Interpolacja - sprawozdanie

Aleksandra Smoter

April 2019

Contents

Listings

List of Figures

1 Interpolacja Lagrange'a

Implementacja interpolacji wielomianowej ze wzoru na wielomian interpolacyjny Lagrange'a (1) w języku Julia

$$L_k(x) = \frac{d}{m} = \prod_{i=0, i=k}^{n} \frac{x - x_i}{x_k - x_i}, \qquad P_n(x) = \sum_{k=0}^{n} L_k(x) f_k(x) \tag{1}$$

Oznaczenia:

- X, Y - tablice przechowujące odpowiednio współrzędne x i y węzłów interpolacji
- n - ilość węzłów interpolacji

Listing 1: Funkcja Lagrange

```
# Funkcja do wyznaczenia wartosci wielomianu interpolacyjnego w pkt x_i
function Lagrange(X, Y, x_i, n)
    result = 0
    for i = 1:n
        y_i = Y[i]
        p_i = 0
        l_i = 1
        for j = 1:n
            if(j != i)
                l_i = l_i*(x_i-X[j])/(X[i]-X[j])
            end
        end
        p_i = l_i * y_i
        result += p_i
    end
    result
end
```

Listing 2: Wykres interpolacji Lagrange'a

```
using Plots, Polynomials

n = 10  # ilosc wezlow interpolacji

# wylosowanie wezlow interpolacji
p =  1.0:n

# zapisanie wspolrzednych x i y odpowiednio do tablicy X i Y
X = [x for x in p]
Y = [rand() for x in X]

# narysowanie na wykresie wezlow interpolacji
scatter(X, Y, label="data points", color = "blue", xlabel = "x", ylabel = "y")

# wybranie zageszczonych punktow do narysowania wykresu funkcji interpolujacej
pd=1:0.01:n

# obliczenie wartosci wielomianu interp.dla wszystkich x z przedzialu pd
# za pomoca funkcji Lagrange
L = [Lagrange(X, Y, x, n) for x in pd]

#narysowanie wielomianu interpolacyjnego na wykresie
plot!(pd, L, label = "Lagrange's Interpolation", color = "red")
```

Po wykonaniu czynności z Listingu 2 otrzymuję poniższy wykres, przedstawiający wielomian interpolacyjny, wyznaczony za pomocą funkcji Lagrange.

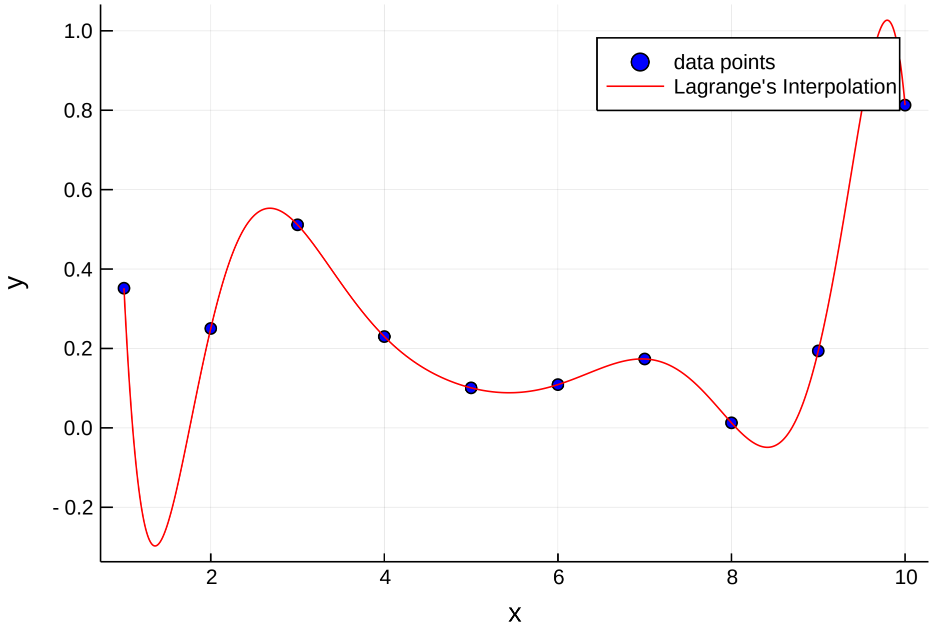

Figure 1: Interpolacja Lagrange'a

Wniosek:
Wielomian przedstawiony na wykresie przechodzi przez punkty interpolacyjne, więc funkcja została napisana poprawnie.

2 Interpolacja Newton'a

Implementacja interpolacji wielomianowej za pomocą metody ilorazów różnicowych (wielomianu interpolacyjnego Newtona (2)) w języku Julia

Wielomian interpolacyjny Newtona:

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + ... + a_n(x - x_0)(x - x_1)...(x - x_{n-1}) \tag{2}$$

Wprowadzam notację:

-> 0-wy iloraz różnicowy względem x_i:

$$x_i : f[x_i] = f(x_i) \tag{3}$$

-> 1-szy iloraz różnicowy względem x_i:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i} \tag{4}$$

-> k-ty iloraz różnicowy względem x_i:

$$f[x_i, x_{i+1}, ..., x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, ..., x_{i+k}] - f[x_i, x_{i+1}, ..., x_{i+k-1}]}{x_{i+1} - x_i} \tag{5}$$

Interpolacyjny wzór Newtona z ilorazami różnicowymi

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, ..., x_k](x - x_0)...(x - x_{k-1}) \tag{6}$$

Listing 3: dividedDiff function

```
# funkcja do wyznaczania ilorazow roznicowych ze wzoru (5)
function  dividedDiff(X, Y, i)
    n = length(X)
    F = zeros(n)
    for j=1:n-i
        F[j]=(Y[j+1, i+1] - Y[j, i+1])/(X[j+i] - X[j])
    end
    Y[:,2+i] = F
end
```

Listing 4: Factors function

```
# funckja do wyznaczenia macierzy wspolczynnikow
function Factors(X, Y)
    n = length(X)
    F = zeros(n, n+2)
    F[:,1] = X
    F[:,2] = Y
    for i=1:n
        dividedDiff(X, F, i)
    end
    return F
end
```

Listing 5: Newton function

```
# funkcja do obliczenia wartosci wielomianu interpolacyjnego w pkt x_i
function Newton(C, x_i)
    result = C[1,2]
    n = length(C[:,1]) - 1
    for i=1:n
        product = C[1, i+2]
        for j=1:i
            product = product * (x_i - C[j,1])
        end
        result = result + product
    end
    return result
end
```

Listing 6: Newton interpolation - plot

```
using Plots, Polynomials

n = 10
p =  1.0:n

X = [x for x in p]
Y = [rand() for x in X]

scatter(X, Y, label="data points", color = "blue", xlabel = "x", ylabel = "y")

pd=1:0.01:n

# obliczenie wartosci wielomianu interp. dla wszystkich x z przedzialu pd
# za pomoca funkcji Newton
C = Factors(X, Y)
N = [Newton(C, x) for x in pd]

plot!(pd, N, label = "Newton's interpolation", color = "blue")
```

Po wykonaniu czynności z Listingu 6 otrzymuję poniższy wykres, przedstawiający wielomian interpolacyjny, wyznaczony za pomocą funkcji Newton.

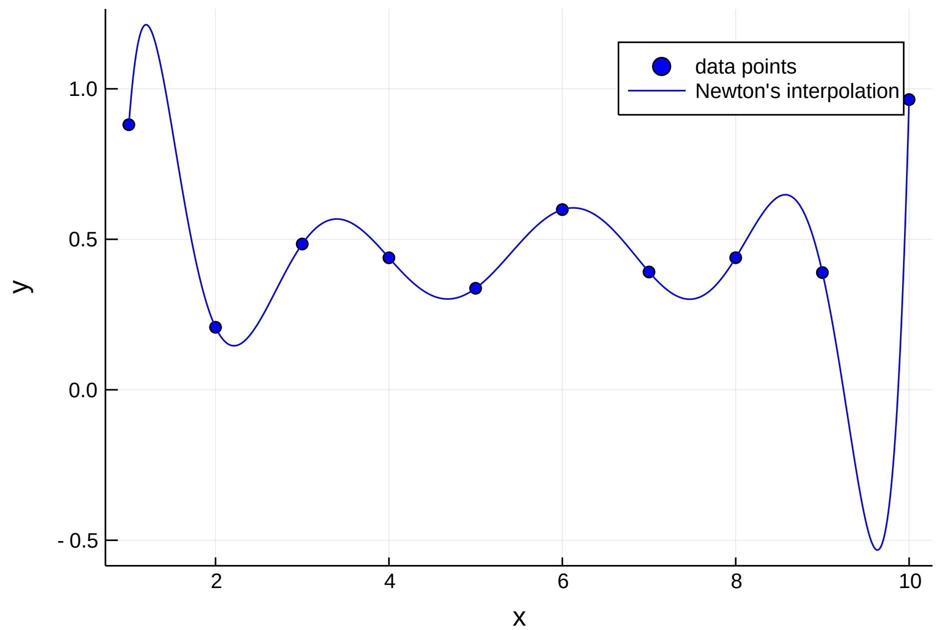

Figure 2: Interpolacja Newtona'a

Wniosek:
Wielomian przedstawiony na wykresie przechodzi przez punkty interpolacyjne, więc funkcja została napisana poprawnie.

3 Porównanie interpolacji Lagrange'a, Newton'a i polyfit

Porównanie interpolacji:
-> Lagrange'a
-> Newton'a
-> funkcją polyfit z modułu Polynomials

Listing 7: Porównanie interpolacji

```
#Pkg.add("Interpolations")
using Plots, Polynomials

n = 10
p =  1.0:n

X = [x for x in p]
Y = [rand() for x in X]

scatter(X, Y, label="data points", color = "blue", xlabel = "x", ylabel = "y")

pd=1:0.01:n

# wartosci wielomianu interp. dla wszystkich x z przedzialu pd
# obliczone za pomoca funkcji Lagrange
L = [Lagrange(X, Y, x, n) for x in pd]
```

```
# obliczone za pomoca funkcji Newton
C = Factors(X, Y)
N = [Newton(C, x) for x in pd]

# obliczone za pomoca funkcji polyfit z modulu Polynomials
fit1=polyfit(p, Y)
P = [fit1(x) for x in pd]

# narysowanie wielomianow interpolacyjnych na wykresie
plot!(pd, L, label = "Lagrange's Interpolation", color = "red")
plot!(pd, N, label = "Newton's interpolation", color = "blue")
plot!(pd, P, label = "Polynomial interpolation", color = "green")
```

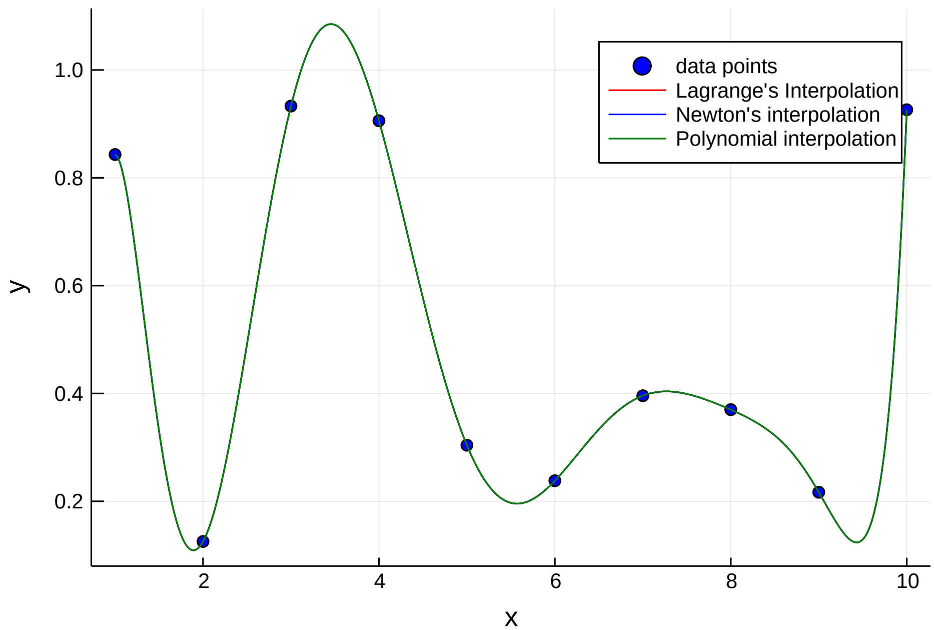

Figure 3: Porównanie rodzajów interpolacji

Wniosek:
Wykresy napisanych przeze mnie funkcji Lagrange i Newton oraz tej z modułu polynomial pokrywają się, z czego wynika, że wielomiany interpolacyjne są wyznaczone jednoznacznie

Wiedząc, że wszystkie wymienione wyżej funkcje działają poprawnie, sprawdzam, która z nich jest najbardziej efektywna pod względem czasu.

Dla każdej z funkcji: Lagrange, Newton i polyfit wykonuję po 10 pomiarów czasu, dla n węzłów interpolacyjnych, gdzie n $\in$ $10, 20, 30, 40, 50, 60, 70, 80, 90, 100$. Wszystkie pomiary zapisywane są do tabeli: Dim (ilość węzłów), Type (rodzaj wykorzystanej funkcji) i Time (czas wykonywania funkcji) (Listing 8)

Listing 8: Porownanie czasow interpolacji

```
using DataFrames, Polynomials
Dim = Int[]
Time = Float64[]
Type = []
for n = 10:10:100
    for i = 1:10
        p =  1.0:n
        X = [x for x in p]
        Y = [rand() for x in X]
        pd=1:0.01:n

        push!(Dim, n)
        push!(Type, "L")
        L = [Lagrange(X, Y, x, n) for x in pd]
        push!(Time, @elapsed L)

        push!(Dim, n)
        push!(Type, "N")
        C = Factors(X, Y)
        N = [Newton(C, x) for x in pd]
        push!(Time, @elapsed N)

        push!(Dim, n)
        push!(Type, "P")
        fit1=polyfit(p, Y)
        P = [fit1(x) for x in pd]
        push!(Time, @elapsed P)
    end
end
```

Następnie tabele Dim, Type i Time łączę w data frame, który poddaję obróbce: agregacja względem rodzaju funkcji, obliczenie średniej wartości czasu oraz niepewności pomiarowej (odchylenia standardowego). Otrzymane wyniki przedstawiam na wykresie (Listing 9)

Listing 9: Porownanie czasow interpolacji - wykres

```
using DataFrames, Statistics, Plots

data = DataFrame()
data[:Dim] = Dim
data[:Type] = Type
data[:Time] = Time

time_df = by(data, [:Dim,:Type], avgTime = :Time => mean, stdDev = :Time => std)

p=scatter(time_df[:Dim],time_df[:avgTime],group=time_df[:Type],
    title="Multiplication time",yerr=time_df[:stdDev],xlabel="N",ylabel="time [ms]",
    size=(900,600),legend=true)

plot(p)
```

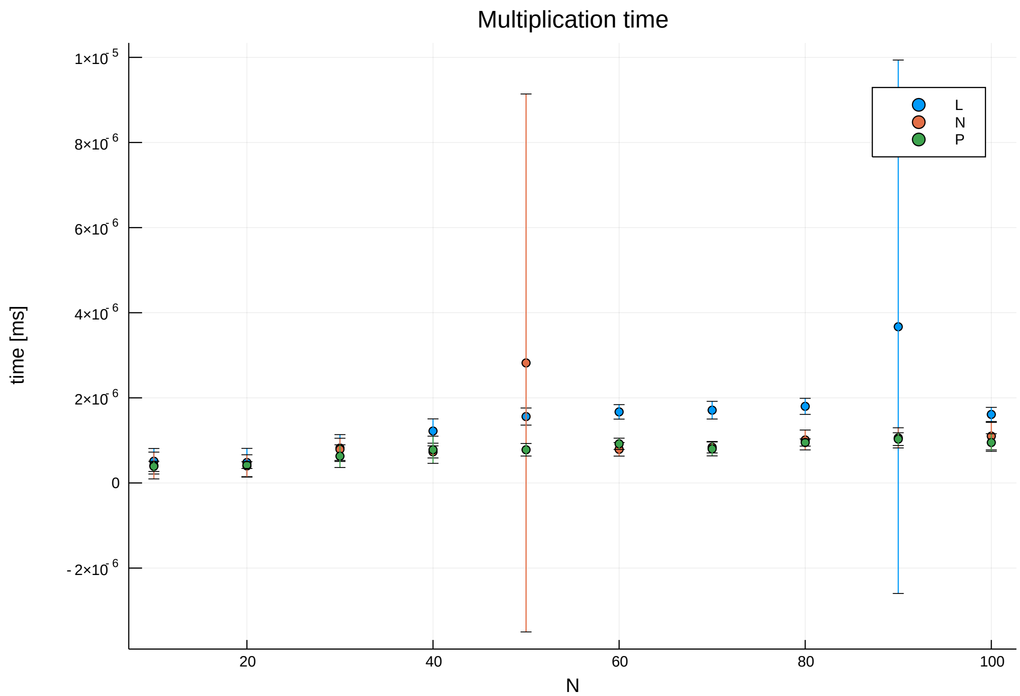

Figure 4: Porównanie czasów interpolacji

Wniosek: Z otrzymanego wykresu wynika, że najmniej efektywna jest funkcja Lagrange. Wynika to z tego, że przy tym rozwiązaniu przy każdej wartości wielomianu wykonujemy te same obliczenia. W metodzie Newtona tylko raz wyznaczamy ilorazy różnicowe, a w kolejnych iteracjach korzystamy z już wyliczonych wartości.

4 Interpolacja funkcjami sklejanymi

Interpolacja funkcjami sklejanymi pierwszego stopnia (liniowymi) - funkcja LinearInterpolation z modułu Interpolations w języku Julia

Listing 10: Interpolacja funkcjami sklejanymi liniowymi

```
using Interpolations

# wylosowanie wezlow interpolacji
n = 10
p =  1.0:n

X = [x for x in p]
Y = [rand() for x in X]

# narysowanie na wykresie wezlow interpolacji
scatter(X, Y, label="data points", color = "blue", xlabel = "x",
    ylabel = "y")

# wybranie zageszczonych punktow do rysowania wykresow funkcji interpolujacych
pd=1:0.01:n

# narysowanie wielomianu interpolacyjnego za pomoca funkcji sklejanych
linear_interp = LinearInterpolation(p, Y)
Ln = [linear_interp(x) for x in pd]
plot!(pd, Ln, title = "Interpolacja funkcjami sklejanymi - liniowa",
    label = "Linear interpolation", color = "dark orange")
```

Po wykonaniu czynności z Listingu 10 otrzymuję poniższy wykres, przedstawiający wielomian interpolacyjny, wyznaczony za pomocą funkcji LinearInterpolation z modułu Interpolations w języku Julia

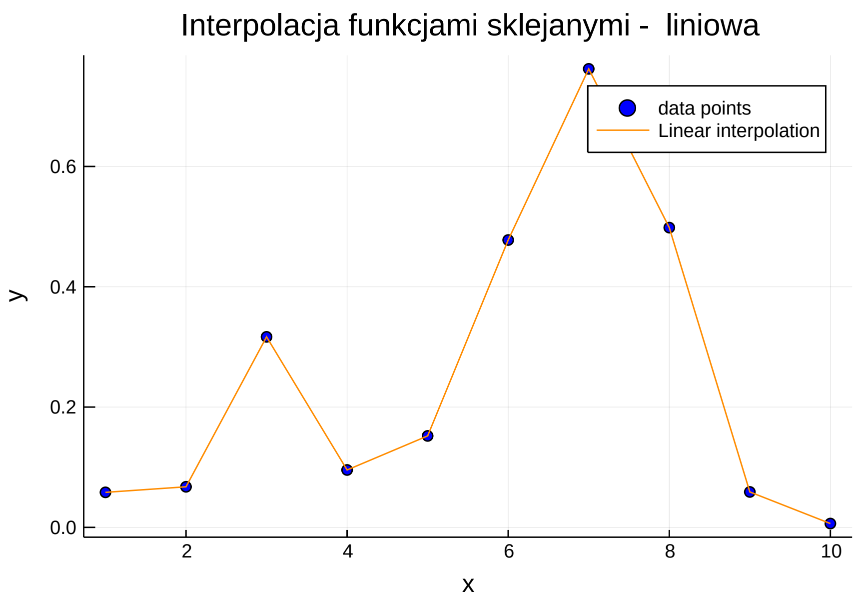

Figure 5: Interpolacja funkcjami sklejanymi liniowymi

Interpolacja funkcjami sklejanymi trzeciego stopnia (sześciennymi) - funkcja CubicSplineInterpolation z modułu Interpolations w języku Julia

Listing 11: Interpolacja funkcjami sklejanymi sześciennymi

```
#  interpolacja szescienna
using Interpolations

# wylosowanie wezlow interpolacji
n = 10
p =  1.0:n

# zapisane wspolrzednych x i y odpowiednio do tablicy X i Y
X = [x for x in p]
Y = [rand() for x in X]

# narysowanie na wykresie wezlow interpolacji
scatter(X, Y, label = "data points", color = "blue", xlabel = "x",
    ylabel = "y")

# wybranie zageszczonych punktow do rysowania wykresow funkcji interpolujacych
pd=1:0.01:n

interp_cubic = CubicSplineInterpolation(p, Y)
C = [interp_cubic(x) for x in pd]
plot!(pd, C, title = "Interpolacja funkcjami sklejanymi - szescienna",
    label = "Cubic interpolation", color = "magenta")
```

Po wykonaniu czynności z Listingu 11 otrzymuję poniższy wykres, przedstawiający wielomian interpolacyjny, wyznaczony za pomocą funkcji CubicSplineInterpolation z modułu Interpolations w języku Julia

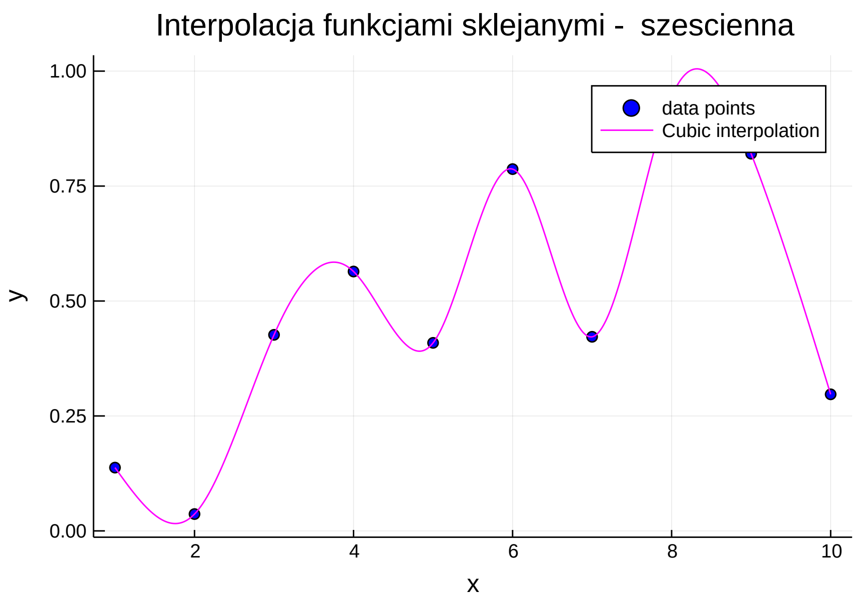

Figure 6: Interpolacja funkcjami sklejanymi sześciennymi

5 Efekt Rungego

Efekt Rungego - zjawisko pogorszenia interpolacji, pomimo zwiększenia ilości węzłów, szczególnie zauważalne na końcach przedziałów.

Listing 12: Efekt Rungego

```
using Interpolations, Polynomials

# wylosowanie wezlow interpolacji
n = 15
p =  1.0:n

# zapisane wspolrzednych x i y odpowiednio do tablicy X i Y
X = [x for x in p]
Y = [rand() for x in X]

# narysowanie na wykresie wezlow interpolacji
scatter(X, Y, label = "data points", color = "blue", xlabel = "x",
    ylabel = "y")

# wybranie zageszczonych punktow do rysowania wykresow funkcji interpolujacych
pd=1:0.01:n

fit1=polyfit(p, Y)
P = [fit1(x) for x in pd]
plot!(pd, P, title = "Runge effect", label = "Polynomial interpolation",
    color = "green")
```

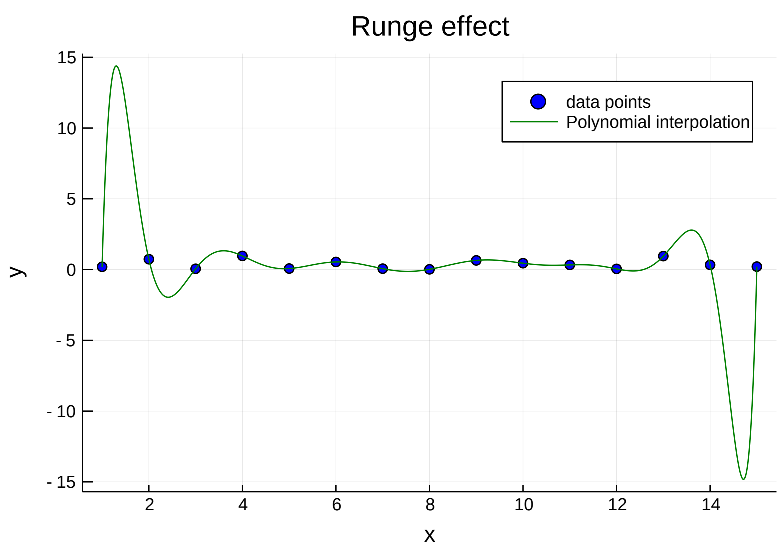

Figure 7: Efekt Rungego

W przypadku dobierania węzłów równoodległych wraz ze wzrostem n dochodzi do wzrostu błędu pomiędzy węzłami, co jest spowodowane wzrostem stopnia wielomianu interpolującego. Wybierając punkty interpolujące w zerach wielomianów Czebyszewa efekt Rungego nie występuje - jest to związane z większą gęstością zer wielomianów na krańcach przedziału.

Listing 13: Efekt Rungego - funkcje sklejane

```
# zniwelowanie efektu Rungego za pomoca funkcji sklejanych
cubic_interp = CubicSplineInterpolation(p, Y)
C = [cubic_interp(x) for x in pd]
plot!(pd, C, label = "Cubic interpolation", color = "magenta")
```

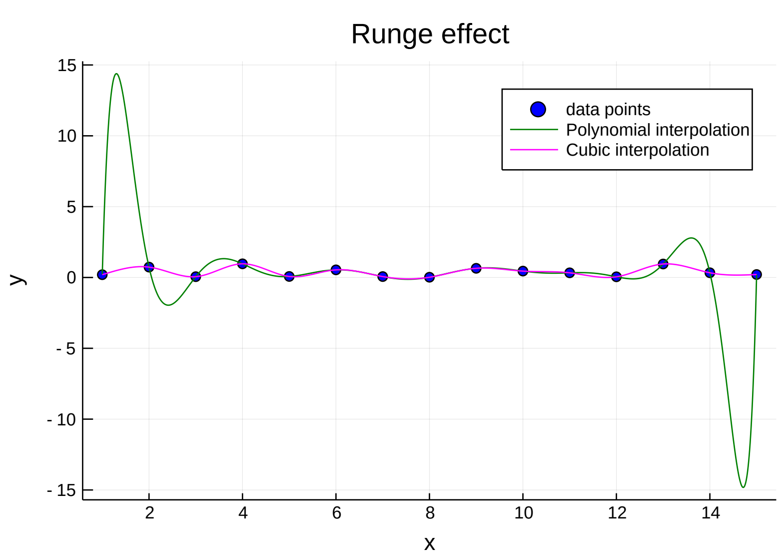

Figure 8: Efekt Rungego - porównanie interpolacji wielomianowej i funkcjami sklejanymi

Wniosek:
Efekt Rungego nie występuje również podczas interpolacji funkcjami sklejanymi.