MNUM Projekt - 2

Bartosz Latosek 310790

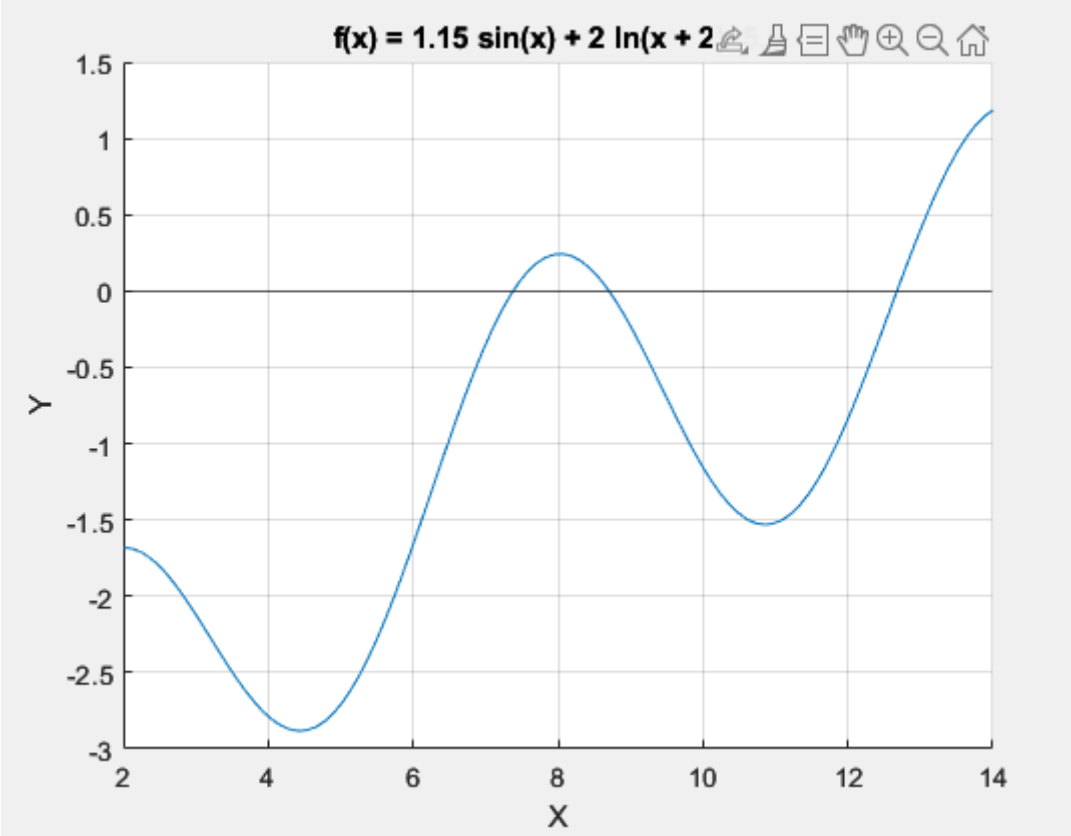
Listopad 2022

1 Zadanie 1 - znajdywanie wszystkich pierwiastków funkcji.

1.1 Badana funkcja

$$f(x) = 1.15\sin(x) + 2\ln(x+2) - 5.5 \quad \text{w przedziale} \quad [5, 11]$$

Rysunek 1: Wzór badanej funkcji f



Rysunek 2: Wykres badanej funkcji f

1.2 Znajdywanie miejsc zerowych funkcji metodą bisekcji.

Algorytmy oraz ich wykorzystanie można znaleźć w plikach *src/bisect.m* oraz *zadanie1.m*.

1.2.1 Algorytm bisekcji

Algorithm 1 Metoda Bisekcji

1: **function** BISECT(f, a, b, δ) ▷ Where f - tested function, [a, b] - interval, δ - minimal acceptable error
    $c = f(a); d = f(b)$
    $e = inf$ ▷ Current approx.error
2: **while** $e > \delta$ **do** ▷ while current error is higher than minimal acceptable
        $middle = (a + b)/2$ ▷ Finding middle of interval
        $y = f(middle)$ ▷ Calculating function value in the middle
        if $c * y < 0.0$
            $b = middle$
        else if $d * y < 0.0$
            $a = middle$
        endif
        $x = (a + b)/2$
        $e = |f(x)|$ ▷ Calculating error of currently calculated x approximation
3: **end while**

4: **end function**

1.2.2 Użycie algorytmu bisekcji do znalezienia miejsc zerowych funkcji

Podany przedział [5, 11] obejmuje w sobie 2 miejsca zerowe. Arbitralnie więc, na podstawie rysunku przeszukiwane będą kolejno przedziały [6, 8], [8, 9].

```
ans =

    'Bisection on: [6, 8]: 7.37948 with error: 5.2581e-09 after 23 iterations
      in total time 0.00151. s. Approximated value: -5.2581e-09'


ans =

    'Bisection on: [8, 9]: 8.70429 with error: 5.0703e-10 after 26 iterations
      in total time 0.00082. s. Approximated value: 5.0703e-10'

>>
```

Rysunek 3: Wyniki uzyskane metodą bisekcji

1.3 Znajdywanie miejsc zerowych funkcji metodą Newtona

Zaimplementowany algorytm znajduje się w pliku newton.m i został pobrany ze strony przedmiotu, w związku z czym pomijam sekcję pseudokodu algorytmu. W uproszczeniu, metoda polega na obliczaniu kolejnych przybliżeń miejsc zerowych funkcji, przez wykorzystywanie właściwości pochodnej funkcji w danym puncie.

W tej metodzie zostały wykorzystane te same, arbitralnie wyznaczone przedziały [6, 8] i [8, 9] a punkt startowy metody newtona jest środkiem kolejno badanego przedziału.

```
ans =

    'Newton on: [6, 8]: 7.37948 with error: -5.7465e-12 after 4 iterations
      in total time 0.00094. s. Approximated value: -5.7465e-12'


ans =

    'Newton on: [8, 9]: 8.70429 with error: -3.3751e-14 after 4 iterations
      in total time 0.00058. s. Approximated value: -3.3751e-14'

>>
```

Rysunek 4: Wyniki uzyskane metodą Newtona

1.4 Porównanie metod bisekcji i Newtona

Metoda	Przedział	Punkt Początkowy	Wartość Funkcji w p	Punkt Końcowy	Wartość Funkcji w PK	Liczba iteracji	Czas
Bisekcji	[6, 8]	n.d.	n.d.	7.37948	-5.2581e-09	23	0.00151
Bisekcji	[8, 9]	n.d.	n.d.	8.70429	5.0703e-10	26	0.00082
Newtona	[6, 8]	7	-0.3500	7.37948	-5.7465e-12	4	0.00094
Newtona	[8, 9]	8.5	0.1210	8.70429	-3.3751e-14	4	0.00058

Rysunek 5: Porównanie metod

Pierwszym, co rzuca się w oczy przy analizowaniu powyższej tabelki, są liczby iteracji potrzebne do uzyskania satysfakcjonującego przybliżenia miejsca zerowego funkcji. W przypadku metody Newtona są to tylko 4 iteracje. Ponieważ w przypadku metody Newtona występuje większy nakład obliczeniowy związany z obliczaniem pochodnej funkcji w punkcie w każdej iteracji, czasy wykonania obydwu metod są porównywalne. W przpadku metody Newtona uzyskujemy również dużo dokładniejsze przybliżenie miejsca zerowego. (Widać to po rubryce Wartośc funkcji w punkcie końcowym, wartości punktu końcowego są różne, ale przez formatowanie do 5 cyfr znaczących nie widać różnicy)

1.5 Wnioski

Metoda Newtona potrzebuje mniejszej ilości iteracji do znalezienia satysfakcjonującego punktu niż metoda Bisekcji, ponieważ metoda bisekcji zawsze dzieli przedział na pół, w przeciwieństwie do metody Newtona, która zbliża się do miejsca zerowego proporcjonalnie do współczynnika nachylenia wykresu w danym punkcie.

Metoda Newtona jest wrażliwa na wybrany punkt początkowy, powinien być on możliwie jak najbliższy miejscu zerowemu, inaczej możemy nie osiągnąć zbieżności. W przypadku metody Newtona, funkcja musi być ciągła i różniczkowalna w sprawdzanym przedziale.

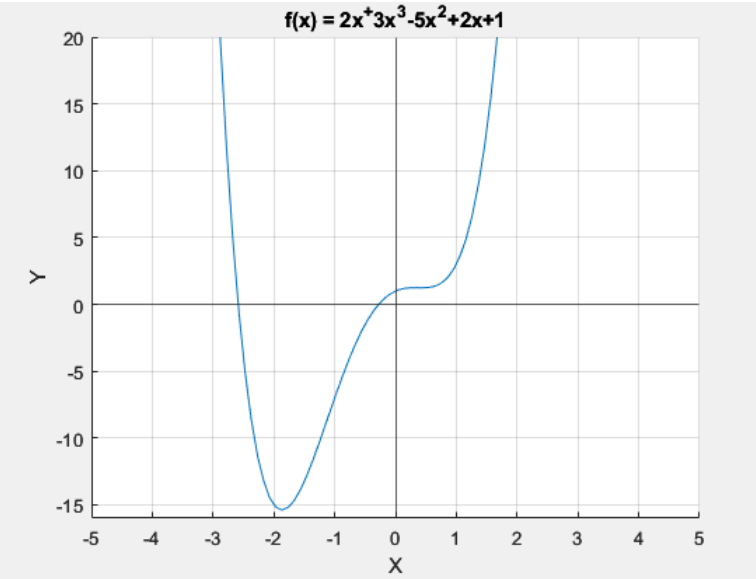
Podsumowując, metoda Newtona daje lepsze wyniki w krótszym czasie, ale jest bardziej niestabilna i wymaga większej ostrożności w przypadku użycia.

2 Zadanie 2 - Wyznaczanie pierwiastków wielomianu metodą Mullera MM2

2.1 Badany wielomian

$$f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0, \qquad [a_4\ \ a_3\ \ a_2\ \ a_1\ \ a_0] = [2\ 3\ -5\ 2\ 1]$$

Rysunek 6: Badany wielomian



Rysunek 7: Wykres badanego wielomianu

2.2 Metoda Mullera MM2

Implementacja metody znajduje się w pliku *src/MM2.m*

Algorithm 2 Metoda Mullera MM2

1: **function** MM2(f, x, δ)▷ Where f - tested polynomial, x - initial x vector, δ - minimum approximation error
    $q = queue(x)$ ▷ Make a queue out of x vector
    $e = inf$ ▷ Current approx.error
2: **while** $e > \delta$ **do** ▷ while current error is higher than minimal acceptable
        $x_{sorted} = sort(q)$
        $poly = LagrangePoly(x_{sorted}, f)$ ▷ Create Lagrange polynomial based on current 3 points
        $r = roots(poly)$ ▷ Calculate roots of lagrange polynomial
        $y_1 = f(r[1])$ ▷ Calculate function values in roots $y_2 = f(r[2])$
        if $|y_1| < |y_2|$ ▷ Pick root that's value closer to 0
            $x_{pred} = r[1]$
        else
            $x_{pred} = r[2]$
        $q- > popFront()$ ▷ Update the queue
        $q- > pushBack(x_{pred})$
        $e = f(x_{pred})$
3: **end while**

4: **end function**

W powyższym pseudokodzie, wielomian Lagrange'a jest obliczany za pomocą wzoru:

$$\bullet\ L_2(x) = f(x_1)\frac{x-x_2}{x_1-x_2}\cdot\frac{x-x_3}{x_1-x_3} + f(x_2)\frac{x-x_1}{x_2-x_1}\cdot\frac{x-x_3}{x_2-x_3} + f(x_3)\frac{x-x_1}{x_3-x_1}\cdot\frac{x-x_2}{x_3-x_2}$$

Rysunek 8: Wielomian Lagrange'a

2.3 Analiza działania algorytmu Mullera MM2

Początkowo wybrany przeze mnie wektor x to [-3, -2, -1]. Przy tak dobranym wektorze wejściowym, algorytm wywoływany przez plik *zadanie2.m* zwraca:

```
ans =

    'Muller MM2 method on : [-3, -2, -1]: -0.27895 with error: -1.3269e-12 after 6 iterations
      in total time 1.05679. s. Approximated value: -1.3269e-12'

>>
```

Rysunek 9: Output zadanie2.m

Gdy za wektor początkowy przyjmiemy [-4, -3, -2] to algorytm zwróci nam:

```
ans =

    'Muller MM2 method on : [-4, -3, -2]: -2.58694 with error: 8.5265e-13 after 5 iterations
      in total time 0.09347. s. Approximated value: 8.5265e-13'

>>
```

Rysunek 10: Output zadanie2.m

Na podstawie powyższych wyników widzimy, że metoda Mullera MM2 jest efektywna - w przeciągu 5 iteracji zwróciła nam bardzo dobre przybliżenie pierwiastka wielomianu. Sprawdzę teraz, jak zachowa się algorytm w przypadku wyboru odległych od pierwiastków punktów startowych.

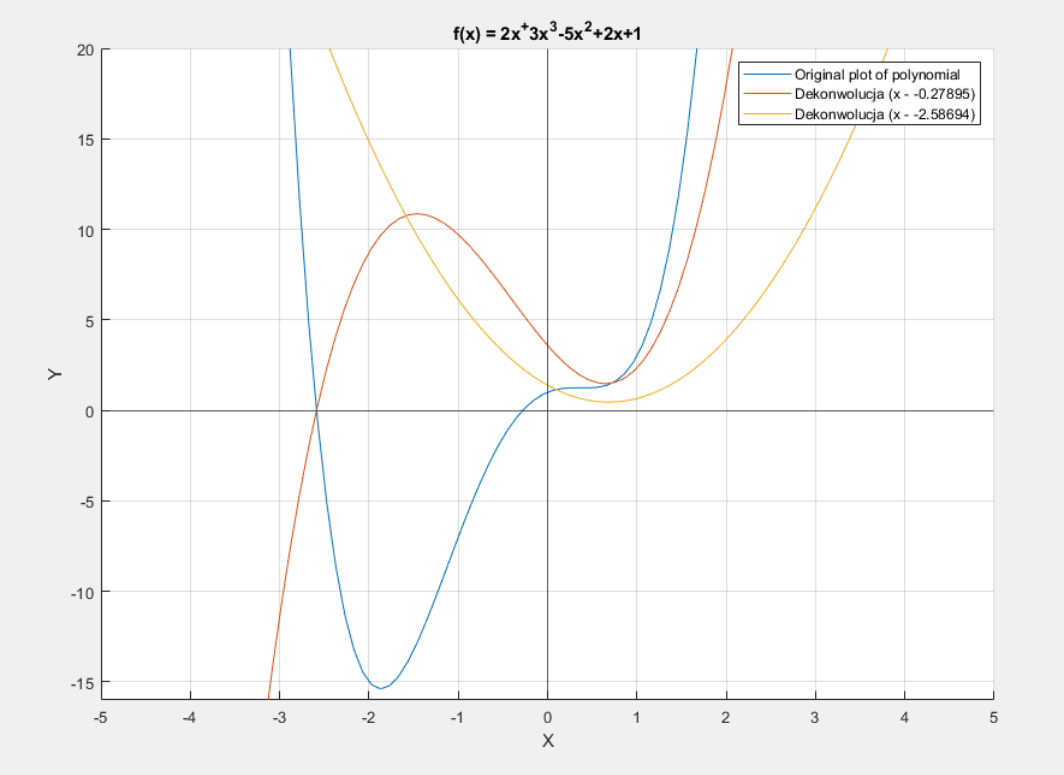
```
ans =

    'Muller MM2 method on : [-500, -499, -498]: -2.58694 with error: -1.1102e-13 after 31 iteration
      in total time 0.60749. s. Approximated value: -1.1102e-13'

>>
```

Rysunek 11: Output zadanie2.m

Algorytm znowu poradził sobie bardzo dobrze. Liczba iteracji jest co prawda pięciokrotnie większa niż poprzednio, ale wybrane punkty startowe były bardzo odległe od pierwiastków wielomianu.

2.4 Znajdowanie pierwiastków wielomianu przy użyciu deflacji wielomianu czynnikiem liniowym

W powyższym problemie zastosowałem rozkład wielomianu metodą Hornera, zaimplementowaną na potrzeby zadania w pliku *myhorner.m*. Dla arbitralnie wybranego wektora początkowego (w tym przypadku dla [-3, -2, -1] wyznaczamy pierwiastek wielomianu, a następnie wykonujemy deflacje wielomianu. Proces powtarzamy do momentu, gdy uzyskane pierwiastki będą liczbami urojonymi. Oznacza to, że obecnie szukane rozwiązania nie należą do zbioru liczb rzeczywistych i należy przerwać dalsze obliczenia.



Rysunek 12: Wykres kolejnych wielomianów po deflacji

Jak widać, taki rozkład oryginalnego wielomianu prowadzi do znalezienia wszystkich jego rzeczywistych pierwiastków.