



**AGH**

**AKADEMIA GÓRNICZO-HUTNICZA  
IM. STANISŁAWA STASZICA  
W KRAKOWIE**

**Złożone systemy cyfrowe  
2019/2020**

Rastrowy projektor laserowy  
Dokumentacja techniczna

Mateusz Wieczorek

October 16, 2019

# Contents

0.1	Cel projektu. . . . .	2
0.2	Założenia projektu . . . . .	2
0.3	Zastosowany układ cyfrowy . . . . .	3
0.4	Kolejne etapy rozwoju projektu. . . . .	3

## 0.1 Cel projektu.

Celem niniejszego projektu jest pogłębienie wiedzy i praktycznego doświadczenia w zakresie projektowania, konstruowania, budowania i programowania złożonego systemu cyfrowego. Ostatecznie celem projektu będzie w miarę możliwości otrzymanie działającego urządzenia, który będzie pełnił przedstawione później funkcje.

Zaprojektowane przeze mnie urządzenie powinno pełnić rolę projektora, który będzie w stanie wyświetlić obraz o wystarczająco dobrej rozdzielczości, który zostanie podany na wejście VGA. Obraz będzie wyświetlany po przez bardzo szybkie skanowanie wiązką laserową w osi pionowej i poziomej, wiązka laserowa będzie tworzyć poziome linie. W ten sposób uzyskamy obraz rastrowy (nie wektorowy jak w przypadku standardowych projektorów laserowych).

## 0.2 Założenia projektu

Głównymi założeniami są:

- a) dostępność materiałów konstrukcyjnych w możliwie najniższych cenach:
  - programowalny układ cyfrowy służący do zarządzania pracą urządzenia
  - lasery na prąd elektryczny służące jako źródło światła
  - silniki elektryczne służące jako układ odchylający wiązkę laserową w poziomie i pionie
  - materiał na walce, które zostaną zamontowane na osiach silników, oraz małe zwierciadła
  - ewentualne soczewki pozwalające na korekcje wiązki laserowej
  - zwierciadła dichroiczne
  - czujniki obrotów
  - wszelakie okablowanie
  - części na obudowę urządzenia, komponentów oraz szkielet
- b) dostępność wolnego czasu
- c) brak większych trudności w zakresie praw fizyki uniemożliwiających konstrukcję urządzenia
- d) dostępność narzędzi do zbudowania urządzenia

### 0.3 Zastosowany układ cyfrowy

Dokładnie jeszcze nie wiem. Najlepiej, żeby układ oferował szybki zegar pozwalający na dokładną synchronizację pracy komponentów, udostępniał łatwy do obsługi interfejs VGA oraz pozwolił na dokładną kontrolę napięć na pinach wyjściowych.

### 0.4 Kolejne etapy rozwoju projektu.

1. Urządzenie powinno rysować z jak największą częstotliwością poziomą linię.

Prosty układ powinien wyświetlać poziomą linię z jak największą częstotliwością. Na tym etapie będzie wymagane odpowiednie podłączenie lasera do układu cyfrowego, który będzie źródłem światła, oraz silnika elektrycznej charakteryzującego się wysokimi obrotami. Laser powinno się dać włączać i wyłączać programowo, tak samo powinno się dać sterować i kontrolować obroty silnika elektrycznego. Na osi silnika powinien być zamontowany płaski walec, na którego powierzchni bocznej będą przyklejone symetrycznie zwierciadélka o wymiarach 1cm x 1cm. Ten komponent będzie odpowiedzialny za odchyłanie wiązki w kierunku poziomym. Ilość rysowanych linii w ciągu sekundy będzie miała znaczenie w kwestii pionowej rozdzielczości oraz częstotliwości odzwierzania obrazu.

2. Urządzenie powinno rysować z jak największą częstotliwością jak najwięcej poziomych linii.

Do układu powinien zostać domontowany dodatkowy silnik elektryczny charakteryzujący się niskimi obrotami, na osi którego będzie zamontowany podobny jak wcześniej walec z przyklejonymi zwierciadélkami na jego powierzchni bocznej. Ten komponent będzie odpowiedzialny za odchylenie odchylonej wcześniej wiązki laserowej w kierunku pionowym.

3. Wszystkie komponenty urządzenia powinny zostać odpowiednio zsynchronizowane.

Należy odpowiednio zsynchronizować prędkości obrotowe obydwu silników elektrycznych według następujących relacji:

$N_{EH}$  - prędkość obrotowa silnika elektrycznego służącego do odchyłania wiązki w kierunku poziomym [obr./min.]

$N_{EV}$  - prędkość obrotowa silnika elektrycznego służącego do odchyłania wiązki w kierunku pionowym [obr./min.]

$M_{EH}$  - ilość zwierciadełek przyklejonych na powierzchni bocznej walca służącego do odchyłania wiązki w kierunku poziomym [j.]

$M_{EV}$  - ilość zwierciadełek przyklejonych na powierzchni bocznej walca służącego do odchyłania wiązki w kierunku pionowym [j.]

$W_V$  - rozdzielczość wyświetlanego obrazu w poziomie [j.]

$H_V$  - rozdzielczość wyświetlanego obrazu w pionie [j.]

$F_V$  - częstotliwość wyświetlania obrazu [Hz]

$$H_V * F_V = N_{EH} * M_{EH} / 60$$

$$F_V = N_{EV} * M_{EV}$$

4. Urządzenie powinno umożliwiać kontrolę nad wyświetlanym obrazem.

Należy zsynchronizować włączanie i wyłączanie lasera razem z prędkościami obrotowymi silników w taki sposób, żeby móc zapalać lub wygaszać odpowiedni piksel wyświetlany na ekranie. Aby zarządzać stanem piksela o współrzędnych  $x$  i  $y$  w danej klatce, należy kontrolować napięcie przyłożone na laser w przedziale czasowym  $t = [(1/F_V) * (y * W_V + x) / (W_V * H_V)], (1/F_V) * (y * W_V + x + 1) / (W_V * H_V)]$

5. Urządzenie powinno odpowiednio analizować sygnał podany na wejściu VGA oraz skalować odbierany obraz.

Układ cyfrowy powinien prawidłowo interpretować sygnał podawany na wejściu VGA, oraz skalować przesyłany obraz do rozdzielczości natywnej urządzenia za pomocą zwykłego algorytmu nearest-neighbour.

6. Urządzenie powinno być w stanie wyświetlić obraz podawany na wejściu VGA w formacie monochromatycznym.

Urządzenie powinno sterować stanem wszystkich pikseli na podstawie odbieranego sygnału na wejściu VGA. Dany piksel powinien się zapalić wtedy i tylko wtedy, gdy chociaż jedna składowa transmitowanego piksela będzie nie mniejsza niż 50% intensywności.

7. Urządzenie powinno być w stanie wyświetlić obraz podawany na wejściu VGA w formacie odcieni szarości.

Urządzenie powinno sterować stanem wszystkich pikseli na podstawie odbieranego sygnału na wejściu VGA. Dany piksel powinien mieć intensywność równą kolorowi RGB transmitowanego piksela rzutowanego na odcienie szarości.

8. Urządzenie powinno być w stanie wyświetlić obraz podawany na wejściu VGA w formacie RGB.

Urządzenie powinno sterować stanem wszystkich pikseli na podstawie odbieranego sygnału na wejściu VGA. Aby osiągnąć model RGB, należy do całego układu domontować lasery o kolorze niebieskim i zielonym, odpowiednio kontrolować napięcie na nich, oraz zwierciadła dichroiczne,

które będą scalać wiązki różnych kolorów w pojedynczą wiązkę. Napięcie dla poszczególnych laserów powinno być wprost proporcjonalne do wartości odpowiednich składowych transmitowanego piksela.

Implementacja interpolacji wielomianowej Lagrange'a:

```
1 using Polynomials
2 function interpolateWithLagrange(nodeSet)
3     interPoly = Poly([0])
4     nodesNum = size(nodeSet,1)
5     for i=1:nodesNum
6         tempPoly = Poly([nodeSet[i,2]])
7         for j=1:nodesNum
8             if i != j
9                 tempPoly = tempPoly * Poly([-nodeSet[j,1], 1.0])
10                tempPoly = tempPoly / (nodeSet[i,1]-nodeSet[j,1])
11            end
12        end
13        interPoly = interPoly + tempPoly
14    end
15    interPoly
16 end
```

Listing 1: Implementacja interpolacji wielomianowej Lagrange'a.

Powyżej przedstawiona funkcja przyjmuje tablicę dwu-wymiarową węzłów do interpolacji w postaci  $[[x_1, y_1], [x_2, y_2], \dots, [x_n, y_n]]$ , gdzie  $x_i$  i  $y_i$  są współrzędnymi  $i$ -tego węzła do interpolacji, a  $n$  to jest liczba wszystkich węzłów (w kodzie przechowywana jako `nodesNum`). W implementowaniu tej metody posłużyłem się wprost końcowym wzorem na wielomian interpolacyjny, który na sam koniec jest przechowywany i zwracany przy użyciu zmiennej `interPoly`. Zewnętrzna pętla `for` oznacza sumie występującej we wzorze, natomiast wewnętrzna odpowiada iloraz.

Zaimplementowaną funkcję testowałem przy użyciu kodu przedstawionego poniżej:

```
1 using Plots
2 using DataFrames
3 using Interpolations
4 setsNum = 4
5 plots = [plot(), plot(), plot(), plot(), plot(), plot(), plot(), plot(), plot(), plot(), plot()]
6 nodesPerSet = 11
7 nodesSets = zeros(setsNum, nodesPerSet, 2)
8 for setId=1:setsNum
9     for i=2:(size(nodesSets,2)-1)
10        nodesSets[setId,i,1] = 200.0*(i-1)/(size(nodesSets,2)-1)
11        -100.0#+(rand()-0.5)*(200.0/(size(nodesSets,2)-1))
12        nodesSets[setId,i,2] = rand()*200-100
13    end
14    nodesSets[setId,1,1] = -100.0
15    nodesSets[setId,1,2] = rand()*200-100
16    nodesSets[setId,size(nodesSets,2),1] = 100.0
17    nodesSets[setId,size(nodesSets,2),2] = rand()*200-100
18 end
19 for setId=1:setsNum
```

```

19     nodes = nodesSets[setId,:,:)
20     lagrangeInterPoly = interpolateWithLagrange(nodes)
21     if true
22         pr = -100:0.1:100
23         plot(title = string("Set ", setId, " (", nodesPerSet, " nodes)
: y(x)", ""), xlabel = "x", ylabel = "y")
24         plot!(pr, polyval(lagrangeInterPoly, pr), label = "Polynomial
from Lagrange's interpolation",
25               linewidth = 2, colour = RGBA(1,0,0,1), linestyle = :solid)
26         sc = scatter!(nodes[:, 1], nodes[:, 2], colour = RGBA(0,0,0,1)
, label = ["Nodes of interpolation"])
27         plots[setId] = sc
28     end
29 end
30 plot(plots[1],plots[2],plots[3],plots[4],#plots[5],plots[6],plots[7],
plots[8],plots[9],plots[10],
31      layout=(4,1), legend=true, normalize = true,
32      size = (1440,2560), xlim = [-101, 101], ylim = [-101, 101])

```

Listing 2: Testowanie funkcji interpolującej węzły za pomocą metody Lagrange'a.

Kod źródłowy z listingu 2. składa się z kilku ważniejszych etapów. Pierwszym z nich jest wygenerowanie 4 zestawów węzłów do interpolacji, po 11 węzłów na każdy zestaw. Ta ilość wynika z ustalonego przeze mnie przedziału na osi  $X = [-100, 100]$  (tak, aby dla  $x = 0$  występował węzeł). Dla potrzeb późniejszego zastosowania algorytmu interpolacji Newtona, węzły te są równoodległe od siebie. Na osi Y również przyjąłem taki sam przedział. Wygenerowane zestawy węzłów zostaną później użyte w innych metodach interpolacji.

Kolejnym etapem jest interpolowanie każdego z 4 zestawów węzłów metodą Lagrange'a oraz wygenerowanie wykresu, na którym będą zaznaczone węzły danego zestawu oraz wielomian interpolujący te węzły.

Ostatnim etapem jest wyświetlenie wszystkich wykresów w odpowiedni sposób.

Wynik uruchomienia powyższego skryptu przedstawia figure 1. Przedstawione są na nim cztery wykresy – każdy jeden został utworzony na podstawie odpowiedniego zestawu węzłów.

Jak można dostrzec, napisana przeze mnie funkcja prawidłowo interpoluje wszystkie węzły, ponieważ wykres wielomianu interpolującego zawiera w sobie każdy z tych węzłów.

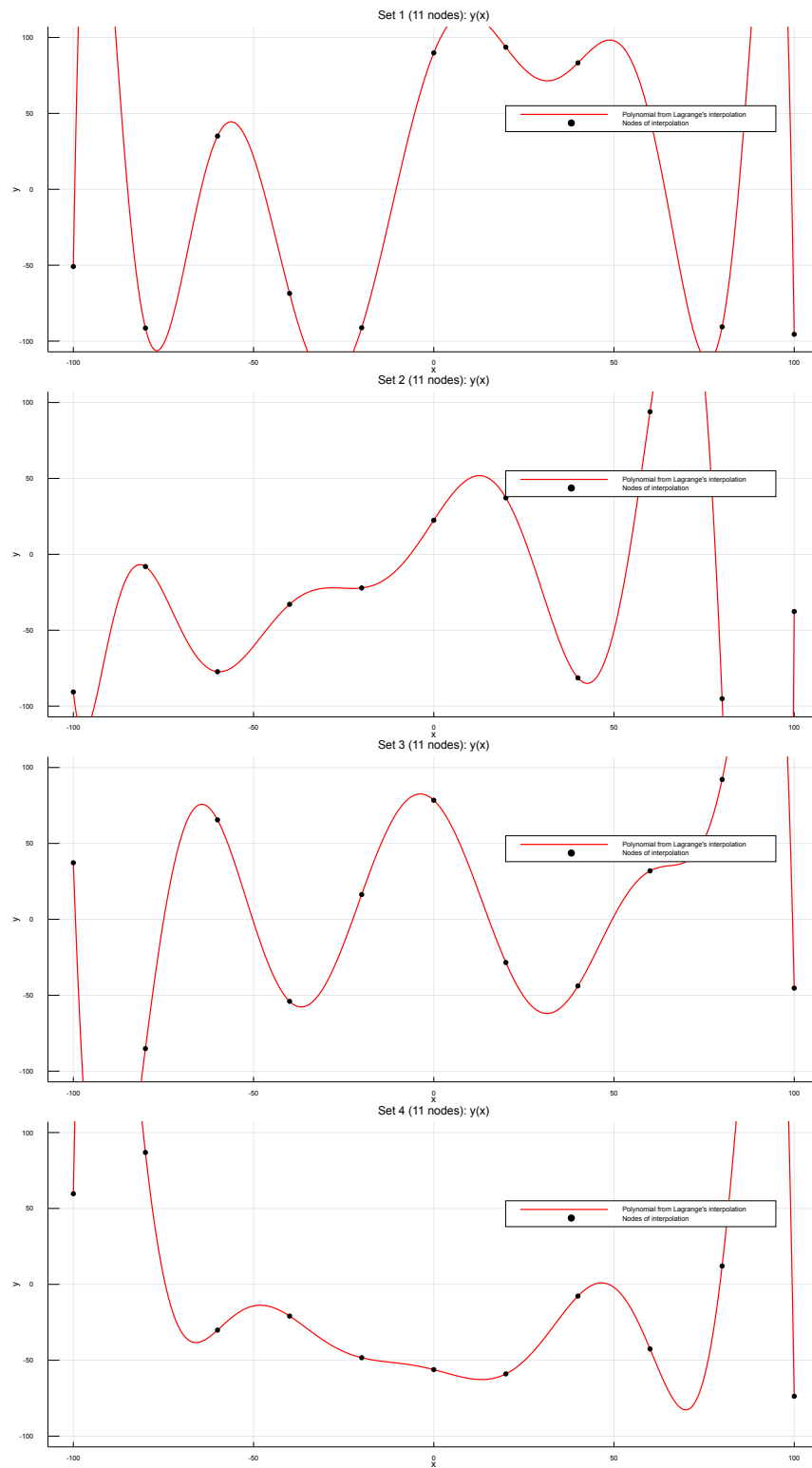


Figure 1: Wynik uruchomienia skryptu z listingu 2.



## 0.5 Własna implementacja interpolacji wielomianowej Newton'a.

Implementacja interpolacji wielomianowej Newton'a:

```
1 function myFactorial(n)
2     f = Int128(1)
3     for i=2:n
4         f = f * i
5     end
6     f
7 end
8
9 function interpolateWithNewton(nodeSet)
10     nodesNum = size(nodeSet,1)
11     tempY = zeros(nodesNum, nodesNum)
12     for i=1:nodesNum
13         tempY[i,1] = nodeSet[i,2]
14     end
15     for i=2:nodesNum
16         for j=1:nodesNum-1
17             tempY[j,i] = tempY[j+1,i-1] - tempY[j,i-1]
18         end
19     end
20     #=for i=1:nodesNum
21         print(nodeSet[i,2], "\t")
22         for j=1:nodesNum-1
23             print(tempY[i,j], "\t")
24         end
25         println()
26     end=#
27     interPoly = Poly([tempY[1,1]])
28     u = Poly([-nodeSet[1,1], 1.0]) / (nodeSet[2,1]-nodeSet[1,1])
29     tempU = u
30     for i=2:nodesNum
31         interPoly = interPoly + (tempU * tempY[1,i]) / myFactorial(i
32         -1)
33         tempU = tempU * (u + 1 - i)
34     end
35     interPoly
36 end
```

Listing 3: Implementacja interpolacji wielomianowej Newton'a.

Powyżej przedstawiona funkcja przyjmuje tablicę dwu-wymiarową węzłów do interpolacji w postaci  $[[x_1, y_1], [x_2, y_2], \dots, [x_n, y_n]]$ , gdzie  $x_i$  i  $y_i$  są współrzędnymi  $i$ -tego węzła do interpolacji, a  $n$  to jest liczba wszystkich węzłów (w kodzie przechowywana jako `nodesNum`). W implementowaniu tej metody również posłużyłem się wprost końcowym wzorem na wielomian interpolacyjny. Dodatkowo wprowadziłem pewne optymalizacje, żeby nie wyliczać kilka razy tych samych danych. Na samym początku obliczane są współczynniki dla każdego z wyrazów sumy wielomianu interpolującego, które będą przechowywane w pierwszym wierszu macierzy, która posłużyła mi do tych obliczeń. Na koniec krokowo tworzę wzór wielomianu interpolującego, który na sam koniec jest przechowywany i zwracany przy

użyciu zmiennej `interPoly`.

Należy zaznaczyć, że funkcja zakłada równoodległość dostarczonych węzłów do interpolacji.

Zaimplementowaną funkcję testowałem przy użyciu kodu przedstawionego poniżej:

```
1 using Plots
2 using DataFrames
3 using Interpolations
4 setsNum = 4
5 plots = [plot(),plot(),plot(),plot(),plot(),plot(),plot(),plot(),plot
6           (),plot()]
7 nodesPerSet = 11
8 for setId=1:setsNum
9     nodes = nodesSets[setId,:,:]
10    newtonInterPoly = interpolateWithNewton(nodes)
11    if true
12        pr = -100:0.1:100
13        plot(title = string("Set ", setId, " (", nodesPerSet, " nodes)
14              : y(x)", ""), xlabel = "x", ylabel = "y")
15        plot!(pr, polyval(newtonInterPoly, pr), label = "Polynomial
16              from Newton's interpolation",
17              linewidth = 2, colour = RGBA(0,1,0,1), linestyle = :solid)
18        sc = scatter!(nodes[:, 1], nodes[:, 2], colour = RGBA(0,0,0,1)
19              , label = ["Nodes of interpolation"])
20        plots[setId] = sc
21    end
22 end
23 plot(plots[1],plots[2],plots[3],plots[4],#plots[5],plots[6],plots[7],
24      plots[8],plots[9],plots[10],
25      layout=(4,1), legend=true, normalize = true,
26      size = (1440,2560), xlim = [-101, 101], ylim = [-101, 101])
```

Listing 4: Testowanie funkcji interpolującej węzły za pomocą metody Newton'a.

Kod źródłowy z listingu 4. składa się z dwóch ważniejszych etapów. Pierwszym z nich jest interpolowanie każdego z 4 zestawów węzłów metodą Newton'a oraz wygenerowanie wykresu, na którym będą zaznaczone węzły danego zestawu oraz wielomian interpolujący te węzły.

Drugim etapem jest wyświetlenie wszystkich wykresów w odpowiedni sposób.

Wynik uruchomienia powyższego skryptu przedstawia figure 2. Przedstawione są na nim cztery wykresy – każdy jeden został utworzony na podstawie odpowiedniego zestawu węzłów.

Jak można dostrzec, napisana przeze mnie funkcja prawidłowo interpoluje wszystkie węzły, ponieważ wykres wielomianu interpolującego zawiera w sobie każdy z tych węzłów. Ponadto porównując te wykresy z odpowiednimi wykresami z figure 1. (odpowiednie zestawy węzłów) można zauważyć, że wykresy są identyczne → obie metody musiały zwrócić ten sam wielomian.

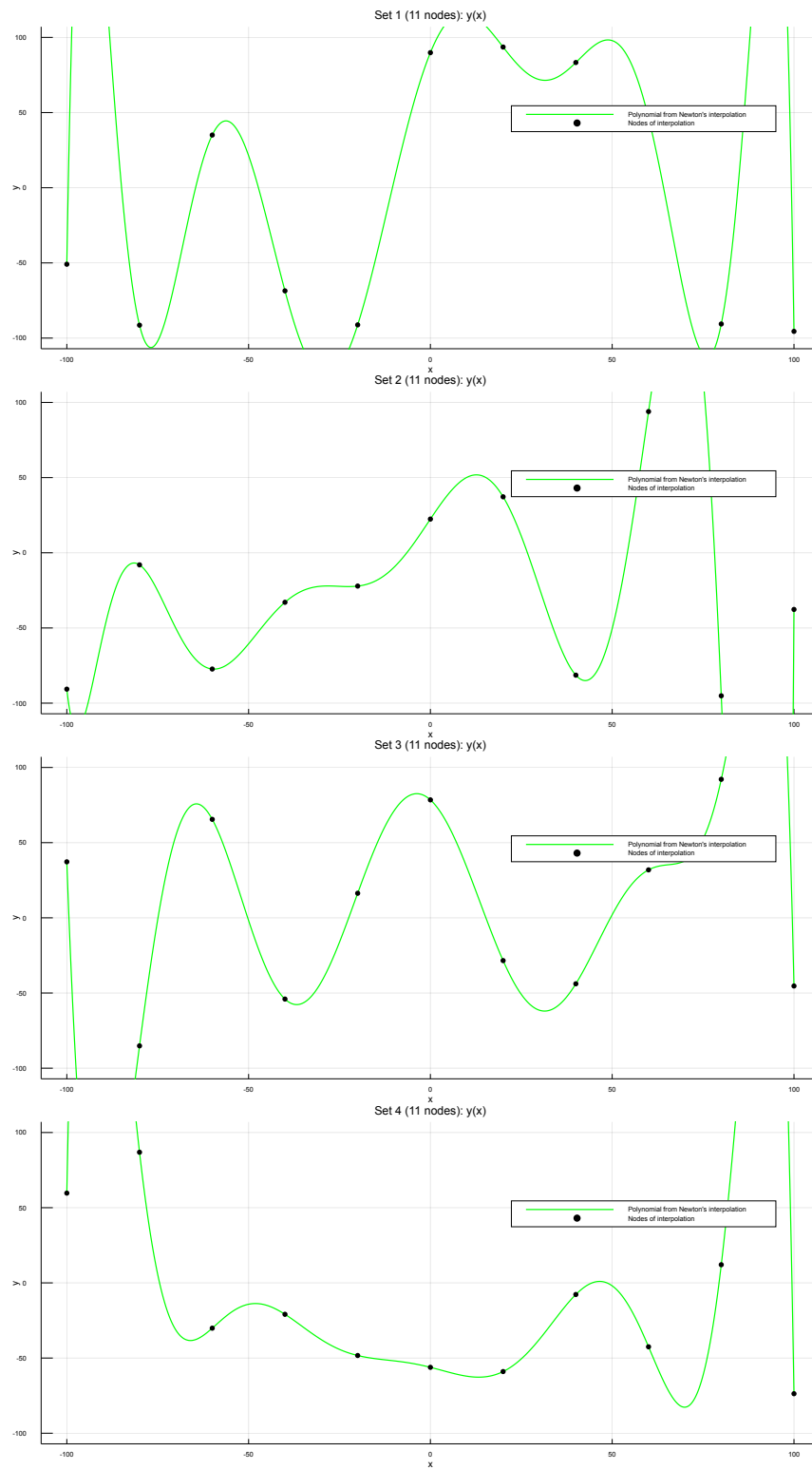


Figure 2: Wynik uruchomienia skryptu z listingu 4.

## 0.6 Wykorzystanie interpolacji wielomianowej z pakietu Polynomials oraz podsumowanie dotychczasowych metod na jednym wykresie.

```
1 using Plots
2 using DataFrames
3 using Interpolations
4 setsNum = 4
5 plots = [plot(),plot(),plot(),plot(),plot(),plot(),plot(),plot(),plot
6           (),plot()]
7 nodesPerSet = 11
8 for setId=1:setsNum
9     nodes = nodesSets[setId,:,:]
10    lagrangeInterPoly = interpolateWithLagrange(nodes)
11    newtonInterPoly = interpolateWithNewton(nodes)
12    juliaInterPoly = polyfit(nodes[:,1], nodes[:,2], nodesPerSet-1)
13    if true
14        pr = -100:0.1:100
15        plot(title = string("Set ", setId, " (", nodesPerSet, " nodes)
16              : y(x)", ""), xlabel = "x", ylabel = "y")
17        plot!(pr, polyval(lagrangeInterPoly, pr), label = "Polynomial
18                  from Lagrange's interpolation",
19               linewidth = 2, colour = RGBA(1,0,0,1), linestyle = :dash)
20        plot!(pr, polyval(newtonInterPoly, pr), label = "Polynomial
21                  from Newton's interpolation",
22               linewidth = 2, colour = RGBA(0,1,0,1), linestyle = :
23                  dashdot)
24        plot!(pr, polyval(juliaInterPoly, pr), label = "Polynomial
25                  from Julia's polynomial interpolation",
26               linewidth = 2, colour = RGBA(0,0,1,1), linestyle = :
27                  dashdotdot)
28        sc = scatter!(nodes[:, 1], nodes[:, 2], colour = RGBA(0,0,0,1)
29                      , label = ["Nodes of interpolation"])
30        plots[setId] = sc
31    end
32 end
33 plot(plots[1],plots[2],plots[3],plots[4],#plots[5],plots[6],plots[7],
34      plots[8],plots[9],plots[10],
35      layout=(4,1), legend=true, normalize = true,
36      size = (1440,2560), xlim = [-101, 101], ylim = [-101, 101])
```

Listing 5: Wykorzystanie funkcji polyfit z pakietu Polynomials w celu znalezienia wielomianu interpolującego wygenerowane wcześniej węzły oraz narysowanie wykresów wielomianów interpolujących osiągniętych za pomocą trzech różnych metod/funkcji.

Kod źródłowy z listingu 5. składa się z dwóch ważniejszych etapów. Pierwszym z nich jest interpolowanie każdego z 4 zestawów węzłów za pomocą metody Lagrange'a, następnie Newton'a, i na koniec za pomocą funkcji polyfit z pakietu Polynomials, jak też chciano w poleceniu.

Drugim etapem jest wyświetlenie wszystkich wykresów

Wynik uruchomienia powyższego skryptu przedstawia figure 3. Przedstawione są na nim cztery wykresy w odpowiedni sposób: jeden wykres = jeden zestaw węzłów = 3 metody interpolacji.

Na przedstawionych wykresach najlepiej widać, jak wszystkie trzy użyte implementacje (w tym dwie własne) zwracają ten sam wynik, zarówno w postaci wykresu wielomianu interpolującego, jak i jego wzoru. Dzieje się tak dlatego, ponieważ wielomiany interpolujące węzły muszą je zawierać, a więc dla skończonej ilości ściśle określonych argumentów muszą przyjmować ściśle określone wartości, przy określeniu jedynie jednego wspólnego wzoru dla całego przedziału. Można z tego wywnioskować, że wielomian interpolujący węzły nie zależy od metody interpolacji, ale tylko od rozmieszczenia węzłów do interpolacji.

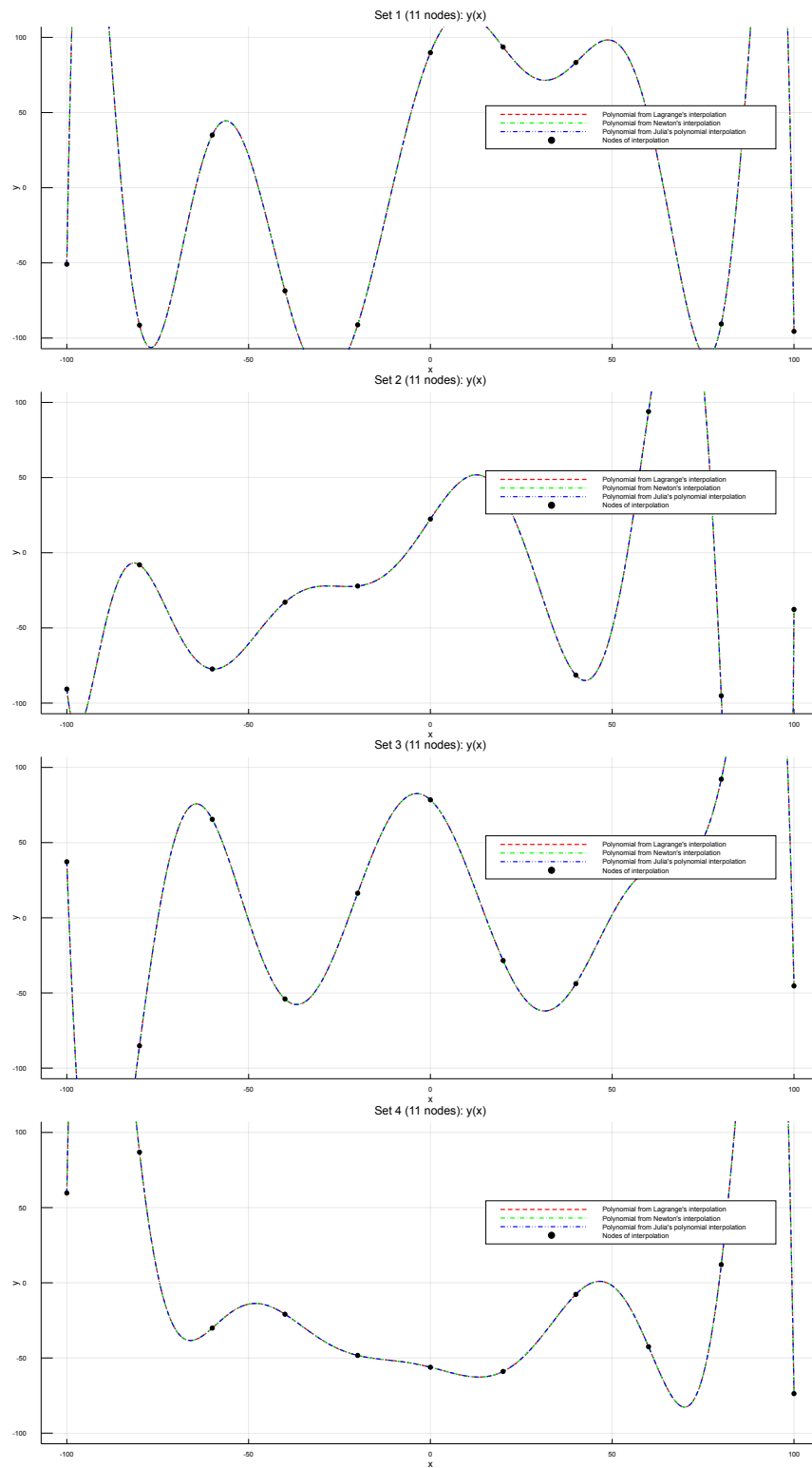


Figure 3: Wynik uruchomienia skryptu z listingu 5.

## 0.7 Porównanie opisanych wcześniej metod interpolacji pod względem czasu interpolowania, interpolacja funkcjami sklejanymi a interpolacja wielomianami, demonstracja efektu Runge’go.

```
1 using Plots
2 using DataFrames
3 using Interpolations
4 setsNum = 10
5 plots = [plot(),plot(),plot(),plot(),plot(),plot(),plot(),plot(),plot
6           (),plot()]
7 timeResults = DataFrame(nodesNum = Int32[], interType = Int32[], time
8                           = Float64[])
9 for nodesPerSet=5:2:21
10     nodesSets = zeros(setsNum, nodesPerSet, 2)
11     for setId=1:setsNum
12         for i=2:(size(nodesSets,2)-1)
13             nodesSets[setId,i,1] = 200.0*(i-1)/(size(nodesSets,2)-1)
14             -100.0#+(rand()-0.5)*(200.0/(size(nodesSets,2)-1))
15             nodesSets[setId,i,2] = rand()*200-100
16         end
17         nodesSets[setId,1,1] = -100.0
18         nodesSets[setId,1,2] = rand()*200-100
19         nodesSets[setId,size(nodesSets,2),1] = 100.0
20         nodesSets[setId,size(nodesSets,2),2] = rand()*200-100
21     end
22     for setId=1:setsNum
23         nodes = nodesSets[setId,:,:]
24         readTime = @elapsed(lagrangeInterPoly =
25                             interpolateWithLagrange(nodes))
26         insert!.(eachcol(timeResults,false), 1, [nodesPerSet, 1,
27             readTime])
28         readTime = @elapsed(newtonInterPoly = interpolateWithNewton(
29             nodes))
30         insert!.(eachcol(timeResults,false), 1, [nodesPerSet, 2,
31             readTime])
32         readTime = @elapsed(juliaInterPoly = polyfit(nodes[:,1], nodes
33            [:,2], nodesPerSet-1))
34         insert!.(eachcol(timeResults,false), 1, [nodesPerSet, 3,
35             readTime])
36         linearInterPoly = LinearInterpolation(nodes[:,1], nodes[:,2])
37         nodesAsRange = -100.0:(200.0/(size(nodesSets,2)-1)):100.0
38         cubicInterPoly = CubicSplineInterpolation(nodesAsRange, nodes
39            [:,2])
40         if setId <= 4 && nodesPerSet == 1+setId*4
41             pr = -100:0.1:100
42             plot(title = string("Set ", setId, " (", nodesPerSet, "
43                 nodes): y(x)", ""), xlabel = "x", ylabel = "y")
44             plot!(pr, polyval(lagrangeInterPoly, pr), label = "
45                 Polynomial from Lagrange's interpolation",
46                 linewidth = 2, colour = RGBA(1,0,0,1), linestyle = :
47                 dash)
48             plot!(pr, polyval(newtonInterPoly, pr), label = "
49                 Polynomial from Newton's interpolation",
```

```

36         linewidth = 2, colour = RGBA(0,1,0,1), linestyle = :
dashdot)
37     plot!(pr, polyval(juliaInterPoly, pr), label = "Polynomial
from Julia's polynomial interpolation",
38         linewidth = 2, colour = RGBA(0,0,1,1), linestyle = :
dashdotdot)
39     linearPlot = [linearInterPoly(x) for x in pr]
40     plot!(pr, linearPlot, label = "Polynomial from Julia's
linear interpolation",
41         linewidth = 2, colour = RGBA(0.5,0,0,1), linestyle = :
dot)
42     cubicPlot = [cubicInterPoly(x) for x in pr]
43     plot!(pr, cubicPlot, label = "Polynomial from Julia's
cubic interpolation",
44         linewidth = 2, colour = RGBA(0,0.5,0,1), linestyle = :
dot)
45     sc = scatter!(nodes[:, 1], nodes[:, 2], colour = RGBA
(0,0,0,1), label = ["Nodes of interpolation"])
46     plots[setId] = sc
47     end
48 end
49 end
50 plot(plots[1], plots[2], plots[3], plots[4], #plots[5], plots[6], plots[7],
plots[8], plots[9], plots[10],
51     layout=(4,1), legend=true, normalize = true,
52     size = (1440,2560), xlim = [-101, 101], ylim = [-101, 101])

```

Listing 6: Wykorzystanie funkcji polyfit z pakietu Polynomials w celu znalezienia wielomianu interpolującego wygenerowane wcześniej węzły oraz narysowanie wykresów wielomianów interpolujących osiągniętych za pomocą trzech różnych metod/funkcji.

```

1 using Statistics
2 groupedResults = by(timeResults, [:nodesNum, :interType], timeMean = :
time => mean, timeStd = :time => std)
3 plot(groupedResults[:nodesNum],
4     groupedResults[:timeMean],
5     group = groupedResults[:interType],
6     colour = [:red :green :blue],
7     yerr = groupedResults[:timeStd])

```

Listing 7: Wyświetlanie wykresu prezentującego analizę czasu wyszukiwania wielomianu interpolującego węzły w zależności od ilości tych węzłów dla trzech różnych implementacji.

Wykonując kolejno skrypty z listingów 6. oraz 7. można otrzymać wykres przedstawiający przykładową analizę czasową wyszukiwania wielomianu interpolującego węzły w zależności od ich ilości oraz użytej implementacji. Przykładową analizę przedstawia wykres z figure 4.



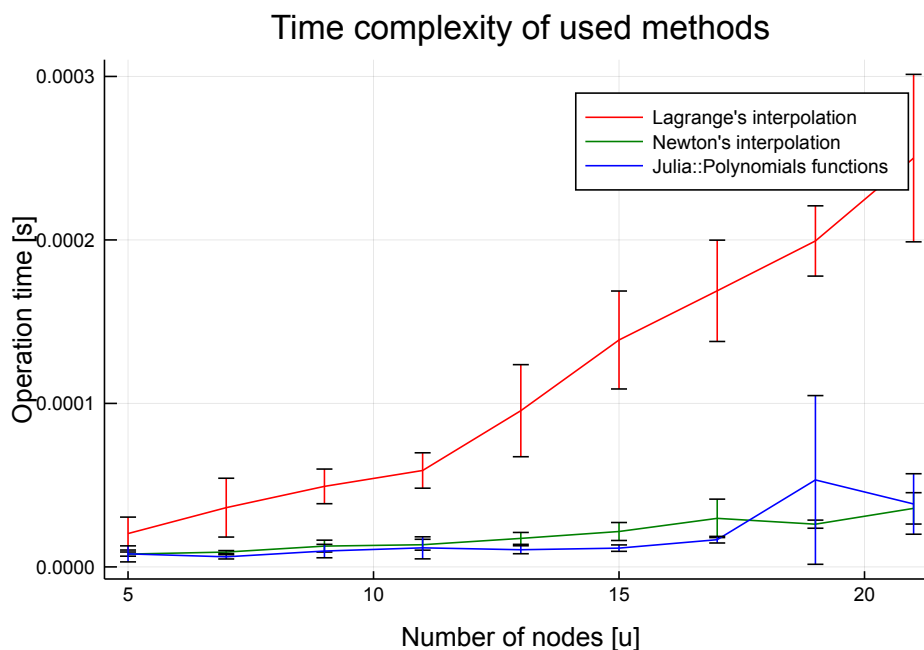


Figure 4: Wykres zależności czasu wyszukiwania wielomianu interpolującego węzły w zależności od ich ilości oraz użytej implementacji.

Na wykresie z figure 4. zostały przedstawione średnie czasy wykonania odpowiednich metod w zależności od ilości przekazanych węzłów, wraz ze słupkami błędów reprezentującymi odchylenie standardowe z 10 pomiarów na każdą sytuację.

Sensowny wykres bardzo trudno było uzyskać z powodu bardzo małych czasów wykonywania i minimalnych zakłóceń wykonywania na serwerach JuliaBox. Dlatego też zredukowałem maksymalną ilość węzłów do 21, gdyż dodatkowo dla większej ilości nie możliwe było obliczenie silni i zapisanie jej wyniku do odpowiedniej zmiennej typu Int.

Z przedstawionego wykresu można wywnioskować, że złożoność czasowa metody Newton'a, jak i funkcji polyfit z pakietu Polynomials jest niemalże jednostkowa, natomiast metody Lagrange'a – liniowa.

Uruchamiając jedynie skrypt z listingu 6. można otrzymać przykładowy zbiór wykresów, przedstawiony na figure 5.

Na wykresach tego zbioru zostały nie tylko pokazane wykresy wielomianów interpolujących węzły za pomocą różnych metod, ale i wykresy interpolacji węzłów funkcjami sklejanymi (w tym funkcjami liniowymi i trzeciego stopnia). Na podstawie tych wykresów można wysunąć poniższe wnioski:

- interpolacja funkcjami liniowymi jest to po prostu zwykłe łączenie punktów odcinkami (nie zalecane do przedstawiania danych na wykresie,

szczególnie w fizyce);

- b) interpolacja funkcjami trzeciego stopnia bardzo ładnie zachowuje się w bliskich otoczeniach węzłów (funkcja jest gładka, nie tak jak w przypadku interpolacji funkcjami liniowymi);
- c) funkcją sklejana interpolująca zadane węzły ma wykres leżący w pobliżu węzłów, natomiast w przypadku interpolacji wielomianem wykres znacząco odbiega od węzłów na krańcach przedziału, amplituda wykresu zaczyna drastycznie rosnąć (opisany później efekt Runge'go).

Na kolejnych wykresach z tego samego zbioru można dostrzec pewną wadę interpolacji wielomianami – wraz ze wzrostem ilości węzłów podlegających interpolacji wartości wielomianu interpolującego węzły na krańcach przedziału zaczynają mocno oscylować z szybko rosnącą amplitudą. Ta wada jest nazywana efektem Runge'go.

Efekt Runge'go jest to pogorszenie jakości interpolacji wielomianowej, pomimo zwiększenia liczby interpolowanych węzłów (naprzeciw rozsądkowi). To zjawisko jest spowodowane głównie interpolowaniem wielomianami wysokich stopni przy nałożonym warunku równoodległości węzłów.

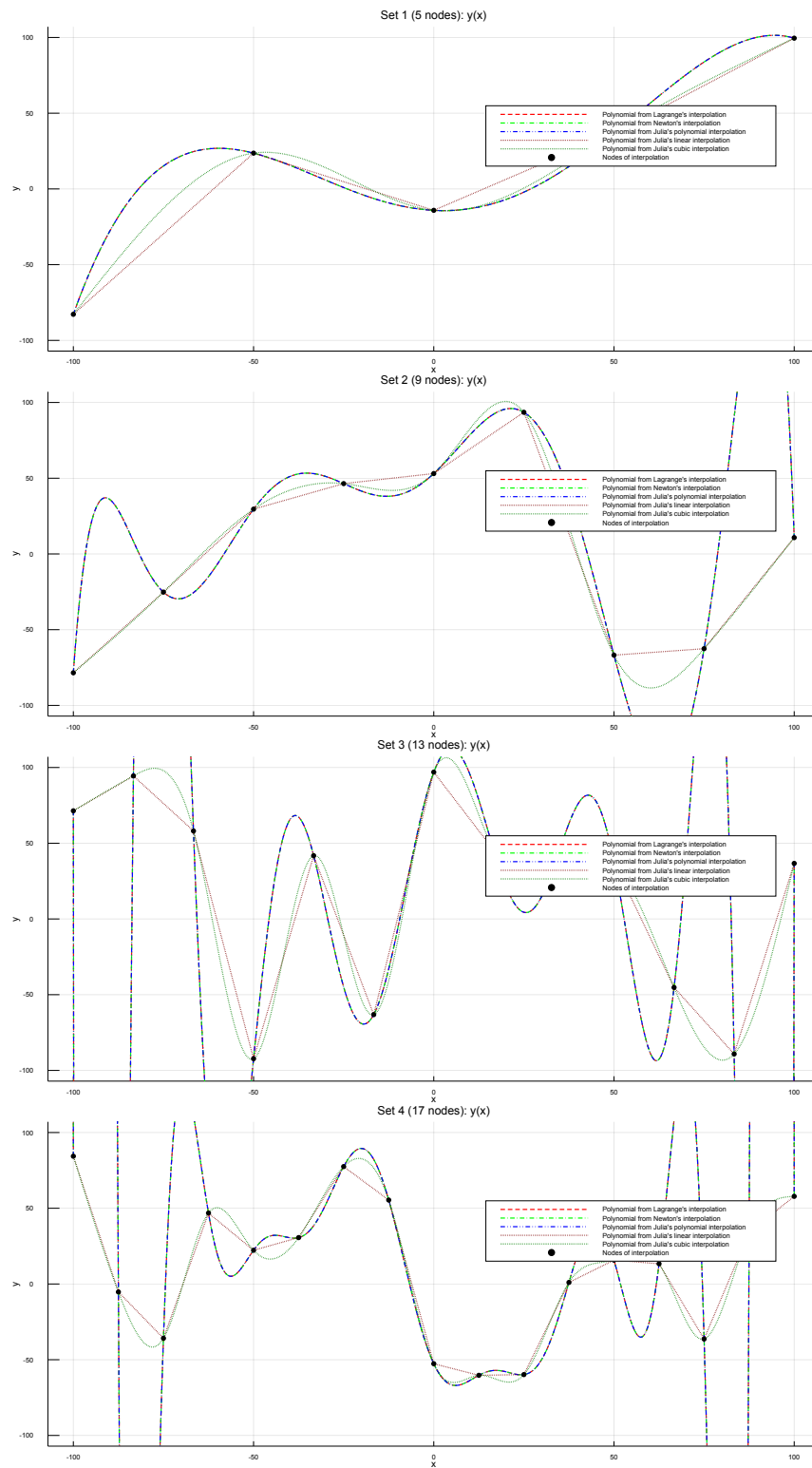
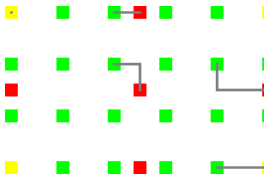
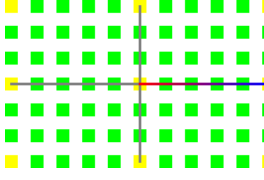
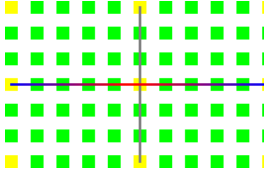


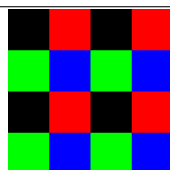
Figure 5: Wynik uruchomienia skryptu z listingu 6.

## 0.8 Algorytmy interpolacji stosowane w grafice komputerowej (na przykładzie powiększania obrazu w programie GIMP).

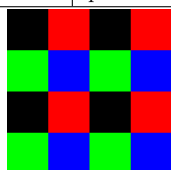
W programie GIMP można się spotkać z następującymi metodami interpolacji podczas skalowania warstwy lub całego obrazu: linear, cubic oraz brak interpolacji - none. Teraz postaram się opisać każdą z tych metod w prosty sposób.

Mając bazowy obraz o wymiarach 16x16, taki jak przedstawiony na ilustracji 1., metoda interpolacji lub brak w przypadku powiększenia obrazu 4-krotnie spowoduje poniżej opisany efekt:

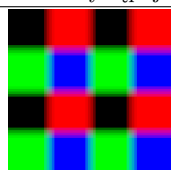
<p>a) none:</p> 	<p>Na przykładzie skalowania obrazka 3x3 px (piksele zostały zaznaczone „obecnością czerwonego koloru”) do rozmiaru 6x4 px (piksele zostały zaznaczone „obecnością zielonego koloru”). Nakładając i odpowiednio rozciągając obrazek wyjściowy na wejściowy, otrzymamy odpowiednie wzajemne rozmieszczenie pikseli. Każdy piksel z obrazka wyjściowego osiągnie kolor piksela obrazka wejściowego, który leży najbliżej niego (przykładowe zostały zaznaczone szarymi liniami). Ilustracja 2.</p>
<p>a) linear:</p> 	<p>Na przykładzie skalowania obrazka 3x3 px (piksele zostały zaznaczone „obecnością czerwonego koloru”) do rozmiaru 11x7 px (piksele zostały zaznaczone „obecnością zielonego koloru”). Nakładając i odpowiednio rozciągając obrazek wyjściowy na wejściowy, otrzymamy odpowiednie wzajemne rozmieszczenie pikseli. Łączymy sąsiednie piksele obrazka wejściowego abstrakcyjnymi liniami, które są liniowym gradientem przechodzącym od koloru piksela na jednym końcu do koloru piksela z drugiego końca. Takie postępowanie można przyrównać do interpolacji funkcją sklejaną (liniowymi), gdzie węzłami są piksele obrazka wejściowego. W taki sposób otrzymamy siatkę funkcji interpolujących (poziomych i pionowych). (przykładowe abstrakcyjne linie zaznaczone szarymi liniami, w tym na jednej został pokazany gradient od czerwonego do niebieskiego). Ilustracja 3.</p>
<p>a) cubic:</p> 	<p>Przykład został przygotowany w sposób analogiczny do przykładu dla interpolacji liniowej. Różnica w tym sposobie interpolacji polega jedynie na zastąpieniu interpolacji funkcji sklejaney (liniowymi) na interpolację funkcją sklejaną (wielomianami trzeciego stopnia). W ten sposób można otrzymać siatkę funkcji interpolujących (poziomych i pionowych), jedna taka siatka na jedną składową koloru. Piksel obrazka wyjściowego skorzysta z wartości składowych z siatek w punktach, gdzie on występuje. Ilustracja 4.</p>



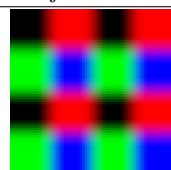
Ilustracja 1. base



Ilustracja 2. none



Ilustracja 3. linear



Ilustracja 4. cubic