Obliczenia Naukowe- Lista 4

Szymon Skoczylas

4 grudnia 2020

Zadanie 1

Cel zadania

Celem tego zadania jest zaimplementowanie algorytmu obliczającego ilorazy różnicowe.

Opis i wykonanie

W celu oblicznia ilorazu posułżymy się, dla następujących warunków początkowych: $f[x_i] = f(x_i)$, $f[x_i, x_j] = \frac{f(x_j)-f(x_i)}{x_j-x_i}$, $i, j \in 0, \ldots, k$ (k to rząd ilorazu różnicowego), poniższym wzorem:

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

Algorytm przedstawia się następująco:

```
function ilorazyRoznicowe(x::Vector{Float64}, f::Vector{Float64})
    fx = Vector{Float64}(undef, Int64(length(f)))

    for i in 1:length(f)
        fx[i] = f[i]
    end

    for i = 2:length(f)
        for j = length(f):-1:i
            fx[j] = (fx[j] - fx[j - 1]) / (x[j] - x[j - i + 1])
        end
    end

    return fx
end
```

Zadanie 2

Cel zadania

Celem jest zaimplementowanie funkcji obliczającą wartość wielomianu interpolacyjnego stopnia n w postaci Newtona $Nn(x)$ w punkcie $x = t$ za pomocą uogólnionego algorytmu Hornera, w czasie $O(n)$.

Opis i wykonanie

Aby obliczyć wartości wielomianu posłużymy się następującym wzorem:

$$N_n(x) = \sum_{k=0}^{n} c_k q_k(x) = \sum_{k=0}^{n} f[x_0, x_1, \ldots, x_k] \prod_{j=0}^{k-1} (x - x_j)$$

Zależność tą można zapisać używając uogólnionych wzorów Hornera:

$$w_n(t) = f[x_0, x_1, \ldots, x_n]$$

$$w_k(t) = f[x_0, x_1, \ldots, x_k] + (t - x_k) w_{k+1}(t)$$

$$N_n(t) = w_0(t)$$

Algorytm obliczającą wartość wielomianu interpolacyjnego za pomocą powyższych wzorów wygląda tak:

```
function warNewton(x::Vector{Float64}, fx::Vector{Float64}, t::Float64)
    nt = fx[length(x)]

    for i = length(x) - 1:-1:1
        nt = fx[i] + (t-x[i]) * nt
    end

    return nt
end
```

Zadanie 3

Cel zadania

Celem tego zadania jest napisanie funkcji która dla danych współczynników wielomianu interpolacyjnego w postaci Newtona $c0 = f[x0], c1 = f[x0, x1], c2 = f[x0, x1, x2], ..., cn = f[x0, ..., xn]$, oraz dla danych węzłów $x0, x2, ..., xn$ oblicza (w czasie $O(n^2)$) współczynniki jego postaci naturalnej $a_0, \ldots, a_n$.

Opis i wykonanie

Algorytm został zaimplementowany tak jak przedstawiono poniżej:

```
function naturalna(x::Vector{Float64}, fx::Vector{Float64})
    a = Vector{Float64}(length(x))
    a[length(x)] = fx[length(x)]

    for i = length(x) - 1:-1:1
        a[i] = fx[i] - a[i + 1] * x[i]

        for j = i + 1:length(x) - 1
            a[j] = a[j] - a[j + 1] * x[i]
        end
    end

    return a
end
```

Algorytm ten opiera się na uogólnionym schemacie Hornera (poprzednie zadanie). Korzystamy tu z faktu, że $a_n = c_n$. W pętli obliczany kolejne wartości a_i i dla każdej z nich przekształcamy wielomian do postaci naturalnej.

Zadanie 4

Cel zadania

Celem tego zadania było zaimplementowanie funkcji, która będzie interpolować $f(x)$ w przedziale $[a, b]$ za pomocą wielomianu stopnia n w postaci Newtona. Następnie narysuje daną funkcję oraz wielomian.

Opis i wykonanie

Aby zinterpolować daną funkcję najpierw trzeba obliczyć odległość pomiędzy kolejnymi wezłami, a potem wartość samej funkcji w tych węzłach. Następnie wyliczamy ilorazy różnicowe, a potem wartści wielomianu w n równoodległych punktach. Do narysowania wykresów został użyty pakiet *PyPlot*. Kod implementujący daną funkcjonalnośc jest przedstawiony poniżej:

```
function rysujNnfx(f, a::Float64, b::Float64, n::Int)
    dist_diff, current_dist = ((b - a) / n), Float64(0.0)

    x, y = Vector{Float64}(undef, Int64(n + 1)), Vector{Float64}(undef, n + 1)
    f_x = Vector{Float64}(undef, n + 1)
    args = Vector{Float64}(undef, 15 * (n + 1))
    f_plot, g_plot = Vector{Float64}(undef, 15 * (n + 1)), Vector{Float64}(undef, 15 * (n + 1))

    for i = 1:(n+1)
        x[i] = a + current_dist
        y[i] = f(x[i])
        current_dist += dist_diff
    end

    f_x = ilorazyRoznicowe(x, y)
    current_dist = Float64(0.0)
    dist_diff = (b - a) / (15 * (n + 1) - 1)

    for i = 1:(15 * (n + 1))
        args[i] = a + current_dist
        g_plot[i], f_plot[i] = warNewton(x, f_x, args[i]), f(args[i])
        current_dist += dist_diff
    end

    clf()
    plot(args, f_plot, label="f(x)")
    plot(args, g_plot, label="g(x)")
    savefig("plot_$(f)_$n.png")
end
```

Zadanie 5

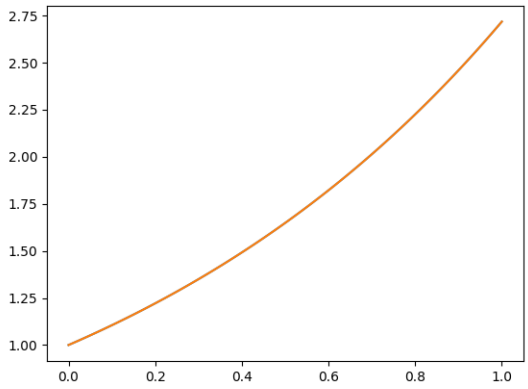
Cel zadania

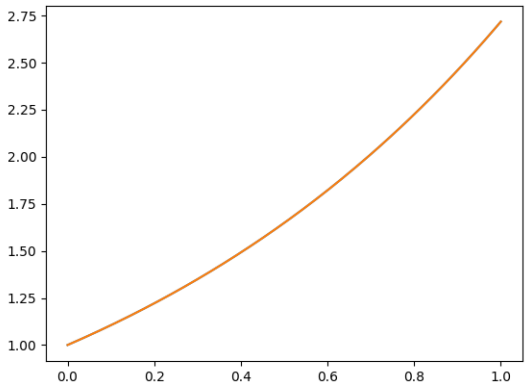
Przetestowanie funkcji z zadania 4 na następujących przykładach:

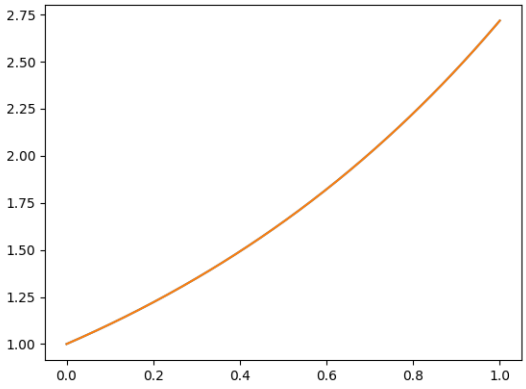
1. e^x, $[0, 1]$, $n = 5, 10, 15$
2. $x^2 sinx$, $[-1, 1]$, $n = 5, 10, 15$

Wykonanie i wyniki

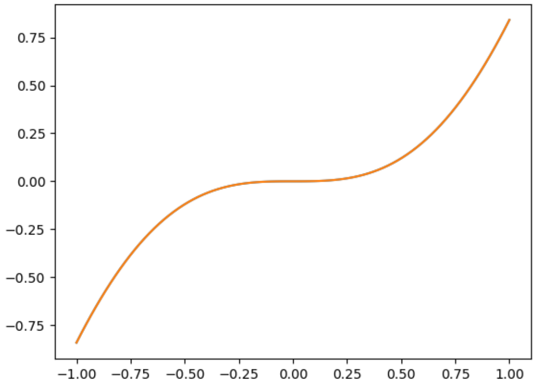
Wykresy dla wywołań funkcji, z zadania 4, wyspecyfikowane w poleceniu:
-Dla funkcji e^x:

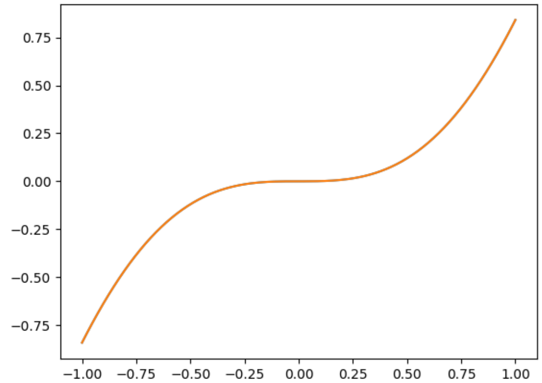


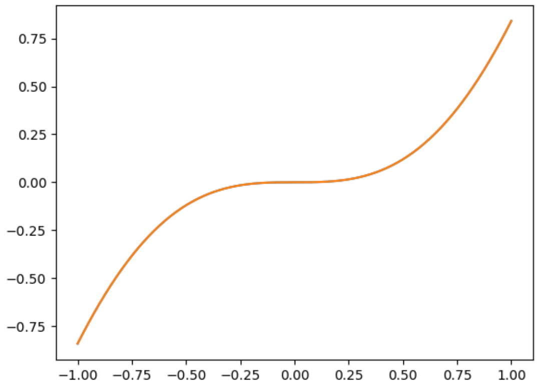




-Dla funkcji $x^2 sinx$:







Wnioski

Jak wydać na wykresach, wartości funkcji oraz ich wielomianów interpolacyjnych są bardzo zbliżone. Otrzymaliśmy bardzo dobre przybliżenie prawwdziwych dancyh. Spowodowane jest to tym, że węzły są od siebie równoodległe.

Zadanie 6

Cel zadania

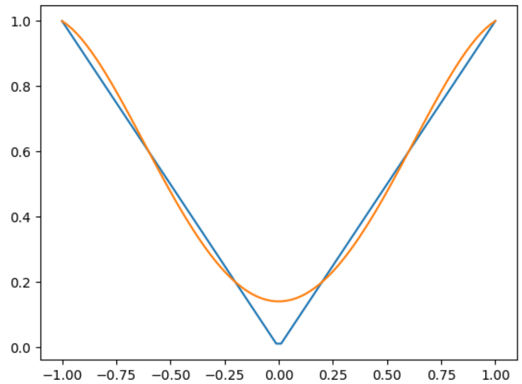
Przetestowanie funkcji z zadania 4 na następujących przykładach:

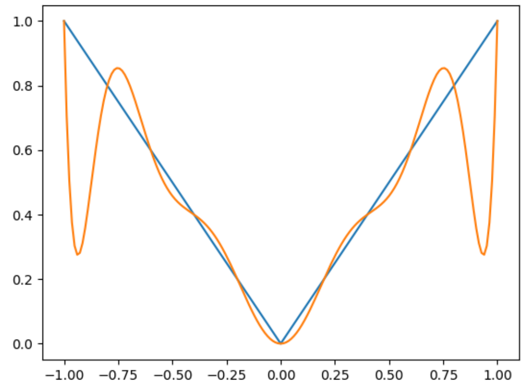
1. $|x|$, $[-1, 1]$, $n = 5, 10, 15$
2. $\frac{1}{1+x^2}$, $[-5, 5]$, $n = 5, 10, 15$

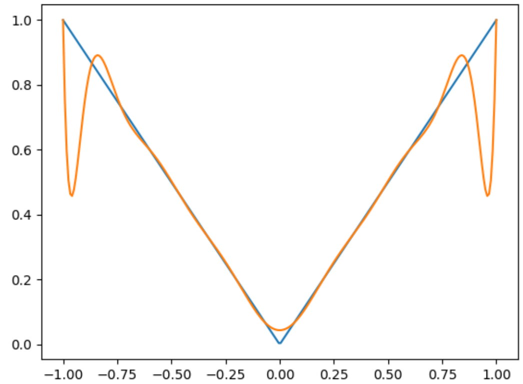
Wykonanie i wyniki

Wykresy dla wywołań funkcji, z zadania 4, wyspecyfikowane w poleceniu:

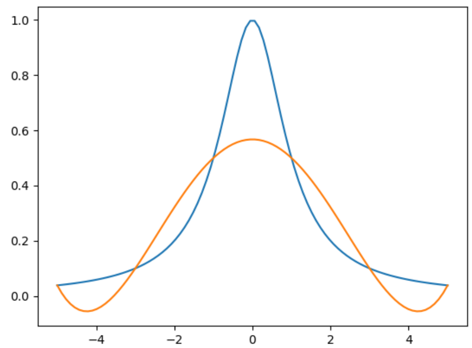
-Dla funkcji $|x|$:

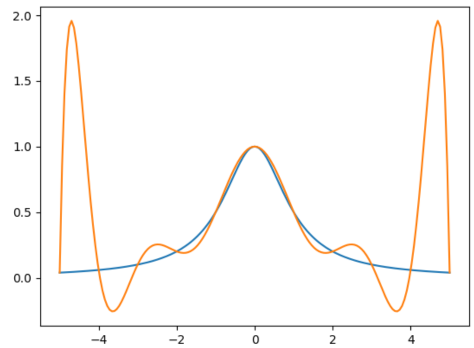


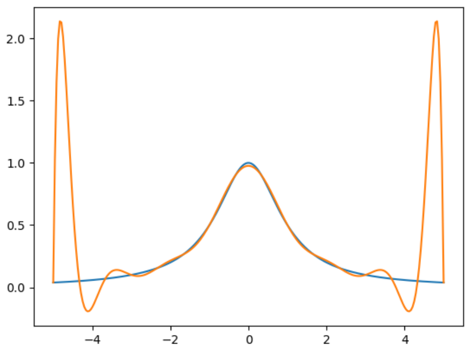




-Dla funkcji $\frac{1}{1+x^2}$:







Wnioski

Z powyższych wykresach można wywnioskować, że wraz ze zwrostem liczby węzłów (n) przybliżenie wartości funkcji poprawia się, z wyjątkiem końców przedziałów. Na końcach przedziałów można zaobserwować znaczne pogorszenie przybliżenia. Jest to tz. efekt Runge'go. Efekt ten polega na spadku dokładności interpolacji, mimo zwiększania liczby węzłów. Aby zniwelować ten efekt, można użyć wielomianu Czebyszewa n-tego stopnia.