Metody Obliczeniowe w Nauce I Technice

Laboratorium 5: Interpolacja – Sprawozdanie

Jakub Pajor

1. Napisać własną implementację interpolacji wielomianowej stosując wprost wzór na wielomian interpolacyjny Lagrange'a . Język implementacji do wyboru (Julia, C). Przetestować swoją implementację na wylosowanych węzłach interpolacji w wybranym przedziale. Narysować wykres wielomianu interpolacyjnego w tym przedziale wraz z wezlami interpolacji.

Definition:[1]

Given a set of $k+1$ data points

$$(x_0, y_0), \ldots, (x_j, y_j), \ldots, (x_k, y_k)$$

where no two x_j are the same, the **interpolation polynomial in the Lagrange form** is a linear combination

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

of Lagrange basis polynomials

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

where $0 \le j \le k$. Note how, given the initial assumption that no two x_j are the same, $x_j - x_m \ne 0$, so this expression is always well-defined. The reason pairs $x_i = x_j$ with $y_i \ne y_j$ are not allowed is that no interpolation function L such that $y_i = L(x_i)$ would exist; a function can only get one value for each argument x_i. On the other hand, if also $y_i = y_j$, then those two points would actually be one single point.

For all $i \ne j$, $\ell_j(x)$ includes the term $(x - x_i)$ in the numerator, so the whole product will be zero at $x = x_i$:

$$\ell_{j \ne i}(x_i) = \prod_{m \ne j} \frac{x_i - x_m}{x_j - x_m} = \frac{(x_i - x_0)}{(x_j - x_0)} \cdots \frac{(x_i - x_i)}{(x_j - x_i)} \cdots \frac{(x_i - x_k)}{(x_j - x_k)} = 0.$$

On the other hand,

$$\ell_i(x_i) := \prod_{m \ne i} \frac{x_i - x_m}{x_i - x_m} = 1$$

In other words, all basis polynomials are zero at $x = x_i$, except $\ell_i(x)$, for which it holds that $\ell_i(x_i) = 1$, because it lacks the $(x - x_i)$ term.

It follows that $y_i \ell_i(x_i) = y_i$, so at each point x_i, $L(x_i) = y_i + 0 + 0 + \cdots + 0 = y_i$, showing that L interpolates the function exactly.

Implementation:

```julia
function l(k, X)
    x_k = X[k]
    X = [x for x in X if x != x_k]
    p = Poly([1.0])
    q = 1
    for x_i in X
        p = p * poly([x_i])
        q = q * (x_k - x_i)
    end
    return (p / q)
end

function L(X, Y)
    p = Poly([0])
    for k in 1:1:length(Y)
        p = p + (Y[k] * l(k, X))
    end
    return p
end
```

Code 1: implementation of given definition

[1] https://en.wikipedia.org/wiki/Lagrange_polynomial#Definition

Then I generated random points to carry out interpolations.

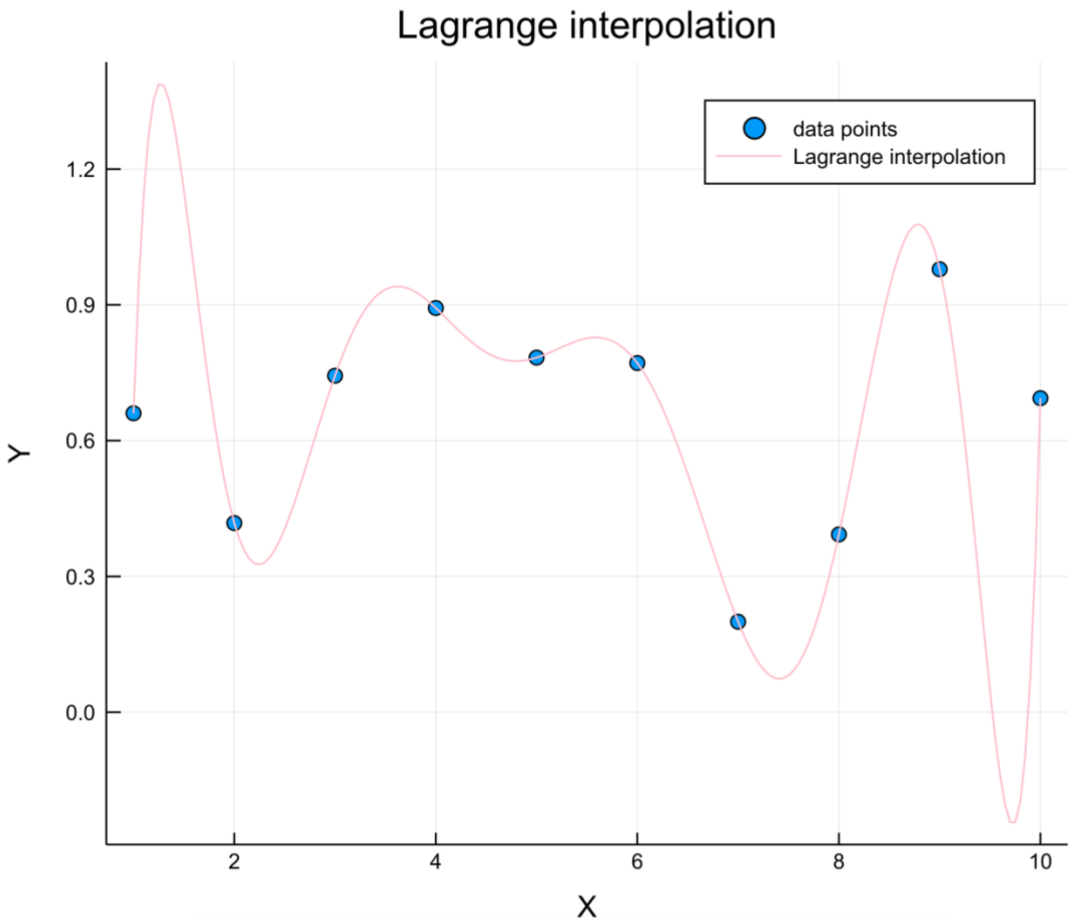
```
x = 1:1:10
y = [rand() for a in x]
xs = 1.0:0.05:10.0
```

Code 2: interpolation points

After that I used written function to calculate Lagrange interpolating polynomial, scatter created points and plot the polynomial.

```
equL = L(x, y)

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

plot!(xs, polyval(equL, xs),
    color=:pink,
    label = "Lagrange interpolation",
    xlabel = "X",
    ylabel = "Y",
    title = "Lagrange interpolation",
    dpi = 120,
    size = (600,500)
    )
```

Code 3: polynomial and plot



Plot 1: Lagrange's interpolation

2. Zrobić to samo dla metody Newtona (metoda ilorazów różnicowych). Zadbać o to, żeby ilorazy wyliczać tylko raz dla danego zbioru węzłów interpolacji. Język implementacji wybrać taki sam, jak w poprzednim punkcie. Narysować wykres wielomianu interpolacyjnego dla tych samych danych, co w poprzednim punkcie.

Definition:[2]

Given a set of $k + 1$ data points

$$(x_0, y_0), \ldots, (x_j, y_j), \ldots, (x_k, y_k)$$

where no two x_j are the same, the Newton interpolation polynomial is a linear combination of **Newton basis polynomials**

$$N(x) := \sum_{j=0}^{k} a_j n_j(x)$$

with the Newton basis polynomials defined as

$$n_j(x) := \prod_{i=0}^{j-1} (x - x_i)$$

for $j > 0$ and $n_0(x) \equiv 1$.

The coefficients are defined as

$$a_j := [y_0, \ldots, y_j]$$

where

$$[y_0, \ldots, y_j]$$

is the notation for divided differences.

Thus the Newton polynomial can be written as

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

Implementation:

```
function comp_n(X, y_k, k, p_k)
    x_k = X[k]
    p = y_k - polyval(p_k, x_k)
    q = 1
    for i in 1:1:k-1
        q = q * (x_k - X[i])
    end
    return (p / q)
end

function N(X, Y, n)
    if n == 1
        Poly(float(Y[1]))
    else
        pp = N(X, Y, n-1)
        c = comp_n(X, Y[n], n, pp)
        poly([X[i] for i in 1:1:n-1]) * c + pp
    end
end

function N(X, Y)
    N(X, Y, length(Y))
end
```

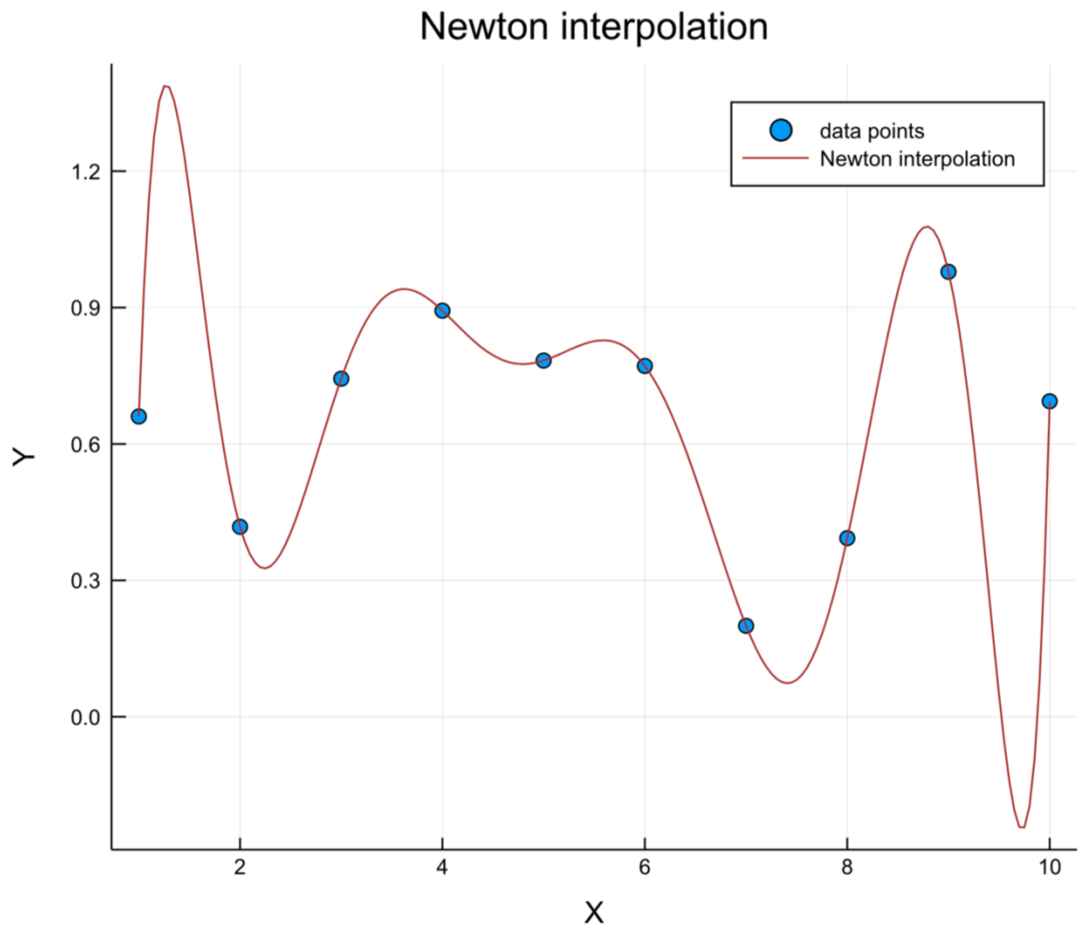
Code 4: implementation of Newton method

[2] https://en.wikipedia.org/wiki/Newton_polynomial#Definition

After that I used written function to calculate Newton's interpolating polynomial, scatter created points and plot the polynomial.

```
equN = N(x, y)

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

plot!(xs, polyval(equN, xs),
    color=:brown,
    label = "Newton interpolation",
    xlabel = "X",
    ylabel = "Y",
    title = "Newton interpolation",
    dpi = 120,
    size = (600,500))
```

Code 5: polynomial and plot



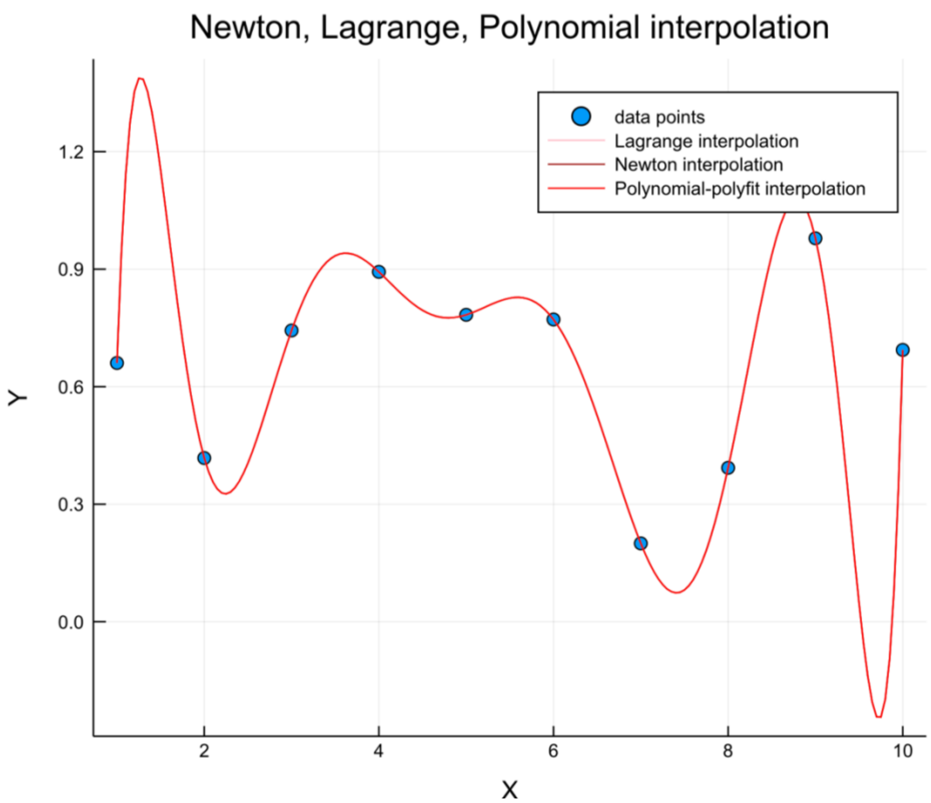
Plot 2: Newton's interpolation

3. Zastosowac interpolację wielomianową z pakietu Polynomials do tych samych danych, co w poprzednich punktach. Porównać wszystkie 3 wyniki interpolacji wielomianowej na jednym wykresie. Co zauważamy? Dlaczego?

So I calculated Polynomial which fits generated points using embedded polyfit function and plotted each kind of interpolation.

```
equP = polyfit(x, y)

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

plot!(xs, polyval(equL, xs),
    color=:pink,
    label = "Lagrange interpolation",
)

plot!(xs, polyval(equN, xs),
    color=:brown,
    label = "Newton interpolation")

plot!(xs, polyval(equP, xs),
    color=:red,
    label = "Polynomial-polyfit interpolation",
    xlabel = "X",
    ylabel = "Y",
    title = "Newton, Lagrange, Polynomial interpolation",
    dpi = 120,
    size = (600,500))
```

Code 6: poly, Lagrange's, Newton's Interpolation



Plot 3: Langrange's, Newton's, Polyfit interpolation

Observation: each of interpolating polynomials goes through the same points. It happens, because for each of given n + 1 points exists only one n'th degree polynomial.

4. Porównać metody poprzez pomiar czasu wykonania dla zmiennej ilości węzłów interpolacji. Dokonać pomiaru 10 razy i policzyć wartość średnią oraz oszacować błąd pomiaru za pomocą odchylenia standardowego.

First I created specific DataFrame and then I filled it 10 times with 100 to 500 points; step 100. In the end determined mean for each “bucket” and it’s standard deviation

```
df = DataFrame(type=String[], size = Int64[], time = Float64[])

types=[];sizes=[];times=[]
range = 100:100:500
for rr in range
    xx = 0:1:rr
    for i in 1:1:10
        yy = rand(Int, rr+1)
        push!(df,["lagrange", rr, @elapsed L(xx,yy)])
        push!(df, ["newton", rr, @elapsed N(xx,yy)])
        push!(df, ["polyfit", rr, @elapsed polyfit(xx,yy)])
    end
end

pol_res = by(df, [1,2]) do dff
    DataFrame(time_mean = mean(dff[3]), time_std = std(dff[3]))
end
```

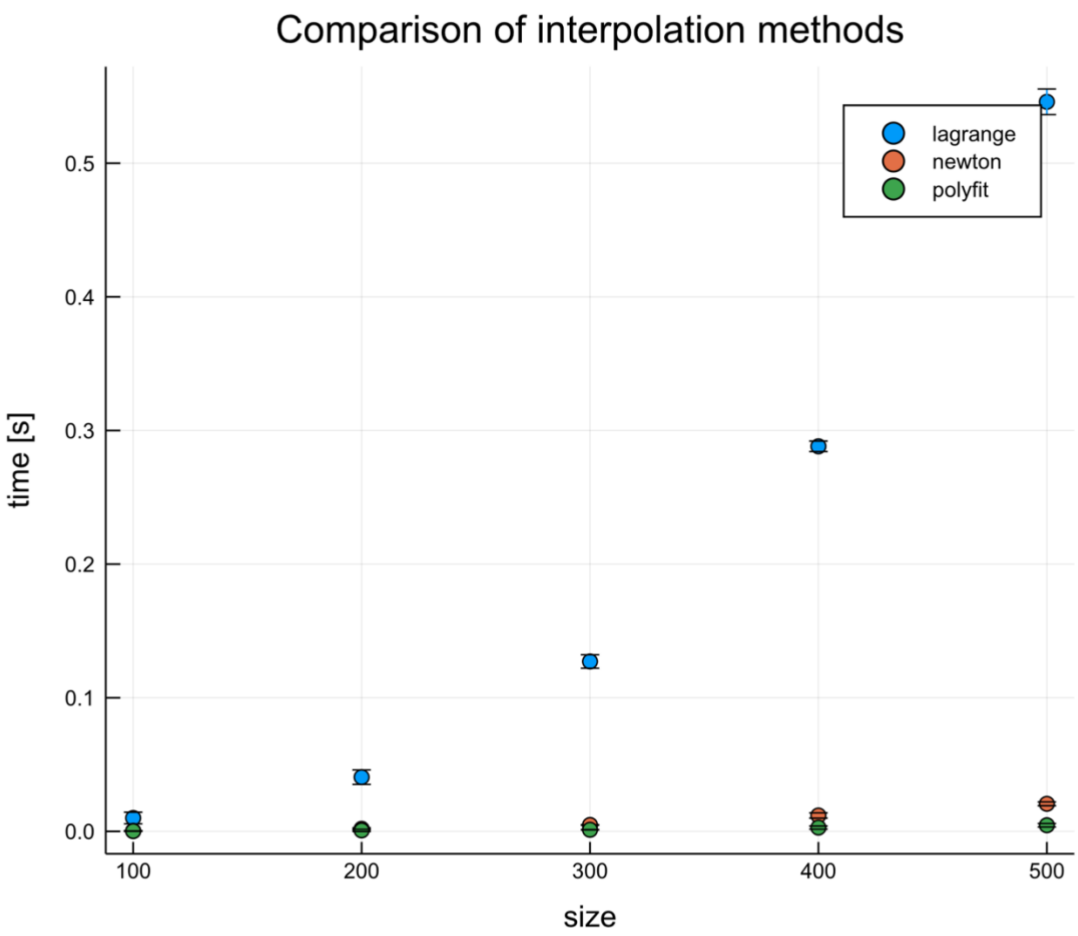
Code 7: dataframe, std, mean

	type	size	time_mean	time_std
	String	Int64	Float64	Float64
1	lagrange	100	0.0100212	0.00435218
2	newton	100	0.000363283	0.000105236
3	polyfit	100	0.000195765	0.000147574
4	lagrange	200	0.0405791	0.00540252
5	newton	200	0.00191023	0.000790464
6	polyfit	200	0.000724788	0.000814945
7	lagrange	300	0.127135	0.00505647
8	newton	300	0.00477466	0.000192925
9	polyfit	300	0.00120259	0.000118685
10	lagrange	400	0.28814	0.0039224
11	newton	400	0.0118855	0.00200529
12	polyfit	400	0.00283653	0.00120733
13	lagrange	500	0.54594	0.0095752
14	newton	500	0.0206329	0.00141171
15	polyfit	500	0.00458597	0.00128937

Table 1: Interpolation method times

```
scatter(pol_res[:size],
        pol_res[:time_mean],
        group = pol_res[:type],
        yerr = pol_res[:time_std],
        xlabel = "size",
        ylabel = "time [s]",
        dpi = 120,
        size = (600,500),
        title = "Comparison of interpolation methods")
```

Code 8: scatter of table 1



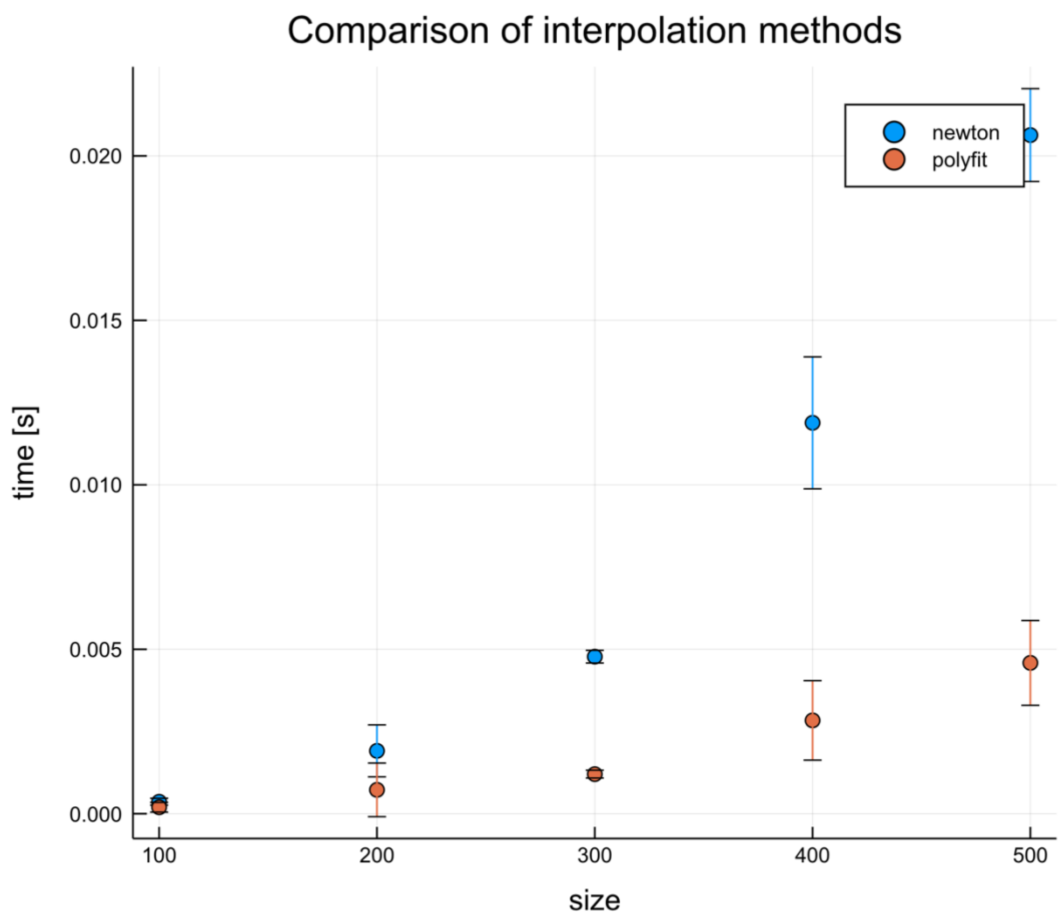
Plot 4: comparison of interpolation methods #1

As we can see using Langrange's polynomial the time is much slower than Newton's or using embedded polyfit.

After that I created another scatter to distinguish Newton's and Polyfit times.

```
## dataframe without lagrange
pol_res_noL = pol_res[pol_res[:type].!= "lagrange", :]

scatter(pol_res_noL[:size],
        pol_res_noL[:time_mean],
        group = pol_res_noL[:type],
        yerr = pol_res_noL[:time_std],
        xlabel = "size",
        ylabel = "time [s]",
        dpi = 120,
        size = (600,500),
        title = "Comparison of interpolation methods")
```

Code 9: Newton's, Polyfit times



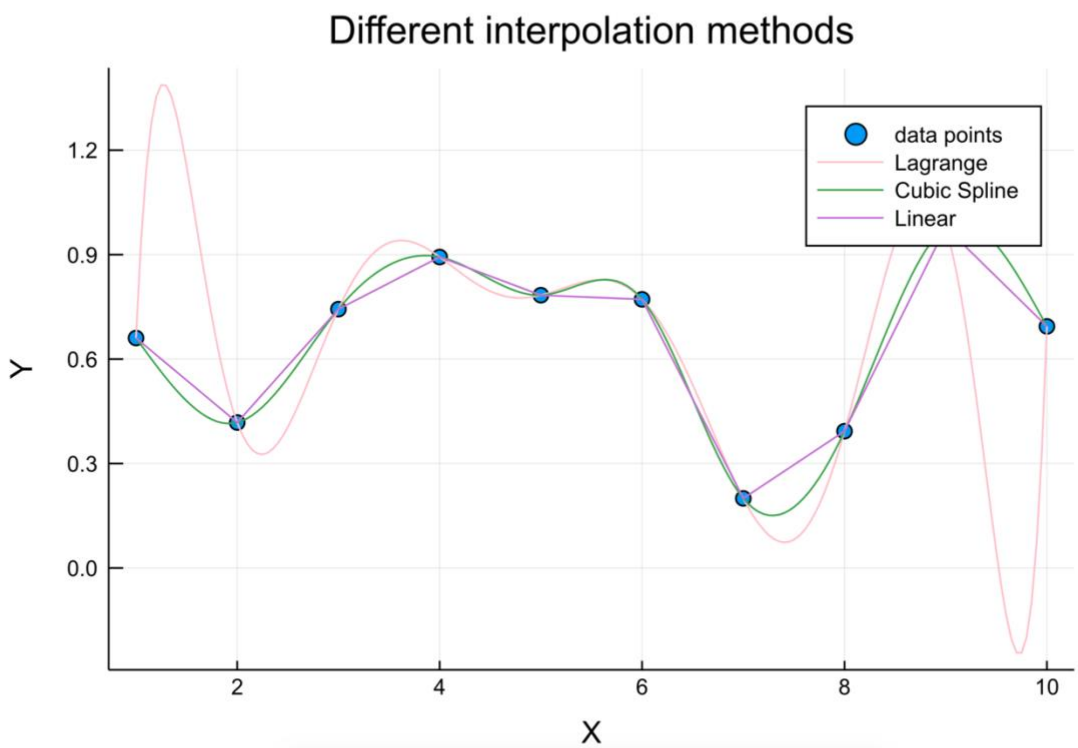
Plot 5: comparison of interpolations methods #2

Even Newton's interpolation is slower by a square of polyfit time.

5. Poeksperymentować z interpolacją funkcjami sklejanymi (minimum dwie rożne funkcje sklejane), narysować wykresy i porównać z wykresami interpolacji wielomianowej.

```
scatter(x, y, label = "data points")

 plot!(xs, polyval(equL, xs),
     color=:pink,
     label = "Lagrange")

plot!(xs, [CubicSplineInterpolation(x,y)(i) for i in xs],
        label = "Cubic Spline")

plot!(xs, [interpolate(y, BSpline(Linear()))(i) for i in xs],
        label = "Linear",
        xlabel = "X",
        ylabel = "Y",
        title = "Different interpolation methods")
```

Code 10: plotting Lagrange, Cubic Spline, Linear interpolation



Plot 6: Lagrange, Cubic Spline, Linear interpolation

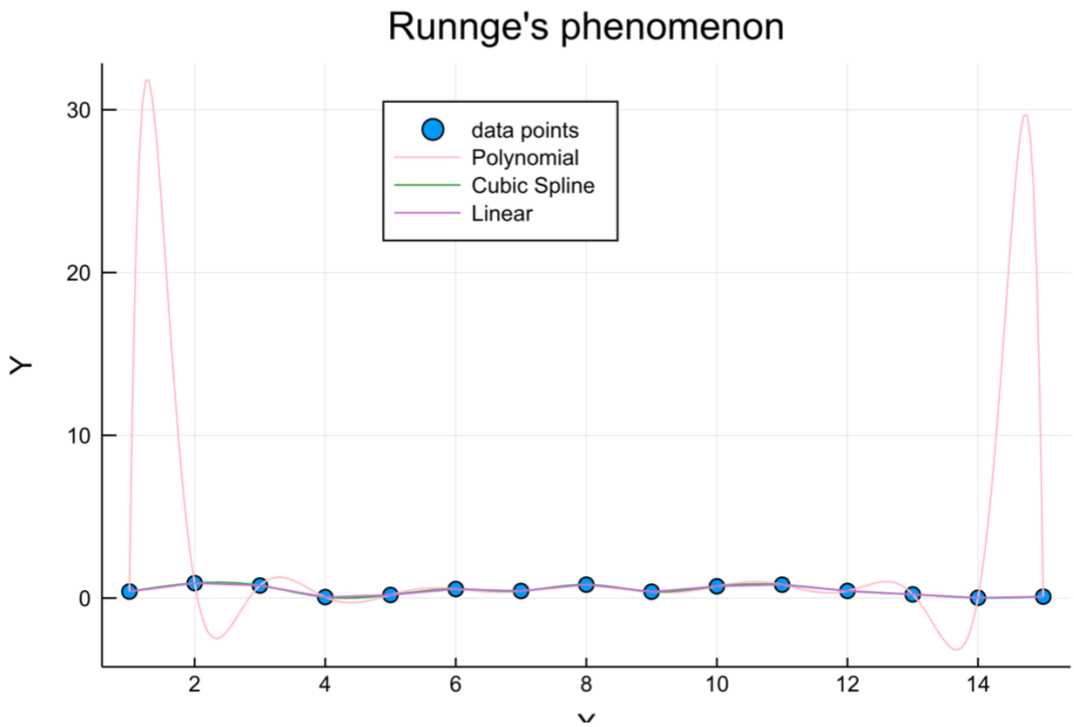
Lagrange's interpolation (so also Newton's interpolation) is in general the least accurate, but mostly on first and last edges. It is Runge's phenomenon.

6. Zademonstrować efekt Rungego.

I generated again random points but this time more of them (15) and plotted them with Polynomial, Cubic Spline and linear interpolation.

```
Rx = 1:1:15
Ry = [rand() for a in Rx]
Rxs = 1.0:0.05:15.0


scatter(Rx, Ry, label = "data points")

plot!(Rxs, polyval(polyfit(Rx,Ry), Rxs),
      color=:pink,
      label = "Polynomial")

plot!(Rxs, [CubicSplineInterpolation(Rx,Ry)(i) for i in Rxs],
      label = "Cubic Spline")

plot!(Rxs, [interpolate(Ry, BSpline(Linear()))(i) for i in Rxs],
      label = "Linear",
      legend = :top,
      xlabel = "X",
      ylabel = "Y",
      title = "Runnge's phenomenon")
```

Code 11: points, polyfit, cublic spline, linear



Plot 7: Runge's Phenomenon

Runge's phenomenon is a problem of oscillation at the edges of an interval that occurs when using polynomial interpolation with polynomials of high degree over a set of equispaced interpolation points. It shows that going to higher degrees does not always improve accuracy.

7. Zbadać i zademonstrować podczas zajęć różne algorytmy interpolacji stosowane w grafice komputerowej (np. do zmiany wielkości obrazu). Można korzystać z gotowych rozwiązań, ale trzeba wiedzieć, jak te algorytmy działają. Do zaliczenia tego zadania potrzebne jest demonstracja i porównanie działania co najmniej dwóch metod.

```
using Images
using FileIO
using ImageMagick
using Colors

totem = load("totem.ascii.pgm")
totem = float32.(totem)

function two_times_smaller(source, srcWidth, srcHeight)
    tgtWidth = srcWidth/2
    tgtHeight = srcHeight/2
    target = zeros(Int32(tgtHeight), Int32(tgtWidth))
    for y = 1:(Int32(tgtHeight) - 1)
        y2 = Int32(2*y)
        for x = 1:(Int32(tgtWidth) - 1)
            x2 = Int32(2*x)
            p = (source[y2, x2] + source[y2, x2 + 1]) / 2
            q = (source[y2 + 1, x2] + source[y2+1, x2+1]) / 2
            target[y, x] = (p + q) / 2
        end
    end
    return target
end
```

Code 12: image size reducing

```
function two_times_bigger(source, srcWidth, srcHeight)
    tgtWidth = srcWidth * 2
    tgtHeight = srcHeight * 2
    target = zeros(Int32(tgtHeight), Int32(tgtWidth))
    for y2 = 1:(Int32(tgtHeight) - 1)
        for x2 = 1:(Int32(tgtWidth) - 1)
            x = Int32(floor(x2/2))
            y = Int32(floor(y2/2))

            if(x < 1)
                x = 1
            elseif(x > srcWidth)
                x = srcWidth
            end

            if(y < 1)
                y = 1
            elseif(y > srcWidth)
                y = srcHeight
            end

            p = source[y,x]

            p1 = source[y, x+1]
            p2 = source[y+1, x]
            p3 = source[y+1,x+1]

            d1 = abs(p - p1)
            d2 = abs(p - p2)
            d3 = abs(p - p3)
            d4 = abs(p1 - p2)

            min = d1
            if min > d2
                min = d2
            elseif min > d3
                min = d3
            elseif min > d4
                min = d4
            end

            if min == d1
                target[y2, x2] = (p + p1)/2
            elseif min == d2
                target[y2, x2] = (p + p2)/2
            elseif min == d3
                target[y2, x2] = (p + p3)/2
            else
                target[y2, x2] = (p + (p1 + p2) / 2) / 2
            end
        end
    end
    return target
end


totem_smaller = two_times_smaller(venus, 640, 480)
totem_smaller = Gray.(venus1)
totem_smaller_to_original = two_times_bigger(venus1, 320, 240)
totem_smaller_to_original = Gray.(venus2)
print()
```
Code 13: image size enlarging


Picture 1: original size


Picture 2: reduced size

Picture 3: reduced and enlarged