Metody Obliczeniowe w Nauce i Technice
Laboratorium
Sprawozdanie z ćwiczenia

Marcin Maleńczuk

AGH

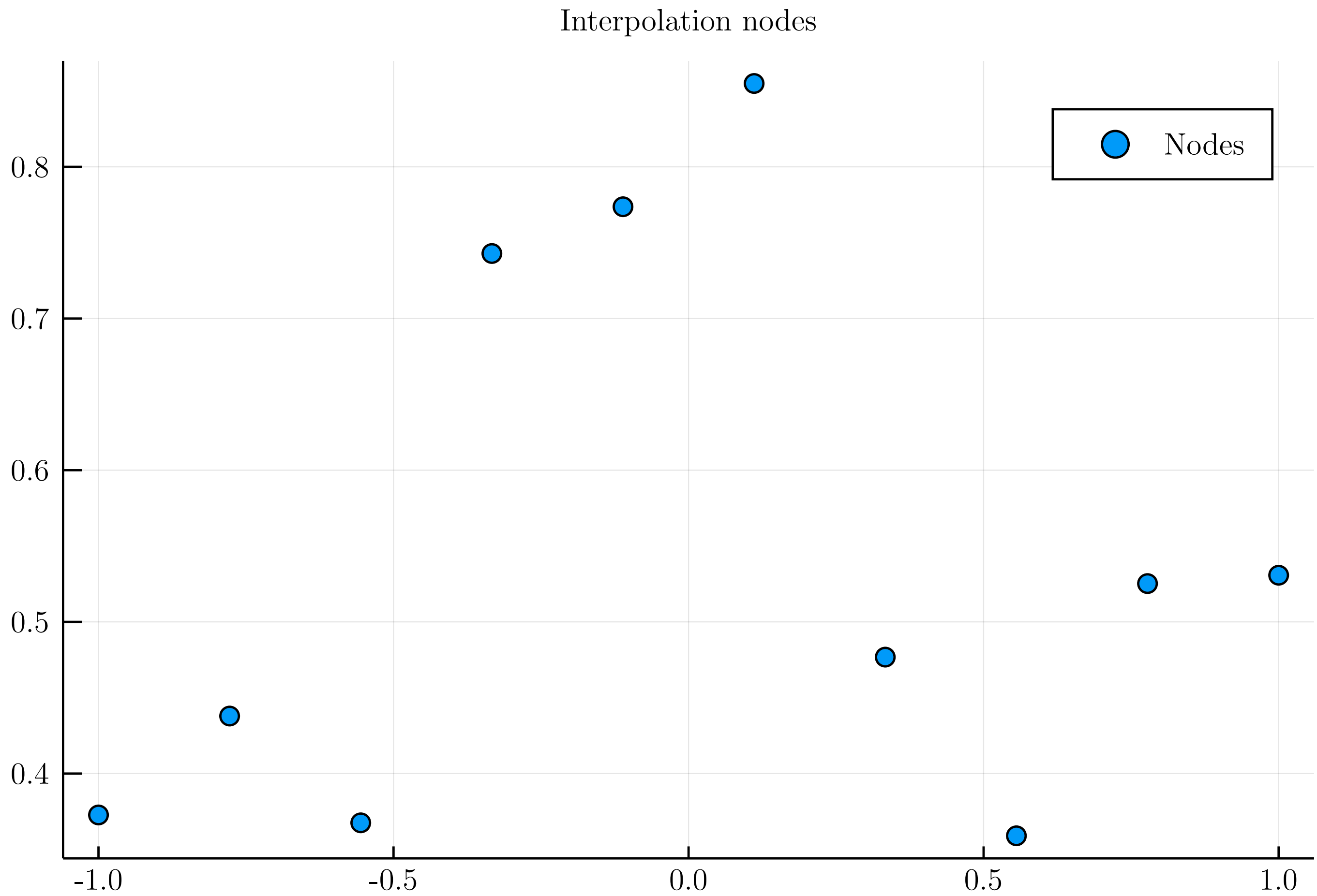
Akademia Górniczo-Hutnicza
im. Stanisława Staszica
w Krakowie

1 Dane

```
X = LinRange(-1, 1, 10)
Y = [rand() for x in X]
xsf = LinRange(-1, 1, 1000)
xs = LinRange(1, length(X), 1000)

scatter(X, Y, label="Nodes", title="Interpolation nodes")
```

Listing 1: Węzły Interpolacji

Listing 1 generuje 10 węzłów interpolacji



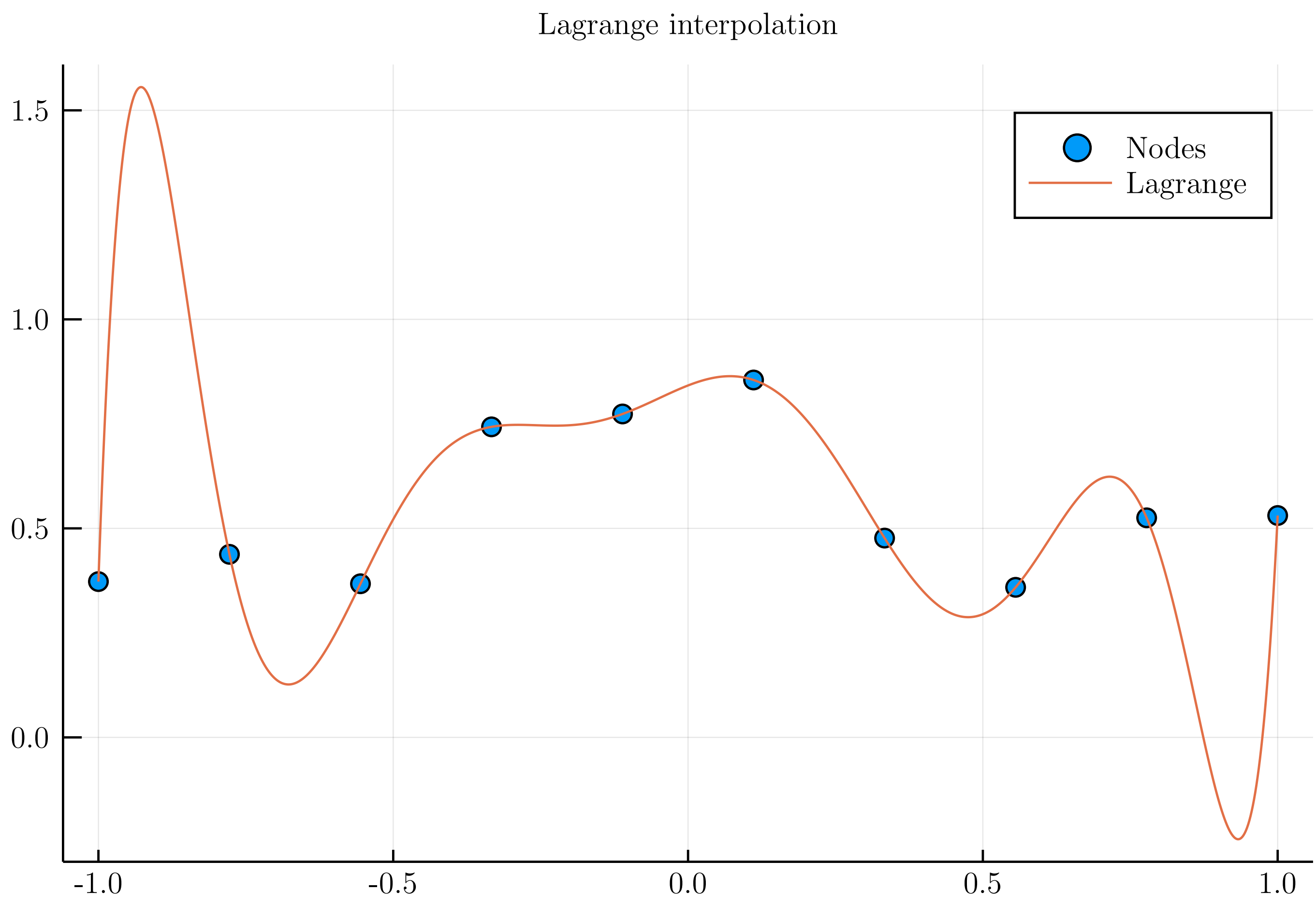
Rysunek 1: Węzły Interpolacji

2 Lagrange

```
function lagrange(X::AbstractArray, Y::AbstractArray, x::Number)
  k = length(X)
  return sum([reduce(*, [(x - X[m])/(X[j] - X[m]) for m in 1:k if m != j]) * Y[j] for j in 1:k])
end

function lagrange(X::AbstractArray, Y::AbstractArray, xx::AbstractArray)
  return [lagrange(X, Y, x) for x in xx]
end

LY = lagrange(X, Y, xsf)
scatter(X, Y, label="Nodes", title = "Lagrange interpolation")
plot!(xsf, LY, label = "Lagrange")
```

Listing 2: Interpolacja Lagrange'a

Listing 2 wykonuję na węzłach wygenerowanych z Listing 1 interpolację wielomianową Lagrange'a



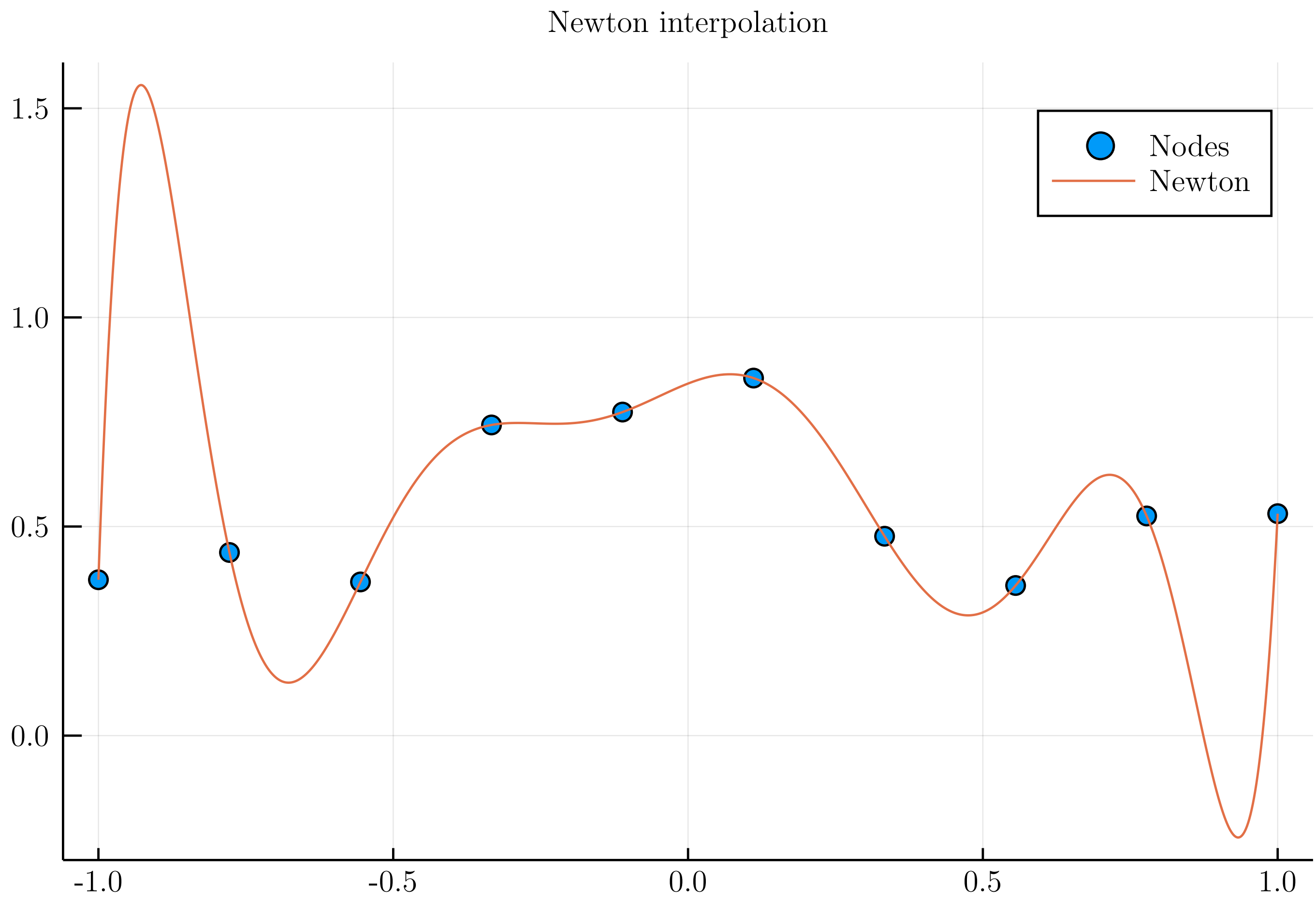
Rysunek 2: Interpolacja Lagrange'a

3 Newton

```
function divdif(X::AbstractArray, Y::AbstractArray)
  n = length(X)
  d = convert(Array{Float64,1}, deepcopy(Y))
  for i=2:n
    for j=1:i-1
      d[i] = (d[j] - d[i])/(X[j] - X[i])
    end
  end
  return d
end

function newtonform(X::AbstractArray, d::AbstractArray, x::Number)
  n = length(d)
  result = d[n]
  for i=n-1:-1:1
    result = result * (x - X[i]) + d[i]
  end
  return result
end

function newton(X::AbstractArray, Y::AbstractArray, x::Number)
  divided = divdif(X, Y)
  result = newtonform(X, divided, x)
  return result
end

function newton(X::AbstractArray, Y::AbstractArray, xx::AbstractArray)
  divided = divdif(X, Y)
  results = [newtonform(X, divided, x) for x in xx]
  return results
end

NY = newton(X, Y, xsf)
scatter(X, Y, label="Nodes", title = "Newton interpolation")
plot!(xsf, NY, label = "Newton")
```

Listing 3: Interpolacja Newton'a

Listing 3 wykonuję na węzłach wygenerowanych z Listing 1 interpolację wielomianową Newton'a



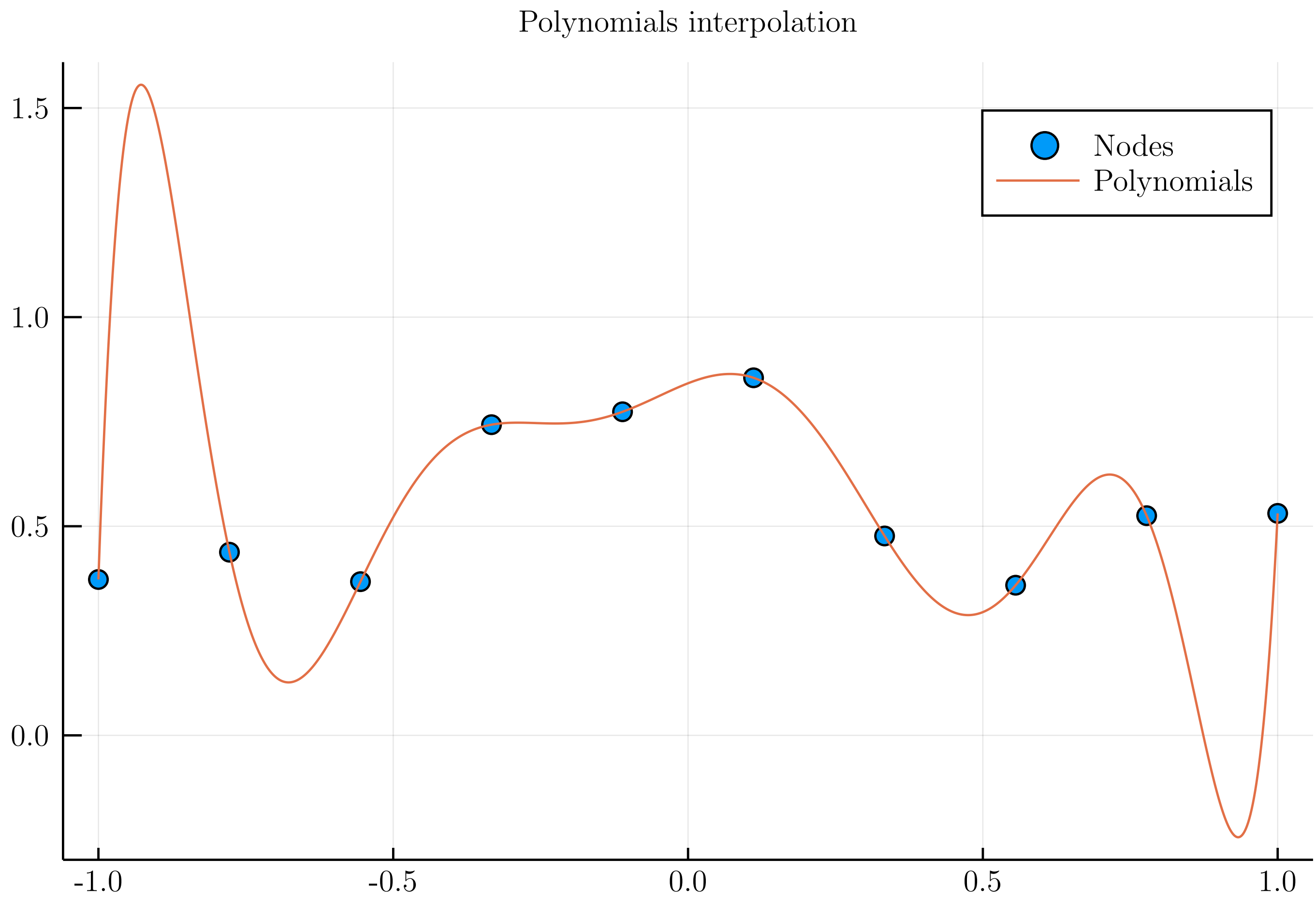
Rysunek 3: Interpolacja Newtona'a

4 Polynomials

```
poly = polyfit(X, Y, length(X) - 1)
PY = polyval(poly, xsf)
scatter(X, Y, label="Nodes", title = "Polynomials interpolation")
plot!(xsf, PY, label = "Polynomials")
```

Listing 4: Interpolacja pakietu Polynomials

Listing 4 wykonuję na węzłach wygenerowanych z Listing 1 interpolację wielomianową dostępną w pakiecie Polynomials



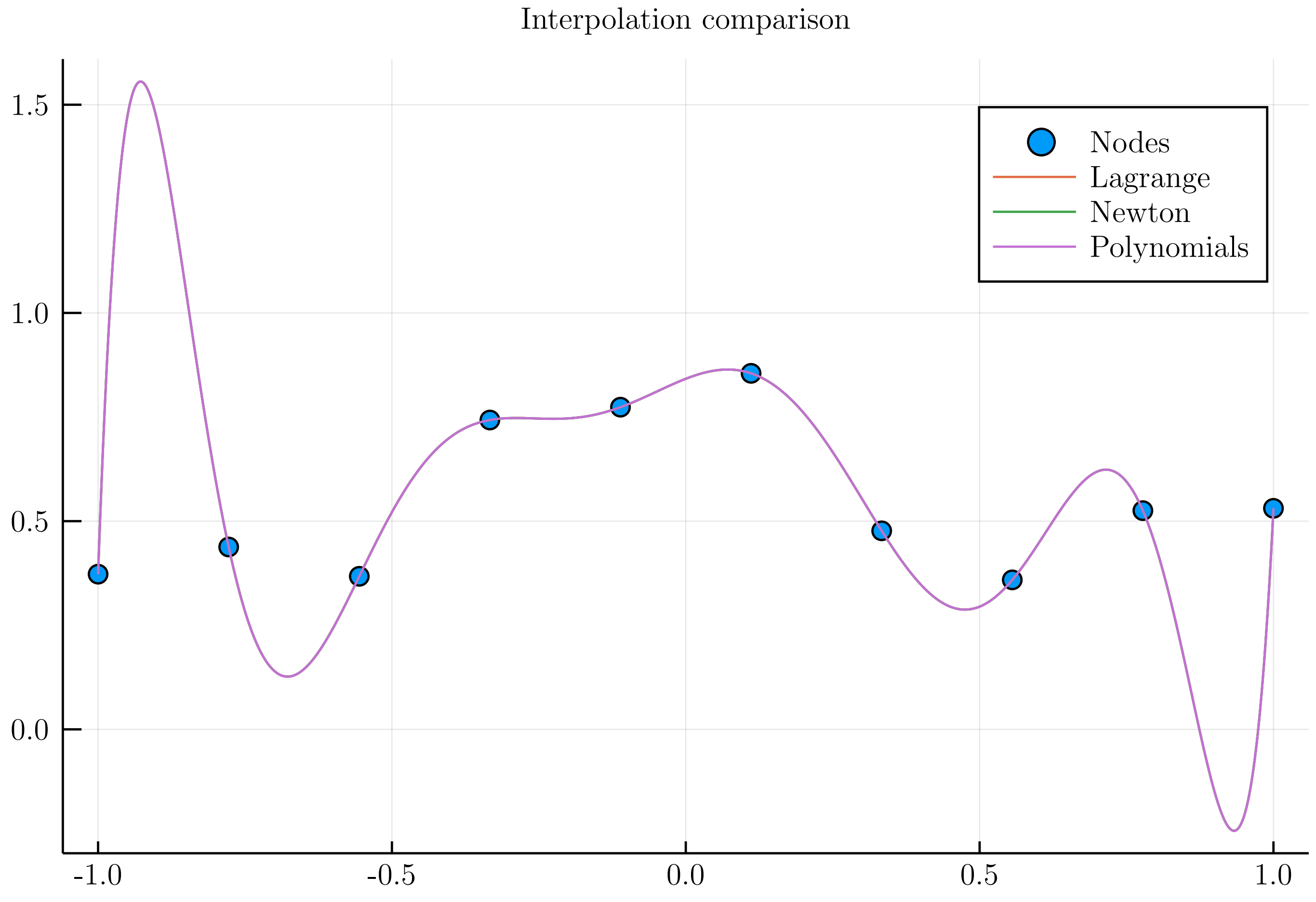
Rysunek 4: Interpolacja pakietu Polynomials

5 Porównanie Interpolacji

```
scatter(X, Y, label="Nodes", title = "Interpolation comparison")
plot!(xsf, LY, label = "Lagrange")
plot!(xsf, NY, label = "Newton")
plot!(xsf, PY, label = "Polynomials")
```

Listing 5: Porównanie Interpolacji

Listing 4 nanosi na Rysunek 5 poprzednie interpolację wielomianowe. Na Rysunek 5 widać że wszystkie interpolacje wilomianowe dają ten sam wielomian. Wynika to z twierdzenia o Jednoznaczności wielomianu interpolacyjnego, które mówi że dla danych węzłów interpolacji istnieje tylko jeden wielomian który je interpoluje.



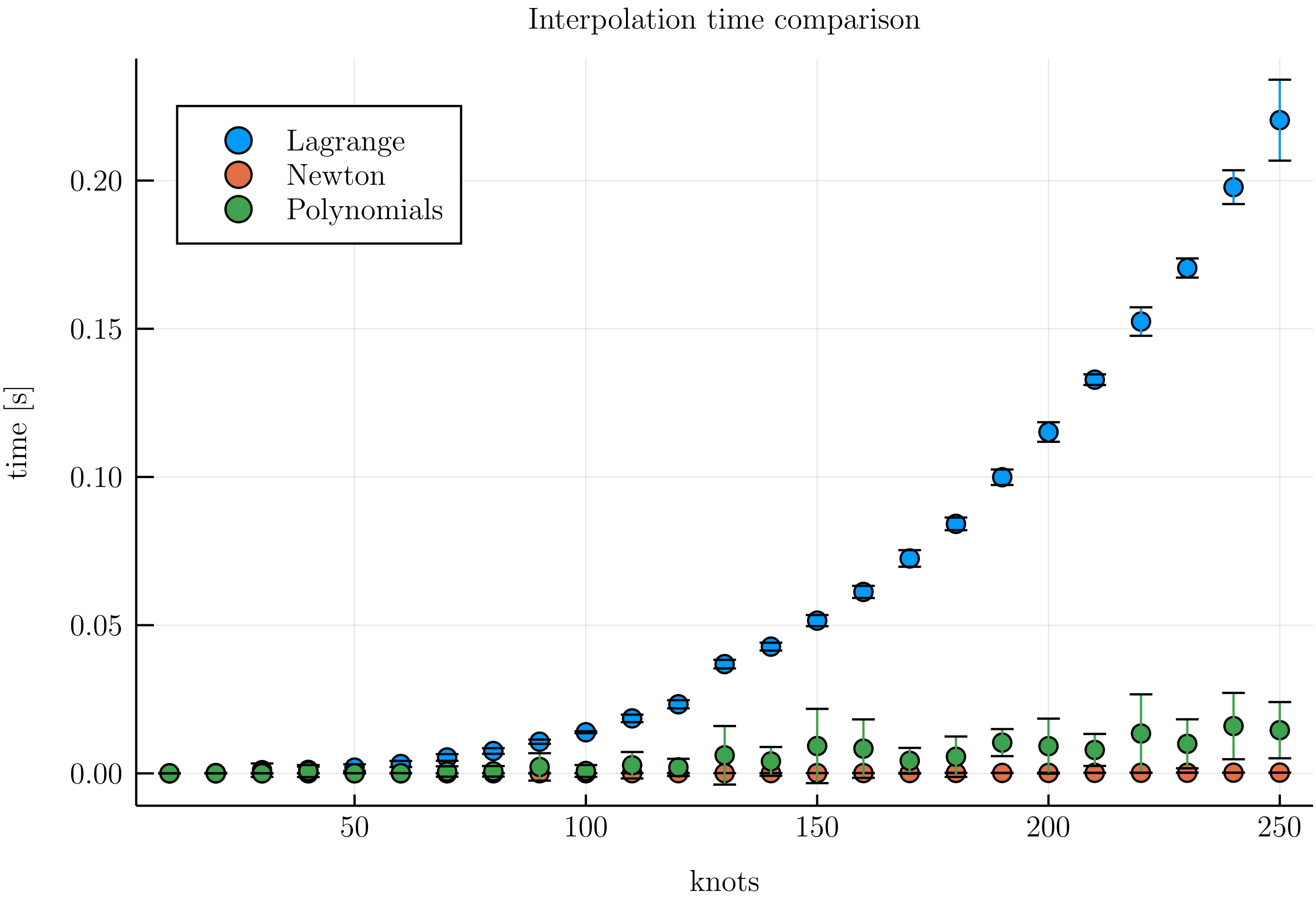
Rysunek 5: Porównanie Interpolacji

```
function times()
  df = DataFrame(Knots = Int64[], Fun = String[], Time = Float64[])
  for l in 10:10:250
    for i in 1:10
      x = LinRange(-1, 1, l)
      y = [rand() for _ in x]
      push!(df, [l, "Polynomials", (@elapsed polyval(polyfit(x, y, length(x) - 1), x))])
      push!(df, [l, "Newton", (@elapsed newton(x, y, x))])
      push!(df, [l, "Lagrange", (@elapsed lagrange(x, y, x))])
    end
  end
  return df
end
```

```
d = aggregate(times(), [:Knots, :Fun], [mean, std])
s = scatter(d[:Knots],
  d[:Time_mean],
  group = d[:Fun],
  yerr = d[:Time_std],
  legend = :topleft,
  title = "Interpolation time comparison",
  ylabel = "time [s]",
  xlabel = "knots")
```

Listing 6: Porównanie szybkości Interpolacji

Listing 6 porównuję czasy wykonania dla poszczególnych interpolacji zaczynając od 10 węzłów a kończąc na 250 co 10. Na Rysunek 6 widać, że czas interpolacji wielomianem Lagrange'a rośnie znacznie szybciej niż wilemoaniam Newton'a dlatego nie jest ona używana tylko do dowdów.



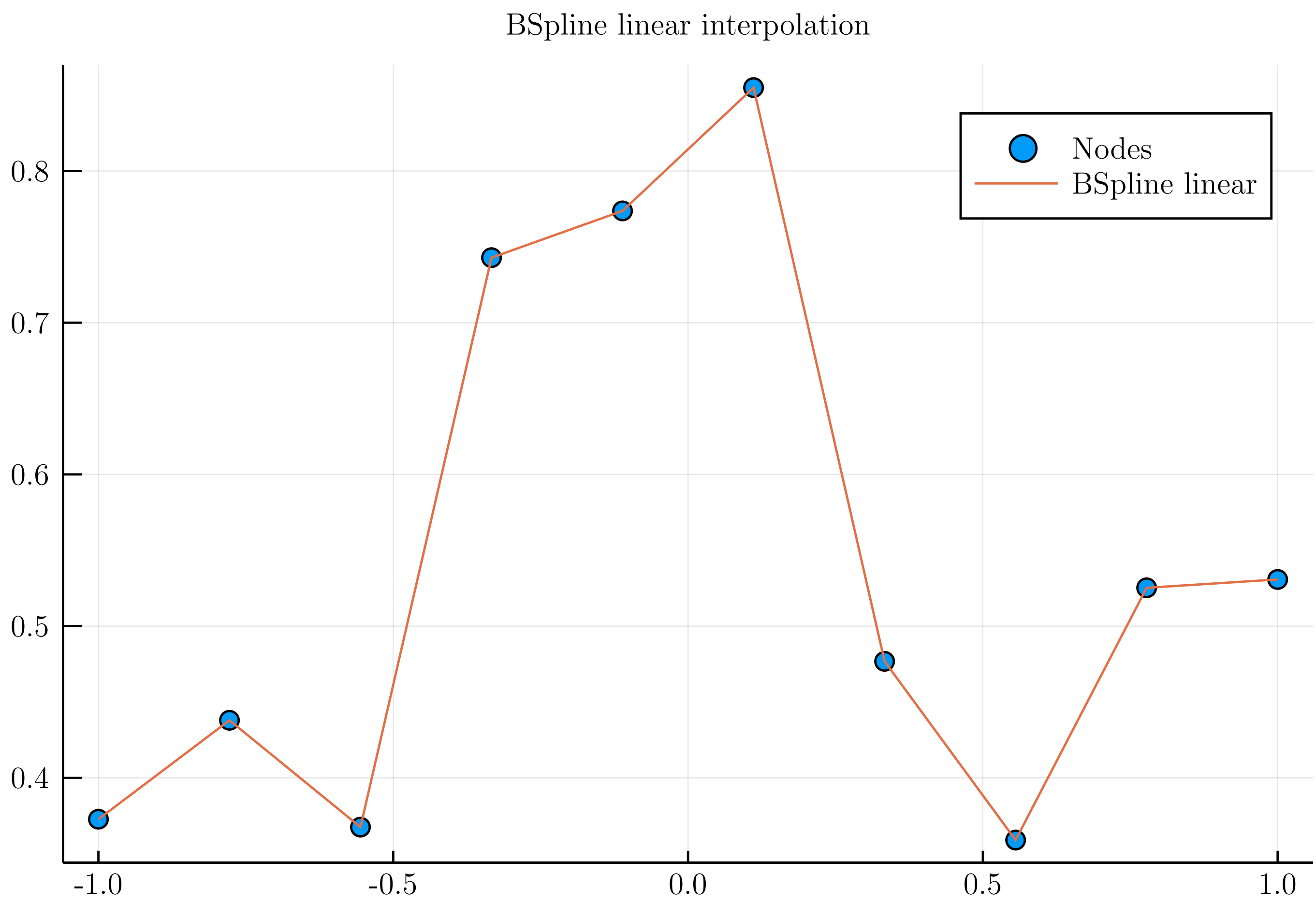
Rysunek 6: Porównanie szybkości Interpolacji

6 Funkcje Sklejane

```
litp = interpolate(Y, BSpline(Linear()))
L = litp(xs)
scatter(X, Y, label="Nodes", title = "BSpline linear interpolation")
plot!(xsf, L, label="BSpline linear")
```

Listing 7: Interpolacja funkcją sklejaną stopnia 1

Listing 7 wykonuję na węzłach wygenerowanych z Listing 1 interpolację funkcjami sklejanymi stopnia pierwszego

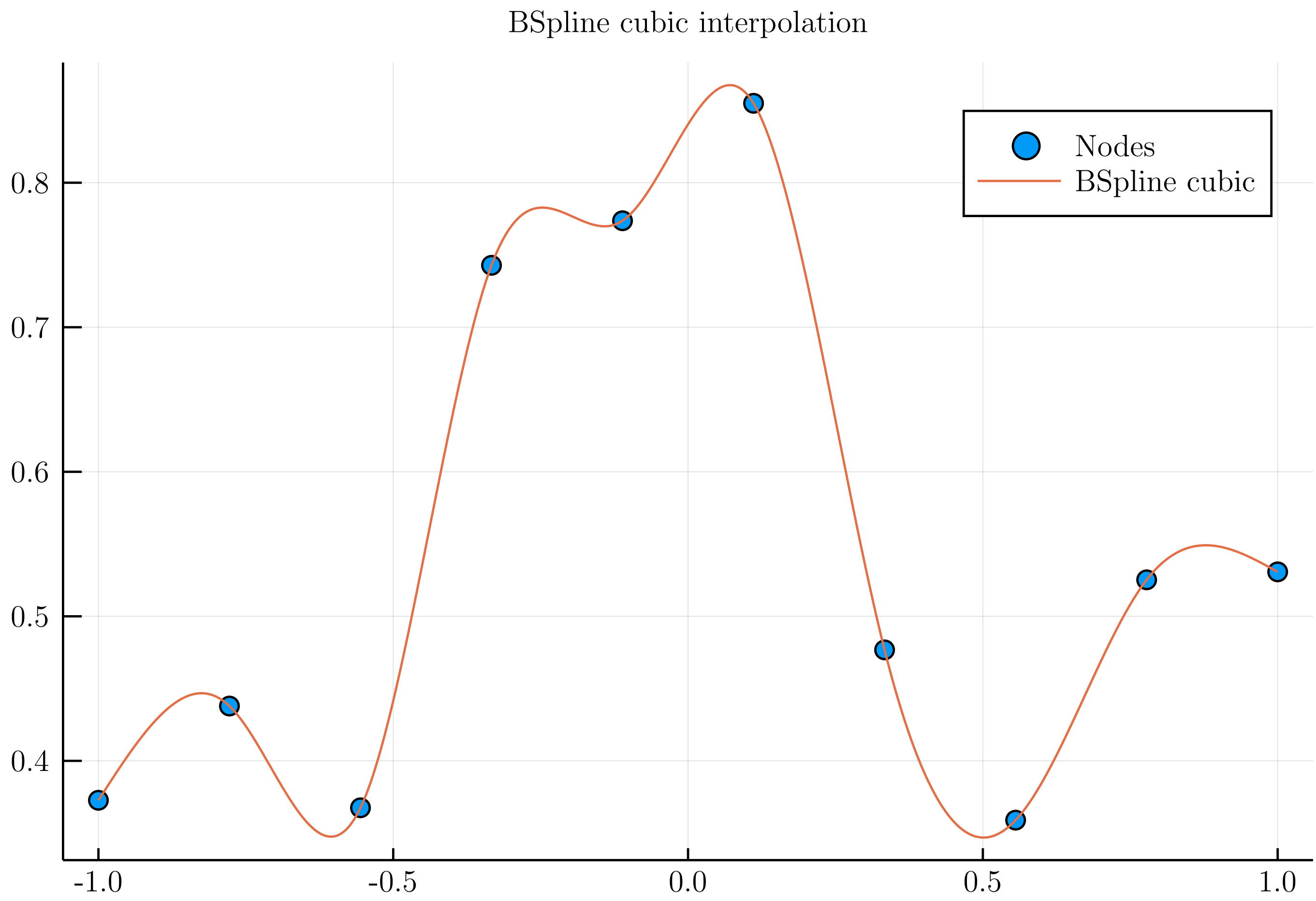


Rysunek 7: Interpolacja funkcją sklejaną stopnia 1

```
citp = interpolate(Y, BSpline(Cubic(Line(OnGrid()))))
C = citp(LinRange(1, length(X), 1000))
scatter(X,Y, label="Nodes", title = "BSpline cubic interpolation")
plot!(xsf,C, label="BSpline cubic")
```

Listing 8: Interpolacja funkcją sklejaną stopnia 2

Listing 8 wykonuję na węzłach wygenerowanych z Listing 1 interpolację funkcjami sklejanymi stopnia drugiego

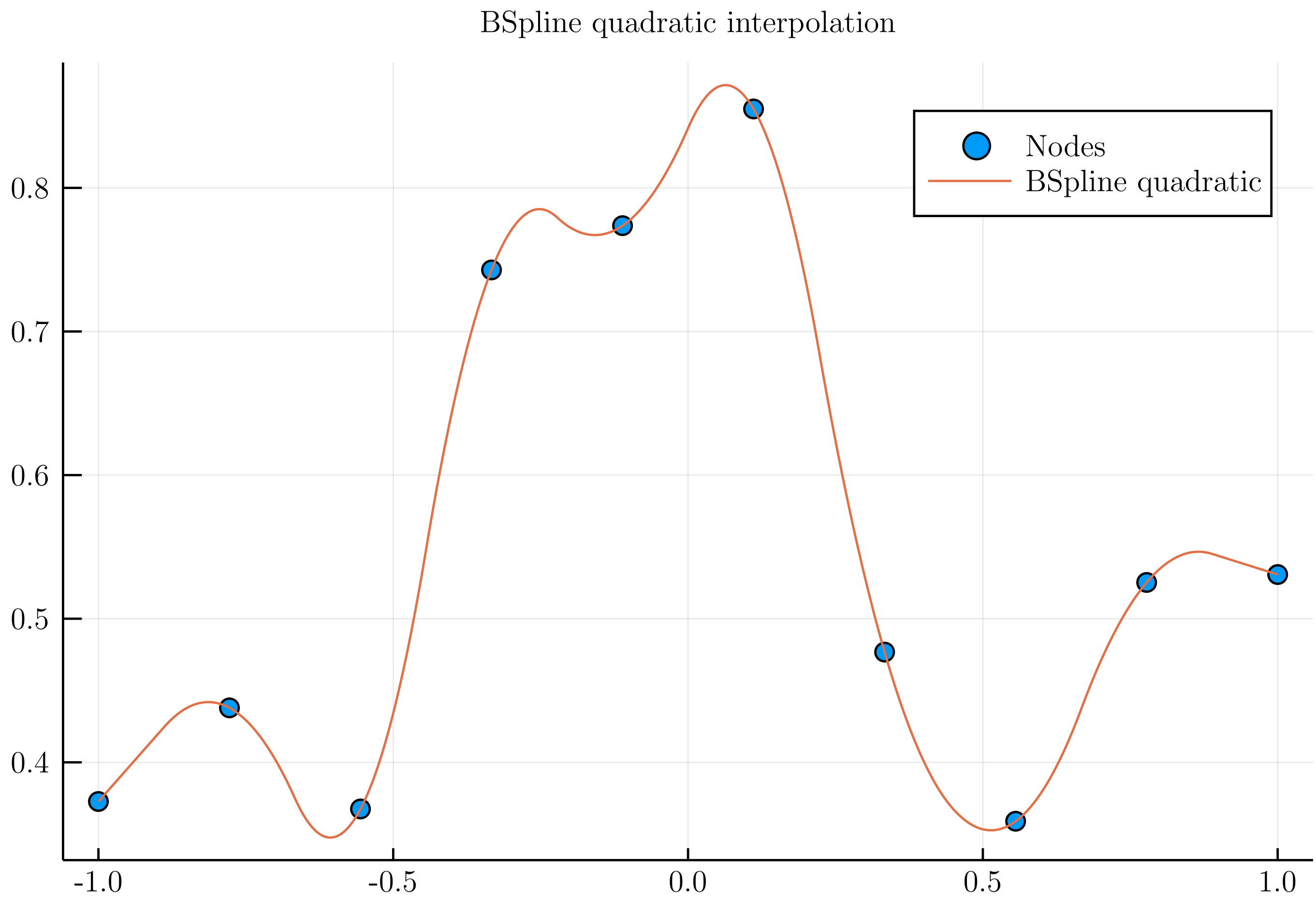


Rysunek 8: Interpolacja funkcją sklejaną stopnia 2

```
qitp = interpolate(Y, BSpline(Quadratic(Line(OnCell()))))
Q = qitp(LinRange(1, length(X), 1000))
scatter(X,Y, label="Nodes", title = "BSpline quadratic interpolation")
plot!(xsf,Q, label="BSpline quadratic")
```

Listing 9: Interpolacja funkcją sklejaną stopnia 3

Listing 9 wykonuję na węzłach wygenerowanych z Listing 1 interpolację funkcjami sklejanymi stopnia trzeciego

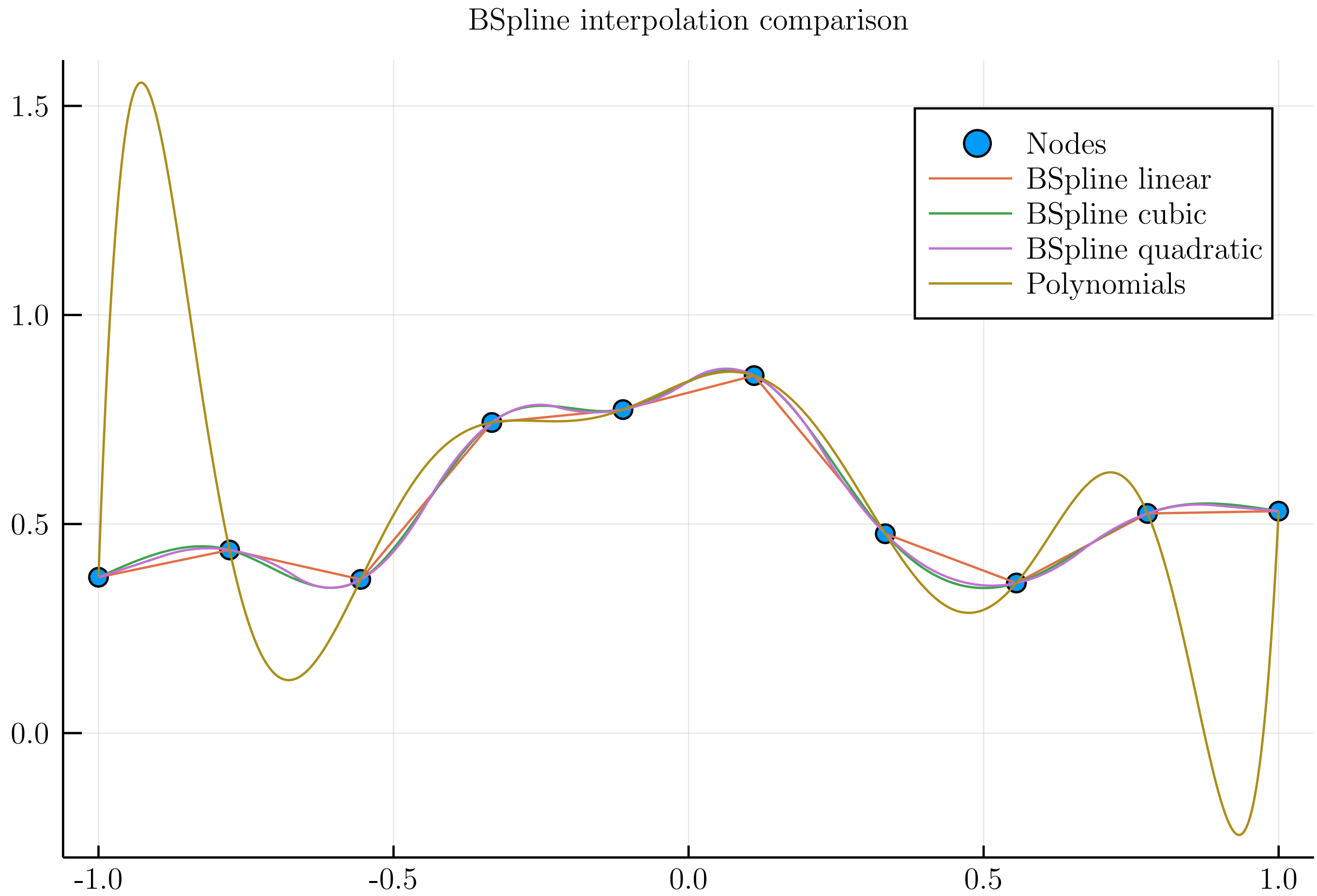


Rysunek 9: Interpolacja funkcją sklejaną stopnia 3

```
scatter(X,Y, label="Nodes", title = "BSpline interpolation comparison")
plot!(xsf, L, label="BSpline linear")
plot!(xsf,C, label="BSpline cubic")
plot!(xsf,Q, label="BSpline quadratic")
plot!(xsf, PY, label = "Polynomials")
```

Listing 10: Porównanie interpolacji funkcjami sklejanymi do interpolacji wielomianem

Rysunek 10 przedstawia porówanie interpolacji wielomanowej z interpolacją funkcjami sklejanymi. Na krańcach przedziału można zauważyć, że dla funkcji sklejanych nie występuję efekt Rungego w przeciwieństwie do wielomanu intepolacyjnego.



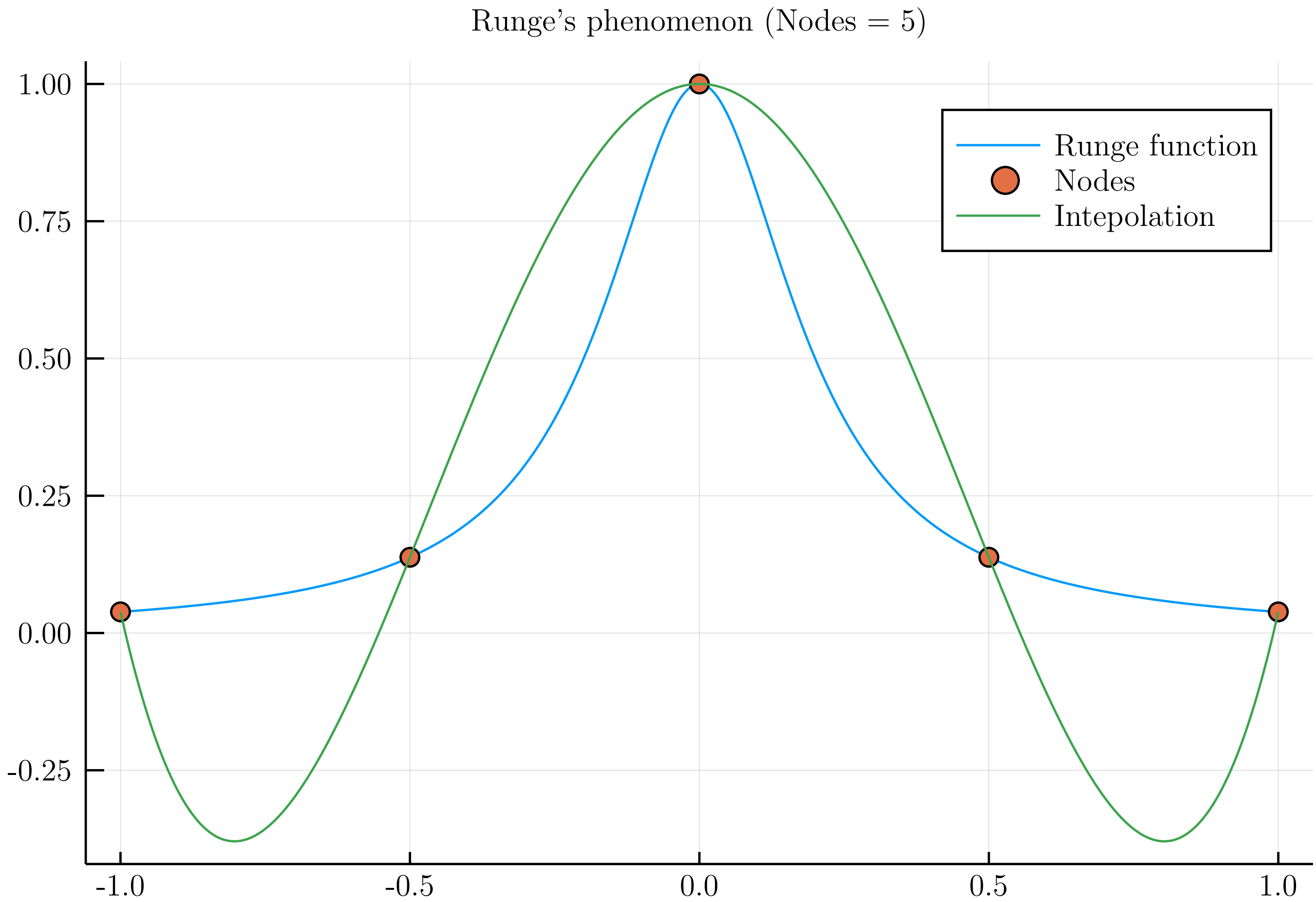
Rysunek 10: Porównanie interpolacji funkcjami sklejanymi do interpolacji wielomianem

7 Efekt Rungego

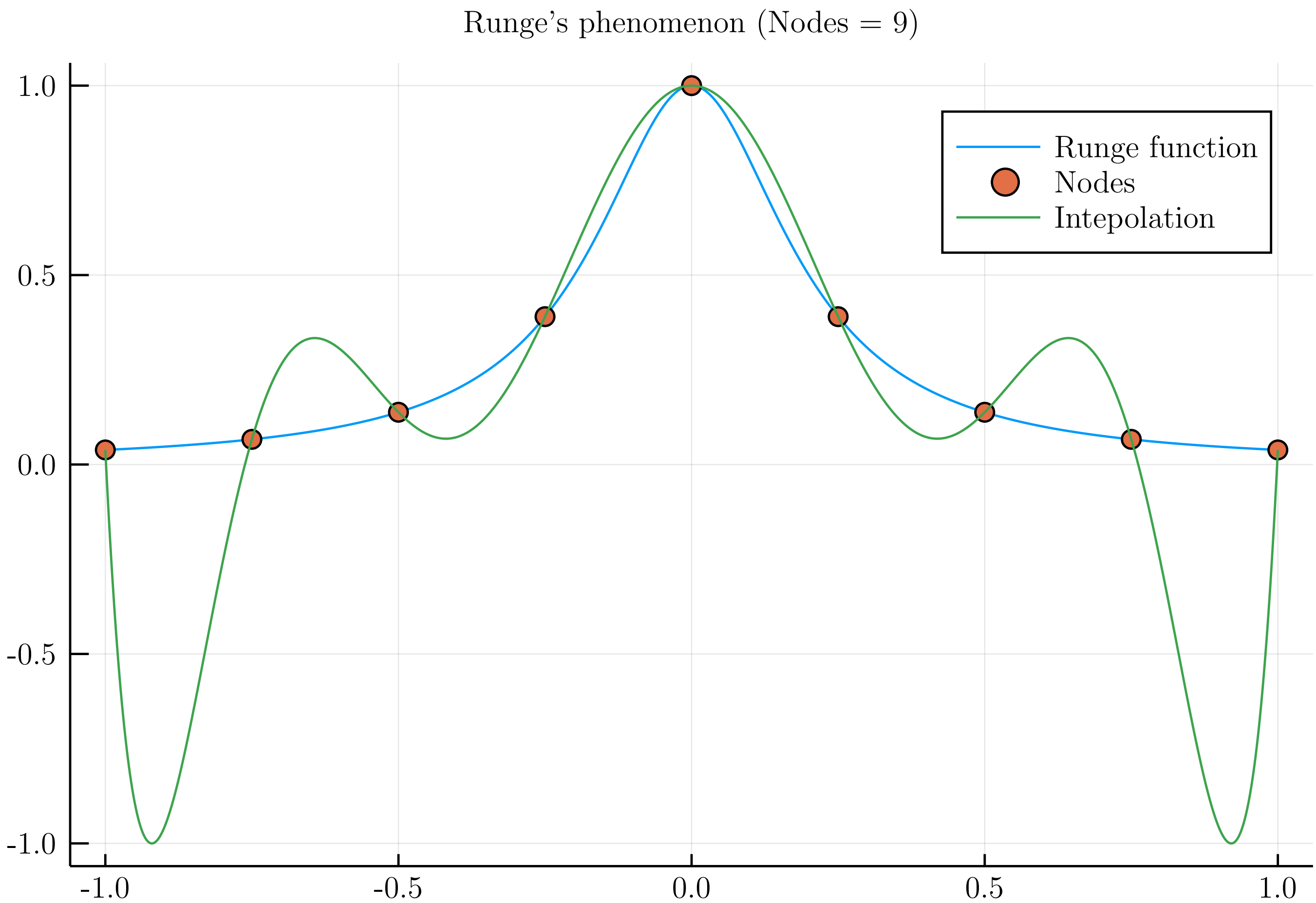
```
runge(x) = 1/(1+25*x^2)

i = 5
p = plot(xsf, map(runge, xsf), label = "Runge function",title = string("Runge's phenomenon
    (Nodes = ", i, ""))
RX = LinRange(-1, 1, i)
RY = map(runge, RX)
scatter!(RX, RY, label = "Nodes")

rpoly = polyfit(RX, RY, length(RX) - 1)
R = polyval(rpoly, xsf)
plot!(xsf, R, label = "Intepolation")
```

Listing 11: Efekt Rungego

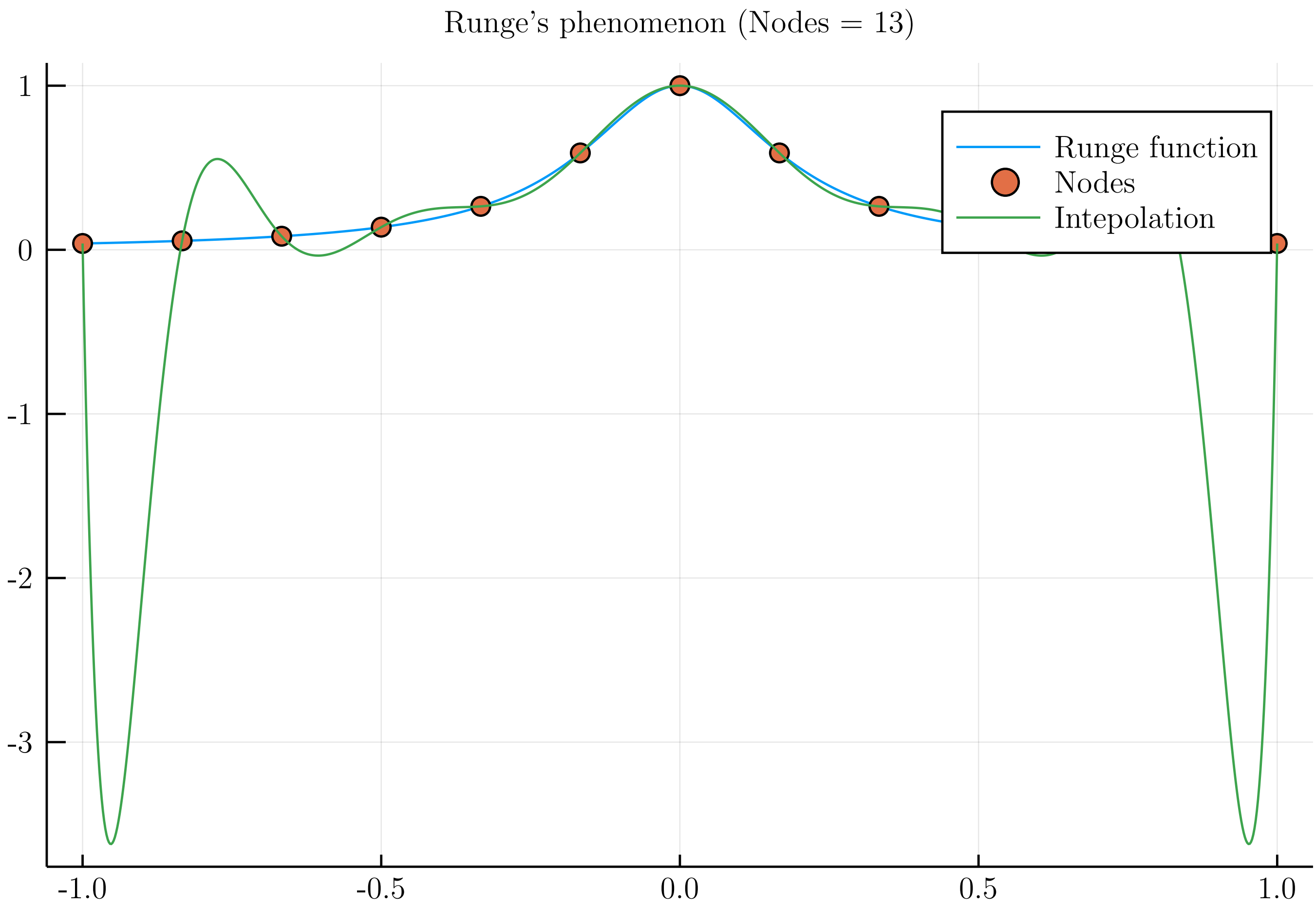
Listing 11 interpoluję wielomianem funkecję Rungego w i równoodległych węzłach. Rysunek 11, Rysunek 12 i Rysunek 13 przedstwiają funkję Rungego wraz z coraz to większą ilością węzłów intepolacji oraz wielomianem je interpolującym. Wraz ze wzrostem węzłów można zauważyć coraz większy błąd interpolacji wielomianem na końcach przedziałow.



Rysunek 11: Interpolacja wielomianem 4 stopmnia funkcji Rungego



Rysunek 12: Interpolacja wielomianem 8 stopmnia funkcji Rungego



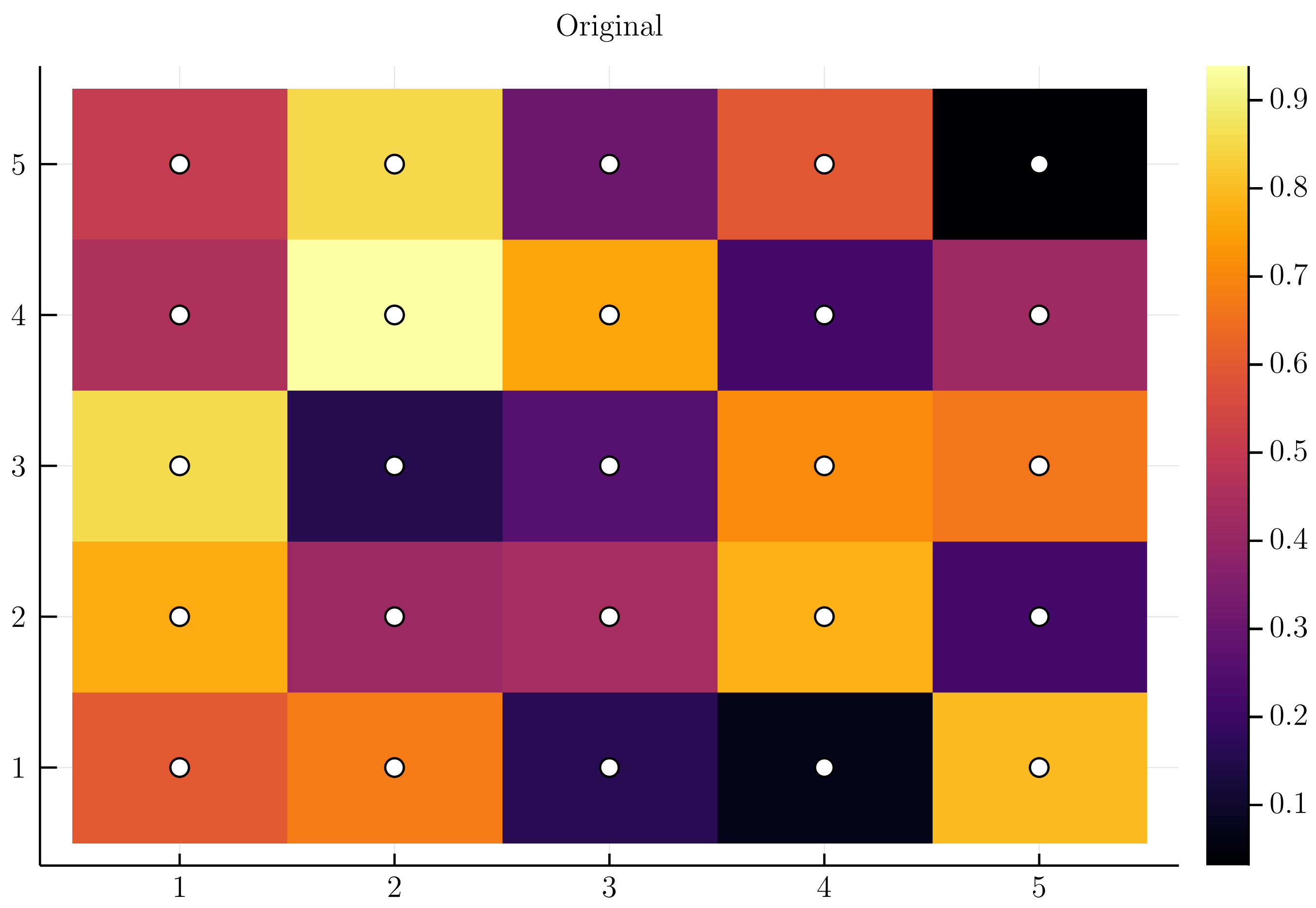
Rysunek 13: Interpolacja wielomianem 12 stopmnia funkcji Rungego

8 Interpolacja w Grafice komputerowej

```
img = rand(Float32, 5, 5)
f = heatmap(img, title = "Original")
scatter!(repeat(1:5,1, 5), transpose(repeat(1:5,1, 5)), legend=false, color = :white)
```

Listing 12: Original Image

Listing 12 tworzy macierz 5x5 z randomowymi wartościami. Zakładamy że jest to nasz obrazek początkowy.



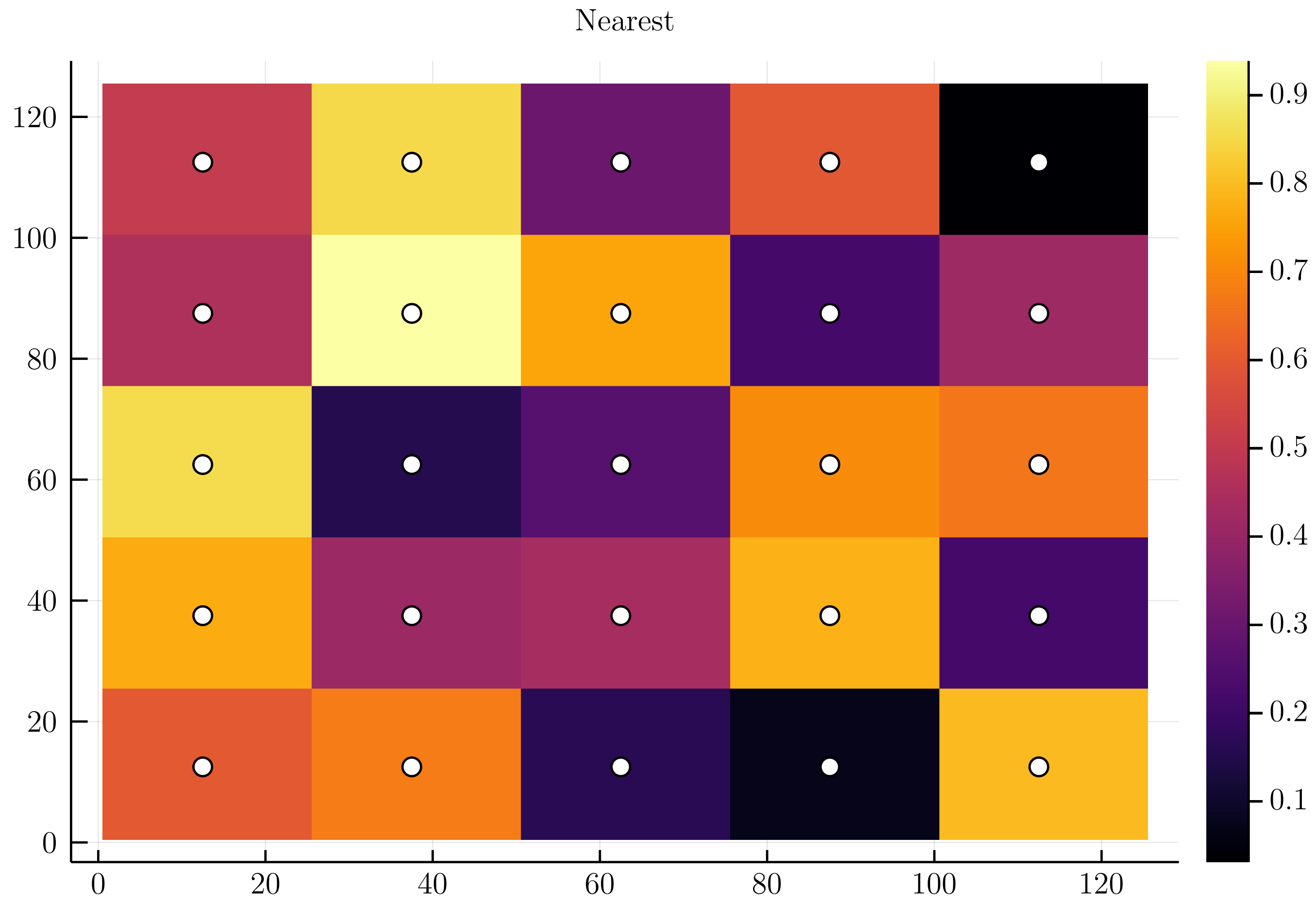
Rysunek 14: Orginal Image

Rysunek 15: Comparison of some 1- and 2-dimensional interpolations

```julia
function nearest(A::Matrix, xfactor::Number, yfactor::Number)
  lx, ly = size(A)
  nx, ny = round.(Int, (xfactor * lx, yfactor * ly))
  vx, vy = LinRange(0.5, lx + 0.5, nx), LinRange(0.5, ly+0.5, ny)
  itp = interpolate(A, BSpline(Constant()))
  etpf = extrapolate(itp, Flat())
  return etpf(collect(vx), collect(vy))
end

scale = 25
f = heatmap(nearest(img, scale, scale), title = "Nearest")
scatter!(repeat(1:5,1, 5)*scale-fill(scale/2,5,5), transpose(repeat(1:5,1, 5)*scale-fill(
    scale/2,5,5)), legend=false, color = :white)
```

Listing 13: Interpolacja nearest

Listing 13 skaluje obrazek 25 krotnie stworzony w Listing 12 korzystając z interpolacji nearest neighbor. Jest to metoda najprostsza, w której przy skalowaniu odbywa się wierne kopiowanie najbliższego piksela Rysunek 15. Metoda wymagająca od komputera najmniejszej mocy obliczeniowej, jednak jest rzadko stosowana, ponieważ w przypadku dużych powiększeń wyraźnie widać grupy identycznych pikseli, a granice pomiędzy nimi są wyraźne, ostre, nie rozmyte.

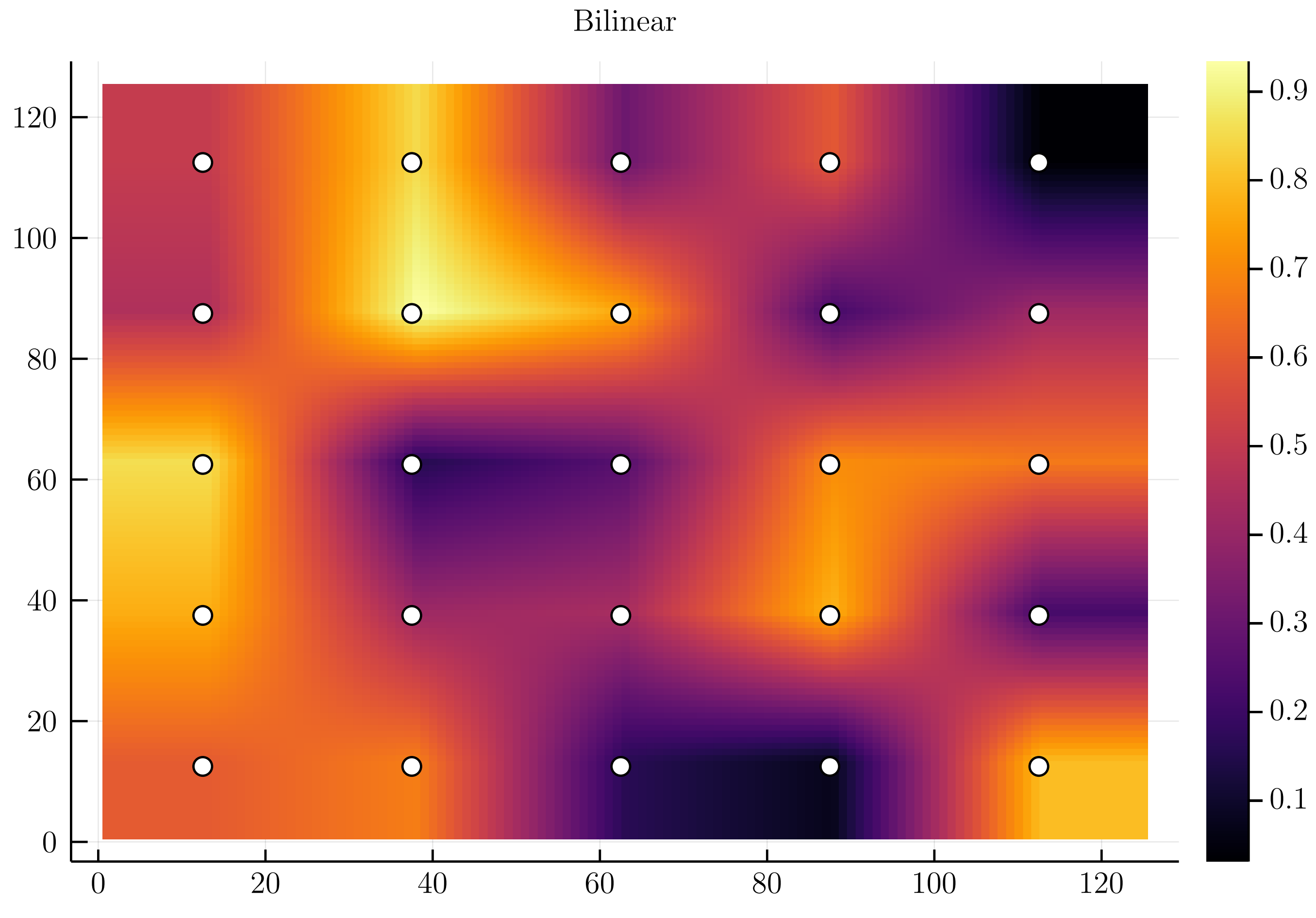


Rysunek 16: Interpolacja nearest

```
function bilinear(A::Matrix, xfactor::Number, yfactor::Number)
  lx, ly = size(A)
  nx, ny = round.(Int, (xfactor * lx, yfactor * ly))
  vx, vy = LinRange(0.5, lx + 0.5, nx), LinRange(0.5, ly+0.5, ny)
  itp = interpolate(A, BSpline(Linear()))
  etpf = extrapolate(itp, Flat())
  return etpf(collect(vx), collect(vy))
end

scale = 25
f = heatmap(bilinear(img, 25, 25), title = "Bilinear")
scatter!(repeat(1:5,1, 5)*scale-fill(scale/2,5,5), transpose(repeat(1:5,1, 5)*scale-fill(scale/2,5,5)), legend=false, color = :white)
```

Listing 14: Interpolacja bilinear

Listing 14 skaluje obrazek 25 krotnie stworzony w Listing 12 korzystając z interpolacji bilinear. Jest to metoda pośrednia, niewiele mocniej obciążająca komputer, ale i dająca lepszy, łagodniejszy dla oczu obraz. Piksele są powielane lub redukowane z uwzględnieniem kolorów czterech sąsiednich pikseli, stykających się bokami z danym pikselem Rysunek 15.

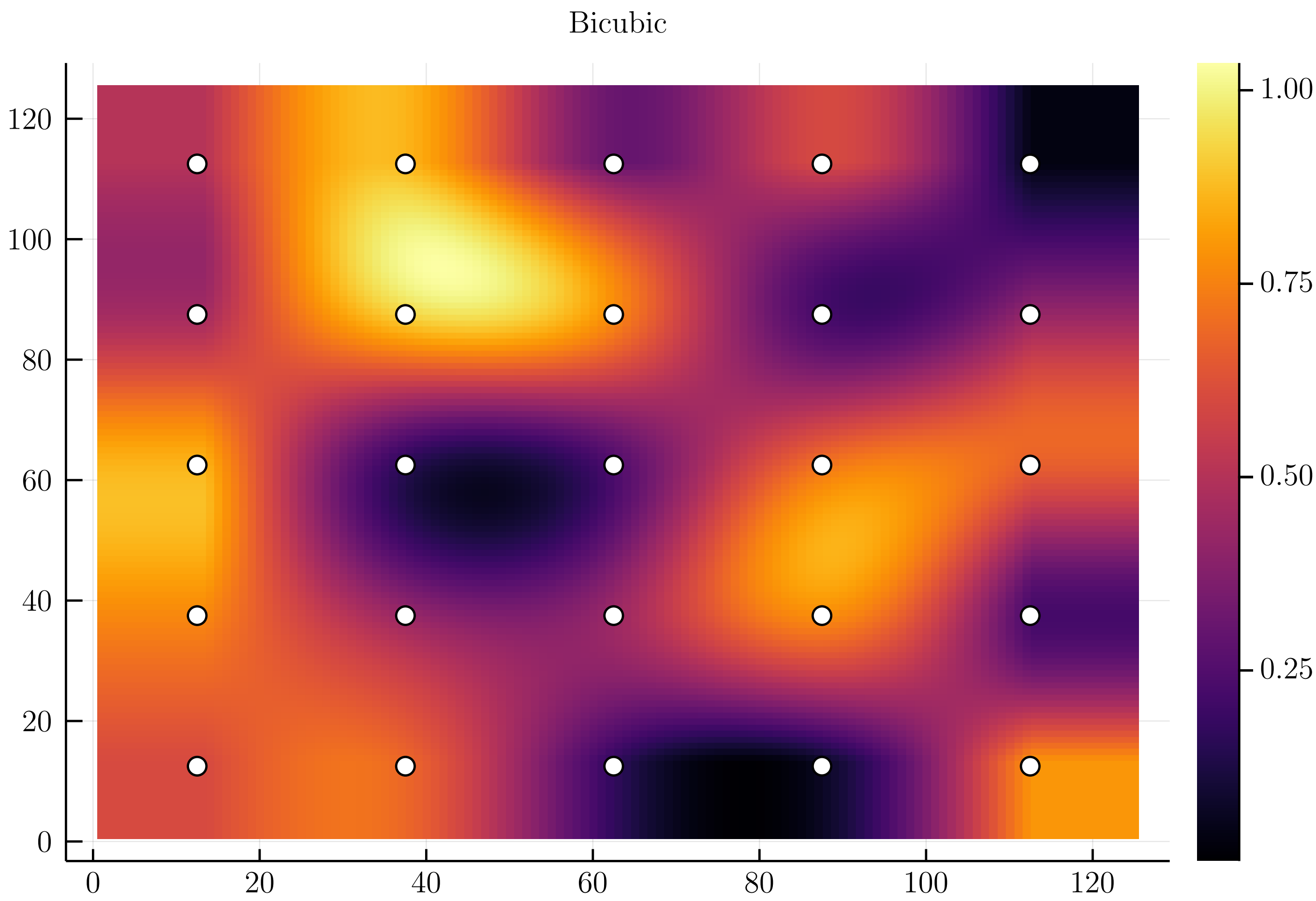


Rysunek 17: Interpolacja bilinear

```
function bicubic(A::Matrix, xfactor::Number, yfactor::Number)
  lx, ly = size(A)
  nx, ny = round.(Int, (xfactor * lx, yfactor * ly))
  vx, vy = LinRange(0.5, lx + 0.5, nx), LinRange(0.5, ly+0.5, ny)
  itp = interpolate(A, BSpline(Cubic(Line(OnGrid()))))
  etpf = extrapolate(itp, Flat())
  return etpf(collect(vx), collect(vy))
end

scale = 25
f = heatmap(bicubic(img, 25, 25), title = "Bicubic")
scatter!(repeat(1:5,1, 5)*scale-fill(scale/2,5,5), transpose(repeat(1:5,1, 5)*scale-fill(
    scale/2,5,5)), legend=false, color = :white)
savefig(f, "Bicubic.svg")
```

Listing 15: Interpolacja bicubic

Listing 15 skaluje obrazek 25 krotnie stworzony w Listing 12 korzystając z interpolacji bicubic. Jest to metoda dająca znacznie lepsze wyniki końcowe, aktualnie opcja domyślna w większości programów przetwarzających obrazy i gier komputerowych. Krawędzie są naturalnie, łagodnie rozmyte, a obraz po transformacji bardzo wiarygodnie przypomina obraz początkowy. Do skalowania obrazu metoda wykorzystuje kolory wszystkich ośmiu pikseli stykających się bokami lub wierzchołkami z danym pikselem Rysunek 15



Rysunek 18: Interpolacja bicubic