Analiza Numeryczna M - Pracownia 1

Konrad Werbliński - 291878

16 grudnia 2017

1 Wstęp

W moim zadaniu na pracownię zajmowałem się zbadaniem metody reprezentacji liczb zmiennopozycyjnych oraz wykonywania obliczeń na liczbach w tej reprezentacji, zaproponowanej w artykule: T. J. Dekker, *A floating-point technique for extending the available precision*, Numerische Mathematatik 18 (1971), 224–242. Główną motywacją do badania tych metod, było zastosowanie ich do zwiększenia dokładności wyników algorytmu obliczania pierwiastków równania kwadratowego (w szczególności zmniejszenia błędu wynikającego z utraty cyfr znaczących pod pierwiastkiem, poprzez zwiększenie dostępnej precyzji arytmetyki wykorzystując metody opisane w wyżej wymienionym artykule).

2 Opis zastosowanych metod

Procedury opisane w artykule bazują na sposobie znajdowania poprawki, dla operacji dodawania liczb zmiennopozycyjnych. Jeżeli x i y są słowami maszynowymi w postaci $mx\beta^{ex}$, $my\beta^{ey}$ oraz:

$$z = fl(x + y)$$

to poszukujemy takiej poprawki zz, że:

$$z + zz = x + y$$

Zakładając bez straty ogólności, że $ex \geq ey$, poszukiwaną poprawkę można obliczyć w następujący sposób:

$$(*)w = fl(z - x)$$

$$(**)zz = fl(y - w)$$

Zauważmy, że

$$z = fl(x + y) = x + y + \delta$$

Skoro $ex \geq ey$ to podczas dodawania mantysa y zostaje dorównana do mantysy x, podczas tej operacji tracimy część cyfr znaczących znajdujących się na końcu mantysy y, co skutkuje powstaniem błędu bezwzględnego δ. Możemy również stracić jedną cyfrę gdy wynik jest normalizowany, co również wpływa na powstawanie błędu bezwzględnego δ. Korzystając z poniższego lematu (udowodnionego w punkcie 4.7 artykułu)(R jest systemem zmiennopozycyjnym, w którym wykonywane są obliczenia):

$$z - x \in R$$

$$y - w \in R$$

oraz założenia, że system R jest wierny(definicja w punkcie 3.1 artykułu) otrzymujemy, że:

$$w = fl(z - x) = y + \delta$$

$$zz = fl(y - w) = fl(y - y - \delta) = -\delta$$

Zatem:

$$z + zz = x + y$$

Możliwość znajdowania poprawki zz jest kluczowa, przy implementacji reprezentacji liczb w postaci pary słów. Rozważmy dwie pary słów (x, xx), (y, yy). Reprezentacja zaproponowana w artykule, w postaci pary słów (r, s), spełnia poniższy warunek (punkt 7.2 artykułu):

$$|s| \leq |r+s|\frac{2^{-t}}{1+2^{-t}} \iff |s| + |s|2^{-t} \leq |r+s|2^{-t} \implies |s| + |s|2^{-t} \leq |r|2^{-t} + |s|2^{-t} \iff |s| \leq |r|2^{-t}$$

Z tego wynika, że:

$$|xx| \leq |x|2^{-t}$$

$$|yy| \leq |y|2^{-t}$$

Wtedy:

$$|xx| + |yy| \leq |x|2^{-t} + |y|2^{-t}$$

Zatem, jeżeli przy obliczaniu $fl(x+y)$, wystąpi błąd względny na poziomie precyzji arytmetyki, czyli $u = \frac{1}{2}2^{-t}$. Wtedy ten błąd będzie porównywalnego rzędu co wynik działania $fl(xx+yy)$. Dlatego znajdowanie poprawki jest niezbędne przy implementacji reprezentacji liczb w postaci pary słów zmiennopozycyjnych.

Wykorzystując tożsamości $(*)$ oraz $(**)$, w artykule opisano procedury dodawania, odejmowania, mnożenia, dzielenia i pierwiastkowania liczb reprezentowanych jako pary słów oraz procedurę mnożenia pojedynczych słów, której wynikiem jest para słów. Implementacja tych procedur w języku Julia, znajduje się w pliku `operations.jl`.

3 Wykonane doświadczenia

3.1 Pojedyncze działania arytmetyczne

W tym doświadczeniu sprawdziłem dokładność proponowanych procedur, w odniesieniu do dokładności standardowych operacji arytmetycznych wykonywanych na liczbach zmiennopozycyjnych typu `BigFloat` o długości 128 bitów. Średni błąd względny testowanych procedur wyniósł: $2.3741126040460738e-33$, czyli 6 rzędów wielkości więcej niż precyzja 128 - bitowej arytmetyki `BigFloat` ($u = \frac{1}{2}2^{-128} = 1.4693679e-39$). Warto nadmienić, że błędy względne arytmetyki `BigFloat128`, były na poziomie jej precyzji. Maksymalny błąd względny testowanych procedur wyniósł: $2.7404699155334106e-32$, czyli 16-rzędów wielkości mniej niż precyzja arytmetyki `Float64` ($u = \frac{1}{2}2^{-52} = 1.110223e-16$). Pozwala to przypuszczać, że wykorzystanie proponowanych procedur przy obliczaniu pierwiastków równania kwadratowego, znacznie zwiększy dokładność wyników.

3.2 Iterowane działania arytmetyczne

Aby sprawdzić dokładność procedur proponowanych w artykule, w sytuacji iterowania operacji arytetycznych, przybliżałem π kolejnymi wyrazami ciągu $x_k = 2^k \sin\frac{\pi}{2^k}$, wyrazy te wyznaczałem za pomocą zależności rekurencyjnej:

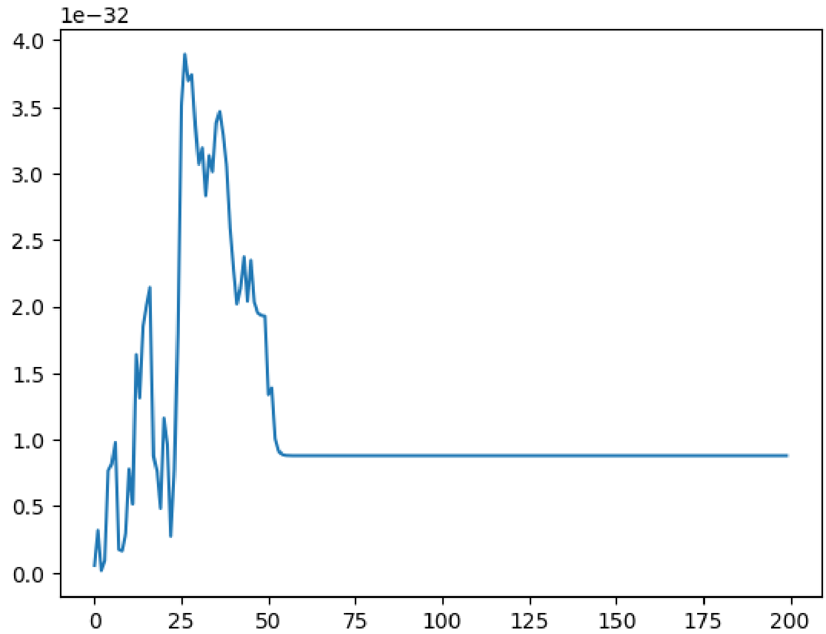
$$k \in \{2, 3, 4, \dots\}$$

$$x_{k+1} = x_k\sqrt{\frac{2x_k}{x_k + x_{k-1}}}$$

$$x_1 = 2$$

$$x_2 = 2\sqrt{2}$$

W kolejnych iteracjach otrzymałem następujące błędy względne:



Rysunek 1: Błąd względy w kolejnych iteracjach

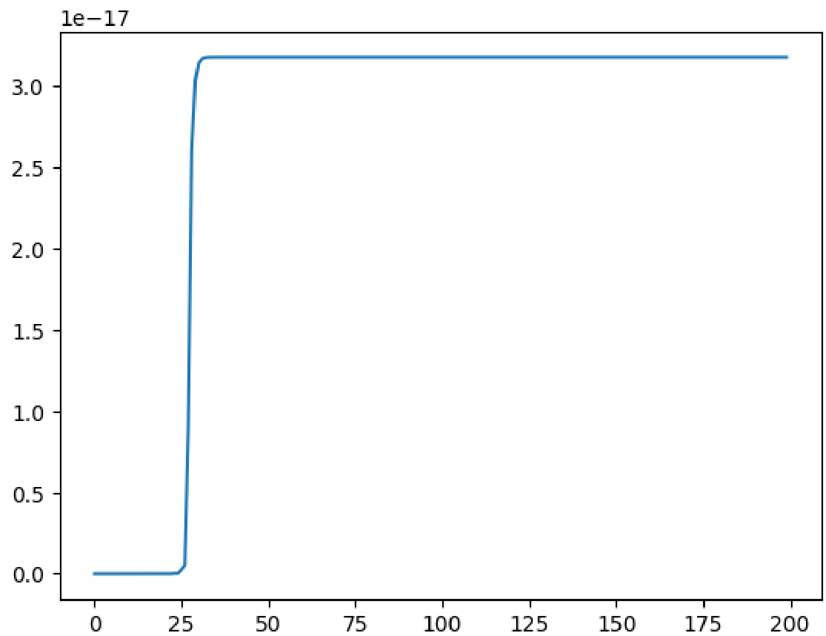
W drugim doświadczeniu wyznaczałem kolejne wartości funkcji sinus w w punktach $\frac{2\pi}{2^k}$ dla $k \in \{1, 2, \ldots\}$, korzystając z zależności rekurencyjnej:

$$\cos\pi = -1$$

$$\cos\frac{2\pi}{2^k} = \sqrt{\frac{\cos\frac{2\pi}{2^{k-1}} + 1}{2}}$$

$$\sin\frac{2\pi}{2^k} = \sqrt{\frac{1 - \cos\frac{2\pi}{2^{k-1}}}{2}}$$

Zauważmy, że gdy $k \to \infty$, to $\cos\frac{2\pi}{2^k} \to 1$, zatem zachodzi zjawisko utraty cyfr znaczących.



Rysunek 2: Błąd względy w kolejnych iteracjach

Maksymalny błąd względny, w teście bez utraty cyfr znaczących, wyniosł $3.894703852084468e - 32$ zatem był na poziomie błędów zaobserowanych przy pojedynczych działaniach. Maksymalny błąd względny, w teście z utratą cyfr znaczących wyniósł $3.174035784072652e - 17$, zatem był porównywalny do precyzji arytmetyki Float64. Co jest bardzo zadowalającym wynikiem, biorąc pod uwagę utratę cyfr znaczących w każdej iteracji. Podobnie, jak w przypadku pojedynczych doświadczeń, wyniki pozwalają przypuszczać, że opisane w artykule procedury skutecznie sprawdzą się przy rozwiązywaniu równań kwadratowych.

3.3 Rozwiązywanie równań kwadratowych

1. W pierwszym doświadczeniu wyznaczyłem, rozwiązania równania kwadratowego: $3x^2 - 4x - 5$. Zauważmy, że w tym teście nie zachodzi zjawisko utraty cyfr znaczących. Zatem dla typów `Float64` oraz `BigFloat128` wyniki algorytmu naiwnego oraz algorytmu wykorzystującego wzory Viete'a są identyczne.

Metoda	Błąd względny x_1	Błąd względny x_2
Algorytm naiwny - `Float64`	$1.3645644245849177e - 16$	$1.7958527604523258e - 16$
Algorytm naiwny - `BigFloat128`	$5.84386552517668e - 40$	$3.4078986294905214e - 39$
Testowane procedury	$1.447508646927704e - 32$	$3.4133599143969575e - 32$
`Float64`	$1.3645644245849177e - 16$	$2.4400248505777377e - 16$
`BigFloat128`	$5.84386552517668e - 40$	$3.4078986294905214e - 39$

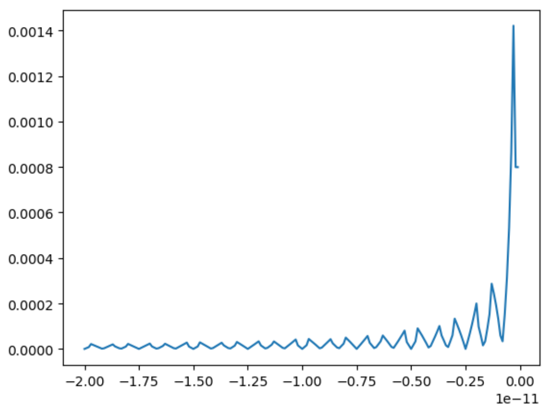
2. W drugim doświadczeniu wykorzystałem poniższe równanie kwadratowe:

$$a = 1$$
$$b = -3$$
$$c \to 0^-$$

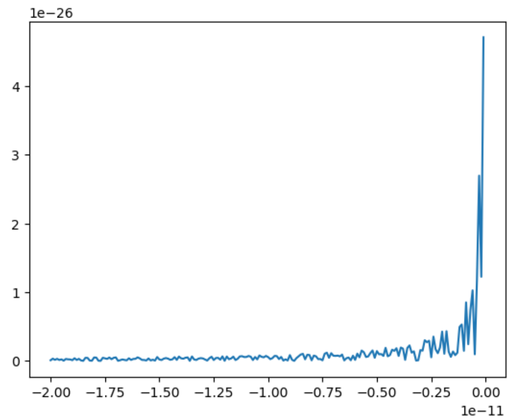
Z czego wynika, że:

$$\sqrt{\Delta} \to 3^+$$

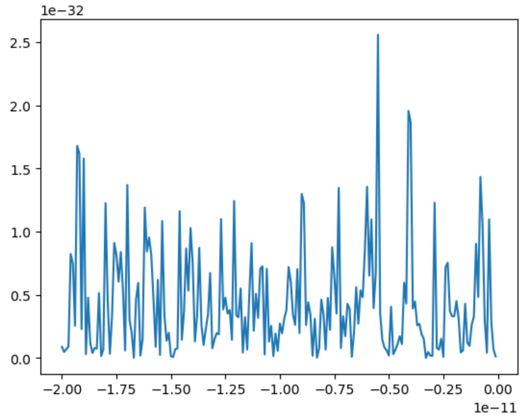
Zatem przy obliczaniu pierwiastka: $x = \frac{-b+\sqrt{\Delta}}{2a}$ dochodzi do utraty cyfr znaczących w naiwnym algorytmie nie wykorzystującym wzorów Vieta'a.



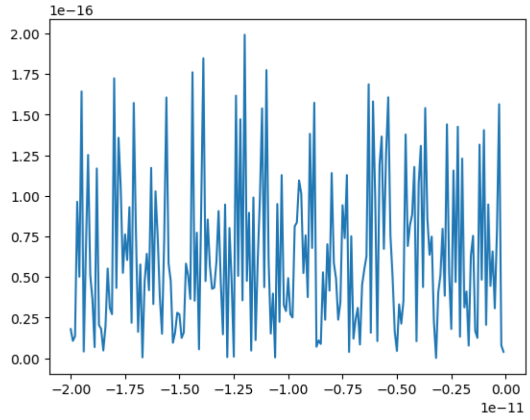
Rysunek 3: Błąd względny naiwnego algorytmu - Float64 w zależności od c.



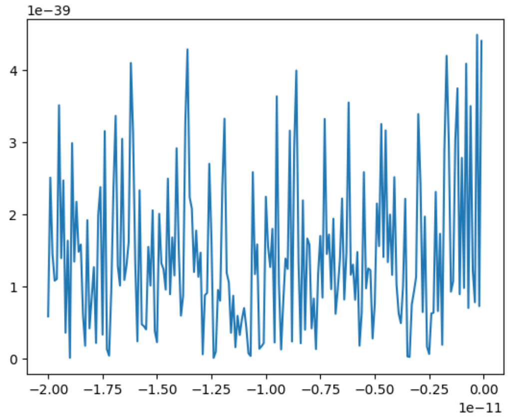
Rysunek 4: Błąd względny naiwnego algorytmu - BigFloat128 w zależności od c.



Rysunek 5: Błąd względny testowanych procedur w zależności od c.



Rysunek 6: Błąd względny algorytmu nienaiwnego - Float64 w zależności od c.



Rysunek 7: Błąd względny algorytmu nienaiwnego - BigFloat128 w zależności od c.

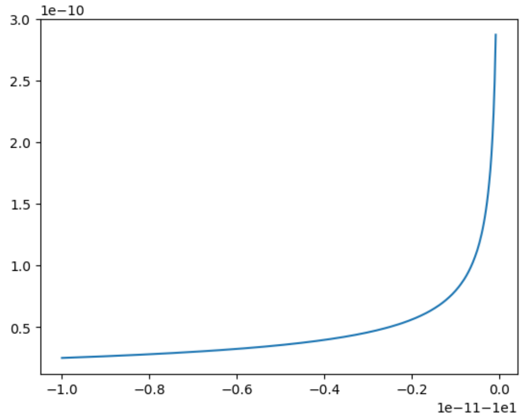
3. W trzecim doświadczeniu wyznaczałem rozwiązania poniższego równania kwadratowego:

$$a = 1$$
$$b \to -10^{-}$$
$$c = 25$$

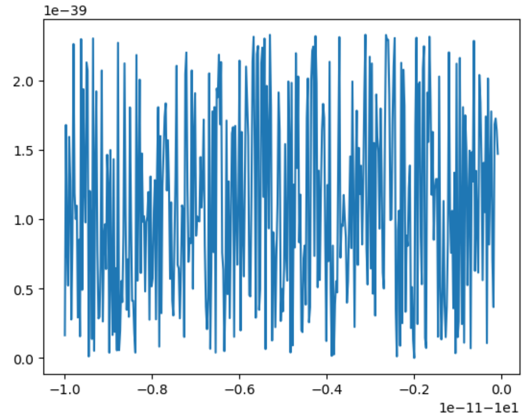
Z czego wynika, że:

$$b^2 \to 100^{+} = 4ac$$

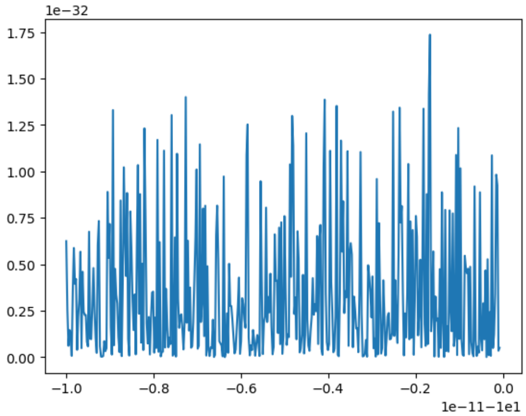
Zatem podczas obliczania Δ dochodzi do utraty cyfr znaczących, której nie zapobiega wykorzystanie wzorów Viete'a (co można zaobserwować na poniższych wykresach).



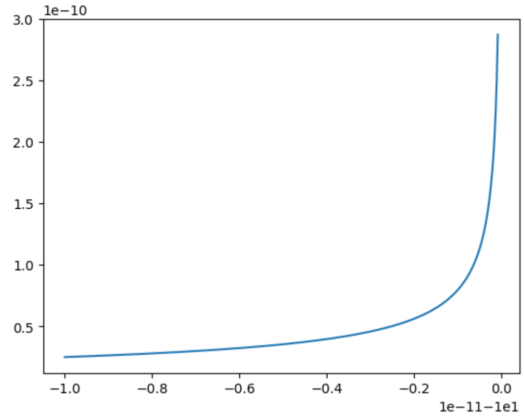
Rysunek 8: Błąd względny naiwnego algorytmu - Float64 w zależności od b.



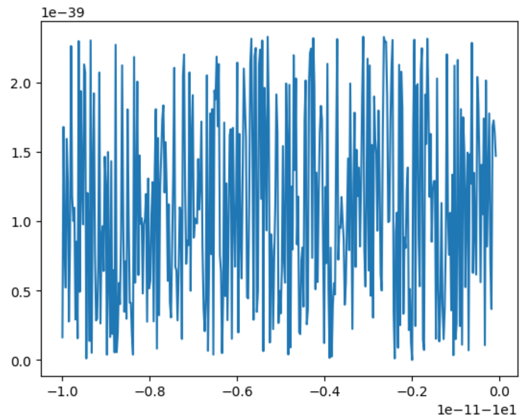
Rysunek 9: Błąd względny naiwnego algorytmu - BigFloat128 w zależności od b.



Rysunek 10: Błąd względny testowanych procedur w zależności od b.



Rysunek 11: Błąd względny algorytmu nienaiwnego - Float64 w zależności od b.



Rysunek 12: Błąd względny algorytmu nienaiwnego - BigFloat128 w zależności od b.

4. W czwartym doświadczeniu wykorzystałem poniższe równanie kwadratowe:

$$a = 1$$

$$b \to 0^+$$

$$c = min(b)^2 * 0.2499999999999999$$

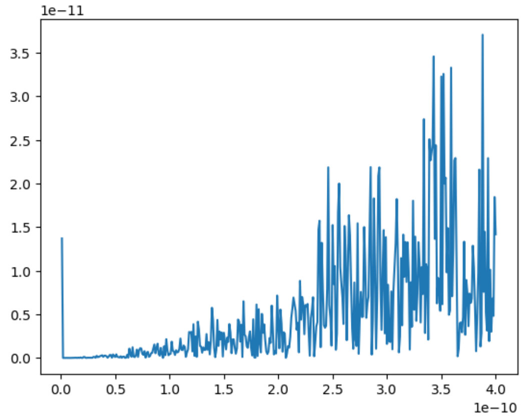
Zatem gdy $b^+ \to 0$:

$$b^2 \to 4 * ac \implies \Delta \to 0^+$$

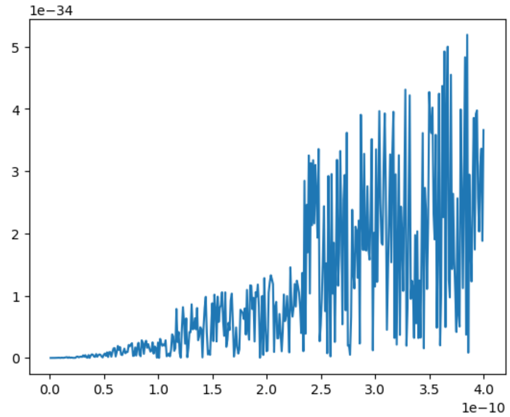
oraz dla $b \gg 4ac$

$$\sqrt{\Delta} \approx b \implies -b + \sqrt{\Delta} \to 0^+$$

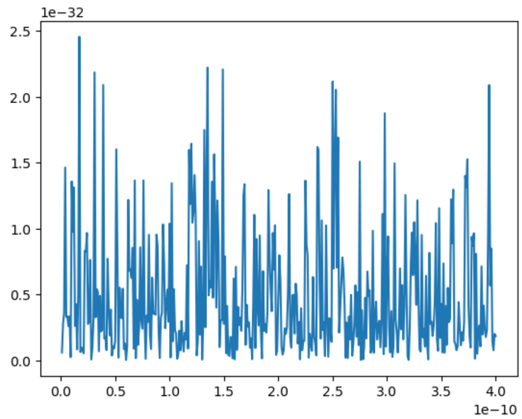
Na poniższych wykresach możemy zauważyć, że utrata cyfr znaczących gdy $b \gg 4ac$, zachodzi tylko dla algorytmów naiwnych `Float64` orza `BigFloat128`.



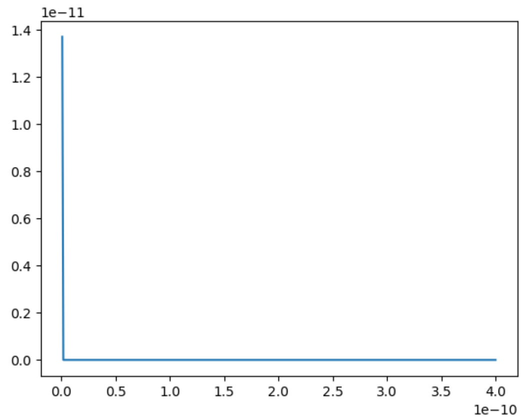
Rysunek 13: Błąd względny naiwnego algorytmu - Float64 w zależności od b.



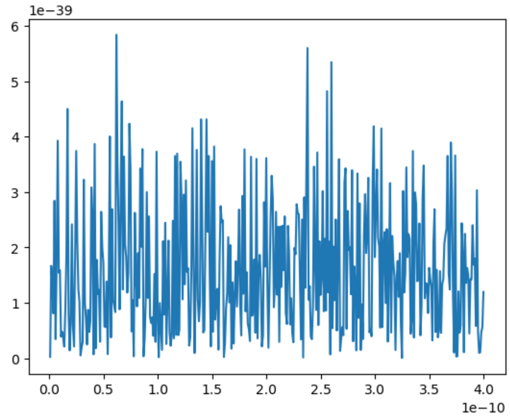
Rysunek 14: Błąd względny naiwnego algorytmu - BigFloat128 w zależności od b.



Rysunek 15: Błąd względny testowanych procedur w zależności od b.



Rysunek 16: Błąd względny algorytmu nienaiwnego - Float64 w zależności od b.



Rysunek 17: Błąd względny algorytmu nienaiwnego - BigFloat128 w zależności od b.

3.4 Test wydajności

W teście wydajności, polegającym na 100000 - krotnym obliczeniu pierwiastka kwadratowego, zarówno obliczenia z wykorzystaniem testowanych procedur jak i arytmetyki `BigFloat` długości 128 bitów, zajęły ~ 0.15 sekundy.

3.5 Podsumowanie przeprowadzonych doświadczeń

Proponowany w artykule sposób reprezentacji składa się z pary dwóch słów typu `Float64`. W standardzie IEEE 754 mantysa słowa 64 bitowego ma 52 bity, zatem łączna długość mantys słów w parze wynosi 102 bity. Z tego wynika, że precyzje reprezentacji zaproponowanej w artykule można oszacować jako:

$$u = \frac{1}{2}2^{-104} = 2.4651903e - 32$$

Warto zauważyć, że we wszystkich doświadczeniach, oprócz drugiego testu iterowanych działań, w którym występowała, powtarzająca się w każdej iteracji, utrata cyfr znaczących, błędy względne testowanych procedur były na poziomie precyzji tej arytmetyki. W szczególności błędy podczas obliczania pierwiastków równania kwadratowego, były rzędu precyzji arytmetyki. Wyniki te zgadzają się z oszacowaniami błędu opisanymi w artykule punktach (7.8) i (7.9). Zatem zastosowanie proponowanych w artykule metod reprezentacji danych i przeprowadzania obliczeń w algorytmie wykorzystującym wzory Viete'a, jest skutecznym sposobem przeciwdziałania utracie cyfr znaczących.

4 Wnioski

Metody zaproponowane w artykule, znacznie zwiększają precyzję obliczeń względem typu `Float64`, zarówno przy wykonywaniu pojedynczych i iterowaych działań arytmetycznych, jak i przy wykorzystaniu ich w algorytmie znajdowania pierwiastków równania kwadratowego. Ich średni błąd względny oraz precyzja arytmetyki są jednak o 6 rzędów wielkości większe niż precyzja arytmetyki 128 - bitowego `BigFloata`. Czas działania, testowanych metod jest bardzo zbliżony do czasu działania `BigFloata`. Zatem nie dają one nam zysku w postaci większej wydajności od `BigFloata`. Z tego wynika, że omawiane metody, w większości przypadków, nie mają zastosowania, ponieważ porównywalnym kosztem pamięci oraz przy takiej samej wydajności możemy skorzystać z typu `BigFloat`, uzyskując o kilka rzędów wielkości mniejsze błędy względne obliczeń. Warto również zwrócić uwagę na powszechną dostępność typu `BigFloat`, czy to w formie biblioteki, czy integralnej części języka, co jest kolejną przewagą nad metodami proponowanymi w artykule (które, musimy zaimplementować sami). Potencjalnym zastosowaniem proponowanych procedur, może być programowanie na komputerach retro, dla których może nie być dostępna biblioteka BigFloat, na przykład w środowiskach zajmujących się Demosceną lub w każdej innej sytuacji gdzie nie mamy dostępu do biblioteki BigFloat. Dużą zaletą jest w tej sytuacji prosta i szybka implementacja testowanych procedur.