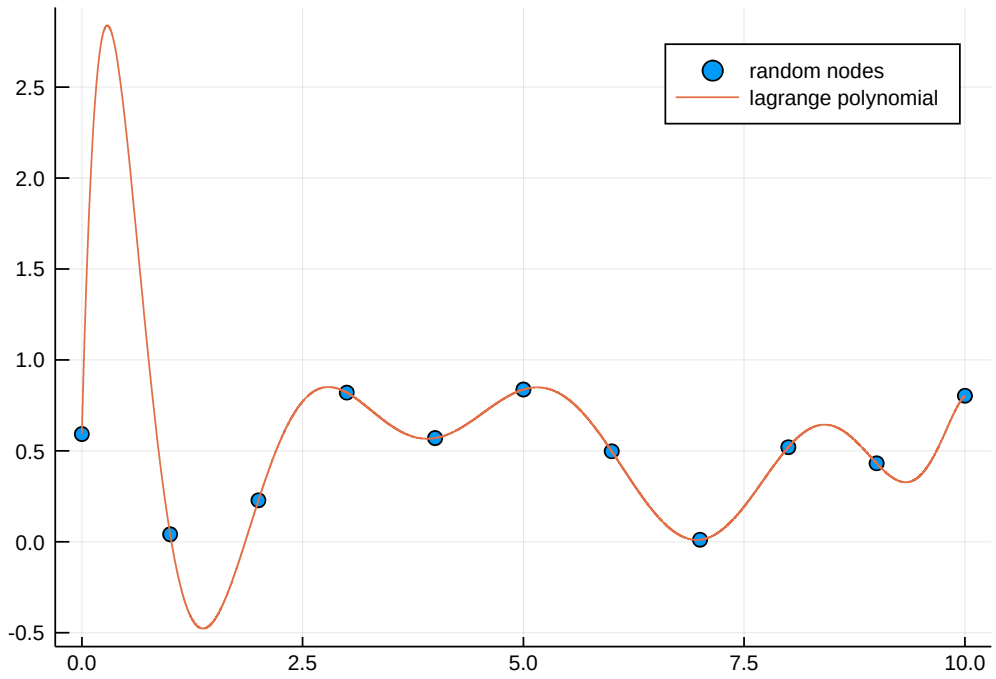
Interpolacja

1. Wielomian interpolacyjny Lagrange’a

Celem ćwiczenia jest interpolowanie wylosowanych węzłów w układzie kartezjańskim wielomianem Lagrange’a. Ćwiczenie zrealizowano przy użyciu języka Julia, a sam algorytm został zaimplementowany ze wzoru użytego na wykładzie.

```
function lagrange(x, y, xi)
    accsum = 0
    for i = 1:size(x,1)
        accpi = 1
        for j = 1:size(x,1)
            if i == j continue end
            accpi = accpi * (xi - x[j]) / (x[i] - x[j])
        end
        accsum = accsum + accpi*y[i]
    end
    accsum
end

x =  0:10
y = [rand() for i in x]

xf = [i/1000 for i in 0:10000]
yf = [lagrange(x,y,i/1000) for i in 0:10000]

using Plots
scatter(x,y, label="random nodes")
plot!(xf,yf, label="lagrange polynomial")
```

Kod źródłowy 1. Funkcja Lagrange’a



Rysunek 1. Wykres losowych punktów i wielomianu który je interpoluje

Algorytm Lagrange’a jest mało efektywny w kontekście złożoności (więcej w rozdziale 4). Polega na obliczeniu wartości wielomianu dla każdego punktu. Samo wyliczenie wielomianu jest kosztownie, ponieważ przetrzymujemy go niejawnie, przez co za kazdym razem trzeba wyliczyć wartość. Funkcja lagrange realizuje całą implementację poniższego wzoru.

$$L_k(x) = \frac{d}{m} = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{(x_k - x_i)}, \quad P_n(x) = \sum_{k=0}^{n} L_k(x) f(x_k)$$

Rysunek 2. Wzór interpolacji Lagrange'a użyty na wykładzie

2. Metoda ilorazów różnicowych

Celem ćwiczenia jest interpolowanie wylosowanych węzłów w układzie kartezjańskim metodą ilorazów różnicowych. Ćwiczenie zrealizowano przy użyciu języka Julia, a sam algorytm został zaimplementowany przy użyciu wzoru użytego na wykładzie.

```
function elems(x, y)
    a = y
    for i = 2:size(x,1)
        for j = size(x,1):(-1):(i)
            a[j] = (a[j] - a[j-1]) / (x[j] - x[j-(i-1)])
        end
    end
    a
end

function newton(x, y, a, xi)
    accsum = a[1]

    for i = 1:size(x,1)-1
        accpi = a[i+1]
        for j = 1:i
            accpi = accpi * (xi - x[j])
        end
        accsum = accsum + accpi
    end
    accsum
end

x =  0:10
y = [rand() for i in x]

using Plots
scatter(x,y)

a = elems(x,y)
xf = [i/1000 for i in 0:10000]
yf = [newton(x,y,a,i/1000) for i in 0:10000]


plot!(xf,yf)
```

Kod źródłowy 2. Funkcja Newtona

Do tworzenia wielomianu wykorzystano pomocniczą funkcję elems, która wyznacza ilorazy tylko raz korzystając z tablicy jednowymiarowej. Cały schemat wyliczania ilorazów został wykorzystany z poniższego wzoru przedstawionego na wykładzie.

Wprowadzamy notację:

- 0-wy iloraz różnicowy wzgl. x_i : $f[x_i] = f(x_i)$
  pozostałe - indukcyjnie:
- 1-szy:

$$f[x_i,\ x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

Gdy zaś określone są ilorazy aż do $(k-1)$, czyli

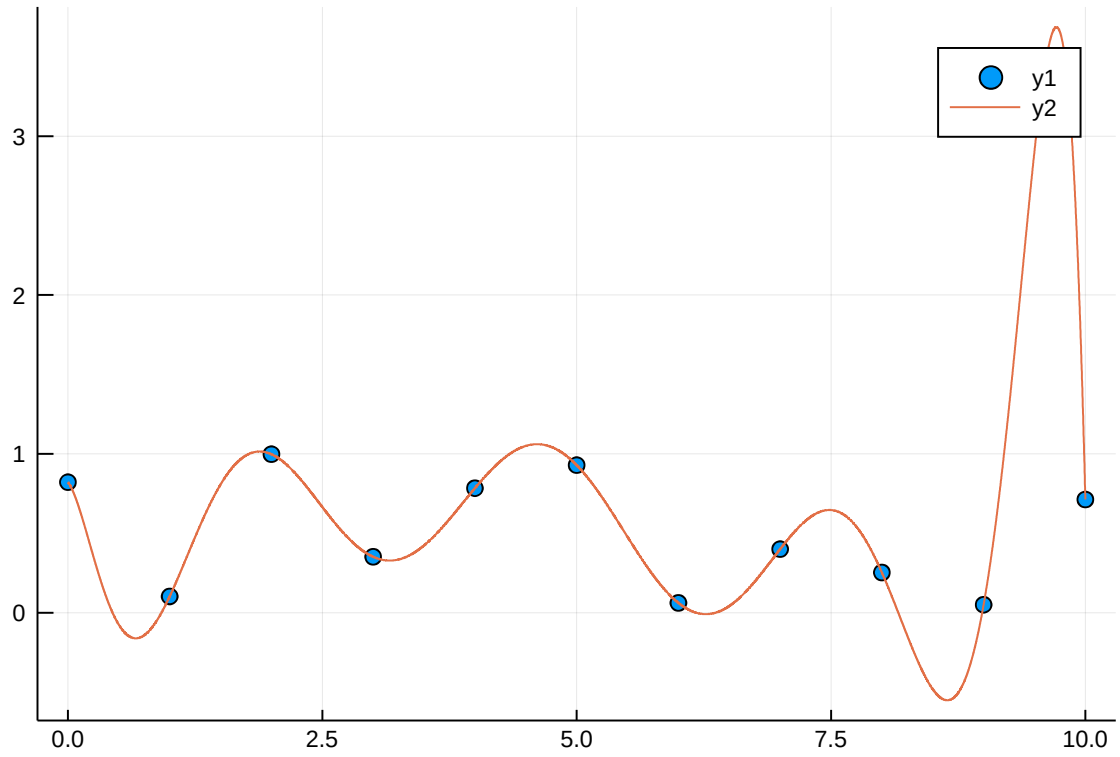
$$f[x_i,\ x_{i+1},\ x_{i+k-1}]\ i\ f[x_{i+1},\ x_{i+2},\ x_{i+k}]$$

- to wtedy k-ty iloraz różnicowy:

$$f[x_i,\ x_{i+1}, \ldots x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, \ldots . x_{i+k}] - f[x_i, x_{i+1}, \ldots . x_{i+k-1}]}{x_{i+k} - x_i}$$

Rysunek 3. Wzór obliczania ilorazów przedstawiony na wykładzie

Interpolacja metodą Newtona jest korzystniejsza w kontekście złożoności czasowej (więcej w rozdziale 4).



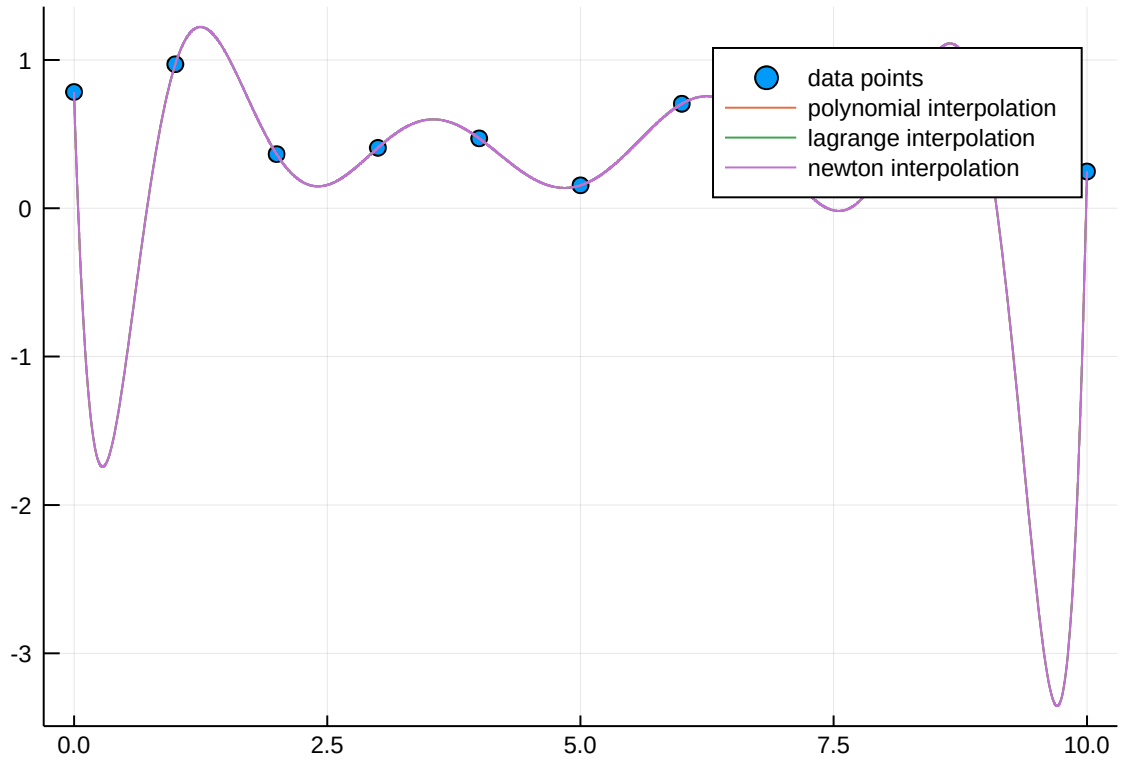
Rysunek 4. Interpretacja graficzna wielomianu interpolującego losowe węzły

3. Porównanie funkcji interpolujących punkty

Celem ćwiczenia było porównanie kształtów funkcji powyższych dwóch interpolacji oraz wbudowanej funkcji interpolującej punkty. Poniżej prezentuję implementację w języku Julia.

```
x =  0:10
y = [rand() for i in x]

using Plots
scatter(x,y, label="data points")

xp = 0:0.01:10

using Polynomials
fit1=polyfit(x, y)
yf=[fit1(i) for i in xp]
plot!(xp,yf, label="polynomial interpolation")

xf = [i/1000 for i in 0:10000]
yf = [lagrange(x,y,i/1000) for i in 0:10000]

plot!(xf,yf, label="lagrange interpolation")

a = elems(x,y)
xf = [i/1000 for i in 0:10000]
yf = [newton(x,y,a,i/1000) for i in 0:10000]


plot!(xf,yf, label="newton interpolation")
```

Kod źródłowy 3. Interpolacja wielomianowa



Rysunek 5. Wykres przedstawiający wszystkie 3 wielomiany interpolujące losowe punkty

Jak widać wszystkie wielomiany pokrywają się. Dzieje się tak gdyż zawsze istnieje tylko jeden wielomian który interpoluje konkretne zadane punkty. Można to udowodnić niewprost. Zakładamy

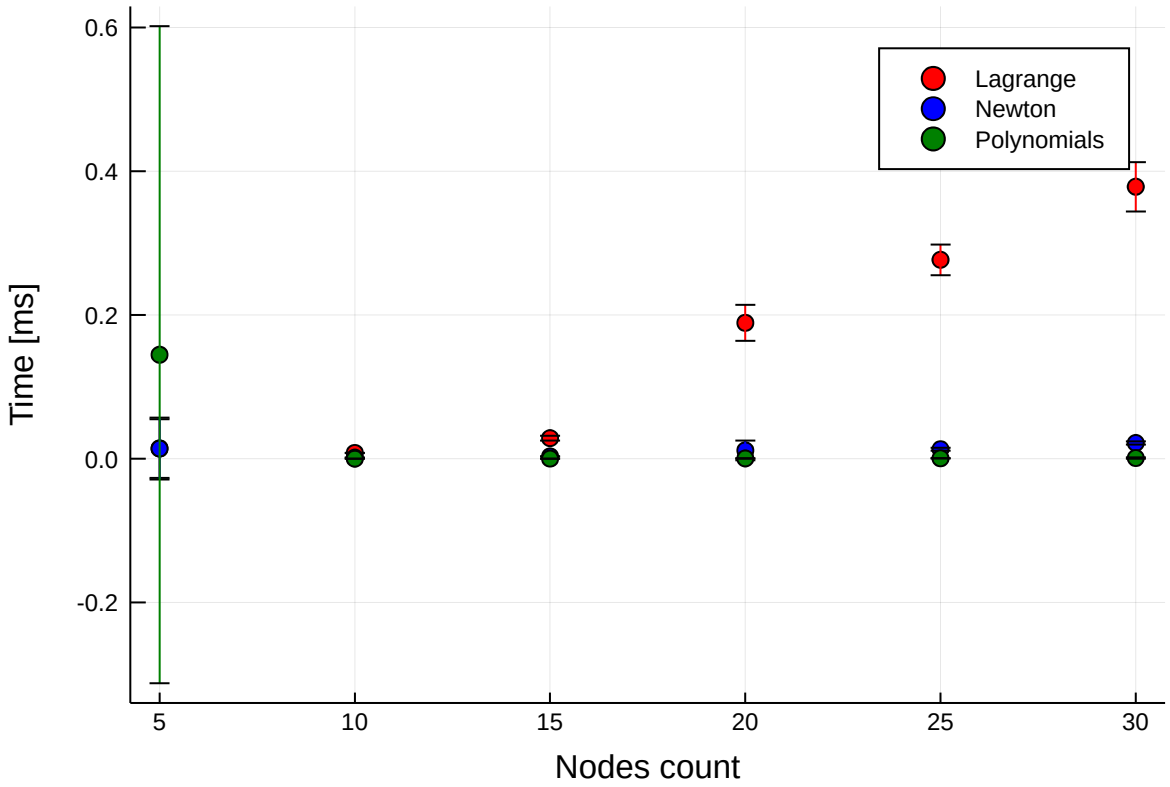
że chcemy interpolować n+1 punktów wielomianem stopnia n. Jeżeli założymy że istnieją dwa takie wielomiany p i q, które interpolują, to ich różnica ma n+1 miejsc zerowych, a ponieważ ich różnica jest wielomianem co najwyżej stopnia n, to znaczy, że jest tożsamościowo równy zero, czyli p = q.

4. Porównanie czasów poszczególnych metod i analiza danych

Celem ćwiczenia jest porównanie czasów działania metod interpolacji. Język użyty do implementacji to Julia.

```
using DataFrames
df = DataFrame(Type = String[], N = Int64[], Time = Float64[])


for n = 5:5:30
    for t = 1:10
        x =  0:n
        y = [rand() for i in x]

        xp = 0:0.01:n
        using Polynomials

        time = @elapsed begin
            fit1=polyfit(x, y)
            yf=[fit1(i) for i in xp]
        end
        df2 = DataFrame(Type = "Polynomials", N = n, Time = time)

        append!(df,df2)

        time = @elapsed begin
            xf = [i/1000 for i in 0:10000]
            yf = [lagrange(x,y,i/1000) for i in 0:n*1000]
        end
        df2 = DataFrame(Type = "Lagrange", N = n, Time = time)

        append!(df,df2)
        time = @elapsed begin
            a = elems(x,y)
            xf = [i/1000 for i in 0:10000]
            yf = [newton(x,y,a,i/1000) for i in 0:n*1000]
        end
        df2 = DataFrame(Type = "Newton", N = n, Time = time)

        append!(df,df2)
    end
end

using Statistics
avg = by(df, [:Type, :N], Mean = :Time => mean, Std = :Time => std)

using Plots
scatter(avg[:N], avg[:Mean], group=avg[:Type], colour = [:red :blue :green],
yerr=avg[:Std], xlabel="Nodes count", ylabel="Time [ms]")
```

Kod źródłowy 4. Funkcje porównujące czasy



Rysunek 6. Wykres prezentujący zależność czasu od ilości punktów do interpolowania

Próbę statystyczną przeprowadzono na losowych punktach w ilości z przedziału 5 do 30 z krokiem 5. Każdą próbę powtórzono 10 razy i wyliczono na tej podstawie średnią wartość i odchylenie standardowe. Jak widać na wykresie (Rysynek 6) metoda Newtona i wbudowana funkcja z pakietu Polynomials mają bardzo podobną złożoność, w przeciwieństwie do metody Lagrange'a.

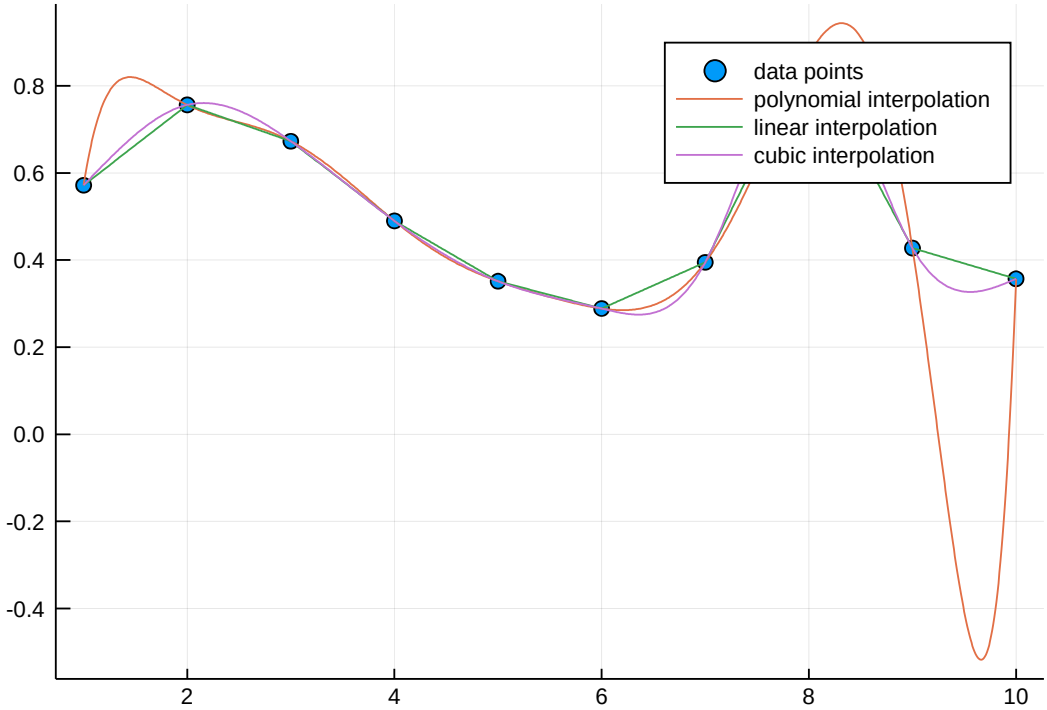
5. Interpolacja funkcjami sklejanymi, a interpolacja wielomianowa

Celem ćwiczenia było porównanie interpolacji funkcjami sklejanymi z interpolacją wielomianem. Język użyty do wykonania ćwiczenia to Julia.

```
x =  1:1:10
y = [rand() for i in x]

using Plots
scatter(x,y, label="data points")
xp = 1:0.01:10

using Polynomials
fit1=polyfit(x, y)
yf=[fit1(i) for i in xp]
plot!(xp,yf, label="polynomial interpolation")

using Interpolations
interp_linear = LinearInterpolation(x, y)
yf=[interp_linear(i) for i in xp]
plot!(xp, yf, label="linear interpolation")

interp_cubic = CubicSplineInterpolation(x, y)
```

```
yf=[interp_cubic(i) for i in xp]
plot!(xp, yf, label="cubic interpolation")
```

Kod źródłowy 5. Interpolacje funkcjami sklejanymi



Rysunek 7. Wykres interpolacji wielomianowej i funkcjami sklejanymi

W ćwiczeniu użyto gotowych mechanizmów do interpolacji wielomianowej lub funkcjami sklejanymi na losowo wybranych punktach.

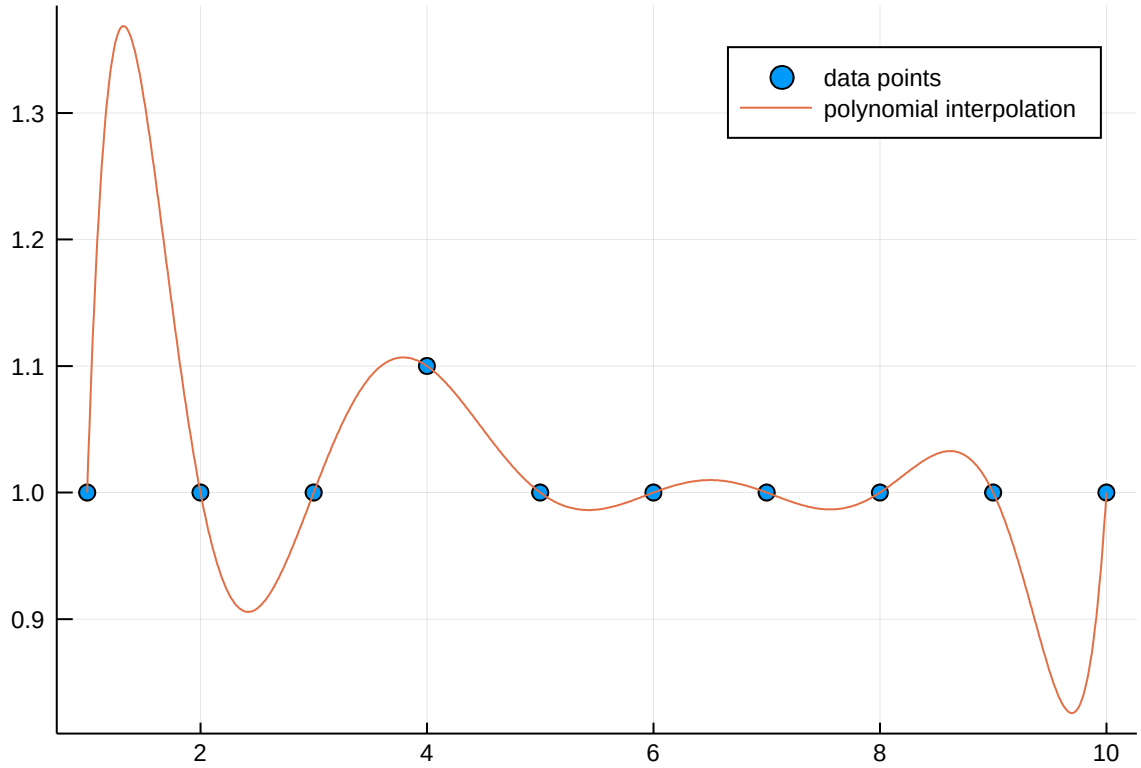
Interpolacja funkcjami sklejanymi jest znacząco inna od wielomianowej. Sklejanie funkcji liniowych daje nam łamaną funkcję, co jest dosyć oczywiste. Natomiast sklejanie fragemtnów wielomianów trzeciego stopnia jest również znacząco inne od interpolacji wielomianowej. Jest jednak gładsze oraz w niektórych miejscach przebiega bardzo podobnie.

6. Efekt Rungego

Efekt Rungego jest związany z pogoraszniem się jakości interpolacji wielomianowej wraz ze wzrostem punktów, zwłaszcza na końcach przedziałów. Demonastrajcę efektu zaprezentuję przy użyciu języka Julia.

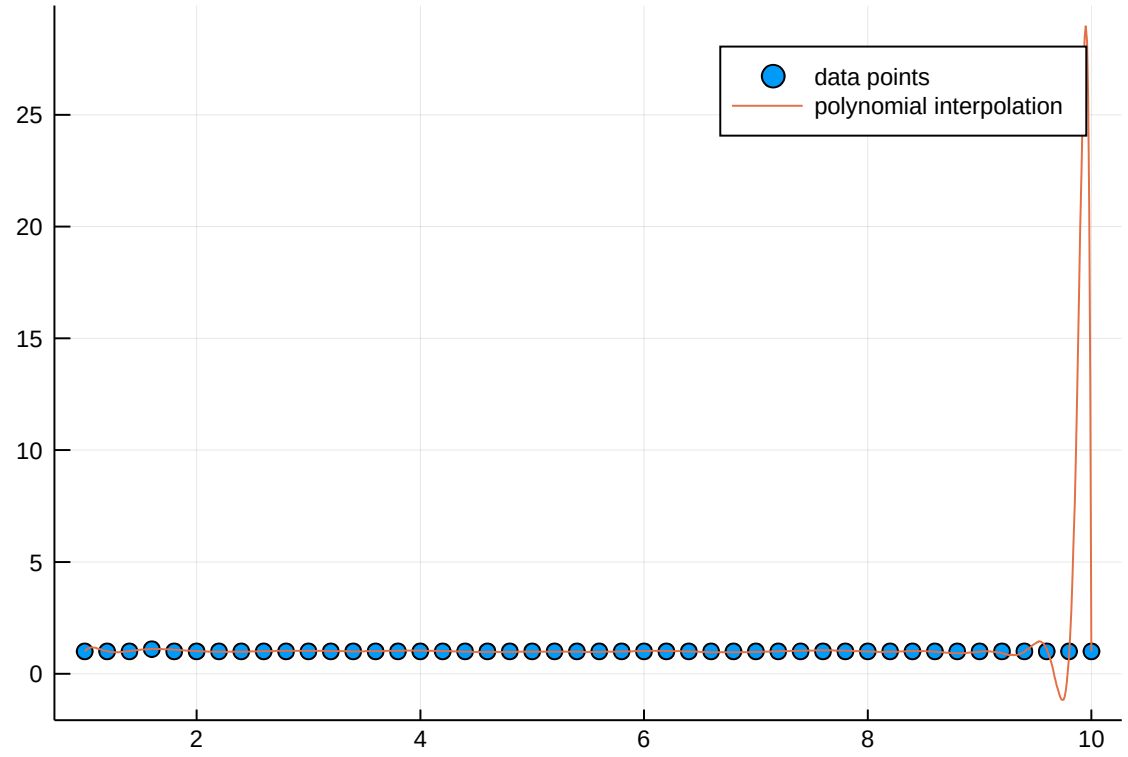
```
x =  1:1:10
y = [1.0 for i in x]
y[4] = 1.1
using Plots
scatter(x,y, label="data points")
xp = 1:0.01:10

using Polynomials
fit1=polyfit(x, y)
yf=[fit1(i) for i in xp]
plot!(xp,yf, label="polynomial interpolation")
```

Kod źródłowy 6. Funkcja prezentujące efektu Rungego



Rysunek 8. Demonstracja efektu Rungego. Mała gęstość punktów

Następnie zwiększono zagęszczenie punktów na podanym przedizale pięciokrotnie.



Rysunek 9. Demonstracja efektu Rungego. Duża gęstość punktów

Mimo że kształt rozmieszczenia punktów został mniej więcej zachowany to na samym końcu funkcja interpolująca ucieka do wartości około 30.

Jeżeli przeanalizujemy również wykresy z poprzednich rozdziałów, zauważymy takie samo zjawisko, zwane efektem Rungego.

7. Różne algorytmy interpolacji stosowane w grafice komputerowej

Celem ćwiczenia jest opis, demonstracja i porównanie działania dwóch metod interpolacji. Do wykonania ćwiczenia użyto języka Python.

Wybrałem dwie najpopularniejsze metody interpolacji: bilinear i bicubic.

Bilinear interpolation

Metoda interpolacji polegająca na obliczeniu wartości obrazu na podstawie siatki ustalonych punktów z jasno zdefiniowanymi wartościami natężenia koloru. Kolor obliczany jest na podstawie czterech najbliższych punktów. W przypadku zbioru współrzędnych

$$Q_{11}(0,0), Q_{12}(0,1), Q_{21}(1,0) \text{ i } Q_{22}(1,1)$$

problem upraszcza się do postaci:

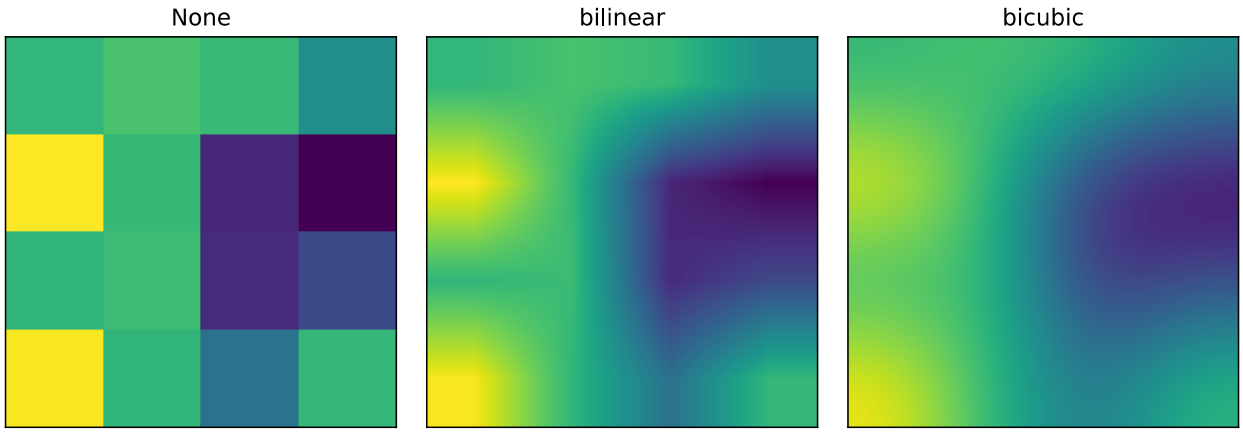
$$f(x,y) \approx f(0,0)\,(1-x)(1-y) + f(1,0)\,x(1-y) + f(0,1)\,(1-x)y + f(1,1)xy.$$

Bicubic interpolation

Metoda interpolacji polegająca na obliczaniu wartości obrazu przez dopasowanie wielomianu trzeciego stopnia, który będzie odzwierdziedlał pośrednie wartości natężenia koloru pomiędzy punktami. Metoda wykorzystuje wielomian trzeciego stopnia, przez co trzeba wyliczyć wartość czterech współczynników wielomianu, co powoduje, że w przypadku dwu wymiarowym potrzeba co najmniej 4*4 = 16 punktów. Istnieją sposoby na obliczenie wielomianu przy uzyciu stablicowanych wartości.

```
import matplotlib.pyplot as plt
import numpy as np

methods = [None, 'bilinear', 'bicubic']

# Fixing random state for reproducibility
np.random.seed(19680801)

grid = np.random.rand(4, 4)

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(9.3, 6),
                        subplot_kw={'xticks': [], 'yticks': []})

fig.subplots_adjust(left=0.03, right=0.97, hspace=0.3, wspace=0.05)

for ax, interp_method in zip(axs.flat, methods):
    ax.imshow(grid, interpolation=interp_method, cmap='viridis')
    ax.set_title(str(interp_method))

plt.tight_layout()
plt.show()
```

Kod źródłowy 7. Funkcje interpolujace w grafice



Rysunek 10. Przedstawienie działania metod interpolacji grafiki

Metoda bicubic korzysta z większej ilości punktów, co jest uwzględniane przy konsturowaniu interpolowanego obrazu, dzięki czemu uzyskujemy gładsze krawędzie. Widać to szczególnie nad ciemnogranatowym kwadratem.