Metody Numeryczne - Projekt 1

Zuzanna Piróg, Krzysztof Sawicki, Wojciech Grabias

Grudzień 2022

Celem projektu jest wyznaczenie zer dowolnego wielomianu postaci

$$w(z) = \sum_{k=0}^{n} a_k z^k$$

oraz wizualizacja szybkości zbieżności metod: Jarratta, Kinga oraz Homeiera w dziedzinie zespolonej. Do obliczania wartości wielomianu i jego pochodnej zastosowany zostanie algorytm Hornera.

Wydział Matematyki
i Nauk Informacyjnych
POLITECHNIKA WARSZAWSKA

Spis treści

1 Algorytm Hornera

1.1 Obliczanie wartości wielomianu

Kluczem do wykorzystania metody Hornera w celu wyliczenia wartości wielomianu oraz jego pochodnej dla danego argumentu jest przedstawienie wielomianu

$$w(z) = a_0 + a_1 z + a_2 z^2 + ... + a_n z^n$$

jako

$$w(z) = a_0 + z(a_1 + z(a_2 + ... + z(a_{n-2} + z(a_{n-1} + za_n))...))$$

Następnie, począwszy od najbardziej zagnieżdżonego nawiasu obliczamy wartość wyrażenia $a_{k-1} + za_k$ i traktujemy je jako nowy współczynnik dla nawiasu $k-1$ -szego względem zagnieżdżenia dla $k = n, n-1, ...0$. Implementacja w języku Matlab wygląda następująco:

```
function [y] = Horner(p, x0)
    % Funkcja Horner sluzy okresleniu wartosci danego wielomianu
    % dla okreslonego argumentu

    % c: Wektor wspolczynnikow wielomianu
    % x: Argument, dla ktorego wartosci szukamy

    n = length(p);
    y = p(1);
    for i = 2:n
        y = y * x0 + p(i);
    end
end
```

Jako przykład spróbujmy policzyć wartość wielomianu $w(z) = z^2 + 1$ dla $z = 2$

```
>> Horner([1 0 1], 2)

ans =

     5
```

1.2 Obliczanie wartości pochodnej wielomianu

Chcąc uzyskać formę analogiczną do tej użytej w ramach obliczenia wartości wielomianu, możemy otrzymać

$$w(z) = a_0 + a_1 z + a_2 z^2 + ... + a_n z^n = a_0 + z p_1$$

$$\text{gdzie } p_1 = p_1(z)$$

Różniczkując względem z:

$$w'(z) = p_1 + z p_1' = p_1 + z(p_2 + z p_2')$$

Ostatecznie dochodzimy do postaci:

$$w'(z) = p_1 + z(p_2 + z(... + z(p_{n-2} + z(p_{n-1} + z p_n))...))$$

Otrzymaliśmy więc formę zupełnie analogiczną do tej z poprzedniego punktu, w algorytmie obliczania wartości pochodnej należy pamiętać jednak o przejściu z wartości współczynników a_i na $p_i(x)$ dla $i = 1, ..., n$. Implementacja powyższej metody w języku Matlab: b
b

```
function p = HornerDerivative(c, x)
    % Funkcja HornerDerivative sluzy okresleniu wartosci pochodnej
    % danego wielomianu w danym punkcie

    % c: Wektor wspolczynnikow wielomianu
    % x: Argument, dla ktorego wartosci szukamy

    n = length(c);

    w = c(1);
    p = w;

    % Nie bierzemy pod uwage wyrazu wolnego w iteracjach
    for k = 2:n-1
        wk = c(k) + x*w;
        pk = wk + x*p;
        p = pk;
        w = wk;
    end
end
```

W celu sprawdzenia poprawności metody spróbujmy policzyć wartość pochodnej wielomianu $w(z) = z^2 + 1$ w punkcie $z = i$:

```
>> HornerDerivative([1 0 1], i)

ans =

   0.0000 + 2.0000i
```

2 Obliczanie miejsc zerowych wielomianu

W projekcie rozważane będą jedynie iteracyjne metody obliczania miejsc zerowych dowolnego wielomianu o współczynnikach zespolonych. Celem zaimplementowanych metod będzie znalezienie wartości najbliższej liczbie $\alpha \in \mathbb{C}$ spełniającej

$$w(\alpha) = 0$$

Wszyskie poniżej przedstawione metody posługują się iteracyjnym wzorem kolejnego przybliżenia o postaci ogólnej:

$$x_{k+1} = T(x_k) \text{ dla } k \geq 0$$

przy czym T jest przekształceniem charakterystycznym dla danej metody, natomiast x_0 - ustalonym argumentem początkowym, względem którego badana będzie liczba wymaganych iteracji. Wszystkie opisy matematyczne metod są uproszczeniem dokładnie opisanych i wyprowadzonych analiz z książki J.M. McNamee - *Numerical methods for roots of polynomials, part I.*

Przyjętym warunkiem stopu dla każdej z metod jest warunek

$$|w(x_k)| < \epsilon$$

dla przyjętego na potrzebę projektu $\epsilon = 0.01$

Każda z metod zostanie zaimplementowana w języku Matlab. Dla każdej z nich potwierdzone zostanie poprawne działanie poprzez porównanie z analitycznie wyznaczonymi wynikami. Ze względu na ograniczenia sprzętowe, maksymalna liczba iteracji ustawiona zostanie na 30. Jeżeli dana metoda przy określonym argumencie początkowym przekroczy ten limit, implementacje automatycznie ustawią liczbę iteracji na 31.

W celu wizualizacji szybkości zbieżności każda z metod zaprezentuje liczbę iteracji wymaganą do uzyskania warunku stopu za pomocą macierzy $A \in \mathbb{R}^{n \times n}$, której współczynniki reprezentowały wymaganą liczbę iteracji $k \in \{1, ..., 30\}$. Każdej z możliwych wartości przypisany zostanie odpowiedni kolor, co ostatecznie graficznie zwizualizuje szybkość zbieżności metody na danym podzbiorze $\mathbb{C}$.

3 Metoda Jarratta

3.1 Opis matematyczny metody

Metoda iteracyjna Jarratta rzędu 3 została zaproponowana przez matematyka o tym samym nazwisku w roku 1966. Ściśle wywodzi się ona z iteracyjnej metody Newtona, wyprowadzenie jej jednak polega na nieco innym założeniu, mianowicie Jarratt przypuścił, że

$$x_{i+1} = x_i - \frac{f(x_i)}{a_1 w_1(x_i) + a_2 w_2(x_i)} \text{ , gdzie}$$

$$w_1(x_i) = f'(x_i),\ w_2(x_i) = f'(x_i + \alpha u(x_i))\ i\ u = \frac{f}{f'}$$

Przypuśćmy teraz, że ζ jest pierwiastkiem jednokrotnym i niech $\epsilon = \epsilon_i = x_i - \zeta$. Z rozszerzenia w szereg Taylora otrzymujemy

$$f(x_i) = c_0 + c_1\epsilon + c_2\epsilon^2 + O(\epsilon^3) \text{ , gdzie } c_r = \frac{f^{(r)}(\zeta)}{r!}$$

Analogicznie definiujemy $f''(x_i)$ oraz $f^{(3)}(x_i)$, stosując odpowiednie podstawienie dostając

$$u(x_i) = \epsilon - \frac{c_2}{c_1}\epsilon^2 + O(\epsilon^3)$$

$$w_2(x_i) = c_1 + 2c_2(1+\alpha)\epsilon + [3c_3(1+2\alpha+\alpha^2) - 2\frac{c_2^2}{c_1}\alpha]\epsilon^2 + O(\epsilon^3)$$

Z czego wynika

$$\begin{aligned} a_1 w_1(x_i) + a_2 w_2(x_i) &= (1+\alpha)a_2]\epsilon + [3c_3 a_1 + a_2 3c_3(1+\alpha^2) - 2\frac{c_2^2}{c_1}\alpha]\epsilon^2 + O(\epsilon^3) \\ &= p_1 + p_2\epsilon + p_3\epsilon^2 + O(\epsilon^3) \end{aligned}$$

Mamy ostatecznie

$$\begin{aligned} \epsilon_{i+1} &= x_{i+1} - \zeta \\ &= \epsilon - (c_1\epsilon + c_2\epsilon^2 + c_3\epsilon^3)(p_1 + p_2\epsilon + p_3\epsilon^2)^{-1} \\ &= \left(1 - \frac{c_1}{p_1}\right)\epsilon_i + \frac{1}{p_1}\left(\frac{p_2}{p_1}c_1 - c_2\right)\epsilon_i^2 + \frac{1}{p_1}\left[\frac{p_2}{p_1}c_2 + \left(\frac{p_3}{p_1} - \frac{p_2^2}{p_1^2}\right)c_1 - c_3\right]\epsilon_i^3 + O(\epsilon_i^4) \end{aligned}$$

Dobierając parametry a_1, a_2, α by zachować 3 rząd wielkości potrzebujemy $a_1 + a_2 = 1$ oraz $p_1 = c_1$, stąd $p_2 = c_2$, z czego otrzymujemy

$$2(a_1 + a_2 + \alpha_2)c_2 = c_2 \Rightarrow 2\alpha_2 = -1$$

Dla $\alpha = -\frac{1}{2}$ otrzymujemy ostateczny wzór dla metody Jarratta:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'\left(x_k - \frac{1}{2}\frac{f(x_k)}{f'(x_k)}\right)}$$

Warto wspomnieć, że ze względu na rząd metoda posiada efektywność $\log \sqrt[3]{3} = 0.159$, co gwarantuje większą efektywność w porównaniu z metodą Newtona, na której bazuje.

3.2 Implementacja w języku Matlab

W zależności od potrzeb projektowych, poniższa funkcja zwraca samą zmienną *count*, x, lub wektor obu tych zmiennych. Funkcja Jarrat(w, x0, tol) wymaga użycia wcześniej zdefiniowanych funkcji obliczających wartość wielomianu i jego pochodnej dla danego argumentu.

```
1   function [x, count] = Jarratt(w, x_0, tol)
2   % Funkcja Jarrat(w,x_0,tol) sluzy okresleniu miejsc zerowych ...
        podanego
3   % wielomianu przy pomocy metody Jarratt'a, jednoczesnie ...
        umozliwiajaca
4   % maksymalnie 30 iteracji tej metody, w przeciwnym wypadku ...
        liczba
5   % iteracji ustalana jest na 31.
6
7   % w: Wspolczynniki wielomianu, ktorego miejsc zerowych szukamy
8   % x_0: Poczatkowe przyblizenie
9   % tol: (Zwykle mala) liczba rzeczywista okreslajaca warunek ...
        stopu
10
11  % x: obliczone miejsce zerowe
12  % count: liczba iteracji
13  x0 = x_0;
14  x1 = x_0;
15
16  fx0 = Horner(w, x0);
17
18  % Zmienna count zlicza dotychczasowe iteracje
19  count = 0;
20
21  while (count < 30)
22
23      % W pierwszym kroku nie nadpisujemy
24      if count ≠ 0
25          x0 = x1;
26          fx0 = Horner(w, x0);
27          if abs(fx0) < tol
28              x=x0;
29              return
30          end
31      end
32
33
34      fix0 = HornerDerivative(w, x0);
35
36      if fix0 == 0
37          disp('Dzielenie przez 0!')
38          return
39      end
```

```
40
41          mianownik = HornerDerivative(w, x0 - (1/2)*fx0/fix0);
42          x1 = x0 - fx0/mianownik;
43          count = count + 1;
44      end
45
46      % W przypadku, gdy po przejsciu 30 iteracji nie spelniamy ...
            warunku stopu
47      x = NaN;
48      count = 31;
49  end
```

3.3 Prezentacja działania implementacji

Na potrzebę poniższych rozważań połóżmy wielomian

$$w(z) = z^4 - 1332z^3 + 443556z^2 + 590816592z - 393483850272$$

Łatwo można pokazać, że równoważną formą w jest

$$w(z) = (z - 666)(z + 666)(z - (666 + 666i))(z - (666 - 666i))$$

skąd natychmiast otrzymujemy wszystkie miejsca zerowe w (na mocy Zasadniczego Twierdzenia Algebry).

3.3.1 Określenie miejsc zerowych wielomianu

W celu sprawdzenia skuteczności metody Jarratta, za pomocą konkretnie ustalonych miejsc startowych sprawdzimy wyniki działania zaimplementowanego algorytmu. Przykład wyniku dla powyższego wielomianu przy $z_0 = -1000$:

```
1  >> Jarratt([1 -1332 443556 590816592 -393483850272], -1000, 0.01)
2
3  ans =
4
5    -666
```

Powtarzając czynność w celu otrzymania pozostałych miejsc zerowych (gdzie k to liczba iteracji, a z to otrzymany wynik), otrzymujemy:

	$z_0 = 1000$	$z_0 = -1000$	$z_0 = 1200 + 1000i$	$z_0 = 800 - 2000i$
z	666	-666	$666 + 666i$	$666 - 666i$
k	4	4	5	6

Użyta metoda nie tylko zwraca więc idealnie przewidziane analitycznie miejsca zerowe, lecz czyni to również w małej ilości iteracji. Sprawdźmy jednak, jak zachowywać się będzie metoda dla zupełnie losowych argumentów początkowych z $\mathbb{C}$:

	$z_0 = 21372137 - 2137.2137i$	$z_0 = 0$	$z_0 = i$	$z_0 = 10^5 i$
z	$666 - 2.5885 \times 10^{-18} i$	666	$666 - 8.4703 \times 10^{-22} i$	$666 + 666i$
k	25	1	2	14

Wartym zauważenia jest fakt, że pomimo przyjętego $\epsilon = 0.01$, odchylenie od analitycznie wyznaczonym miejsc zerowych są kilkanaście rzędów wielkości mniejsze od warunku stopu. Zaskakiwać może to, że przejście z punktu $z_0 = 0$ do $z = 666$ zajęło zaledwie jedną iterację.

3.3.2 Porównanie szybkości zbieżności

Bazując na poprzednich testach empirycznych przyjęte zostało, że dla dowolnego argumentu startowego $z_0 \in \mathbb{R}$ takiego, że $z_0 >> 0$, metoda zbieżna będzie do punktu $z = 666$. Porównanie będzie polegało więc na porównaniu rzędów wielkości punktu startowego z_0:

z_0	1×10^4	1×10^5	1×10^6	1×10^7	1×10^8	1×10^9
k	9	14	19	24	29	$30 <$

Przy okazji rzeczywiście potwierdziły się wcześniejsze przypuszczenia, każdy z przyjętych wyżej punktów startowych (za wyjątkiem największego) zbiega do punktu $z = 666$. Swój limit metoda osiągnęła przy zbieżności zaczynając od punktów większych od 1×10^8, wówczas zgodnie z wcześniejszymi założeniami, liczba iteracji dla takiego punktu ustawiana jest na 31. Sukcesywnie jednak widać liniową zależność pomiędzy rzędem wielkości punktu startowego oraz liczbą iteracji. Możemy pokusić się więc o wyprowadzenie pewnej przybliżonej zależności:

$$k(z_0 \times 10^p) = c_1(z_0) + 5p \text{ dla } p \geq 0$$

gdzie $c1$ to zmienna mówiąca ile iteracji potrzebuje metoda Jarratta do spełnienia warunku stopu dla punktu początkowego z_0

3.3.3 Ograniczenia i wady metody

Wcześniejszy problem ustalenia dokładnego miejsca zerowego nie wynikał jednak z samej metody, ale z ograniczeń wynikających z ograniczeń sprzętowych (pomimo tego, że wyliczenie pojedynczego miejsca zerowego wymagającego nawet bardzo dużej liczby iteracji nie jest kosztowne, na poczet dalszej części projektu ograniczenie iteracji uogólnione zostało do całości projektu, chcąc uniknąć dwuznaczności).
Niech teraz

$$w(z) = z^4 - 42z^3 - 4564959z^2 + 191804298z - 8265851890$$

Łatwo można pokazać, że równoważną formą w jest

$$w(z) = (z - 2137)(z + 2137)(z - (21 - 37i))(z - (21 + 37i))$$

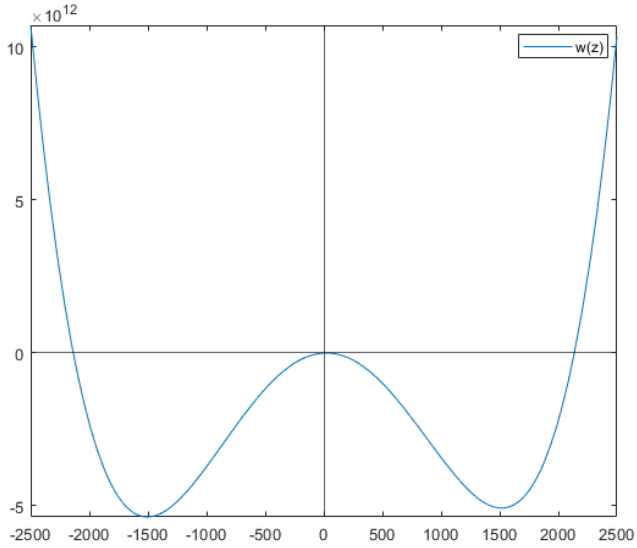
Ponownie otrzymaliśmy analitycznie wszystkie miejsca zerowe wielomianu w. Zbadajmy więc, używając wcześniej zaimplementowanego algorytmu, do którego z nich zbiegać będzie metoda Jarratta przy argumencie początkowym $z_0 = 0$:

```
>> Jarratt([1 -42 -4564959 191804298 8265851890], 0, 0.01)

ans =

   -26.4505
```

Algorytm uznał za poprawną wartość $z = -26.4505$, która nijak nie pokrywa się z żadnym z wyliczonych analitycznie miejsc zerowych. Powód może wyjaśnić się po ukazaniu wykresu funkcji $w(z)$:



Z wykresu można zauważyć, że w otoczeniu punktu $(0, 0)$ funkcja przyjmuje wartości bliskie 0, jednak nigdy go nie osiąga (wielomian stopnia 4 o współczynnikach rzeczywistych nie może mieć więcej niż 4 wyznaczone już przez nas pierwiastki). Dokładnie przez tą własność przyjętego wielomianu, algorytm uznaje punkt $z = -26.4505$ za pewne miejsce zerowe, bowiem przyjęty warunek stopu: $|w(z)| < \epsilon,\ \epsilon = 0.01$ jest spełniony przez w w punkcie z.

Zauważmy, że dla dowolnego z_k takiego, że $f'(z_k) >> f(z_k)$, iteracyjna metoda Jarratta przyjmuje postać analogiczną do metody Newtona:

$$z_{k+1} \approx z_k - \frac{f(z_k)}{f'(z_k)}$$

wtedy jednak

$$z_{k+1} \approx z_k$$

Jeżeli więc nasz punkt startowy będzie znacząco odległy od miejsca zerowego, metoda nie jest w stanie być skuteczna. Przykładem niech będzie

$$w(z) = z^{1000} + 0.00000001z^{30}$$

Różnica pomiędzy wartością pochodnej, a wartością wielomianu w punkcie $z < 1$, wynosi kilka rzędów wielkości. Możemy spodziewać się więc, że metoda ta nie będzie zbieżna.
Demonstracja dla wielomianu o analogicznej własności:

```
>> [x,k]=Jarratt([1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
     0 0 0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
     0 0 0 0 0 0 0 1/2000  0 0 0 0 0 0 0 0 1/700 1/100 1/20 1/2 0 ...
     0 0 0], 2, 0.01)

x =

   NaN


k =

    31
```

Pomimo tego, że miejscem zerowym powyższego wielomianu jest $z = 0$, metoda nie była w stanie obliczyć miejsca zerowego nawet w 30 krokach przyjmując przybliżenie początkowe odległe o 2 od miejsca zerowego.

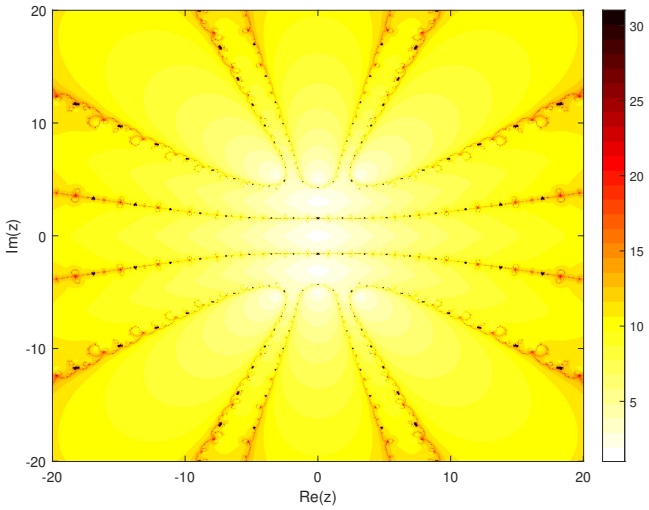
3.4 Wizualizacja zbieżności

Przykład 1

Rozpatrzmy przybliżenie funkcji $f(x) = \sin x$ za pomocą wielomianu:

$$w(z) = \frac{1}{362880}z^9 - \frac{1}{5040}z^7 + \frac{1}{120}z^5 - \frac{1}{6}z^3 + z$$

Ze względu na bliskie rozmieszczenie miejsc zerowych wielomianu, istnieją podstawy oczekiwania małej liczby iteracji potrzebnych do uzyskania pożądanego rezultatu:



Rysunek 1: Wizualizacja zbieżności dla **Przykładu 1**

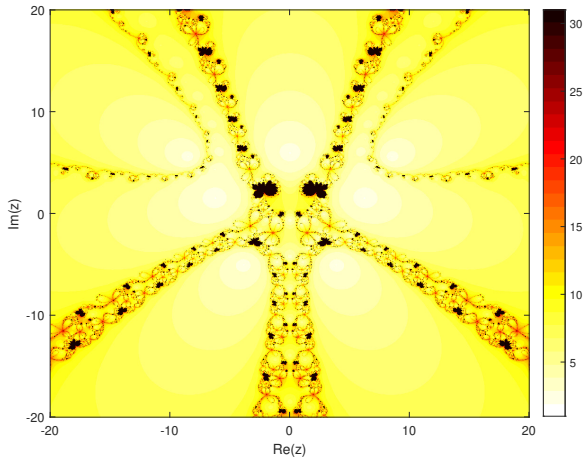
Zgodnie z oczekiwaniami, na ustalonym przedziale funkcja jest zbieżna prawie wszędzie (poza pojedynczymi przypadkami). Kwestią przewidywalną była gorsza zbieżność dla punktów najdalej oddalonych od $z = 0$. Mimo tego, nawet wśród z takich, że $\mathrm{Re}(z) = 0$, zauważyć można pojedyncze czarne ślady - w przeciwieństwie do liczb ściśle rzeczywistych.

Przykład 2

Zbadajmy, czy zbieżność metody ma bezpośredni związek ze współczynnikami wielomianu. Rozpatrzmy więc wielomian stopnia 7, którego ciąg współczynników jest ściśle rosnący:

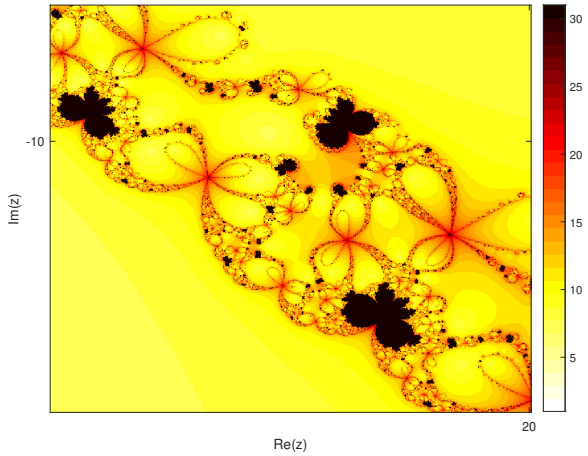
$$w(z) = \frac{1}{1000}z^7 + \frac{1}{100}z^6 + \frac{1}{10}z^5 + z^3 + 10z^2 + 100z + 1000$$

Spójrzmy jednak na wizualizację zbieżności



Rysunek 2: Wizualizacja zbieżności dla **Przykładu 2**

Przypuszczenia tym razem nie sprawdziły się, podzbiorów $\mathbb{C}$, dla których metoda przekracza limit 31 iteracji jest znacznie więcej oraz stanowią one niemałą część całej wizualizacji. Warto jednak zwrócić uwagę na symetrię zachowaną na powyższej graficznej reprezentacji zbieżności. Może to szokować głównie ze względu na fakt, że sam wielomian w nie jest funkcją symetryczną w dziedzinie rzeczywistej, parzystą, czy nieparzystą. Interesującym jest również to, że funkcja przekracza liczbę 30 iteracji dla punktów bliskich 0 pomimo, że dla dowolnego z - zera wielomianu w, $\mathrm{Re}(z) < 6$ oraz $\mathrm{Im}(z) < 9$.
Po raz pierwszy jednak ukazała się fraktalna struktura wizualizacji, którą możemy zauważyć na zbliżeniu powyższej wizualizacji:

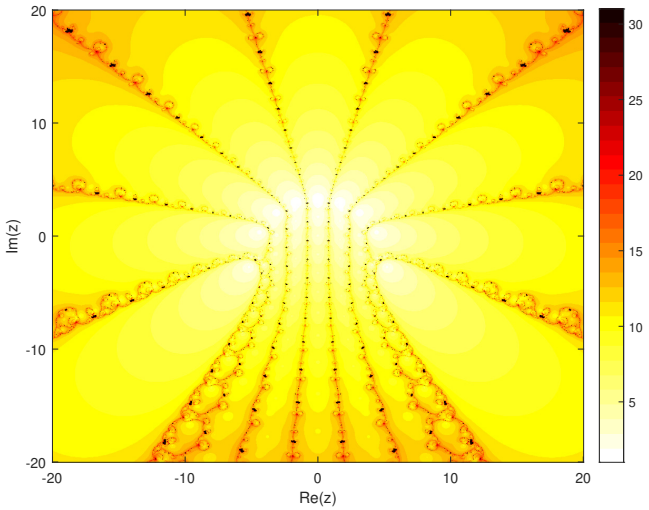


Rysunek 3: Przedstawienie fraktalnej struktury wizualizacji

Przykład 3

Rozpatrzmy teraz funkcję $f(x) = e^x$ oraz przedstawmy ją jako suma pierwszych wyrazów szeregu Maclaurina dla tej funkcji:

$$w(z) = \frac{1}{362880}z^9 + \frac{1}{40320}z^8 + \frac{1}{5040}z^7 + \frac{1}{720}z^6 + \frac{1}{120}z^5 + \frac{1}{24}z^4 + \frac{1}{6}z^3 + \frac{1}{2}z^2 + z + 1$$



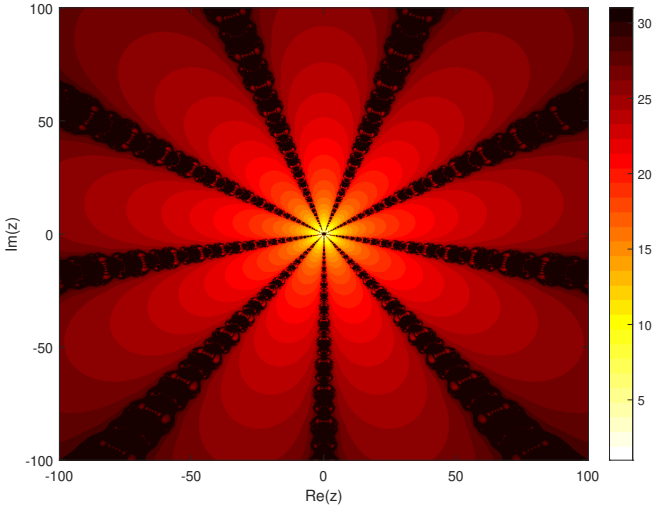
Rysunek 4: Wizualizacja zbieżności dla **Przykładu 3**

Po raz kolejny dostrzec można ścisłą symetrię względem osi urojonej. Kolejnym istotnym wnioskiem jest fakt, że stopień rozważanego wielomianu jest większy, niż w poprzednich przypadkach, a mimo to dostrzec można poprawę w zbieżności względem poprzedniego wielomianu. Po raz kolejny dostrzec można jednak pojedyncze czarne fragmenty, nietypowe dla pozostałych metod. Tym głównie charakteryzuje się metoda Jarratta - kosztem za ogólną lepszą zbieżność jest występowanie pojedynczych miejsc, w których metoda wymagała będzie znacznie więcej iteracji.

Przykład 4

Do tej pory metoda Jarratta spełniała, wręcz przekraczała oczekiwania względem szybkości zbieżności. Metoda ta nie jest pozbawiona wad, co uwydatnia wielomian:

$$w(z) = -55z^9 + 5z^8 + 5z^7 + 5z^6 + 5z^5 + 5z^4 + 5z^3 + 5z^2 + 5z - 55$$



Rysunek 5: Wizualizacja zbieżności dla **Przykładu 4**

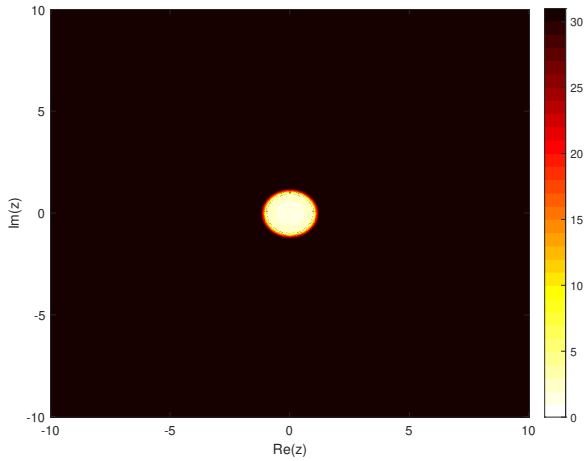
Warto zwrócić uwagę, że zwiększony został rozważany przedział, aby pokazać ograniczenia metody Jarratta. W projekcie rozważany już był wielomian, dla którego wychylenia nawet o kilka rzędów wielkości punktu startowego względem zera nie wpływały znacząco na liczbę wymaganych iteracji. Inaczej jest w przypadku rozważanego tutaj wielomianu - metoda Jarratta nie spełnia swojego zadania niemal w ogóle w przypadku punktów nieznacznie wychylonych względem zer wielomianu, natomiast stopniowo zwiększa wymaganą liczbę iteracji konsekwentnie na przestrzeni całego podzbioru. Dla z takich, że $|z| > 80$ wymagane jest już ponad 25 iteracji, co burzy teorię o niezawodności metody względem punktów startowych dużych co do modułu. Po raz pierwszy nieuwidoczniona została również przewaga liczb ściśle rzeczywistych nad zespolonymi względem szybkości zbieżności - wnioskiem może być więc to, iż reguła ta działa dla liczb znajdujących się jedynie w pewnym otoczeniu 0.

Przykład 5

Zwizualizujmy teraz szybkość zbieżności dla wielomianu rozważanego w ramach wcześniejszej części projektu:

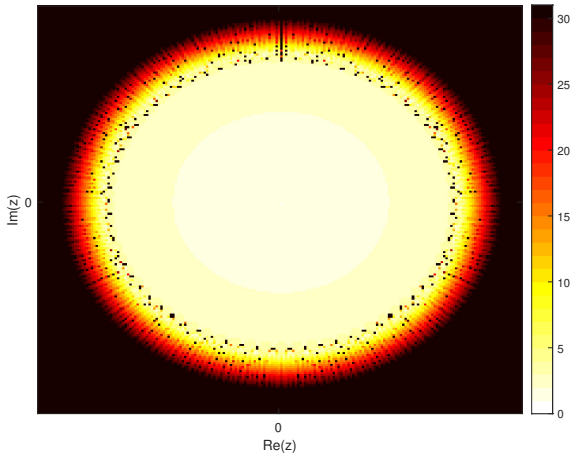
$$w(z) = z^{221} + z^{68} + \frac{1}{2000}z^{17} + \frac{1}{700}z^{7} + \frac{1}{100}z^{6} + \frac{1}{20}z^{5} + \frac{1}{2}z^{4}$$

Pamiętając ograniczeniach metody Jarratta, spodziewany jest słaby wynik zbieżności dla dowolnych z takich, że $|z| < 1$:



Rysunek 6: Wizualizacja zbieżności dla **Przykładu 5**

Wizualizacje ściśle potwierdza przypuszczenia - metoda nie jest zbieżna w ogóle z ustalonym limitem iteracji dla $|z| > 1$. Zaskakiwać może jednak to, że metoda zbiega stosunkowo szybko dla $|z| < 1$. Wynikać to może z faktu, że wielomian pozbawiony jest wyrazu wolnego, przez co miejsce zerowe $z_1 = 0$ jest bardzo szybko osiągane przez metodę. Podobne zachowania przejawiały poprzednie przykłady - metoda Jarratta bardzo sprawnie znajduje miejsca zerowe o niewielkim module. Interesującą może być obserwacja, iż przejście pomiędzy punktami początkowymi spełniającymi warunek stopu w mniej niż 31 iteracji nie jest dyskretne, co lepiej uwidocznione jest na przybliżonej wersji powyższej wizualizacji:

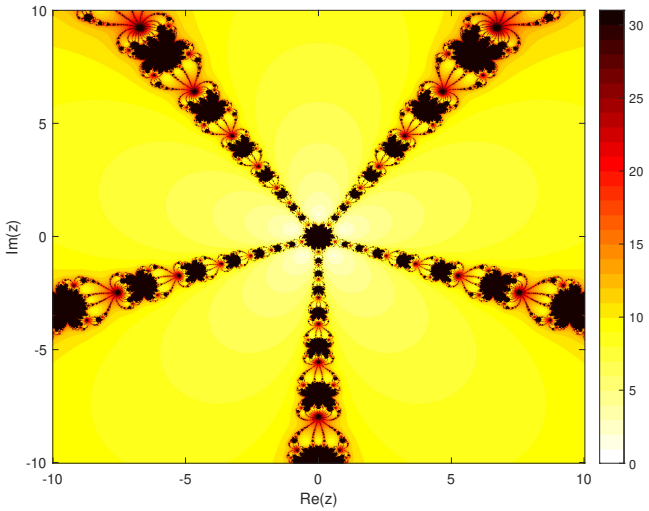


Rysunek 7: Wizualizacja niedyskretnego przejścia między wymaganymi iteracjami

Przykład 6

Na koniec, by uwidatnić zarówno wady, jak i zalety metody Jarratta posłużymy się prostym przykładem:

$$w(z) = z^5 + 1$$



Rysunek 8: Wizualizacja zbieżności dla **Przykładu 6**

Symetryczne rozmieszczenie miejsc zerowych wielomianu jest więc słabym punktem metody Jarratta. Mimo faktu, iż metoda zasadniczo dobrze spełnia swoje zadanie w w punktach z, których argument jest równy któremuś z miejsc zerowych, wzdłuż prostej wychylonej o 15 stopni od niego, metoda nie sprawdza się właściwie w ogóle. Niemal identyczne zjawisko zobrazowane zostało na większą skalę w ramach przykładu 4. Zauważyć można również sytuację zupełnie w pewnym sensie zarówno analogiczną, jak i odwrotną do poprzedniego przykładu - dla $|z| < 1$ metoda nie jest zbieżna w ogóle. Ciekawą obserwacją może być również to, iż na tym przykładzie jasno widać fraktalną strukturę czarnych pasów z przykładu 4.

3.5 Podsumowanie

Metoda Jarratta szczególnie dobrze sprawdza się, gdy posługujemy się nią w celu obliczania nierównomiernie rozłożonych pierwiastków na płaszczyźnie zespolonej. W przypadku zer rzeczywistych, w przypadku, gdy posłużymy się również rzeczywistym miejscem startowym, metoda zaledwie liniowo zwiększa liczbe iteracji w stosunku do zwiększenia rzędu wielkości miejsca początkowego. Zdecydowaną wadą metody jest szukanie zespolonych miejsc zerowych w przypadku, gdy argumentem początkowym jest wychylona od danego zera o około 15 stopni. Kolejną potencjalną wadą metody Jarratta jest sytuacja, gdy w danym argumencie wartość pochodnej wielomianu znacznie przewyższa wartość samego wielomianu. Wówczas możemy spodziewać się nawet kilkaset iteracji, by osiągnąć względnie duży warunek stopu. Cechą charakterystyczną dla metody Jarratta jest to, że w przypadku wielomianów, z którymi metoda radzi sobie najlepiej, kosztem mniejszej liczby iteracji, występują pojedyncze podzbiory $\mathbb{C}$, dla których metoda nie jest zbieżna w ogóle. Jeżeli więc jesteśmy świadomi tej cechy, korzystanie z metody Jarratta jest w stanie znacząco przyspieszyć proces obliczeń.

4 Metoda Kinga

4.1 Opis matematyczny metody

Metoda została zaproponowana przez R.F. Kinga w 1976r. Poniższy wzór wykorzystuje dwie funckje i dwie kalkulacje pochodnych, zatem złożoność wynosi $log(\sqrt[4]{5}) \approx 0.175$.

$$y_k = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = y_k - \frac{f(y_k)}{f'(y_k)} - \frac{f(x_k)}{f'(x_k)} \left(\frac{f(y_k)}{f(x_k)}\right)^3$$

4.2 Implementacja w języku Matlab

W celu implementacji iteracyjnego wzoru Kinga

$$y_k = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = y_k - \frac{f(y_k)}{f'(y_k)} - \frac{f(x_k)}{f'(x_k)} \left(\frac{f(y_k)}{f(x_k)}\right)^3$$

Musimy go najpierw przekształcić otrzymując w rezultacie:

$$y_k = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = y_k - \frac{f(y_k)f'(x_k)f(x_k)^3 + f(x_k)f'(y_k)f(y_k)^3}{f'(y_k)f'(x_k)f(x_k)^3}$$

Następnie w tej postaci poddajemy go implementacji w jezyku Matlab

```
1  function [k,x,w_x] = metodaKinga(p, x0, tol)
2
3  % Funkcja metodaHomeiera(w,x0,tol) s u y okre leniu miejsc ...
       zerowych podanego
4  % wielomianu przy pomocy metody Kinga, jednocze nie umo liwiaj ca
5  % maksymalnie 30 iteracji tej metody, w przeciwnym wypadku liczba
6  % iteracji ustalana jest na 31.
7  %Jako argumenty metoda przyjmuje:
8  % p - wspolczynniki wielomianu
9  % x0 - pocz tkowe przybli enie
10 % tol - dok adno
11 % max_iter - maksymalna i lo    iteracji
12 % Metoda zwraca:
13 % k - liczba wykonanych iteracji
14 % x - przybli enie
15 % w_x=w(x)
16
17 max_iter = 30;
```

```
18  k = 0;
19
20  while k ≤ max_iter
21      w = Horner(p,x0);
22      y0 = x0 - w/HornerDerivative(p, x0);
23      if abs(w) ≤ tol               % warunek na dok adno
24          x = x0;
25          w_x = w;
26          return
27      end
28      mianownik = HornerDerivative(p, x0)*HornerDerivative(p, ...
            y0)*(Horner(p,x0))^3;
29
30      if mianownik == 0
31          disp(' Dzielenie przez zero! ');
32          return
33      end
34
35      licznik1 = Horner(p,y0)*HornerDerivative(p, ...
            x0)*(Horner(p,x0))^3;
36      licznik2 = HornerDerivative(p, ...
            y0)*(Horner(p,y0))^3*Horner(p,x0);
37      licznik = licznik1 + licznik2;
38      dx = licznik/mianownik;
39      x1 = y0 - dx;         % obliczanie xk+1
40      k = k + 1;            % aktualizacja liczby iteracji
41      x0 = x1;              % aktualizacja xk
42  end
43
44  k = 31;
45  x = NaN;
```

4.3 Prezentacja działania implementacji

Na potrzebę poniższych rozważań połóżmy wielomian

$$w(z) = z^4 - 20z^3 + -1049900z^2 + 21012500z - 761703125$$

Łatwo można pokazać, że równoważną formą w jest

$$w(z) = (z - 1025)(z + 1025)(z - (10 + 25i))(z - (10 - 25i))$$

4.3.1 Określenie miejsc zerowych wielomianu

W celu sprawdzenia skuteczności metody Kinga, za pomocą konkretnie ustalonych miejsc startowych sprawdzimy wyniki działania zaimplementowanego algorytmu. Przykład wyniku dla powyższego wielomianu przy $z_0 = 1000$:

```
1  >> metodaKinga([1, -20, -1049900, 21012500, -761703125], 1000, 0.01)
2
3  ans =
4
5          1025
```

Powtarzając czynność w celu otrzymania pozostałych miejsc zerowych (gdzie k to liczba iteracji, a z to otrzymany wynik), otrzymujemy:

	$z_0 = 1000$	$z_0 = -1525$	$z_0 = 9 + 12i$	$z_0 = -12 + 15i$
z	1025	-1025	$10 + 25i$	$10 - 25i$
k	2	3	4	3

Użyta metoda nie tylko zwraca więc idealnie przewidziane analitycznie miejsca zerowe, lecz czyni to również w małej ilości iteracji. Sprawdźmy jednak, jak zachowywać się będzie metoda dla zupełnie losowych argumentów początkowych z $\mathbb{C}$:

	$z_0 = 2594058903 + 3i$	$z_0 = 1000$	$z_0 = 700$	$z_0 = 1900000$
z	1025	1025	1025	1025
k	28	2	22	16

Zwróćmy uwagę na fakt, że aby przejść z punktu $z_0 = 1000$ do $z = 1025$ potrzebne były jedynie 2 iteracje, jednak przejście z $z_0 = 700$ również do $z = 1025$ to aż 22 potrzebne iteracje. Zatem przesunięcie naszej liczby z_0 o jedynie $\frac{3}{10}z_0$ wywołało zwiększenie liczby iteracji o ponad 10 razy. Zaś dla $z_0 = 1900000$ czyli liczby większej od początkowego z_0 aż 1900 razy, liczba potrzebnych iteracji była dużo mniejsza.

4.3.2 Porównanie szybkości zbieżności

Bazując na testach empirycznych można przyjąć, że dla dowolnego argumentu startowego $z_0 \in \mathbb{R}$ takiego, że $z_0 >> 1025$, metoda zbieżna będzie do punktu $z = 1025$. Porównanie będzie polegało więc na porównaniu rzędów wielkości punktu startowego z_0:

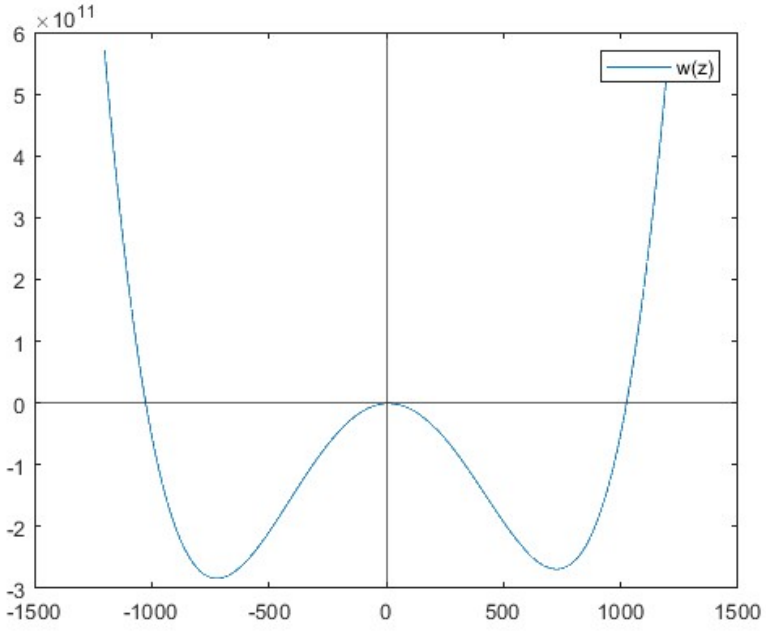
z_0	1×10^5	1×10^6	1×10^7	1×10^8	1×10^9	1×10^{10}	$1 \times 10^{10.1}$
k	11	15	18	22	26	30	30 <

Potwierdza się przypuszczenia, że każdy zprzyjętych wyżej punktów startowych z = 1025. Swój limit metoda osiągnęła przy zbieżności zaczynając od punktów większych od $10^{10.1}$, wówczas zgodnie z wcześniejszymi założeniami, liczba iteracji ustawiana jest na 31. Analizując wyniki z tabeli można zauważyć liniową zależność pomiędzy rzędem wielkości punktu startowego oraz liczbą iteracji.

4.3.3 Ograniczenia i wady metody

Wcześniejsze precyzyjne wyliczenia były spowodowane odpowiednim doborem przyliżenia początkowego, jednak nie zawsze jest to łatwe zadanie. Niech $x_0 = 700$. Przy takim wyborze przybliżenia znajdziemy miejsce zerowe = 1025 w 22 iteracjach. Popatrzmy teraz na wykres funkcji:

$$w(z) = z^4 - 20z^3 + -1049900z^2 + 21012500z - 761703125$$



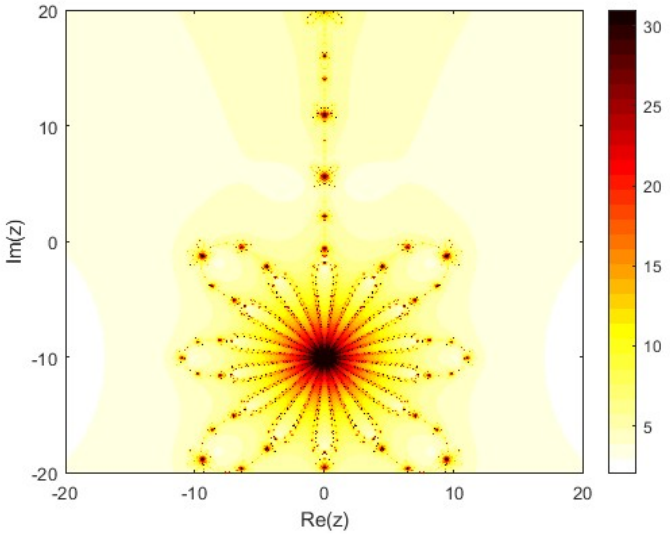
Po analize niektórych wariantów początkowego przybliżenia x_0 pojawił się ciekawy przypadek. Funkcja dla początkowego $x_0 = 9$ odnajduje przybliżenie -1025. Z kolei w 30 iteracjach nie odnajduje miejsca zerowego dla $x_0 = 10$, zaś dla $x_0 = 11$ odnajduje przybliżenie 1025. Wydaje się zatem, że dla metody Kinga problematyczny jest obszar bliski $x_0 = 10$ w przypadku omawianej funkcji. Ponadto metoda Kinga nie radzi sobie dla wyjątkowo dużych początkowych przybliżeń dla $x_0 = 10^{16}$ odnajduje pierwiastek 1025 w 54 przybliżeniach. Jendak dla $x_0 = 10^{17}$ nie jest w stanie znaleźć rozwiązania nawet dla $k = 10^9$ iteracji.

4.4 Wizualizacja zbieżności

Przykład 1

$$w(z) = z^4 - 20z^3 + -1049900z^2 + 21012500z - 761703125$$

Jest to wielomian który wykorzystaliśmy przy prezentacji działania implementacji metody i jej zbieżności



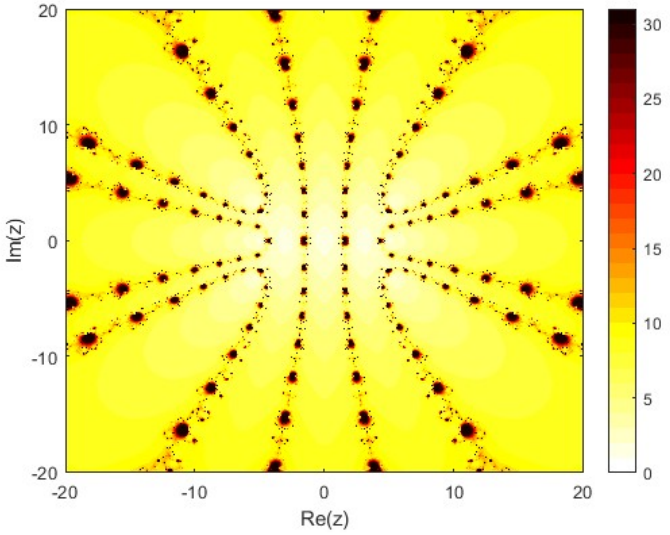
Rysunek 1: Wizualizacja zbieżności dla **Przykładu 1**

Możemy zauważyć że metoda w liczbie operacji mniejszej niż 5 zdaje się przybliżać znaczną część punktów. Dopiero anomalię jesteśmy w stanie zauważyć w okolicy punktu, Re(z) = 0 oraz Im(z) = -10. Interesującym faktem jest również symetryczność wizualizacji względem osi urojonej. Istotną obserwacją jest to, że wartość modułu punktu startowego jest wręcz odwrotnie proporcjonalny do wymaganej liczby iteracji potrzebnych do uzyskania warunku stopu. Tendencja ta nie została napotkana przy żadnej z innych metod.

Przykład 2

$$w(z) = \frac{1}{362880}z^9 + \frac{1}{5040}z^7 + \frac{1}{120}z^5 + \frac{1}{6}z^3 + z$$

Powyższy wielomian przybliża $f(x) = \sinh(x)$

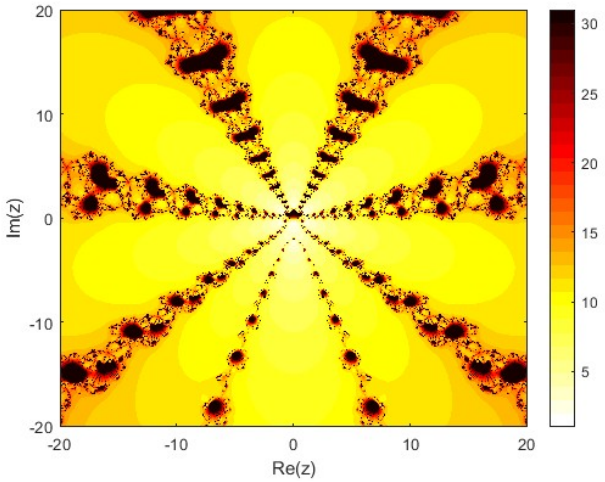


Rysunek 2: Wizualizacja zbieżności dla **Przykładu 2**

Metoda odnajduje rozwiązanie w mniej niż 10 iteracjach dla większości punktów. Najlepiej przybliża te, które znajdują się na osi $\mathrm{Re}(x) = 0$, wtedy zawsze odnajduje rozwiązanie w mniej niż 10 iteracjach i wartość ta maleje im bliżej 0. Ponadto poza 4 małymi skupiskami na osi $\mathrm{Im}(x) = 0$, funkcja również skutecznie znajduje rozwiązania. Można zauważyć symetryczność zarówno względem osi rzeczywistej, jak i osi urojonej.

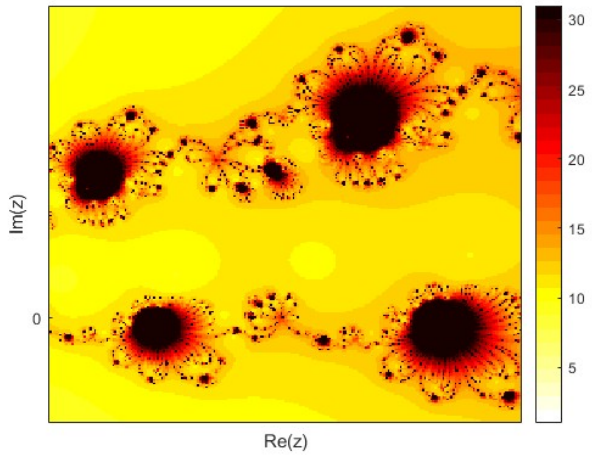
Przykład 3

$$w(z) = -6z^7 + 7z^6 + 8z^5 + 9z^4 + 10z^3 + 11z^2 + 12z - 13$$



Rysunek 3: Wizualizacja zbieżności dla **Przykładu 3**

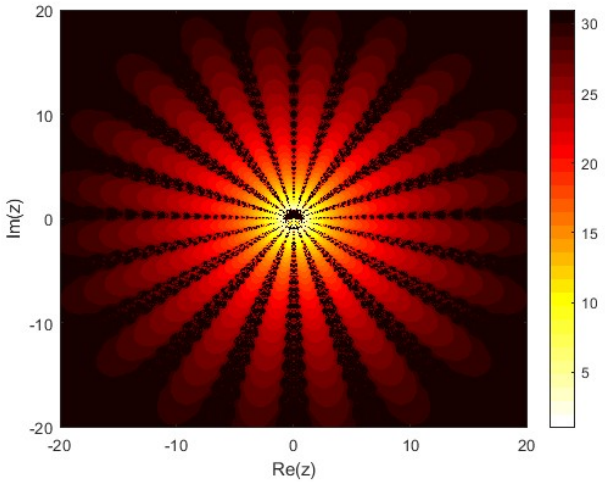
Jak możemy zauważyć metoda radzi sobie znacznie lepiej z przybliżaniem punktów, które znajdują się w dolnej części osi urojonej. Pojawia się za to więcej obszarów z którymi ma problem by odnaleźć przybliżenie. To zjawisko występuje szczgólnie dla puntków pomiędzy $\mathrm{Im}(x) < 5$ oraz $\mathrm{Im}(x) > 0$. Podobnie jak w przypatku metody Jarratta, zauważyć można fraktalną strukturę wizualizacji:



Rysunek 4: Przedstawienie fraktalnej struktury wizualizacji

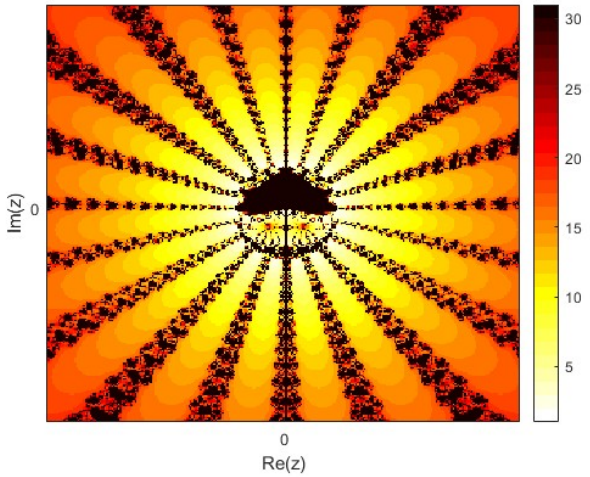
Przykład 4

$$w(z) = z^{20} + z^3 + z^2 + z + 1$$



Rysunek 5: Wizualizacja zbieżności dla **Przykładu 4**

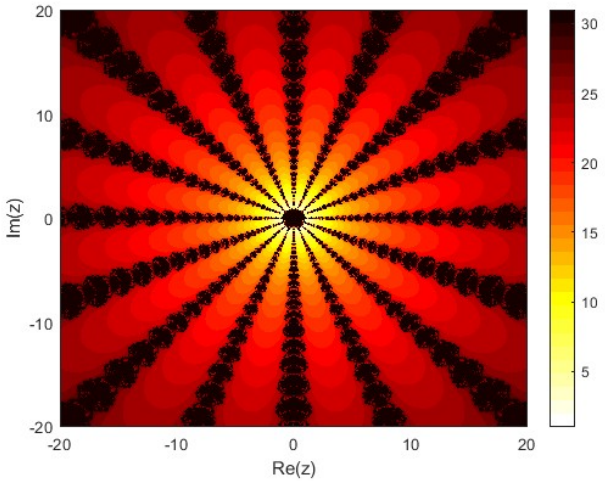
Powyższy wielomian jest przykładem na to, że metoda Kinga nie jest idealna. Możemy zauważyć że w większości punktów funkcja nie radzi sobie, i nie znajduje szukanych przybliżeń w wymaganej liczbie iteracji, a jeżeli już to robi, to dzieje się to dla liczby iteracji przekraczającej 20 i 25. Warto zwrócić uwagę, że w bliskim otoczeniu 0 funkcja również nie radzi sobie najlepiej i nawet tam napotyka problemy:



Rysunek 6: Zachowanie metody w otoczeniu 0

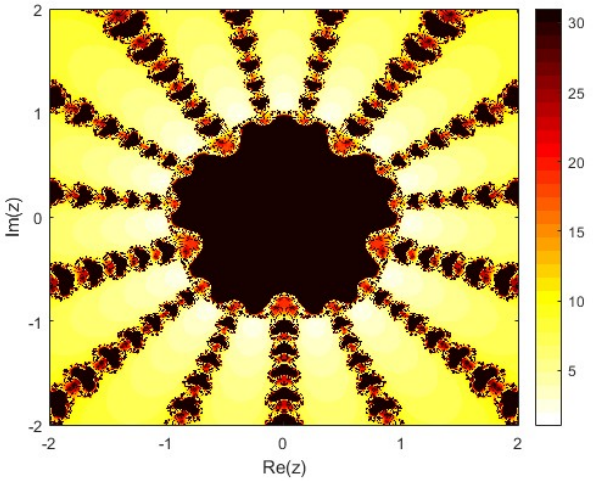
Przykład 5

$$w(z) = 20z^{15} + 15z^{10} + 10z^5 + 5z + 50$$



Rysunek 7: Wizualizacja zbieżności dla **Przykładu 5**

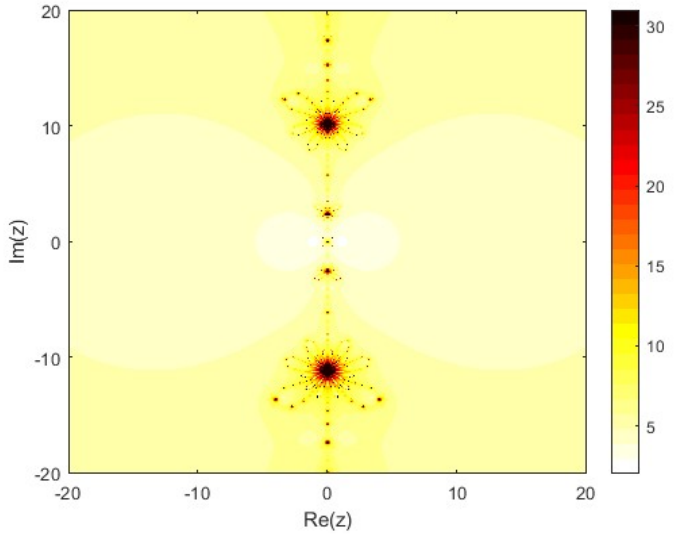
Metoda nie odnajduje przybliżenia dla punktów bliskich zeru. Warto zwrócić uwagę na to, że nie odnajduje przybliżeń dla punktów które znajdują się na osi $\mathrm{Re}(x) = 0$ oraz $\mathrm{Im}(x) = 0$. Pozostałe przybliżenia wymagały w większosci powyżej 20 iteracji. Interesującym może być, iż metoda jeszcze gorzej radzi sobie z punktami w otoczeniu 0:



Rysunek 7: Wizualizacja zachowania metody w otoczeniu 0

Przykład 6

$$w(z) = z^6 - 38z^5 - 3947838z^4 + 150332560z^3 - 32768541961z^2 + 150332598z - 32764594122$$



Rysunek 8: Wizualizacja zbieżności dla **Przykładu 6**

Powyższy wielomain posiada następujące pierwiastki: to $z_1 = -19 + 89i$, $z_2 = -i$, $z_3 = i$,$z_4 = 19 - 89i$, $z_5 = 1989$, $z_6 = -1989$ Możemy zauważyć, że metoda Kinga zadziałała dla tego wielomianu nadzwyczaj sprawnie. Praktycznie wszystkie punkty wymagały poniżej 5 iteracji, a dla osi rzeczywistej za każdym razem odnaleźliśmy rozwiązanie. Jedyne odchyły jakie wystąpiły znalazły się na osi $\mathrm{Im}(x)$ podczas gdy część rzeczywista była równa 0.

4.5 Podsumowanie

Na podstawie powyższych wizualizajci i wcześniejszych badań szybkości zbieżności, możemy wysnuć ciekawe wnioski. Metoda Kinga radzi sobie najlepiej dla wielominaów posiadających pierwiastki, które są symetryczne względem osi OX. Mogliśmy to zaobserwować na podstawie przykładu 6 oraz przykładu obrazującego implementację metody. Ponadto metoda Kinga nie radzi sobie wystarczająco efektywnie z wielomianami dużego stopnia. Odnalezienie szukanego wyniku staje się prawie niemożliwe do wykonania w 30 operacjach. Kolejną wadą tej metody jest problem z wyborem odpowiedniego przybliżenia, żeby znaleźć preferowany przez nas pierwiastek, różnica w wyborze przybliżenia początkowego x_0 nawet o 1 może spowodować wybór innego niż zakładaliśmy rozwiązania lub w ogóle do niego nie doprowadzić. Niemniej jednak metoda jest w większości przypadków skuteczna, szczególnie dla wielomianów niewielkiego stopnia, które posiadają symetrycznie rozłożone pierwiastki.

5 Metoda Homeiera

5.1 Opis matematyczny metody

Metoda została zaproponowana przez Homeiera w roku 2005 i wykorzystuje twierdzenie całkowe Newtona. Przejawia się to w zaproponowanym przez Homeira wzorze:

$$x(y) = x(y_k) + \int_{y_k}^{y} x'(t)\,dt$$

w którym całka, dzięki interpolacji reguły kwadraturowej poprzez regułe trapezów, zostala zastąpiona dając nam ostateczny wzór:

$$y_k = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = x_k - \frac{1}{2} f(x_k) \left(\frac{1}{f'(x_k)} + \frac{1}{f'(y_k)} \right)$$

Metoda jest rzędu 3 zatem jej efektywność jest równa $log(\sqrt[3]{3}) \approx 0.159$ co daje nam lepszy wynik niż w przypadku metody newtona, której efektywnośc wynosi 0.15.

5.2 Implementacja w języku Matlab

W celu implementacji interacyjnego wzoru Homeier'a

$$y_k = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = x_k - \frac{1}{2} f(x_k) \left(\frac{1}{f'(x_k)} + \frac{1}{f'(y_k)} \right)$$

Poddajemy go przekształceniu w rezultacie otrzymując:

$$y_k = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = x_k - \frac{1}{2} f(x_k) \frac{f'(y_k) + f'(x_k)}{f'(x_k) f'(y_k)}$$

W takiej postaci wzór implementujemy w języku Matlab

```
1  function [k,x,w_x] = metodaHomeiera(p, x0, tol)
2
3  % Funkcja metodaHomeiera(w,x0,tol) sluzy okresleniu miejsc ...
       zerowych podanego
4  % wielomianu przy pomocy metody Homeiera, jednoczesnie umozliwiajaca
5  % maksymalnie 30 iteracji tej metody, w przeciwnym wypadku liczba
6  % iteracji ustalana jest na 31.
7  %Jako argumenty metoda przyjmuje:
```

```
% p - wspolczynniki wielomianu
% x0 - poczatkowe przyblizenie
% tol - dokladnosc
% Metoda zwraca:
% k - liczba wykonanych iteracji
% x - przyblizenie
% w_x=w(x) - wartosc wielomianu p w punkcie na ktorym konczymy ...
    iterowac

k = 0;
max_iter = 30;

while k < max_iter
    w = Horner(p,x0);
    y0 = x0 - w/HornerDerivative(p, x0);
    if abs(w) ≤ tol               % warunek na dokladnosc
        x = x0;
        w_x = w;
        return
    end
    mianownik = 2*HornerDerivative(p, x0)*(HornerDerivative(p, y0));

    if mianownik == 0
        disp('Dzielenie przez zero');
        return
    end

    dx = (w*(HornerDerivative(p, x0)+ HornerDerivative(p, ...
        y0)))/mianownik;
    x1 = x0 - dx;         % obliczanie xk+1
    k = k + 1;            % aktualizacja liczby iteracji
    x0 = x1;              % aktualizacja xk
end

k = 31;
x = NaN;
```

5.3 Prezentacja działania implementacji

Na potrzebę poniższych rozważań połóżmy wielomian

$$w(z) = z^4 - 42z^3 - 4472559z^2 + 187875450z - 2979167850$$

Łatwo można pokazać, że równoważną formą w jest

$$w(z) = (z - 2115)(z + 2115)(z - (21 + 15i))(z - (21 - 15i))$$

5.3.1 Określenie miejsc zerowych wielomianu

W celu sprawdzenia skuteczności metody Homeiera, za pomocą konkretnie ustalonych miejsc startowych sprawdzimy wyniki działania zaimplementowanego algorytmu. Przykład wyniku dla powyższego wielomianu przy $z0 = 1945$:

```
>> metodaHomeiera([1,-42,-4472559,187875450,-2979167850], 1945, ...
    0.01)

ans =

        2115
```

Powtarzając czynność w celu otrzymania pozostałych miejsc zerowych (gdzie k to liczba iteracji, a z to otrzymany wynik), otrzymujemy:

	$z_0 = 1945$	$z_0 = -1525$	$z_0 = 14 + 10i$	$z_0 = -9 - 72i$
z	2115	-2115	$21 + 15i$	$21 - 15i$
k	3	6	3	4

Użyta metoda nie tylko zwraca więc idealnie przewidziane analitycznie miejsca zerowe, lecz czyni to również w małej ilości iteracji. Sprawdźmy jednak, jak zachowywać się będzie metoda dla zupełnie losowych argumentów początkowych z $\mathbb{C}$:

	$z_0 = 123456789 + i$	$z_0 = 1500$	$z_0 = 1600$	$z_0 = 100$
z	$2.1150e + 03 - 1.9259e - 34i$	2115	2115	2115
k	25	13	5	12

Zauważmy, że przejscie z punktu $z_0 = 1600$ do $z = 2115$ zajęło 5 iteracji natomiast z $z_0 = 1500$ do tego samego z aż 13 iteracji. Oznacza to, że przesunięcie punktu startowego o $\frac{1}{15}z_0$ w prawo spowodowało znaczne zwiększenie liczby wymaganych iteracji. Z kolei dla $z_0 = 100$ zbiegającego do tego samego miejsca zerowego potrzebowaliśmy 12 iteracji, czyli o jedną mniej niż dla $z_0 = 1500$.

5.3.2 Porównanie szybkości zbieżności

Bazując na testach empirycznych można przyjąć, że dla dowolnego argumentu startowego $z_0 \in \mathbb{R}$ takiego, że $z_0 >> 2115$, metoda zbieżna będzie do punktu $z = 2115$. Porównanie będzie polegało więc na porównaniu rzędów wielkości punktu startowego z_0:

z_0	1×10^4	1×10^5	1×10^6	1×10^7	1×10^8	1×10^9	$1 \times 10^{9.8}$
k	6	11	15	19	23	27	$30 <$

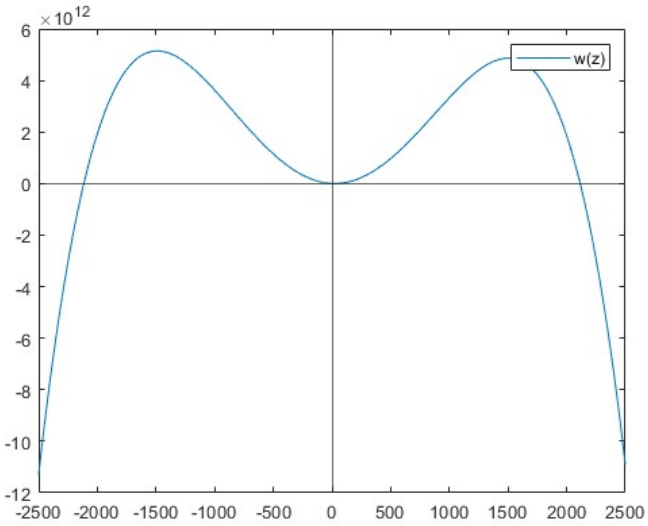
Potwierdza się przypuszczenia, że każdy zprzyjętych wyżej punktów startowych z = 2115. Swój limit metoda osiągnęła przy zbieżności zaczynając od punktów większych od $10^{9.8}$, wówczas zgodnie z wcześniejszymi założeniami, liczba iteracji ustawiana jest na 31. Analizując wyniki z tabeli można zauważyć liniową zależność pomiędzy rzędem wielkości punktu startowego oraz liczbą iteracji.

5.3.3 Ograniczenia i wady metody

Wcześniejsze precyzyjne wyliczenia były spowodowane odpowiednim doborem przyliżenia początkowego, lecz nie zawsze jest to łatwym zadaniem. Niech $x_0 =$

100. Przy takim wyborze znajdziemy miejsce zerowe = 2115 w 12 iteracjach. Spójrzmy teraz na wykres funkcji:

$$w(z) = -z^4 + 42z^3 + 4472559z^2 - 187875450z + 2979167850$$



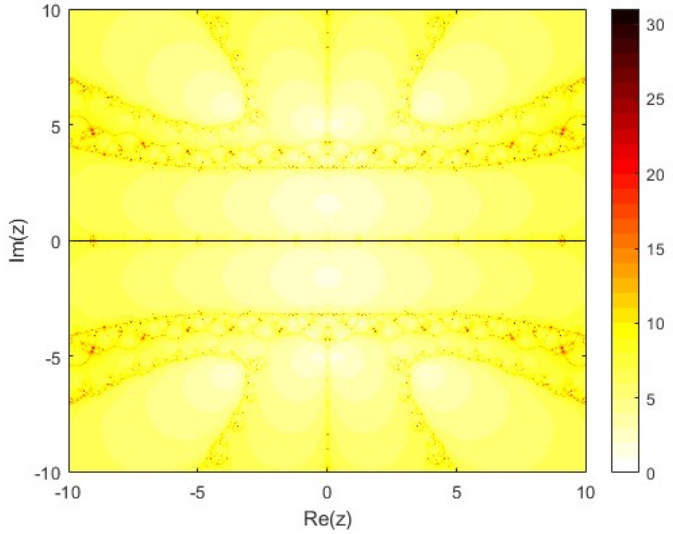
Spodziewalibyśmy się, że wybierając $x_0 > 10$ również znajdziemy to samo miejsce zerowe w mniejszej bądź równej liczbie operacji. Tak jednak się nie dzieje, gdyż dla $x_0 = 1200$ liczba iteracji jest co prawda mniejsza i wynosi 4,lecz znalezione miejsce zerowe jest inne i wynosi -2115. Gdy z x_0 przesuniemy się na rok Bitwy pod Grunwaldem to dla $x_0 = 1410$ metoda nie jest zbieżna w 30 iteracjach. Aby odnaleźć pierwiastek 2115 potrzebujemy aż 123 iteracji.

5.4 Wizualizacja zbieżności

Przykład 1

$$w(z) = -\frac{1}{3628800}z^{10} + \frac{1}{40320}z^8 - \frac{1}{720}z^6 + \frac{1}{24}z^4 - \frac{1}{2}z^2 + 1$$

Jest to wielomian, który przybliża $f(x) = \cos(x)$



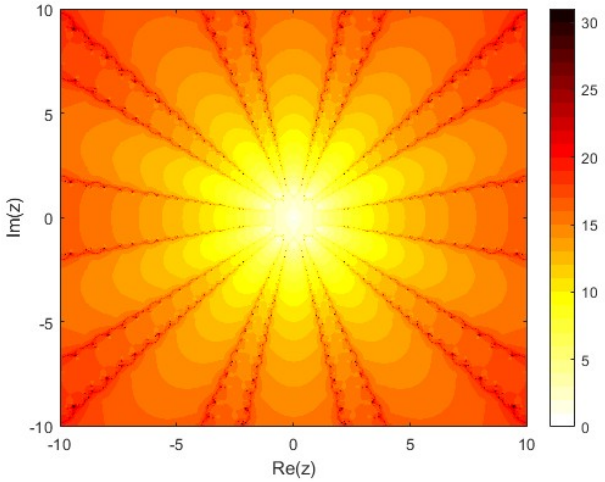
Rysunek 1: Wizualizacja zbieżności dla **Przykładu 1**

Można zauważyć, że metoda Homeiera dobrze sobie radzi z przybliżaniem $w(z)$. Dla liczb zespolonych w badanym przedziale potrzebuje od 5 do 15 iteracji. Zdecydowane trudności pojawiają się jednak dla $z \in \mathbb{R}$. Dla takich z metoda nie radzi sobie w wymaganych 30 iteracjach.

Przykład 2

$$w(z) = -\frac{1}{6}z^{12} + \frac{1}{5}z^{10} - \frac{1}{4}z^8 + \frac{1}{3}z^6 - \frac{1}{2}z^4 + z^2$$

Jest to wielomian, który przybliża $f(x) = \ln(x)$

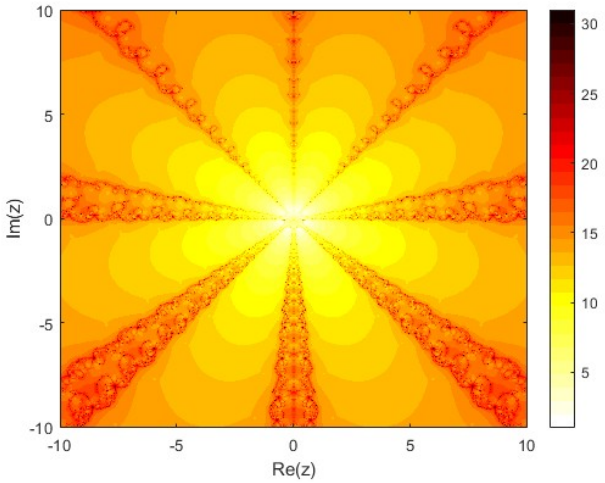


Rysunek 2: Wizualizacja zbieżności dla **Przykładu 2**

Dzięki wizualizacji możemy zauważyć, że metoda Homeiera dobrze sobie radzi z przybliżaniem $w(z)$ w punktach w okolicy $z = 0$. Natomiast im bardziej oddalamy się od 0 tym więcej iteracji potrzebujemy.

Przykład 3

$$w(z) = 420z^9 - 4z^8 + 2z^7 + 4z^5 - 2z^4 + 4z + 2$$

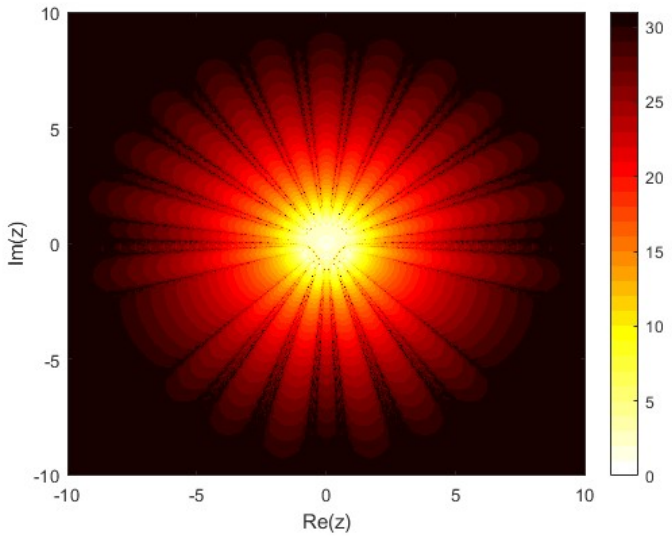


Rysunek 3: Wizualizacja zbieżności dla **Przykładu 3**

Dla punktów w okolicy $z = 0$ metoda potrzebuje co najwyżej kilka iteracji. Na badanym podzbiorze $\mathbb{C}$ metoda Homeiera wymaga nię więcej niż 30 iteracji. Najgorzej wypadają punkty $z = a+bi$, gdzie $a \approx b$ oraz a lub b równają się zero.

Przykład 4

$$w(z) = z^{24} + z^{21} + z^{20} + z^{17} + z^{16} + z^{13} + z^{12} + z^9 + z^8 + z^5 + z^4 + z$$

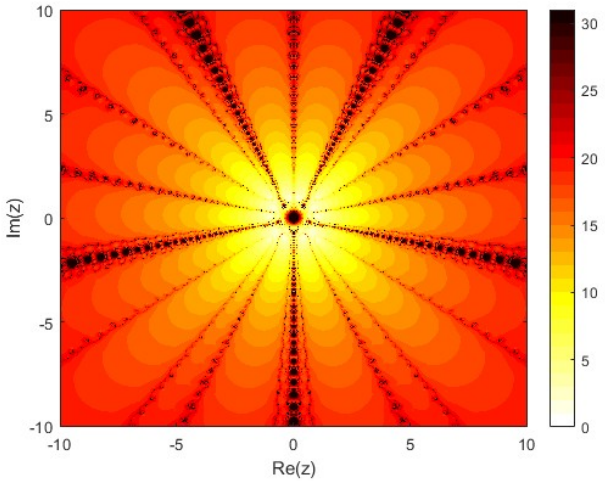


Rysunek 4: Wizualizacja zbieżności dla **Przykładu 4**

W tym przpadku wyraźnie zauważalny jest promień, który wyznacza nam okrąg w którym rozpatrywane punkty wymagają mało iteracji. Dla punktów w których $|z| > r$ metoda potrzebuje więcej iteracji. Dosyć szybko dochodzimy do przypadku w którym liczba iteracji przekracza 30.

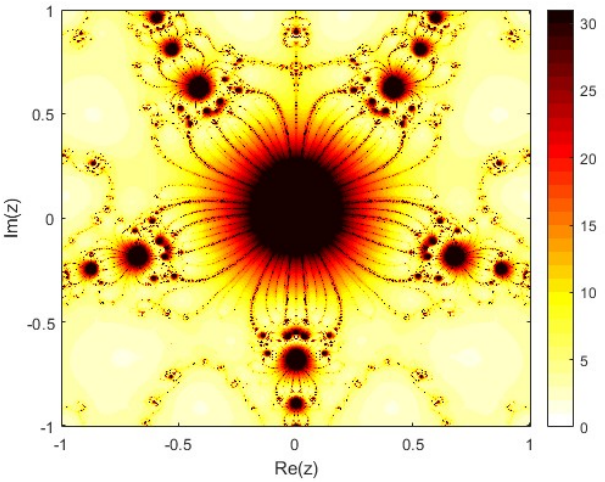
Przykład 5

$$w(z) = z^{14} + 4z^{10} + z^9 + 4z^5 + z^4 + 4$$



Rysunek 5: Wizualizacja zbieżności dla **Przykładu 5**

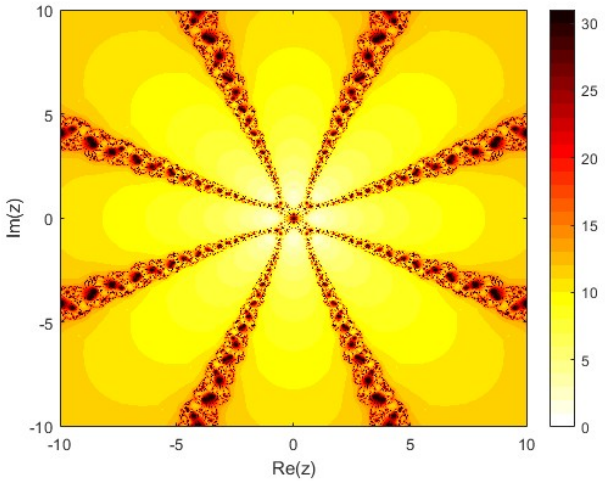
Ciekawy przykład w którym metoda nie sprawuje się dobrze w bliskim otoczeniu 0. Gdy oddalimy się delikatnie od 0 to metoda zaczyna dobrze działać, lecz dalsze przesunięcie punktu początkowego ponownie zwiększy liczbę iteracji. Przyjrzyjmy się fraktalnej strukturze wizualizacji w otoczeniu 0:



Rysunek 6: Zachowanie metody Homeiera w otoczeniu 0

Przykład 6

$$w(z) = z^8 + 3z^4 - 4$$



Rysunek 7: Wizualizacja zbieżności dla **Przykładu 6**

Pierwiastkami tego wielomianu są: $z_1 = 1 + i$, $z_2 = -1 - i$, $z_3 = -1 + i$, $z_4 = -i$, $z_5 = i$,$z_6 = 1 - i$, $z_7 = 1$, $z_8 = -1$. Są one równo rozłożone na kole jednostkowym, dzięki czemu rysunek zbieżnosci jest symetryczny i podzielony na tyle sekcji dobrego zbiegania ile mamy pierwiastków.

5.5 Podsumowanie

Metoda Homeiera świetnie sobie radzi z przybliżaniem wielomianów o pierwiastkach równo rozłożonych na kole jednostkowym. Przy wyborze odpowiedniowego rzeczywistego punktu początkowego jest wyjątkowo szybka, metoda zaledwie liniowo zwiększa liczbe iteracji w stosunku do zwiększenia rzędu wielkości miejsca początkowego. Niewątpliwie wadą metody homeira jest trudność z wyborem odpowiedniego punktu początkowego. Wybrany punkt pomimo, że znajduje sie blisko szukanego pierwiastka może zbiegać do pierwiastka zupełnie innego bądź nie zbiegać wcale. Delikatne przesunięcie zdecydowanie zwiększyć liczbę potrzbnych iteracji. Jednakże jeśli jesteśmy świadomi tych wad to przy rozważnym wyborze punktu startowego jesteśmy w stanie znajdować pierwiastki w zadawalającym tempie.

6 Dodatek

6.1 Generowanie macierzy o równoodległych współczynnikach zespolonych

W tym celu zdefiniujemy funkcję GenerateMatrix, która na podstawie krańców dwóch przedziałów liczb rzeczywistych podzieli je na odpowiednio n oraz m części i odpowiednio doda do macierzy wynikowej.

Implementacja funkcji GenerateMatrix w języku Matlab:

```
function [A] = GenerateMatrix(a,b,c,d,n,m)

    % Funkcja GenerateMatrix(a,b,c,d,n,m) to funkcja generujaca
    % macierz A o wspolczynnikach zespolonych, przy czym kazda ...
        kolumna
    % oraz kazdy wiersz rozni sie o dokladnie ta sama wartosc ...
        wzgledem
    % poprzednika czesci zarowno rzeczywistej, jak i urojonej.
    % Roznica ta okreslana jest przez srednia arytmetyczna ...
        ustalonych
    % przez uzytkownika parametrow:

    % a,b: Krance dyskretnego podzialu wartosci czesci urojonej
    % c,d: Krance dyskretnego podzialu wartosci czesci rzeczywistej
    % n,m: Ustalaja podzial roznicy pomiedzy kolejnymi wartosciami,
    %      jednoczesnie wyznaczajac rozmiar macierzy (z dokladnoscia
    %      do pierwszego wyrazu)

    h1 = (b-a)/n;
    h2 = (d-c)/m;

    y = c:h2:d;

    A = zeros(n+1, m+1);

    % Zamiana podanej wartosci rzeczywistej na urojona
    for k = 1:m+1
        y(k) = y(k) * 1i;
    end

    % Dodanie czesci urojonej do macierzy
    for k = 1:n+1
        A(k, :) = y;
    end

    % Dodanie czesci rzeczywistej do macierzy
    for k = 1:m+1
        x = reshape(a:h1:b, n+1, 1);
        A(:, k) = reshape(A(:,k),n+1,1) + x;
    end
end
```

6.2 Tworzenie macierzy iteracji

Funkcja generująca macierz iteracji - ResultMatrix - różnić będzie się dla każdej z trzech przedstawionych w projekcie metod.

Implementacja funkcji w języku Matlab (na przykładzie metody Jarratta):

```
function [I] = ResultMatrix(A, p, tol)

    % Funkcja ResultMatrix(A, p, tol) sluzy zamianie wartosci
    % wspolczynnikow podanej macierzy A na liczbe iteracji wymagana
    % do uzyskania warunku stopu o podanym parametrze przyjmujac ...
        za punkt
    % startowy wartosc wspolczynnika w danej komorce, poslugujac sie
    % wczesniej okreslona funkcja Jarratt(w, x_0, tol)

    % A: Macierz wspolczynnikow (byc moze zespolonych)
    % p: Wektor wspolczynnikow wielomianu, ktorego miejsc zerowych
    %    poszukujemy
    % tol: (Zwykle mala) liczba rzeczywista okreslajaca warunek ...
        stopu

    % Deklaracja macierzy wynikowej
    I = zeros(size(A,1), size(A,2));

    % Dla kazdej komorki macierzy wywolujemy funkcje Jarratt() z ...
        punktem
    % poczatkowym rownym wartosci wspolczynnika
    for k = 1:size(A,1)
        for j = 1:size(A,2)
            i = Jarratt(p, A(k,j), tol);
            I(k, j) = i;
        end
    end
end
```

6.3 Graficzne przedstawienie wyniku

W tym celu zdefiniowana została funkcja Demonstrate, która wymaga implementacji dwóch poprzednio zdefiniowanych funkcji: ResultMatrix oraz GenerateMatrix. Parametr rozmiaru macierzy wynikowej ustawiony został na 1500. Pozostałe parametry funkcji potrzebne są jedynie do wywoływania dwóch pozostałych, samo Demonstrate nie zwraca bowiem żadnej wartości - jej zadaniem jest zwrócenie graficznego przedstawienia macierzy.
Implementacja funkcji Demonstrate w języku Matlab:

```
function [] = Demonstrate(a,b,c,d,p)
    % Funkcja Demonstrate(a,b,c,d,p) sluzy wygenerowaniu graficznej
    % reprezentacji szybkosci zbieznosci metody Jarratt'a poprzez
    % okreslenie kolorow odpowiadajacych wymaganym liczbom iteracji
    % potrzebnych do uzyskania warunku stopu.

    % W tym celu funkcja wykorzystuje wczesniej zdefiniowana funkcje
    % ResultMatrix(A, p, tol) oraz GenerateMatrix(a,b,c,d,n,m).

    % Argumenty przekazywane funkcji GenerateMatrix(a,b,c,d,n,m):
    % a,b: Krance dyskretnego podzialu wartosci czesci urojonej
    % c,d: Krance dyskretnego podzialu wartosci czesci rzeczywistej

    % Argumenty przekazywane funkcji ResultMatrix(A, p, tol):
    % p: Wektor wspolczynnikow wielomianu, ktorego miejsc ...
        zerowych szukamy

    % Macierz A generujemy funkcja GenerateMatrix
    size = 1500;
    I = ResultMatrix(GenerateMatrix(a,b,c,d,size,size), p, 0.01);

    % Recznie ustawiamy kolorystyke
    map = hot(31);

    figure

    imagesc(I);

    % Niech miejsca poczatkowe wymagajace najwiecej iteracji beda
    % najciemniejsze
    colormap(flipud(map))
    colorbar

    % Nazwy osi zmieniamy w ten sposob, by odpowiadaly wybranym ...
        przez
    % uzytkownika przedzialom
    xticks([1 size/4 size/2 size*3/4  size-1])
    xticklabels({a, -abs(b-a)/4,a+b,abs(b-a)/4,b})
    yticks([1 size/4 size/2 size*3/4  size-1])
    yticklabels({d, abs(d-c)/4,c+d, -abs(d-c)/4,c})

    xlabel('Re(z)')
    ylabel('Im(z)')
end
```