Raport z projektu 1

Damian Skowroński

4 grudnia 2021

1 Wprowadzenie

W projekcie miałem zaimplementować w programie Matlab wyznaczanie zer wielomianu $w(z) = \sum_{k=0}^{n} a_k z^k$ w dziedzinie zespolonej używając metody Jarratta, a także zwizualizować szybkość zbieżności tej metody. W celu obliczenia potrzebnych w metodzie wartości wielomianu i jego pochodnych zastosowałem algorytm Hornera.

1.1 Metoda Jaratta

Metoda Jarratta używana jest w celu wyznaczania jednego pierwiastka równania $f(x) = 0$, gdzie f jest funkcją jednej zmiennej. Niech $x_0 \in \mathbb{C}$ będzie danym przybliżeniem początkowym. Kolejne przybliżenia $x_1, x_2, x_3, \ldots$ wyznacza się za pomocą wzoru iteracyjnego:

$$x_{k+1} = x_k - \frac{f(x_{k)}}{f'(x_k - \frac{1}{2}\frac{f(x_k)}{f'(x_k)})}, \qquad k = 0, 1, \ldots$$

1.2 Algorytm Hornera

Dla wielomianu f zapisanego jako:

$$f(x) = a_0 + x(a_1 + x(\ldots + x(a_{n-2} + x(a_{n-1} + xa_{n)}) \ldots))$$

i argumentu $\widehat{x}$ będziemy liczyć $f(\widehat{x})$ i $f'(\widehat{x})$. Oznaczamy $w_n = a_n$, oraz $p'_n = w_n$ i obliczamy po kolei $w_{n-1} = a_{n-1} + \widehat{x}w_n$, $w_{n-2} = a_{n-2} + \widehat{x}w_{n-1}$ itd., a także $p'_{n-1}(\widehat{x}) = w_{n-1} + \widehat{x}p'_n(\widehat{x})$, $p'_{n-2}(\widehat{x}) = w_{n-2} + \widehat{x}p'_{n-1}(\widehat{x})$ itd. Ostatecznymi wartościami będą $w_0 = f(\widehat{x})$, oraz $w'_1(\widehat{x}) = f'(\widehat{x})$.

2 Implementacja w Matlabie

W celu implementacji zadania stworzyłem trzy funkcje:

1. `Horner`
2. `Jarratt`
3. `visualise`
4. `generate_coeffs`

Warto zaznaczyć, że zamieszczonych w tym raporcie fragmentach kodu brakuje komentarzy, które są w rzeczywistych plikach funkcji, w których kod jest lepiej wytłumaczony.

2.1 Funckcja Horner

Przyjmuje argument $\widehat{x} \in \mathbb{C}$ jako `variable` oraz współczynniki wielomianu `coefficients` w wektorze, gdzie współczynnik a_i stojący przy x o potędze i znajduje się na i-tym miejscu wektora $(i = 0, 1, ..., n)$. Implementuje wcześniej opisany algorytm Hornera. Zwraca wartość `value`, oraz pochodną `derivative` wielomianu w podanym argumencie.

```
function [value,derivative] = Horner(coefficients,variable)
n = length(coefficients);
w = coefficients(n);
p = w;

for k = (n-1):-1:2
    w = coefficients(k) + variable.*w;
    p = w + variable.*p;
end

w = coefficients(1) + variable.*w;
value = w;
derivative = p;

end
```

2.2 Funkcja Jarratt

Funkcja przyjmuje wcześniej zdefiniowana funkcję `Horner` jako `fun_horner`, współczynniki wielomianu jako `coefficient`, wektor argumentów startowych `x_start`, oraz wielkość dopuszczalengo błędu `error`. Implementuje wcześniej opisaną metodę Jarratta i zwraca `x` oraz `steps`, którymi są odpowiednio informacje o tym, do którego pierwiastka udało się dojść i ile iteracji było na to potrzebne.

```
function [x,steps] = Jarratt(fun_horner,coeficient,x_start,error)
if nargin <4
    error = 10^(-12);
end
value = inf;
n = length(x_start);
x = x_start;
steps = Inf(1,n);

step_counter = 0;
while any(abs(value)>error)
    [value, derivative] = fun_horner(coeficient,x);
    [value2, derivative2] = fun_horner(coeficient,x - value./(2.*derivative));
    x = x - value./derivative2;
    steps((abs(value)<error) & (steps > step_counter)) = step_counter;
    step_counter = step_counter + 1;
    if step_counter > 30
        steps(steps > step_counter)= step_counter;
        break
    end
end
x;
steps;
end
```

2.3 Funkcja visualise

Funkcja wyświetla wyniki z funkcji Jarratta dla podanych współczynników `coeffs` wielomianu. Wizualizacje podane potem w przykłądach są wynikiem wywołania tej funkcji.

```
function [] = visualise(coeffs)
%dimensions of meshgrid:
n = 1000; m = 1000; a = -10; b = 10; c = -10; d = 10;
[X,Y] = meshgrid(linspace(a,b,n+1),linspace(c,d,m+1));
test = complex(X,Y);
A = zeros(n+1,m+1);
B = A;
C = A;
error = 10^(-13);
%finding roots
for k = 1:(n+1)
    x_start = test(k,:);
    [A(k,:),B(k,:)] = Jarratt(@Horner,coeffs,x_start,error);
end
%things that have to do with both legends
A = round(A,2) + complex(0);
compare = round(roots(flip(coeffs)),2);
A(~ismember(A,compare)) = max(real(A(:))) + 1 ;
uniA = unique(A)
tmp = size(uniA)
numberOfColors = tmp(1)
for k = 1:numberOfColors
    C(A == uniA(k)) = k - 0.5;
end
%legend lebels
leb1 = string(uniA)
leb1(end) = "No root found"
leb2 = string(5:5:30)
leb2(end + 1) = "No root found"
%visualising the outcome
figure
ax(1) = subplot(1,2,1) %which root
imagesc([a b], [c,d],C);
set(gca,'YDir','normal');
cmap1 = lines(numberOfColors)
cmap1(end,:) = zeros(3,1)
colormap(ax(1),cmap1)
colorbar("Ticks",unique(C),"TickLabels",leb1)
caxis(gca,[0, numberOfColors])
xlabel("Real axis")
ylabel("Imaginary axis")
ax(2) = subplot(1,2,2)  %how many steps
imagesc([a b], [c,d],B);
set(gca,'YDir','normal');
cmap2 = turbo(35);
cmap2(31:35,:)= zeros(5,3);
colormap(ax(2),cmap2)
caxis(gca,[0 35])
colorbar("Ticks",[4.5:5:34.5],"TickLabels",leb2)
xlabel("Real axis")
ylabel("Imaginary axis")
end
```

2.4 Funkcja generate_coeffs

Funkcja generuje i zwraca współczynniki `coeffs` w dziedzinie zespolonej wielomianu danego stopnia `degree`. Funkcja przydała się do szukania ciekawie wyglądających wizualizacji.

```
function [coeffs] = generate_coeffs(deg)
minim = -10; maxim = 10 %range
real_part = (abs(minim) + abs(maxim))*rand( 1, deg ,'double') - abs(minim);
imaginary_part = (abs(minim) + abs(maxim))*rand( 1, deg ,'double') - abs(minim);
coeffs = complex(round(real_part,2),round(imaginary_part,2);
end
```

3 Przykłady

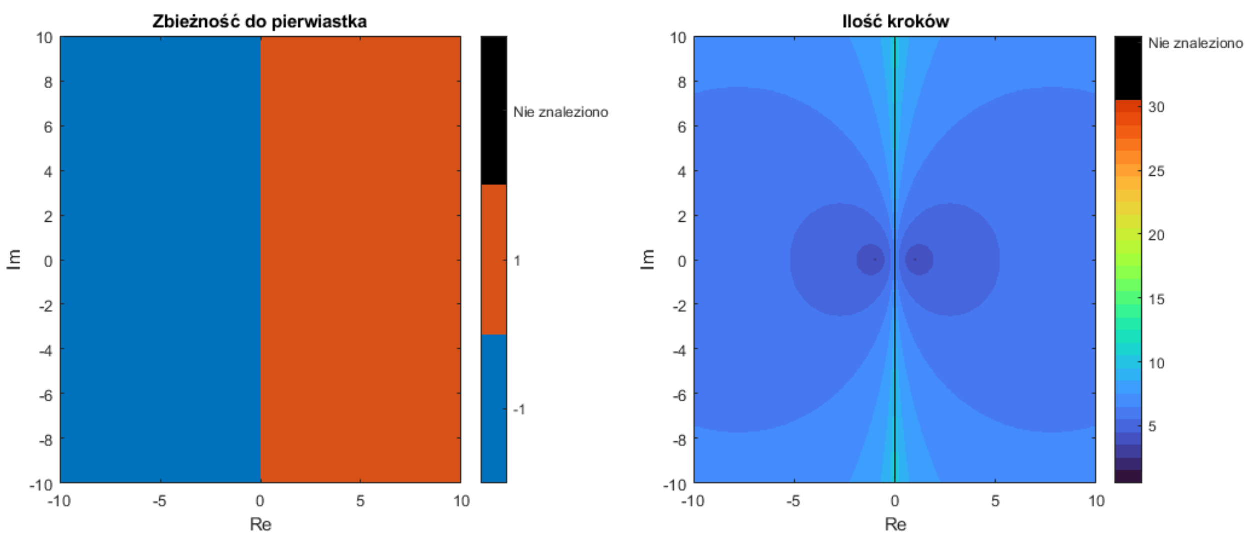
W tej sekcji przedstawię kilka przykładów działania programu. Zacznę od od mniej skomplikowanych wielomianów z wytłumaczeniem co widać na wizualizacjach, a zakończę na ciekawiej wyglądajych wizualizcjach wielomianów, o wygenerowanych przez funkcję `generate_coeffs` współczynnikach.

3.1 Własne przykłady

Przykłady, których współczynniki wybierałem ręcznie, w celu pokazania czegoś konkretnego.

3.1.1 Przykład 1

Na początek biorę prosty wielomian $z^2 - 1$. Spodziewamy się oczywiście dwóch pierwiastków: $z_1 = -1$, oraz $z_2 = 1$



Otrzymaliśmy takie pierwiastki jakich się spodziewaliśmy. Jak widać program miał problem w znalezieniu pierwiastka dla tych $x_0 \in \mathbb{C}$, które miały $Re(x_0) = 0$. Można policzyć dlaczego tak się dzieje.
Dla $x_0 = 0$ w metodzie Jarratta nie istnieje x_1, bo występuje dzielenie przez zero:

$$x_1 = 0 + \frac{-1}{2(0 - \frac{-1}{2*2*0})}$$

W pozostałych x_0, dla których nie znaleziono pierwiastka występuje jakiegoś rodzaju nieskończona pętla (przynajmniej tak podejrzewam, bo nawet jak zwiększałem maksymalną liczbę kroków do miliona to dla żadnej z tych wartości nie znalazło pierwiastka). Na przykład dla $x_0 = i$, będzie:

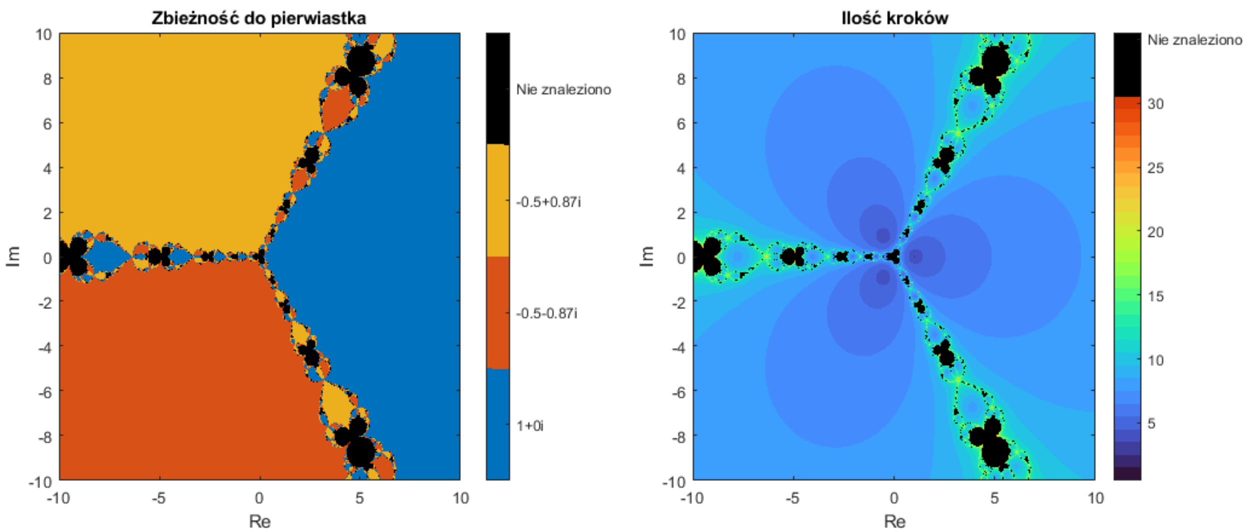
$$x_1 = i - \frac{-2}{2(i - \frac{-2}{2*2i})} = i + \frac{1}{i + \frac{1}{2i}} = i + \frac{2i}{2i^2+1} = i + \frac{2i}{-1} = -i$$

$$x_2 = -i - \frac{-2}{2(-i - \frac{-2}{-2*2i})} = -i + \frac{1}{-i - \frac{1}{2i}} = -i + \frac{2i}{-2i^2-1} = -i + 2i = i \quad ,$$

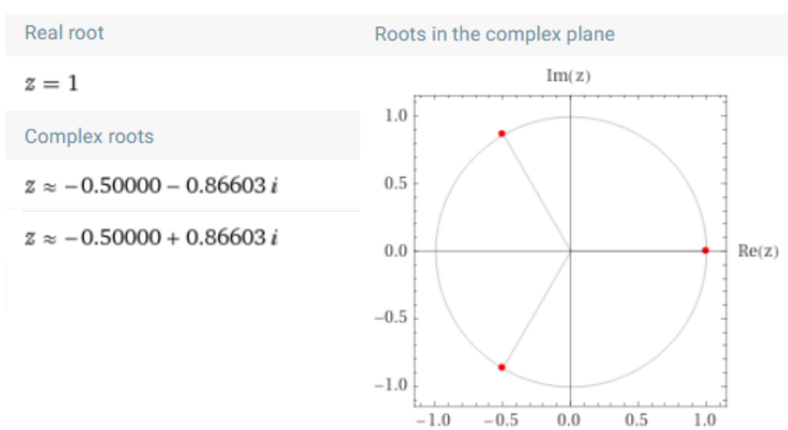
więc jest tu powrót do x_0.

3.1.2 Przykład 2

W przykładzie drugim wielomianem jest $z^3 - 1$. Wielomian ten jest podobny do poprzedniego, z przykładu 1, jednak zwraca już zupełnie inne, ciekawsze wyniki.



Można porównać pierwiastki, do których doszedł program z wynikami w WolframAlpha.

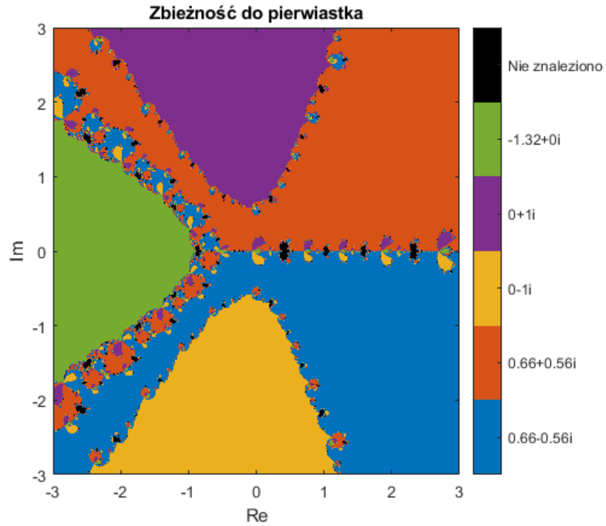


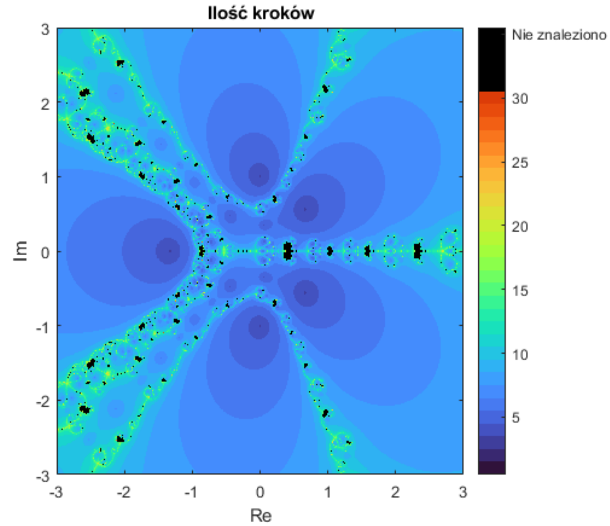
Widać, że pierwiastki są w przybliżeniu takie same, i do tego dzięki grafice *Roots in the complex plane*, możemy lepiej zrozumieć wizualizację *Zbieżność do pierwiastka*.
Próbowałem ustalić, dlaczego istnieją punkty w których programowi nie udało się znaleźć pierwiastka. Okazuje się, że te punkty po kilku krokach są bliskie punktu $x = 0$, i w kolejnych krokach zbiegają do niego, ale o bardzo małą wartość, to znaczy im więcej robią kroków tym o mniej się zliżają. Dla samego $x_0 = 0$ zachodzi podobna sytuacja jak w przykładzie 1, czyli mamy dzielenie przez 0, gdy szukamy x_1:

$$x_1 = 0 + \frac{-1}{3(0 - \frac{-1}{2*3*0})}$$

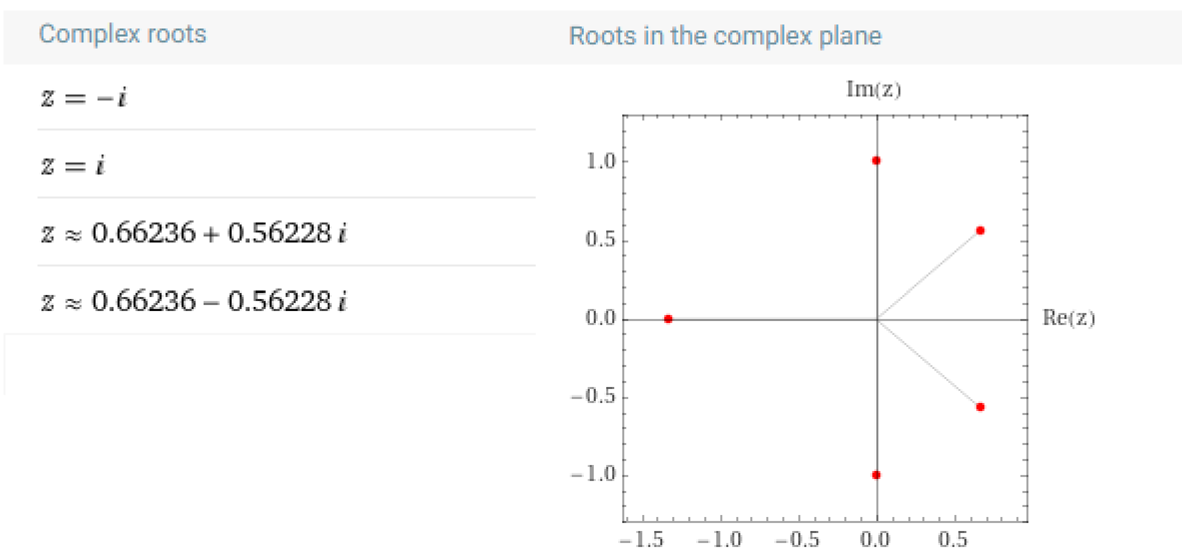
3.1.3 Przykład 3

Następnym wielomianem jest $z^5 + z^2 - z + 1$





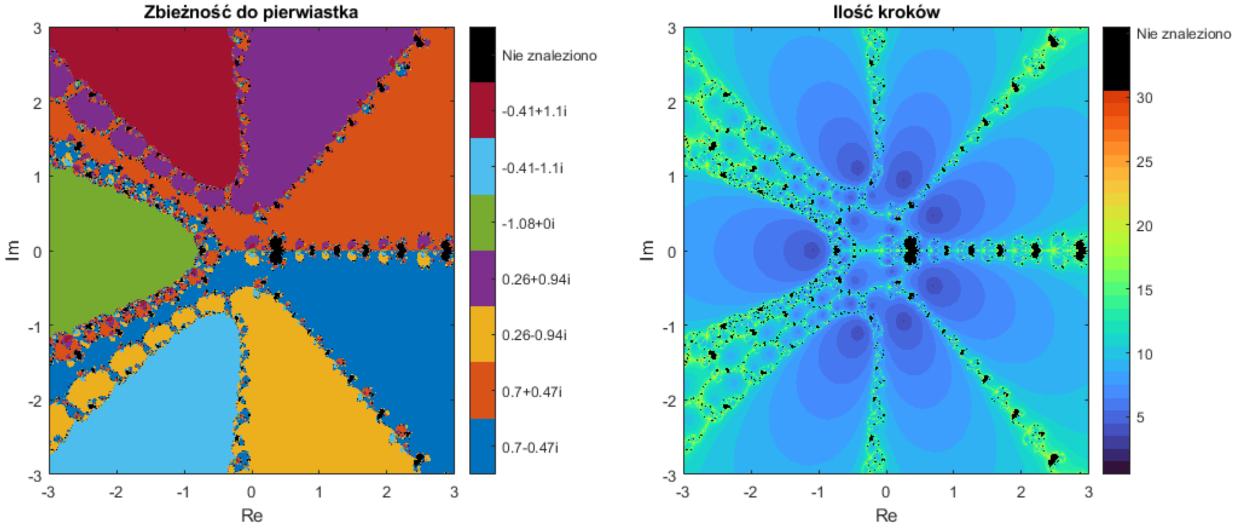
Na wizualizacji *Zbieżność do pierwiastka* widać, że dla tego wielomianu granica zielonego pola jest ciekawsza niż granice w poprzednich wielomianach. Jeśli spojrzy się na granicę między polem zielonym, a polem niebieskim widać,że w tych znajdują się tam też wszystkie pozostałe kolory w powtarzających się figurach.
Po raz ostatni można porównać wyniki z wynikami w WolfarmAlpha, aby się upewnić, że program działa poprawnie.



Jak widać wyniki w przybliżeniu znowu się zgadzają.

3.1.4 Przykład 4

Teraz biorę wielomian $x^7 + x^5 + x^3 - x + 1$. W tym przypadku wielomian różni się od poprzedniego tym, że jest dodatkowy wyraz x^7. Wydaje się, że to niewielka zmiana, jednak wizualizacja zmienia się znacząco.



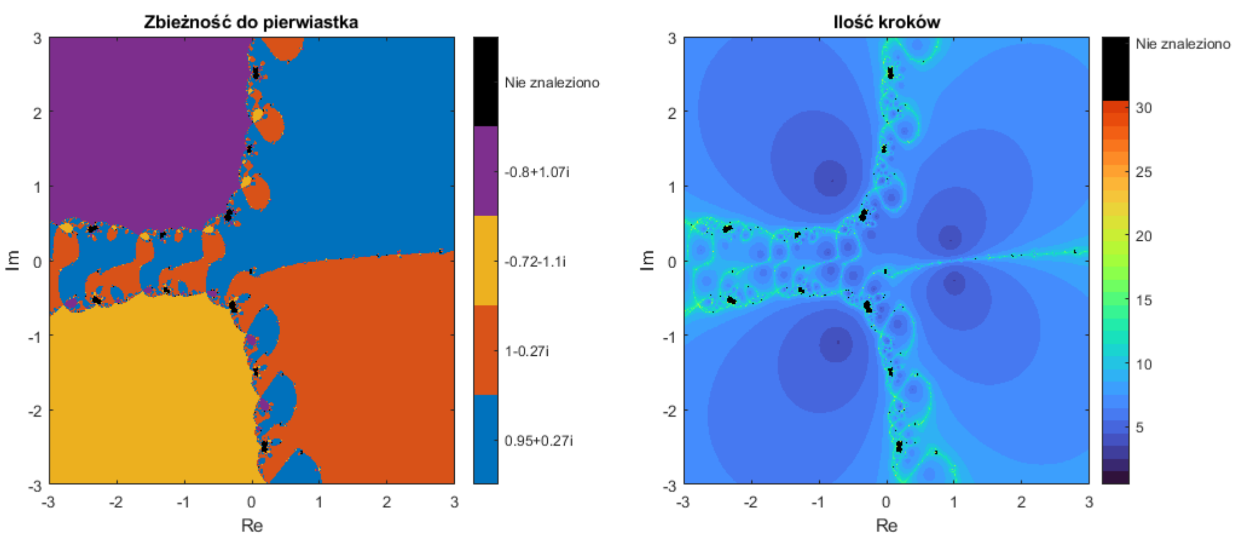
Wygląda na to, że im większy stopień wielomianu, a tym samym większa ilość pierwiastków, tym bardziej zawiła wizualizacja. Widać też, że potrzeba było zrobić więcej kroków, aby dojść do pierwiasku, niż w poprzednich wielomianach.

3.2 Przykłady z generatora

W tej sekcji pokażę kilka wizualizacji, które otrzymałem generując współczynniki funkcją `generate_coeffs` i wytłumaczę, dlaczego uważam je za ciekawe.

3.2.1 Przykład 5

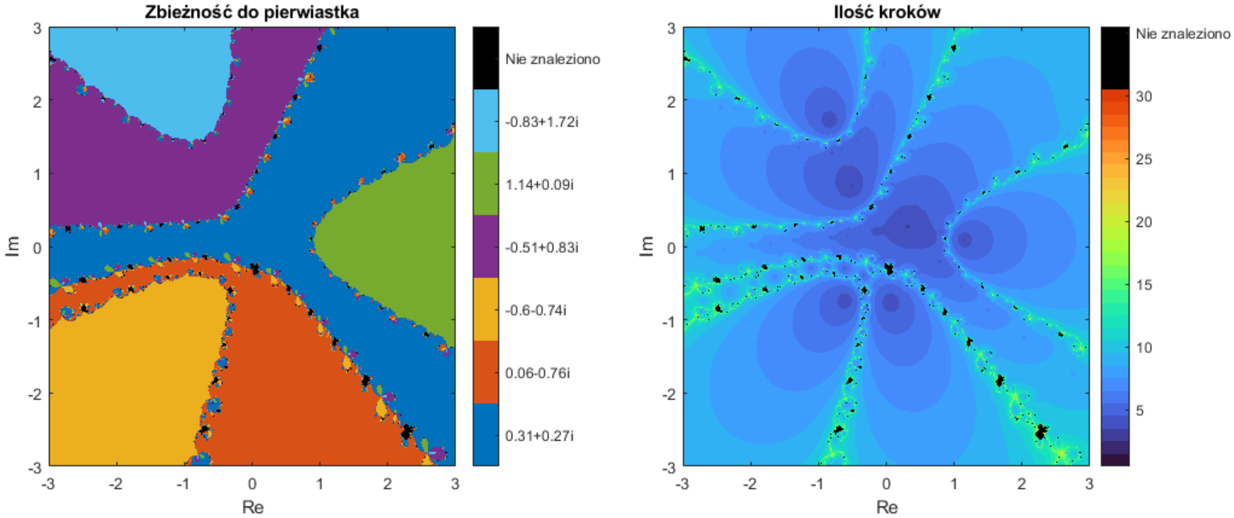
Wielomian $(5.31 + 2.93i)z^4 + (-2.37 - 1.09i)z^3 + (-1.23 - 0.2i)z^2 + (-9.31 - 6.26i)z + (9 + 5.9i)$



Moim zdaniem ciekawe tu jest to jak nietypowo ustawiają się pomarańczowe kształty po lewej stronie wizualizacji *Zbieżność do pierwiastka*.

3.2.2 Przykład 6

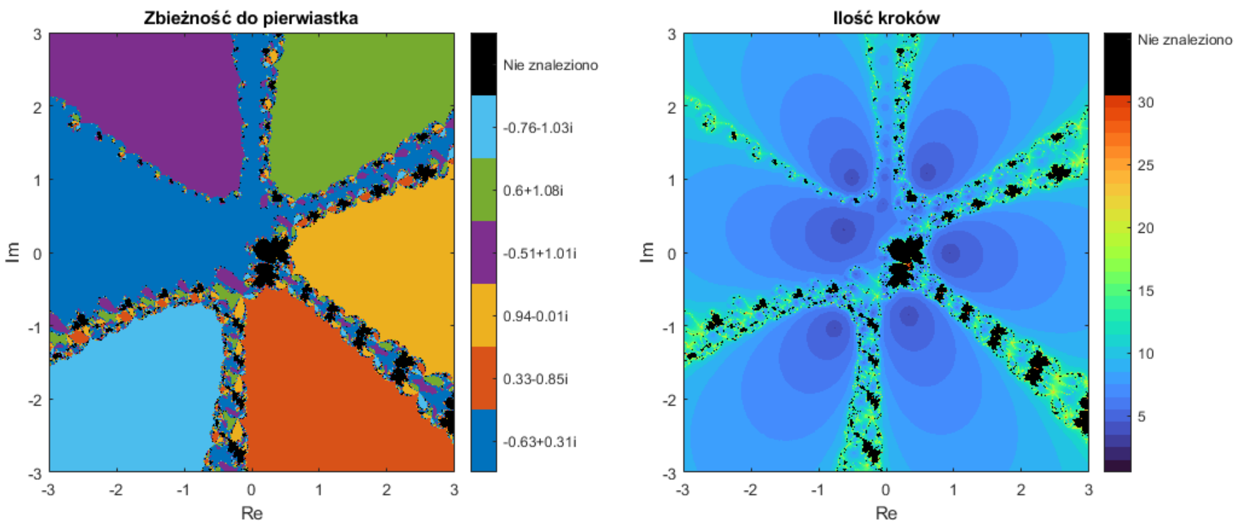
Wielomian $(-7.01 - 3i)z^6 + (0.94 - 5.13i)z^5 + (-7.23 + 8.59i)z^4 + (7.82 - 4.91i)z^3 + (9.19 + 6.29i)z^2 + (0.12 - 4.85i)z + (3.98 + 6.81i)$



Tutaj zaskakujące jest to w jaki sposób pierwiastki $-0.83 + 1.72i$ i $-0.6 - 0.74i$, są odgrodzone przez odpowiednio pierwiastki $-0.51 + 0.83i$, oraz $0.06 - 0.76i$, a także to, że granice między kolorami są stosunkowo cienkie.

3.2.3 Przykład 7

Wielomian $(-6.69 - 8.32i)z^6 + (2.04 - 5.42i)z^5 + (-4.74 + 8.27i)z^4 + (3.08 - 6.95i)z^2 + (3.78 + 6.52i)z^2 + (4.96 + 0.77i)z + (-0.99 + 9.92i)$



W tym przykładzie interesujący jest fakt, że jest dużo obszarów, na czarno, czyli tych, w których nie udało się znaleźć rozwiązania.

4 Podsumowanie

Mając na uwadze podane przykłady widać, że metoda Jarratta ma swoje ograniczenia. W każdym pokazanym przeze mnie wielomianie, a także na wszystkich innych, które testowałem (oprócz takich trywialnych jak x^2, x^3, itd.) istnieją punkty początkowe, dla których nie udaje się znaleźć żadnego rozwiązania podanego wielomianu. Same wizualizacje wynikające z szukania tych rozwiązań, szczególnie te pokazujące *Zbieżność do pierwiastka* wyglądają zadziwiająco. Wydaje się, że gdyby można było je obserwować w bardzo dużej rozdzielczości, to znaczy dla dużo większej ilości punktów, to przybliżając się coraz bardziej w miejscu granic można by obserwować ciągle nowsze, mniejsze kształty.