Obliczenia Naukowe

Sprawozdanie z laboratorium nr 2

Piotr Kawa

Wrocław 05.11.2017

0. Wprowadzenie

Celem tejże listy było zapoznanie z tematem uwarunkowania zadań oraz stabilności algorytmów. Zadania wykonano przy użyciu języka Julia (w zadaniach 1-6) oraz Python (zadanie 2).

1. Zadanie pierwsze

1.1 Opis zadania

Celem zadania było powtórzenie obliczeń z zadania 5 z listy 1 używając innych danych.

1.2 Rozwiązanie

Zadanie 5 z listy 1 polegało na obliczeniu za pomocą czterech algorytmów iloczynu skalarnego dwóch wektorów używając precyzji $Float32$ oraz $Float64$.

Pierwszy algorytm polegał na wyliczeniu iloczynu metodą „w przód": S=S+$xi*yi$ w pętli 1..5, gdzie x i y to wektory. Druga metoda („w tył") prowadziła pętlę od 5 do 1. Trzeci algorytm polegał na dodaniu do siebie dodatnich wyników mnożenia $xi*yi$ w kolejności od największego do najmniejszego, ujemnych natomiast odwrotnie. Następnie należało obliczyć sumę obydwu z nich. Ostatnia metoda była odwrotnością trzeciej.

Wyliczenia zostały przeprowadzone dla innego zestawu danych (z ostatnich miejsc w x_4 oraz x_5 zostały usunięte kolejno cyfry 9 oraz 7).

1.3 Wyniki

Wyniki otrzymane przy zastosowaniu nowych danych:

LP	$Float32$	$Float64$
1.	-0.499944	$-\mathbf{4.296342}73989158537\mathrm{e}-3$
2.	-0.454345	$-\mathbf{4.296342}99871395342\mathrm{e}-3$
3.	-0.5	$-\mathbf{4.296342}84228086472\mathrm{e}-3$
4.	-0.5	$-\mathbf{4.296342}84228086472\mathrm{e}-3$

Wyniki zadania 5 z listy 1:

LP	$Float32$	$Float64$
1.	-0.499944	$1.0251881368e-10$
2.	-0.454345	$-1.5643308870e-10$
3.	-0.5	0.0
4.	-0.5	0.0

1.4 Wnioski

Jak widać po powyższych wynikach nie zmieniły się one dla arytmetyki $Float32$. Jest to spowodowane faktem, iż zmiany w liczbach w tejże arytmetyce były na zbyt dalekiej pozycji - dziesiątej, a to nie mogło w żaden sposób wpłynąć na wynik – $Float32$ nie mieści tylu cyfr znaczących.

Sprawa wyglądała jednak inaczej w przypadku $Float64$. Zmiany znajdowały się na pozycjach, które miały jeszcze znaczenie w tejże arytmetyce, co doprowadziło do zmian. Jak widać w powyższej tabeli wyniki były zgodne ze sobą w pewnym stopniu (do 9 cyfry po przecinku) – w odróżnieniu od tych zupełnie nieskorelowanych wyników z listy 1.

2. Zadanie drugie

2.1 Opis problemu

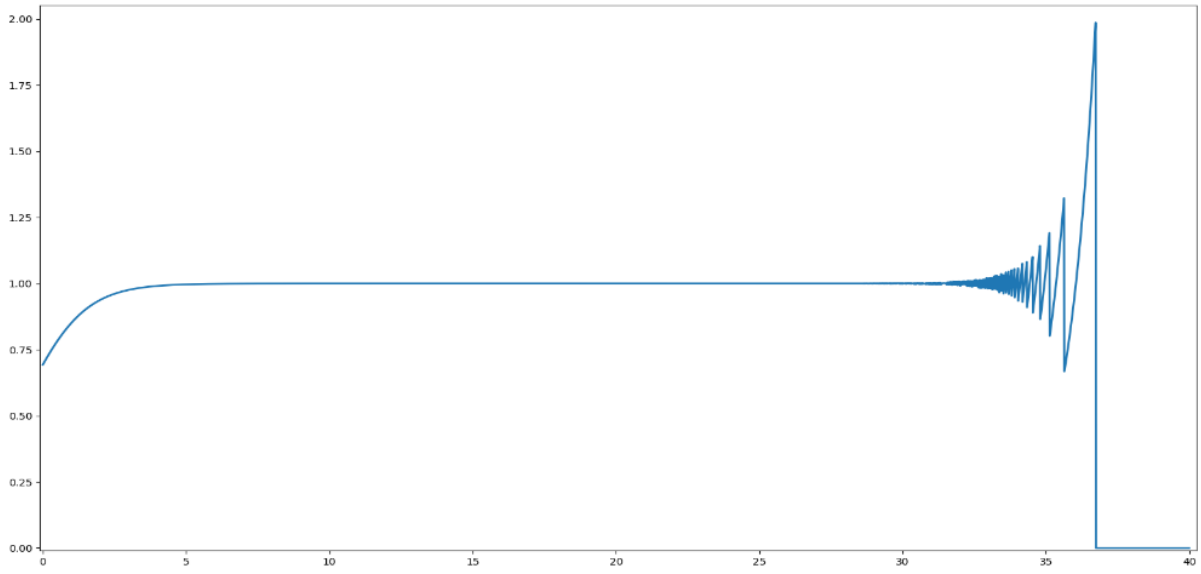
W zadaniu drugim należało narysować wykres funkcji $f(x) = e^x \ln(1 + e^{-x})$ w co najmniej dwóch programach do wizualizacji, a następnie porównać z policzoną wcześniej granicą funkcji $\lim_{x \to \infty} f(x)$.

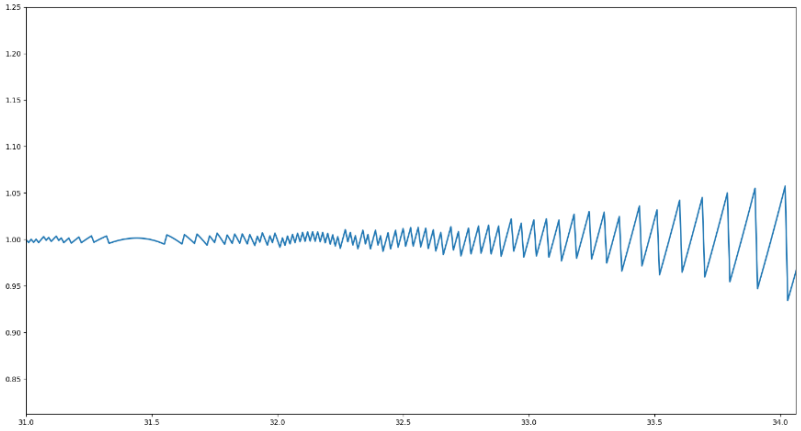
2.2 Rozwiązanie i wyniki

W celu wygenerowania wykresu funkcji $f(x)$ skorzystano z kilku narzędzi.

2.2.1 Python

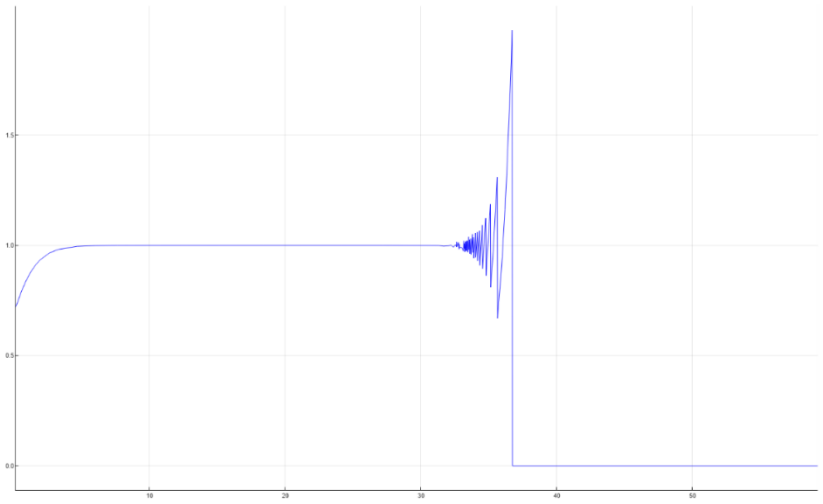
Pierwszym z nich był program napisany w języku $Python$ korzystając z biblioteki $matplotlib$. Wygenerowany wykres wyglądał następująco.





2.2.2 Julia

Drugi wykres został stworzony przy pomocy biblioteki $PlotlyJS$ używając języka $Julia$.



2.3 Wnioski

Wynikiem działania $\lim_{x\to\infty} f(x)$ jest 1. Jak widać po powyższych wykresach do pewnego momentu występuje zgodność pomiędzy wykresem funkcji, a jej granicą. W okolicach liczby 30 funkcja zaczyna przyjmować nieoczekiwane wartości. Wynika to z wartości składowych tejże funkcji. e^x już w okolicach $x = 30$ jest duża (rośnie wykładniczo). Drugą składową funkcji $f(x)$ jest natomiast logarytm, który w tym momencie ma bardzo małą wartość. Mnożenie tychże liczb powoduje widoczne wyżej anomalie.

3. Zadanie trzecie

3.1 Opis problemu

Celem zadania drugiego było rozwiązanie układu równań liniowych $Ax = b$, gdzie A to macierz, natomiast b to wektor wyliczony za pomocą wzoru $b = Ax$, gdzie $x = (1, \dots, 1)^T$. Wynik miał zostać uzyskany z wykorzystaniem dwóch sposobów: metody eliminacji Gaussa oraz $x =$

$A^{-1}b$. Powyższe algorytmy miały następnie zostać przetestowane na dwóch rodzajach macierzy: macierzy Hilberta o rosnącym stopniu oraz kwadratowej macierzy o losowo wybranych elementach z zachowaniem podanego wskaźnika uwarunkowania $c = 1, 10, 10^3, 10^7, 10^{12}, 10^{16}$.

3.2 Rozwiązanie i wyniki

W celu stworzenia danych macierzy wykorzystane zostały podane wcześniej funkcje $hilb(n)$ oraz $matcond(n, c)$, gdzie n to wielkość macierzy, a c to wskaźnik uwarunkowania. Wykonane zostały na nich podane wcześniej algorytmy: eliminacja Gaussa $x = A \setminus b$ oraz wykorzystanie odwrotności macierzy $x = A^{-1}b$. Błędy względne przedstawione w poniższej tabeli zostały obliczone za pomocą funkcji $\frac{norm(matrix - ones(n,1))}{norm(ones(n,1)}$.

Macierz Hilberta			
n	Wskaźnik	Metoda Gaussa (błąd)	Odwrotność (błąd)
1	1.0	0.0	0.0
2	$1.9281470067e1$	$5.661048867003676e-16$	$1.1240151438116956e-15$
5	$4.7660725024e5$	$1.6828426299227195e-12$	$8.500055777753297e-12$
10	$1.6024416992e13$	$6.329153722983848e-4$	$4.5521422517408853e-4$
15	$3.6743929534e17$	1.177294734836712	4.8273577212576475
20	$1.3553657908e18$	$1.4930069669294001e1$	$2.153949860251383e1$

Macierz losowa			
n	Wskaźnik	Metoda Gaussa (błąd)	Odwrotność (błąd)
5	1	1.9860273225978183e-16	1.719950113979703e-16
5	10	4.2998752849492583e-16	3.2934537262255424e-16
5	10^3	1.5632476364782915e-14	1.0783117927540637e-14
5	10^7	3.313971681915597e-10	2.07498248713627e-10
5	10^{12}	1.2156213399240275e-5	1.1520680193223685e-5
5	10^{16}	1.2889100101428277e-1	1.6237333550006244e-1
10	1	2.4575834280036907e-16	2.4575834280036907e-16
10	10	3.2177320244274193e-16	2.5559253454202263e-16
10	10^3	2.1058534033865817e-14	2.3099390002898322e-14
10	10^7	3.113504564448224e-11	3.0036728796487764e-11
10	10^{12}	1.9083838560935246e-5	1.9051775929281237e-5
10	10^{16}	1.2693694455977542e-1	1.1421982884318328e-1
20	1	3.9642891350404605e-16	3.4666702832799586e-16
20	10	6.57754439519148e-16	8.731323549074581e-16
20	10^3	1.857121395500645e-14	1.683941546322223e-14
20	10^7	1.7072533333147238e-11	4.704119019223323e-11
20	10^{12}	2.8606789343780592e-5	3.1122529739533846e-5
20	10^{16}	2.2399003675612475e-1	2.274581568877353e-1

3.3 Wnioski

Macierz Hilberta jest przykładem macierzy, która jest źle uwarunkowana (wykazuje się niestabilnością numeryczną). Składa się ona z liczb, które nie sposób zapisać dokładnie w arytmetyce zmiennoprzecinkowej (takich jak np. $\frac{1}{3}$, $\frac{1}{6}$ czy $\frac{1}{7}$). W celu przedstawienia ich przez komputer zapisywane są one z przybliżeniem. Generuje to pewien błąd, który powiększa się wraz z wykonywanymi działaniami. Im większa macierz tym więcej wykonywanych jest obliczeń co skutkuje generowaniem coraz większego błędu. Z dwóch zastosowanych metod – eliminacji Gaussa oraz odwrotności, większymi błędami odznaczała się ta druga. Powodem tego stanu rzeczy jest użycie przez metodę Gaussa tzw. postaci schodkowej macierzy, która zmniejsza ilość działań na liczbach zaokrąglonych co w efekcie powoduje mniejsze błędy.

Na wyniki otrzymane przy użyciu macierzy losowej również miały wpływ jej rozmiar jak i użyty algorytm. Tendencja była również ta sama – im większa macierz, tym większy błąd, eliminacja Gaussa obarczona była mniejszymi błędami. Podczas tworzenia tychże macierzy stosowany był jednak jeszcze jeden czynnik – c, czyli wskaźnik uwarunkowania macierzy. Im był on większy, tym większymi błędami obarczone były wyniki, nawet jeśli macierz miała ten sam rozmiar.

4. Zadanie czwarte

4.1 Opis problemu

Celem tego zadania było obliczenie 20 zer wielomianu $P(x)$ (postaci naturalnej wielomianu Wilkinsona), a następnie porównać je z tym samym wielomianem w postaci $p(x)$. Rozwiązanie powinno było zawierać funkcje z pakietu $Polynomials$.

Eksperyment następnie miał być powtórzony na innych danych (współczynnik -210 miał zostać zamieniony na $-210 - 2^{-23}$).

4.2 Rozwiązanie i wyniki

W celu wykonania zadania skorzystano z funkcji znajdujących się w pakiecie $Polynomials$. Do wygenerowania pierwiastków wielomianu użyta została funkcja $roots(Polynomial)$. Funkcje $poly(x)$ oraz $Poly(x)$ służyły do generowania wielomianów – pierwsza z nich na podstawie pierwiastków, druga na podstawie współczynników. Wykorzystana została również funkcja $polyval(p, x)$, która wyliczała wartość wielomianu p dla danego x.

k	$roots(Polynomial)$	$\lvert z_k - k \rvert$	$\lvert P(x) \rvert$	$\lvert p(x) \rvert$
1	0.9999999999996	$3.0e-13$	$3.6352e4$	$3.8400e4$
2	2.0000000000283	$2.831e-11$	$1.8176e5$	$1.98144e5$
3	2.9999999995920	$4.0790e-10$	$2.09408e5$	$3.01568e5$
4	3.9999999837375	$1.626246e-8$	$3.106816e6$	$2.844672e6$
5	5.0000006657697	$6.6576979e-7$	$2.4114688e7$	$2.3346688e7$
6	5.9999892458247	$1.075417522e-5$	$1.20152064e8$	$1.1882496e8$
7	7.0001020027930	$1.0200279300e-4$	$4.80398336e8$	$4.78290944e8$
8	7.9993558296077	$6.4417039223e-4$	$1.682691072e9$	$1.67849728e9$
9	9.0029152943620	$2.91529436205e-3$	$4.465326592e9$	$4.457859584e9$
10	9.9904130424817	$9.58695751827e-3$	$1.2707126784e10$	$1.2696907264e10$
11	11.025022932909	$2.50229329093e-2$	$3.5759895552e10$	$3.5743469056e10$
12	11.953283253846	$4.67167461531e-2$	$7.2167715840e10$	$7.2146650624e10$
13	13.074314032447	$7.43140324473e-2$	$2.15723629056e11$	$2.15696330752e11$
14	13.914755591802	$8.52444081978e-2$	$3.65383250944e11$	$3.653447936e11$
15	15.075493799699	$7.54937996994e-2$	$6.13987753472e11$	$6.13938415616e11$
16	15.946286716607	$5.37132833920e-2$	$1.555027751936e12$	$1.554961097216e12$
17	17.025427146237	$2.54271462374e-2$	$3.777623778304e12$	$3.777532946944e12$
18	17.990921352716	$9.07864728351e-3$	$7.199554861056e12$	$7.1994474752e12$
19	19.001909818299	$1.90981829943e-3$	$1.0278376162816e13$	$1.0278235656704e13$
20	19.999809291236	$1.9070876336e-4$	$2.7462952745472e13$	$2.7462788907008e13$

Poniżej zostały przedstawione wyniki dla drugiego zestawu danych.

k	$roots(Polynomial)$	$\lvert P(x) \rvert$
1	$0.9999999999 + 0.0im$	$2.0992e4$
2	$2.0000000000 + 0.0im$	$3.49184e5$
3	$2.9999999966 + 0.0im$	$2.221568e6$
4	$4.0000000897 + 0.0im$	$1.046784e7$
5	$4.9999985738 + 0.0im$	$3.9463936e7$
6	$6.0000204766 + 0.0im$	$1.29148416e8$
7	$6.9996020704 + 0.0im$	$3.88123136e8$
8	$8.0077720290 + 0.0im$	$1.072547328e9$
9	$8.9158163679 + 0.0im$	$3.065575424e9$
10	$10.0955 - 0.644933im$	$7.143113638035824e9$
11	$10.0955 + 0.644933im$	$7.143113638035824e9$
12	$11.7939 - 1.65248im$	$3.357756113171857e10$
13	$11.7939 + 1.65248im$	$3.357756113171857e10$
14	$13.9924 - 2.51882im$	$1.0612064533081976e11$
15	$13.9924 + 2.51882im$	$1.0612064533081976e11$
16	$16.7307 - 2.81262im$	$3.315103475981763e11$
17	$16.7307 + 2.81262im$	$3.315103475981763e11$
18	$19.5024 - 1.94033im$	$9.539424609817828e12$
19	$19.5024 + 1.94033im$	$9.539424609817828e12$
20	$20.8469 + 0.0im$	$1.114453504512e13$

4.3 Wnioski

Wielomian Wilkinsona jest zwany wielomianem „złośliwym”. Wartość każdego z obliczonych pierwiastków odbiegała od oczekiwanych wyników. Sytuacja wygląda podobnie w przypadku wartości $P(z_k)$ oraz $p(z_k)$. Sytuacja spowodowana jest przez błąd, który został wygenerowany już na początku obliczeń. Wraz z kolejnymi operacjami powiększał się on, co doprowadziło do wyników przedstawionych w powyższych tabelach.

Jak widać istnieje różnica między wynikami otrzymanymi dla pierwszego, jak i drugiego zestawu danych - zadanie jest źle uwarunkowane – mała zmiana danych (odjęcie 2^{-23}) spowodowała duże różnice wyników – niektóre z pierwiastków zawierały liczby zespolone.

5. Zadanie piąte

5.1. Opis problemu

Celem zadania było przeprowadzenie dwóch eksperymentów dla $n > 0$ używając do tego następującego równania rekurencyjnego $p_{n+1} = p_n + rp_n(1 - p_n)$.

5.1.1 Pierwszy eksperyment

Dla danych $p_0 = 0.01$ i $r = 3$ należało wykonać 40 iteracji powyższego równania rekurencyjnego. Następnie powtórzyć je z pewnymi modyfikacjami – po każdych 10 iteracjach wykonać obcięcie wyniku odrzucając cyfry po 3 miejscu po przecinku. Wykonane obliczenia miały wykorzystywać Float32.

5.1.2 Drugi eksperyment

Eksperyment polegał na wykonaniu 40 iteracji ww. równania rekurencyjnego dla danych $p_0 = 0.01$ i $r = 3$. Obliczenia miały być wykonane w arytmetyce $Float32$ oraz $Float64$.

5.2 Rozwiązanie i wyniki

n	$Float32$	$Float32$ (obcięcie)	$Float64$
1	**0.0397**	**0.0397**	**0.0397**
5	**0.1715188**	**0.1715188**	**0.17151**914210917552
10	**0.7229306**	**0.7229306**	**0.7229**14301179573
11	**1.32**38364	**1.32**41479	**1.32**38419441684408
15	**1.2**704837	**1.2**572169	**1.2**702617739350768
20	0.5799036	1.3096911	0.5965293124946907
25	**1.**0070806	**1.**0929108	**1.**315588346001072
30	0.7529209	1.3191822	0.37414648963928676
35	1.021099	0.034241438	0.9253821285571046
40	0.25860548	1.093568	0.011611238029748606

5.3 Wnioski

Jak widać na podstawie pogrubionych miejsc w powyższej tabeli w przypadku pierwszego eksperymentu wyniki od 10 iteracji nie pokrywają się – jest to naturalnie wynik zastosowanego obcięcia wyniku do 3 cyfry po przecinku. Kolejne iteracje różnią się od siebie coraz bardziej. Wyjaśnienie tego zjawiska można znaleźć w książce H.O. Peitgen, H. Jürgens, D. Saupe „Granice chaosu. Fraktale, część 1": co każdą iterację następowało podniesienie do kwadratu liczby zmiennoprzecinkowej – operacja skutkująca podwojeniem ilości cyfr znaczących danej liczby. W odpowiedzi na to komputer zmuszony był zaokrąglać wynik operacji, co generowało pewien błąd. Błędny wynik był wykorzystywany jako zmienna w kolejnych obliczeniach, co powodowało to, że z każdą iteracją błąd nawarstwiał się co doprowadziło do ogromnych różnic pomiędzy wynikami obydwu algorytmów. Efekt ten naukowo nazywany jest niestabilnością numeryczną.

Podobna sytuacja ma miejsce w etapie drugim zadania. Jedynie początkowe wyniki algorytmu były podobne zarówno dla $Float32$ jak i $Float64$. Kolejne iteracje powodowały coraz większe odbieganie od siebie obydwu wyników. $Float64$ z racji większej ilości cyfr precyzji dłużej odznaczał się wiarygodnością wyniku. Do podobnych rezultatów doszli autorzy wcześniej podanej książki, którzy przeprowadzili eksperyment porównując wyniki tego samego algorytmu na kalkulatorze Casio (mającego 10 cyfr precyzji) oraz HP (12 cyfr). Obydwa urządzenia nie były w stanie dokładnie reprezentować wyników już od 3 iteracji.

6. Zadanie szóste

6.1 Opis problemu

Celem zadania szóstego było wykonanie w arytmetyce Float64 czterdziestu iteracji wyrażenia $x_{n+1} = x_n^2 + c$ dla $n = 0,1, \ldots$, gdzie c to pewna stała. Przeprowadzone miały zostać one na siedmiu zestawach danych:

1) $c = -2, x_0 = 1$

2) $c = -2, x_0 = 2$

3) $c = -2, x_0 = 1.999999999999999$

4) $c = -1, x_0 = 1$

5) $c = -1, x_0 = -1$

6) $c = -1, x_0 = 0.75$

7) $c = -1, x_0 = 0.25$

6.2 Rozwiązanie

W celu rozwiązania powyższego problemu stworzona została funkcja $computeRecursiveEquation(x,\ c,\ n,\ maximumNoOfIterations)$, która rekursywnie wyliczała kolejne wartości równania $x_n^2 + c$ dla zadanego w argumencie zestawu danych.

6.2 Wyniki i wnioski

Wyniki dla pierwszego zestawu danych ($c = -2, x_0 = 1$) oraz drugiego ($c = -2, x_0 = 2$) były zgodnie z oczekiwaniami takie same dla każdej iteracji, wynosiły mianowicie odpowiednio -1.0 oraz 2.0.

Jak wiadomo z poprzedniego zadania podniesienie do kwadratu liczby zmiennoprzecinkowej to operacja skutkująca podwojeniem ilości cyfr znaczących danej liczby. Dla zestawu trzeciego tą liczbą było $x_0 = 1.999999999999999$, co skutkowało tym, że już pierwsza iteracja powodowała wykroczenie poza zakres arytmetyki $Float64$, przez co powstawał pewien błąd. Kolejne iteracje powielały go, co zaprezentowane zostało w poniższej tabeli.

Zestaw 3	
Iteracja	**Wynik**
1	1.9999999999999555911
2	1.9999999999998223643
3	1.9999999999992894573
4	1.9999999999971578291
5	1.9999999999886313162
6	1.9999999999545252649
7	1.9999999998181010596
8	1.9999999992724042386
9	1.9999999970896169543
10	1.9999999883584678173
15	1.9999880790722173174
20	1.9987794210450759907
30	−0.68810193087555382441
31	−1.52651573272533447323
40	0.11000795163075816063

Zarówno czwarty zestaw danych ($x_0 = 1.0,\ c = -1.0$), jak i piąty ($x_0 = -1.0,\ c = -1.0$) nie wygenerowały żadnego błędu. Zwrócone zostały oczekiwane wyniki – na przemian 0.0 i -1.0.

Sprawa wyglądała jednak inaczej w przypadku zestawu szóstego oraz siódmego (odpowiednio $x_0 = 0.75, c = -1$ oraz $x_0 = 0.25, c = -1$).

Iteracja	$x_0 = 0.75, c = -1$	$x_0 = 0.25, c = -1$
1	-0.43750000000000000000	-0.93750000000000000000
2	-0.80859375000000000000	-0.12109375000000000000
3	-0.34617614746093750000	-0.98533630371093750000
4	-0.88016207492910325527	-0.02911236858926713467
5	-0.22531472185649559226	-0.99915246999512263848
6	-0.94923327611473007348	-0.00169434170264559647
7	-0.09895618751649659650	-0.99999712920619465706
8	-0.99020767295219991322	-0.00000574157936927833
9	-0.01948876442658908914	-0.99999999996703425875
10	-0.99962018806112495906	-0.00000000006593148250
11	-0.00075947962064115693	-1.0
12	-0.99999942319070578289	0.0
13	-0.00000115361825570037	-1.0
14	-0.99999999999866917566	0.0
15	-0.00000000000266164868	-1.0
16	-1.0	0.0
17	0.0	-1.0
18	-1.0	0.0
25	0.0	-1.0
26	-1.0	0.0
39	-1.0	0.0
40	0.0	-1.0

Jak widać w tabeli powyżej, zwracane przez program wartości wynosiły od pewnego momentu na przemian 0.0 i -1.0. Powinno się to stać dopiero w nieskończoności. To zachowanie zostało również wyjaśnione w książce „Granice chaosu. Fraktale, część 1”. Jest to zjawisko stabilności – jest ono bardzo pożądane podczas wykonywania obliczeń. Polega ono na przewidywalności wyników oraz tym, że pomijane są małe błędy (które wcześniej prowadziły poprzez nawarstwianie do sporych różnic).