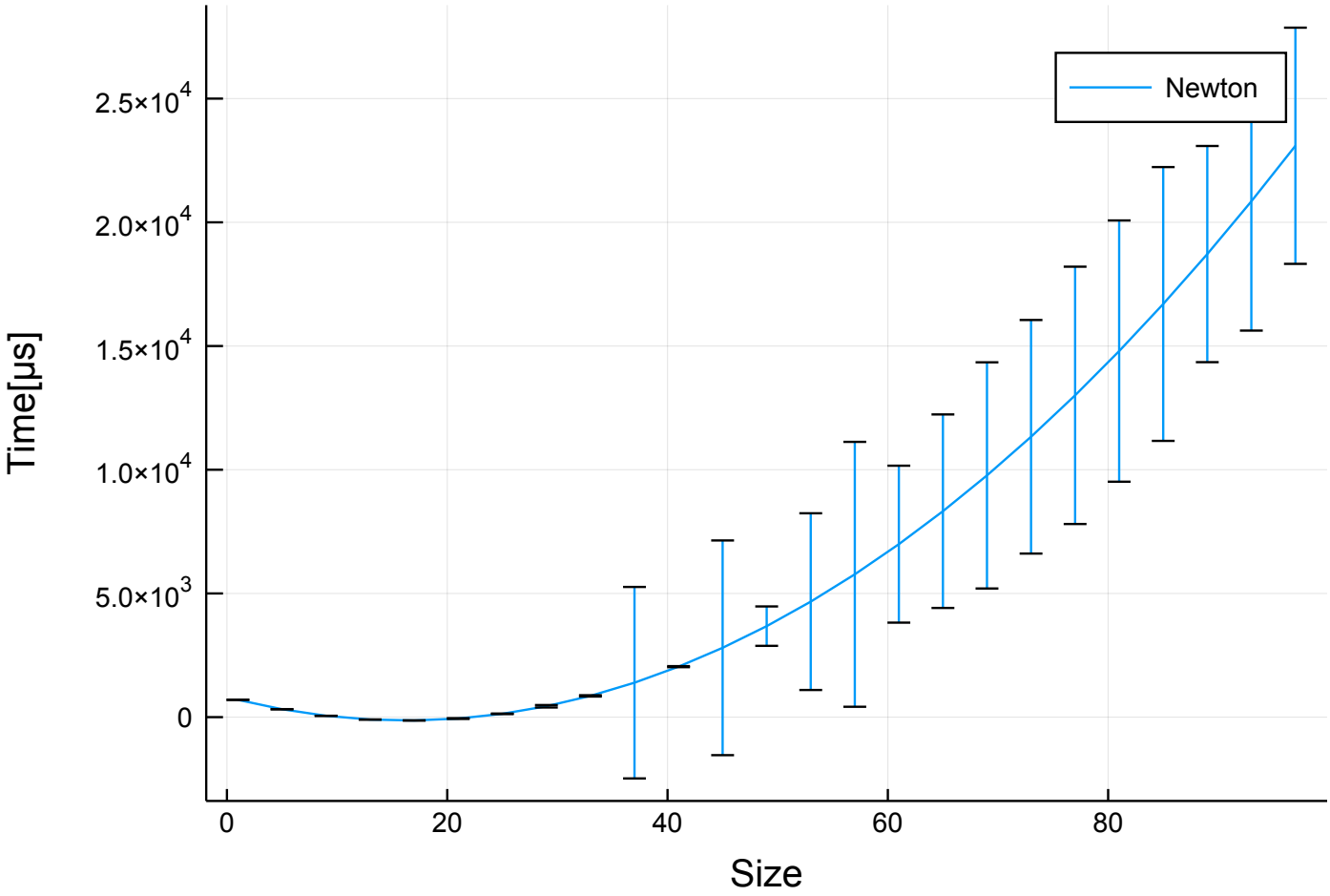


wyk. 4

Newton:

```
polyFit2 = polyfit(Newton[:Size],Newton[:Mean],2)
plot(Newton[:Size],polyval(polyFit2,Newton[:Size]),
label = "Newton",xlabel="Size",
ylabel="Time[μs]",yerror=Newton[:STD])
```

lis. 9

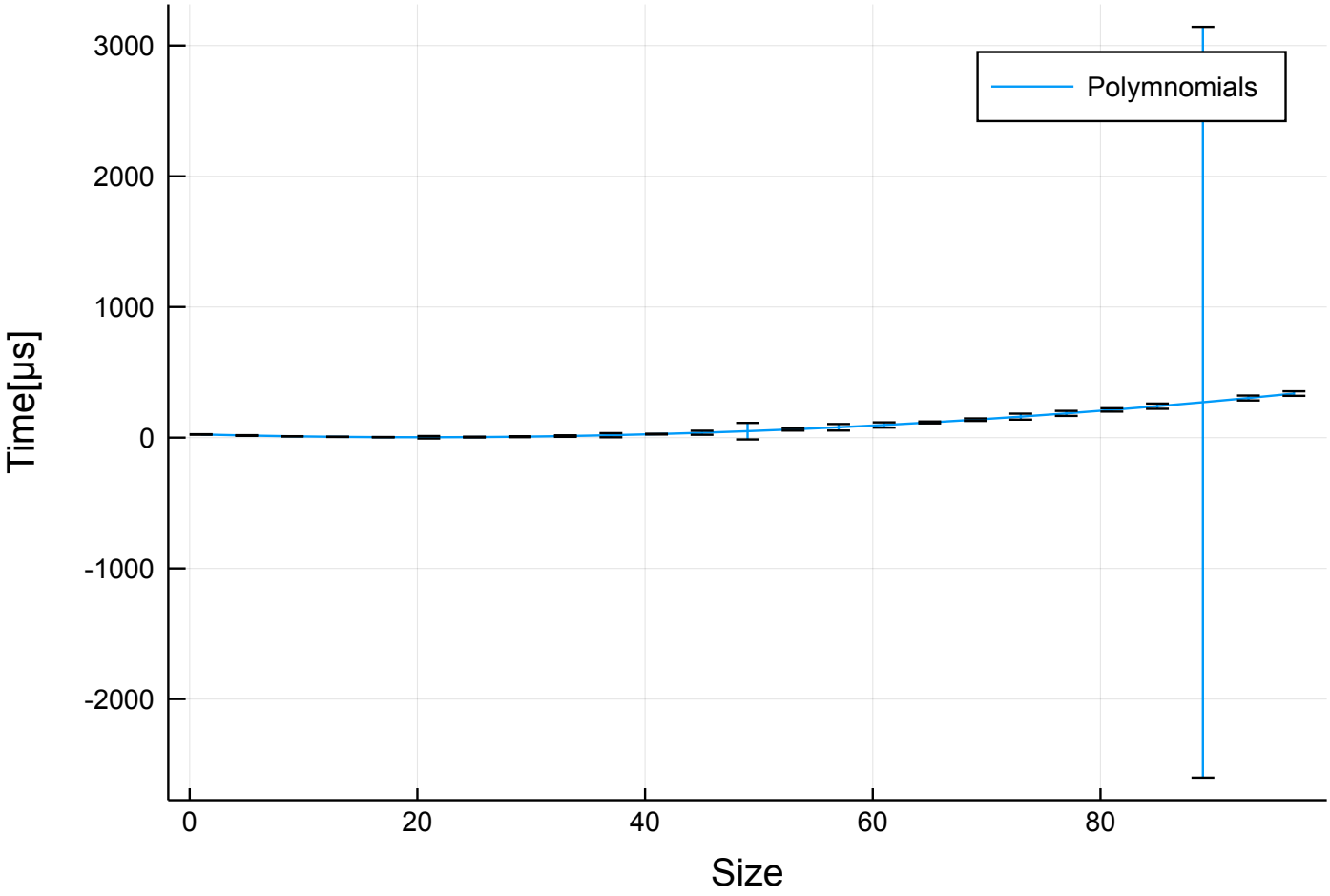


wyk. 5

Polymnomials:

```
polyFit3 = polyfit(Polymnomials[:Size],Polymnomials[:Mean],2)
plot(Polymnomials[:Size],polyval(polyFit3,Polymnomials[:Size])
,label = "Polymnomials",xlabel="Size",
ylabel=„Time[μs]",yerror=Polymnomials[:STD])
```

lis. 10



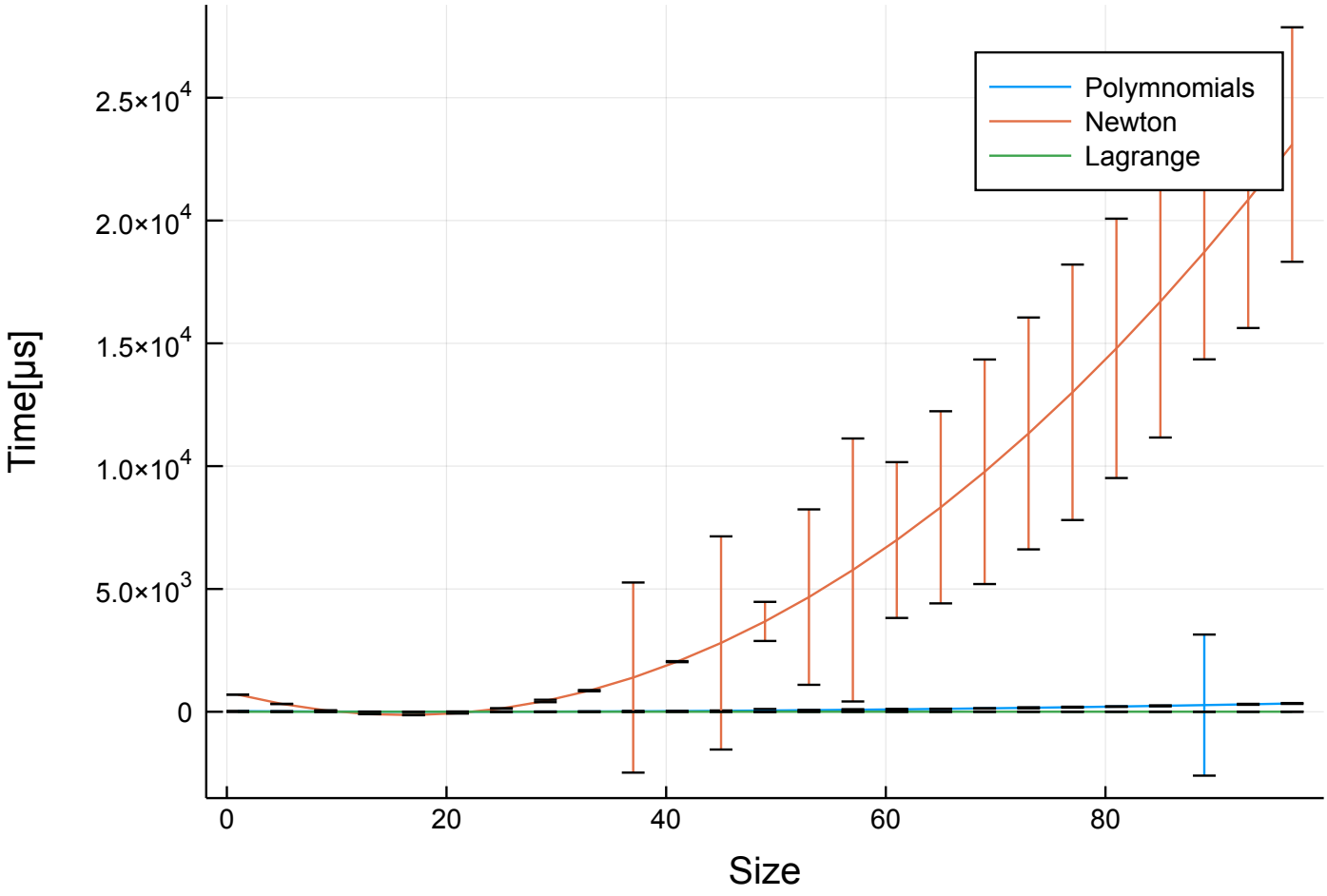
wyk. 6

Osie specjalnie zostawiłem nieprzeskalowane by móc porównać wykresy poszczególnych interpolacji.

Porównanie wydajności:

```
plot(Polymnomials[:Size],polyval(polyFit3,Polymnomials[:S
ize]),label = "Polymnomials",xlabel="Size",
ylabel="Time[μs]",yerror=Polymnomials[:STD])
plot!(Newton[:Size],polyval(polyFit2,Newton[:Size]),label
= "Newton",xlabel="Size",
ylabel="Time[μs]",yerror=Newton[:STD])
plot!
(Lagrange[:Size],polyval(polyFit1,Lagrange[:Size]),label
= "Lagrange",xlabel="Size",
ylabel=„Time[μs]",yerror=Lagrange[:STD])
```

lis. 11



wyk. 7

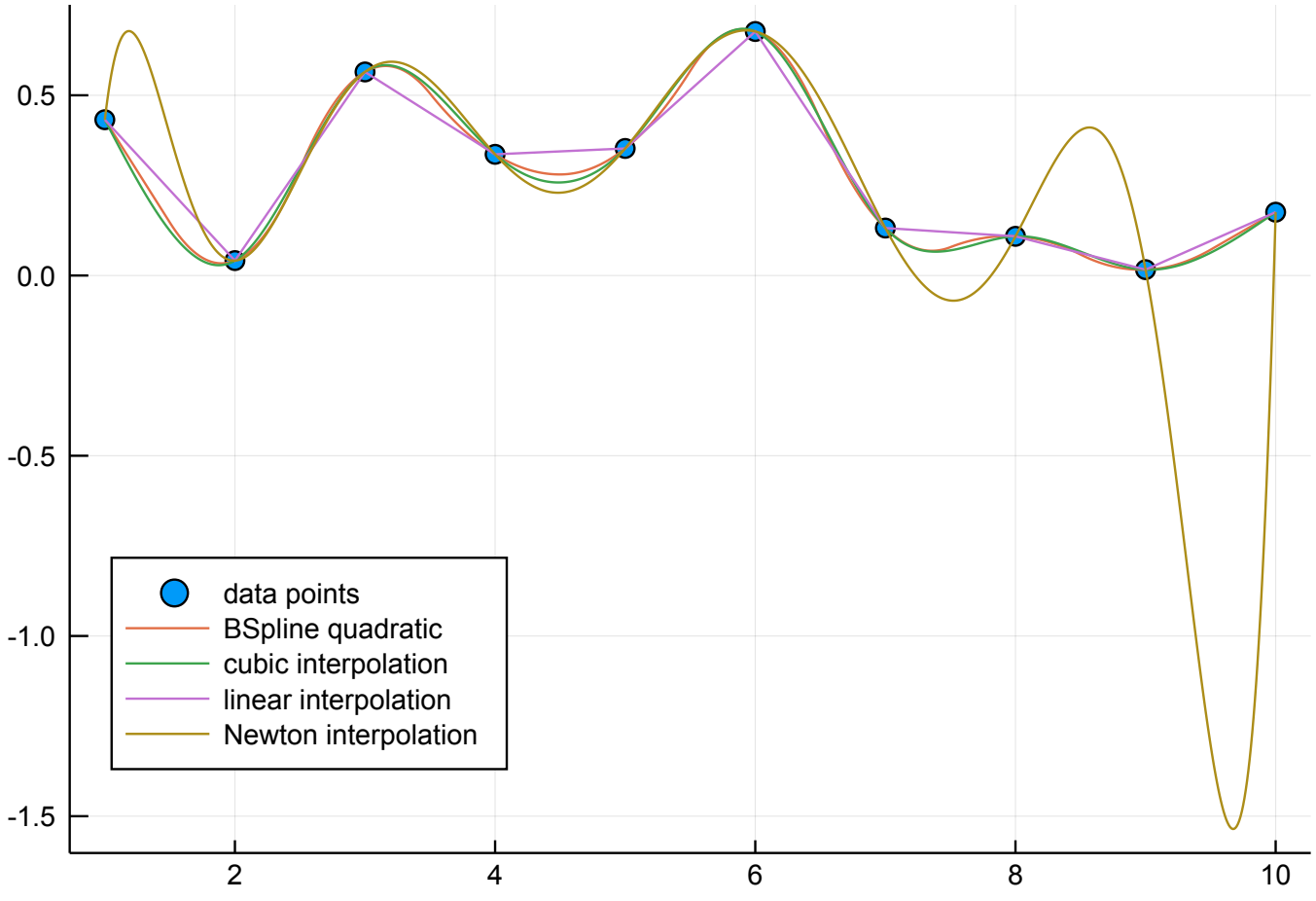
Porównanie wykresów interpolacji i użycia różnych funkcji sklejanych:

1) pierwszy zbiór punktów:

(dane z poprzednich podpunktów)

```
interp_cubic = CubicSplineInterpolation(X, Y)
interp_linear = LinearInterpolation(X, Y)
itp_quadratic = interpolate(Y,
BSpline(Quadratic(Line(OnCell()))))
interp_newton = newtonInterpolation(X,Y)
YN = [interp_newton(x) for x in Xs]
YQ = [itp_quadratic(x) for x in Xs]
YL = [interp_linear(x) for x in Xs]
YC = [interp_cubic(x) for x in Xs]
scatter(X, Y, label ="data points", legend=:bottomleft)
plot!(Xs, YQ, label ="BSpline quadratic")
plot!(Xs, YC, label = "cubic interpolation")
plot!(Xs, YL, label = "linear interpolation")
plot!(Xs, YN, label = "Newton interpolation")
```

lis. 12

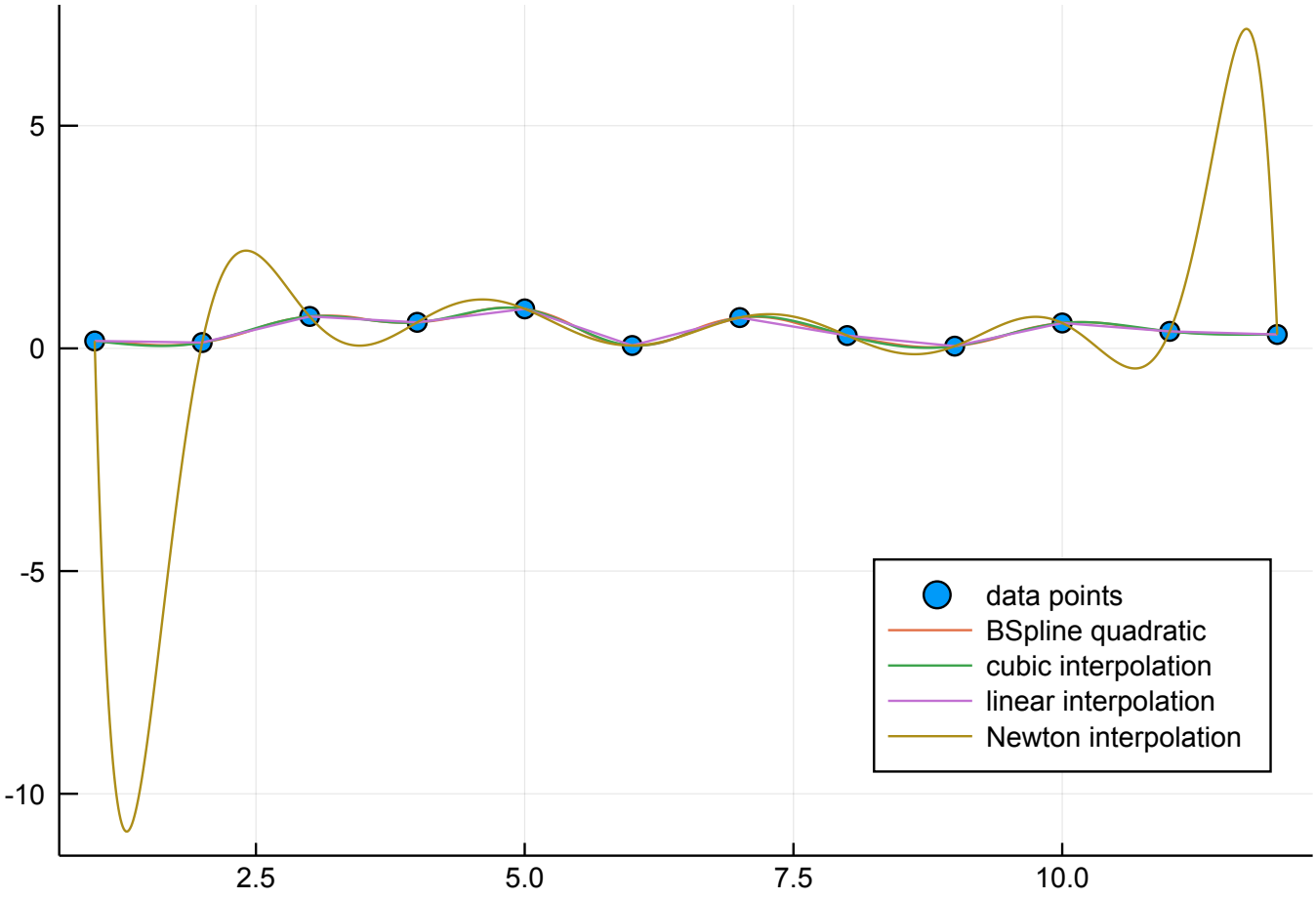


wyk. 8

2) drugi zbiór punktów:

```
X1 = 1:1:12
Xs1 = 1:0.01:12
Y1 = [rand() for x in X1]
using Interpolations
interp_cubic = CubicSplineInterpolation(X1, Y1)
interp_linear = LinearInterpolation(X1, Y1)
itp_quadratic = interpolate(Y1,
BSpline(Quadratic(Line(OnCell()))))
interp_newton = newtonInterpolation(X1,Y1)
YN = [interp_newton(x) for x in Xs1]
YQ = [itp_quadratic(x) for x in Xs1]
YL = [interp_linear(x) for x in Xs1]
YC = [interp_cubic(x) for x in Xs1]
scatter(X1, Y1, label ="data points", legend=:bottomright)
plot!(Xs1, YQ, label ="BSpline quadratic")
plot!(Xs1, YC, label = "cubic interpolation")
plot!(Xs1, YL, label = "linear interpolation")
plot!(Xs1, YN, label = "Newton interpolation")
```

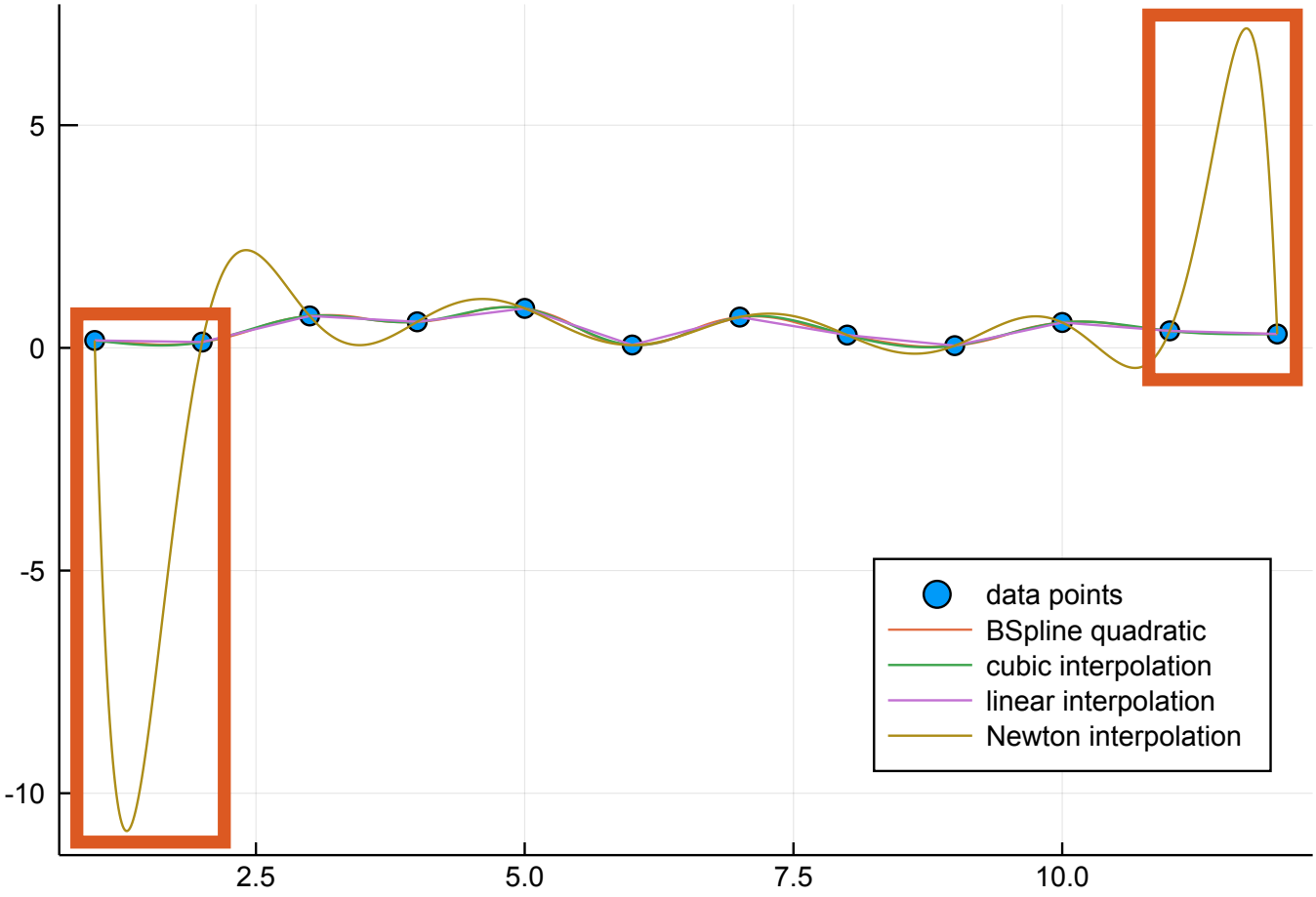
lis. 13



wyk. 9

4. Efekt Rungego

Na poniższym wykresie widać wyraźnie anomalie jakie powstały na początku i na końcu wykresu dla interpolacji Newtona.



wyk. 10

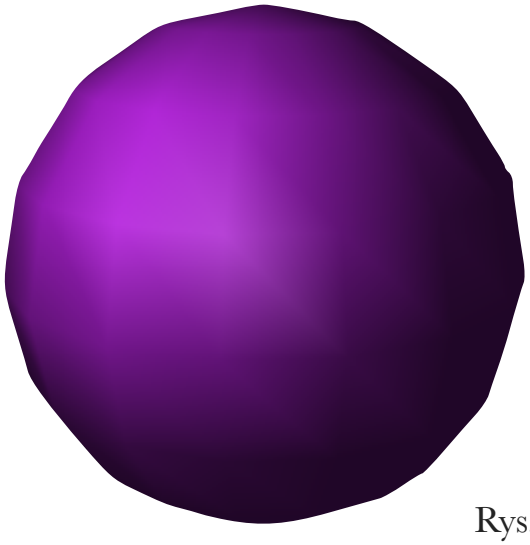
5. Algorytmy interpolacji stosowane w grafice komputerowej.

Wybrałem do porównania dwie metody cieniowania obrazu:

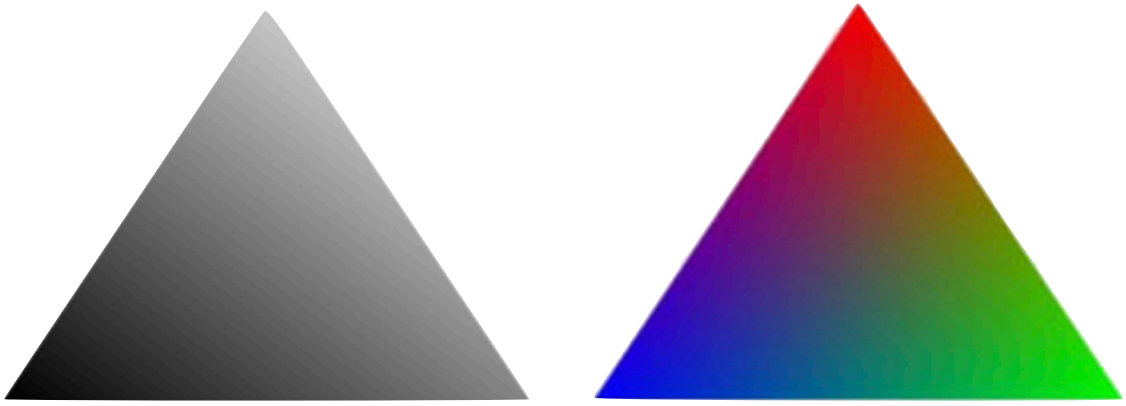
- cieniowanie Gourauda
- cieniowanie Phonga

Cieniowanie Gourauda:

Metoda ta jest najczęściej wykorzystywana do cieniowania trójkątów, inne wielokąty są w tym celu przerabiane na trójkąty. Metoda polega na obliczaniu barw wierzchołków trójkąta, a wnętrze trójkąta jest interpolowane liniowo



Rys. 1



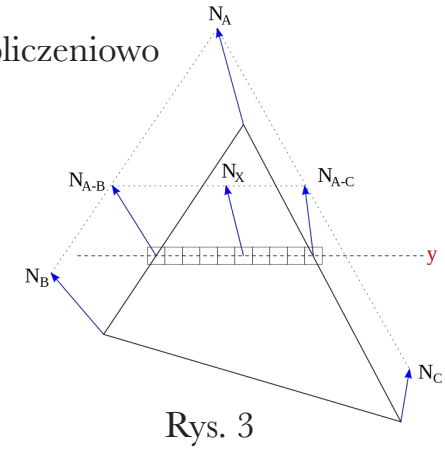
Rys. 2

W przypadku koloru 24-bitowego należy interpolować każdą ze składowych koloru oddzielnie.

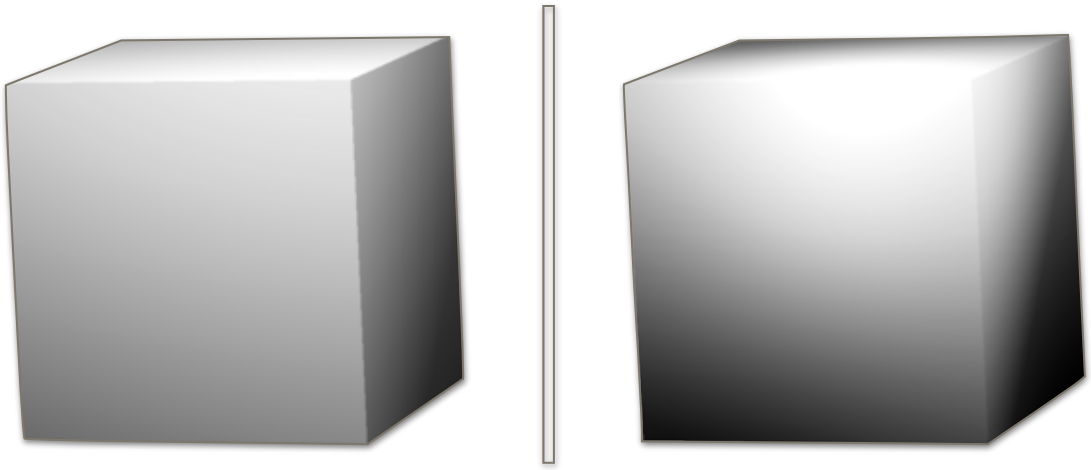
Cieniowanie Phong Buoi-Tuonga:

W odróżnieniu od metody Gouraud'a cieniujemy wielokąty, a nie trójkąty. Ta metoda interpoluje się wierzchołki, lecz wektory normalne.
Dla każdego piksela wyliczamy wektor normalny powierzchni interpolując wektory normalne w narożnikach i po unormowaniu podstawiamy do wzoru odpowiadającego naszemu modelowi oświetlenia (Phonga, Lamberta, Blinna…)

Metoda jest bardziej skuteczne, ale dużo bardziej złożona obliczeniowo od metody Gouraud.



Rys. 3



Rys. 4