Obliczenia Naukowe - L2

Piotr Maciejończyk

3 listopada 2023

1 Wpływ niewielkich zmian danych na wyniki

Należało powtórzyć zadanie 5 z listy 1., ale zastosować drobne zmiany w wektorze x. Wektory te, przed zmianą, prezentowały się następująco:

$$x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$$
$$y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]$$

W wektorze x zmieniłem dane poprzez usunięcie cyfry 9 z x_4 oraz cyfry 7 z x_5, przez co powstał wektor $\tilde{x}$ o postaci:

$$\tilde{x} = [2.718281828, -3.141592654, 1.414213562, 0.577215664, 0.301029995]$$

Za pomocą algorytmów z zadania 5 z listy 1., obliczyłem iloczyn skalarny wektorów $x \cdot y$ oraz $\tilde{x} \cdot y$. Obliczenia wykonywałem zarówno w arytmetyce Float32, jak i Float64:

	$x \cdot y$ (Float32)	$\tilde{x} \cdot y$ (Float32)	$x \cdot y$ (Float64)	$\tilde{x} \cdot y$ (Float64)
algorytm a)	-0.4999443	-0.4999443	$1.0251881368296672 \cdot 10^{-10}$	-0.004296342739891585
algorytm b)	-0.4543457	–0.4543457	$-1.5643308870494366 \cdot 10^{-10}$	-0.004296342998713953
algorytm c)	-0.5	-0.5	0.0	-0.004296342842280865
algorytm d)	-0.5	-0.5	0.0	-0.004296342842280865

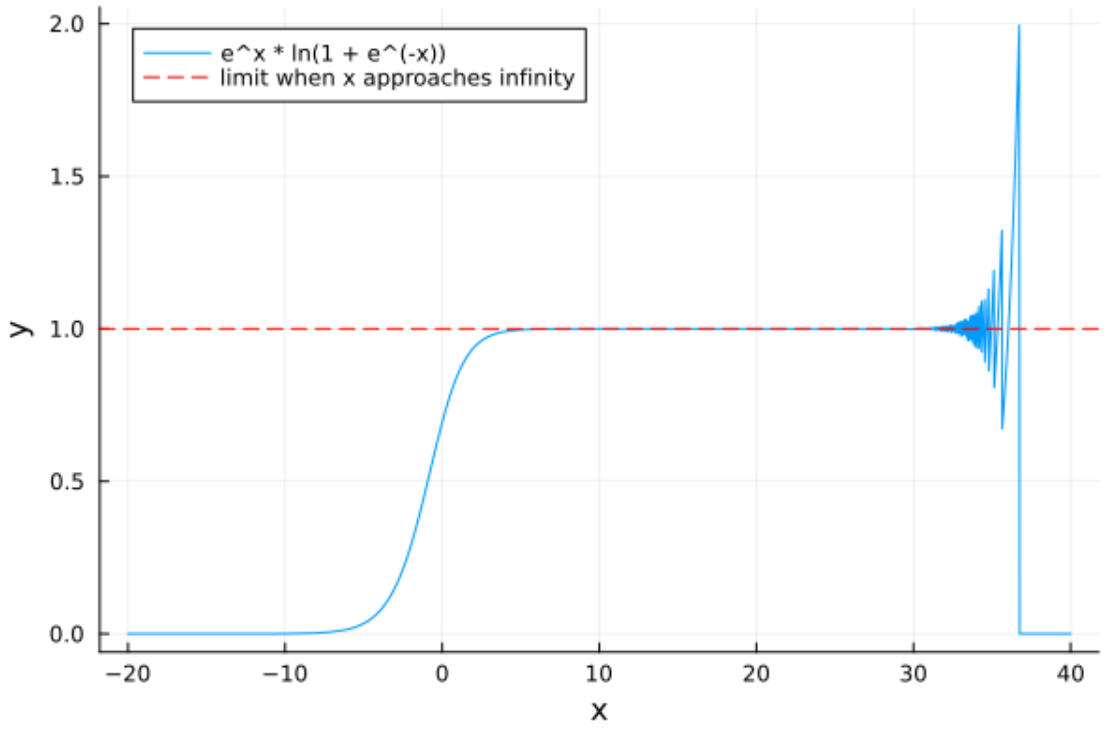
Dla Float32 precyzja wynosi około 7 cyfr po przecinku, a zmiany danych dotyczyły usunięcia 10. cyfr po przecinku. Z tego powodu dla tej precyzji zmiennoprzecinkowej wyniki są identyczne zarówno dla $x \cdot y$, jak i $\tilde{x} \cdot y$. Natomiast w przypadku Float64, gdzie precyzja wynosi około 15 cyfr po przecinku, powyższe zmiany znacząco wpływają na wyniki działania algorytmów. Nasuwa się przez to wniosek, że zadanie jest źle uwarunkowane, gdyż zaimplementowanie małych zmian w danych wejściowych powoduje dużą zmianę w wynikach.

2 Wizualizacja funkcji $f(x) = e^x \cdot ln(1 + e^{-x})$

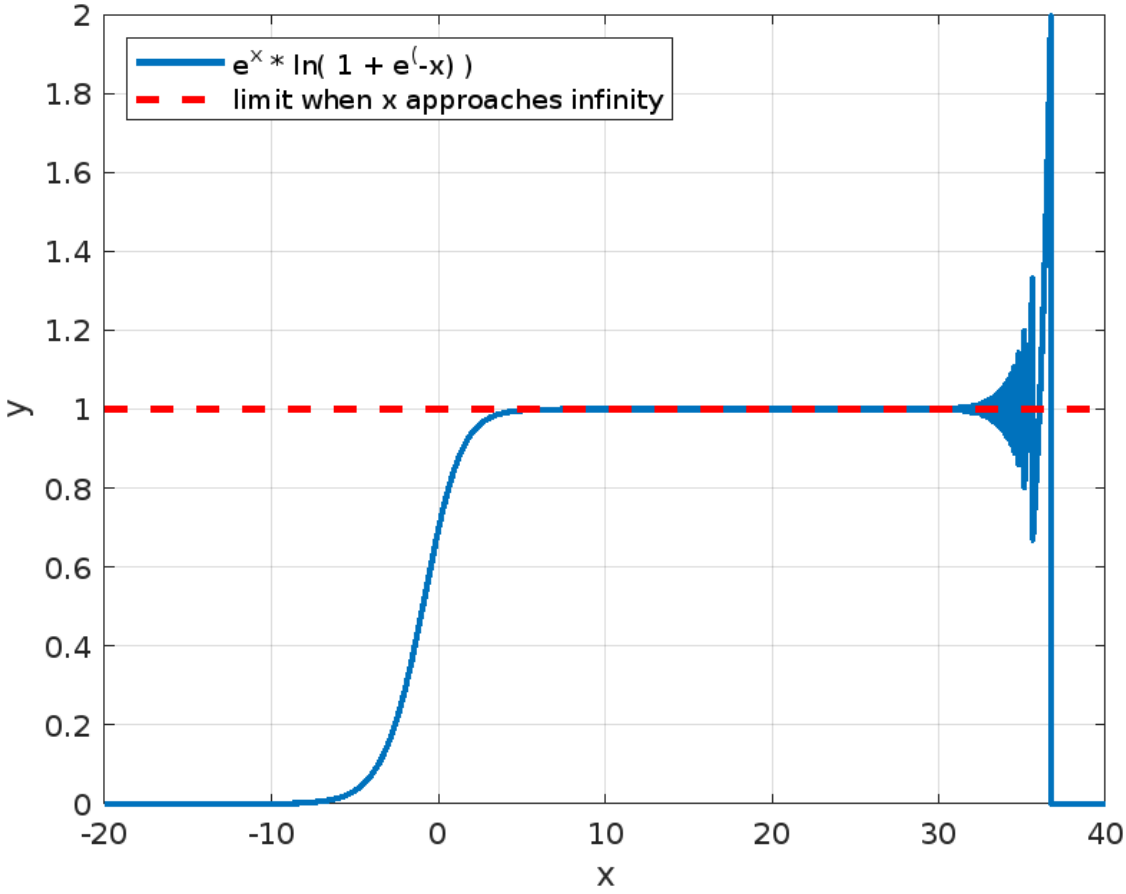
Zadanie polegało na narysowaniu powyższej funkcji w dwóch różnych programach do wizualizacji, oraz porównanie wyników z rzeczywistą wartością granicy $\lim_{x \to \infty} f(x)$.

Wykresy narysowałem w programach: **Julia** (a) oraz **MATLAB** (b), a granicę w nieskończoności dla tej funkcji obliczyłem w następujący sposób:

$$\lim_{x \to \infty} e^x \cdot ln(1 + e^{-x}) = \lim_{x \to \infty} \frac{ln(1+e^{-x})}{e^{-x}} \overset{de\ l'Hôpital}{=} \lim_{x \to \infty} \frac{(ln(1+e^{-x}))'}{(e^{-x})'} = \lim_{x \to \infty} \frac{-1}{e^x+1} \cdot \frac{-1}{e^{-x}} = \lim_{x \to \infty} \frac{1}{1+e^{-x}} = \mathbf{1}$$



(a) Wykres przy użyciu Plots w **Julii**



(b) Wykres w **MATLABie**

Jak można zauważyć na powyższych wykresach, dla wartości $x > 30$ wyniki zostają stanowczo zaburzone i odbiegają od rzeczywistej wartości granicy w nieskończoności. Wynika to z ograniczonej precyzji arytmetyki zmiennopozycyjnej i dla $x \approx 37$ wartości spadają do zera. Wartość e^{-x} dla $x \geq 37$ zaczyna przekraczać epsilon maszynowy, przez co $ln(1 + e^{-x}) \approx ln(1) = 0$.

3 Rozwiązywanie układów liniowych

W zadaniu chodziło o znalezienie rozwiązania układu równań liniowych $\mathbf{Ax} = \mathbf{b}$ dla danej macierzy współczynników $\mathbf{A} \in \mathbb{R}^{n \times n}$ oraz wektora prawych stron $\mathbf{b} \in \mathbb{R}^n$. Dla macierzy Hilberta oraz macierzy wygenerowanej losowo należało porównać rozwiązania otrzymane za pomocą algorytmu eliminacji Gaussa oraz $x = inv(A) \cdot b$. Moje wyniki prezentują się następująco:

n	c	cond(A)	rank(A)	Gauss	Macierz odwrotna
5	1.0	1.0000000000000007	5	9.930136612989092e-17	1.4895204919483638e-16
5	10.0	9.999999999999988	5	4.328446199157272e-16	3.2177320244274193e-16
5	1000.0	1000.000000000067	5	1.4375223285337227e-14	1.18896540830928e-14
5	1.0e7	9.999999991669139e6	5	4.188248723384368e-10	3.577411597737172e-10
5	1.0e12	9.999504753053225e11	5	6.063797489123467e-5	7.498573227115243e-5
5	1.0e16	7.583290350678896e15	4	0.23639748942549696	0.1996089927833914
10	1.0	1.0000000000000009	10	3.020133145511626e-16	2.673771110915334e-16
10	10.0	9.999999999999998	10	5.747482092719373e-16	4.723343357811471e-16
10	1000.0	1000.0000000000518	10	6.39665764893529e-15	7.149698361352146e-15
10	1.0e7	1.0000000009263769e7	10	7.169785619164074e-11	3.904688598413118e-11
10	1.0e12	1.0000568724817673e12	10	8.440327275802296e-6	3.6189395830174018e-6
10	1.0e16	9.99637417633785e15	9	0.4203185381836076	0.4064508968411421
20	1.0	1.0000000000000018	20	5.376277206893598e-16	5.063396036227354e-16
20	10.0	10.000000000000005	20	6.596257219332524e-16	4.637756528637166e-16
20	1000.0	999.9999999999653	20	3.085796942719421e-14	3.0111737320537164e-14
20	1.0e7	1.0000000001357146e7	20	2.8380727387857127e-10	3.550488394222987e-10
20	1.0e12	1.0000194143509363e12	20	3.869286679663308e-5	3.733622768961989e-5
20	1.0e16	2.2052859015268796e16	19	0.2002878701481808	0.1552113918882084

Tabela 1: Wyniki dla macierzy losowych **matcond(n,c)**

n	cond(A)	rank(A)	Gauss	Macierz odwrotna
1	1.0	1	0.0	0.0
2	19.28147006790397	2	5.661048867003676e-16	1.4043333874306803e-15
3	524.0567775860644	3	8.022593772267726e-15	0.0
4	15513.73873892924	4	4.137409622430382e-14	0.0
5	476607.2502425855	5	1.6828426299227195e-12	3.3544360584359632e-12
6	1.4951058642254734e7	6	2.618913302311624e-10	2.0163759404347654e-10
7	4.753673567446793e8	7	1.2606867224171548e-8	4.713280397232037e-9
8	1.5257575538060041e10	8	6.124089555723088e-8	3.07748390309622e-7
9	4.9315375594102344e11	9	3.8751634185032475e-6	4.541268303176643e-6
10	1.602441698742836e13	10	8.67039023709691e-5	0.0002501493411824886
11	5.222701316549833e14	10	0.00015827808158590435	0.007618304284315809
12	1.7515952300879806e16	11	0.13396208372085344	0.258994120804705
13	3.1883950689209334e18	11	0.11039701117868264	5.331275639426837
14	6.200786281355982e17	11	1.4554087127659643	8.71499275104814
15	3.67568286586649e17	12	4.696668350857427	7.344641453111494
16	7.046389953630175e17	12	54.15518954564602	29.84884207073541
17	1.249010044779401e18	12	13.707236683836307	10.516942378369349
18	2.2477642911280653e18	12	10.257619124632317	24.762070989128866
19	6.472700911391398e18	13	102.15983486270827	109.94550732878284
20	1.1484020388436145e18	13	108.31777346206205	114.34403152557572
21	3.2902428208431683e18	13	44.089455838364245	34.52041154914292
22	1.093638219471544e19	13	17.003425713362045	102.60161611995359
23	6.313779002744782e17	13	25.842511917947366	22.272314298730727
24	2.1608428256899587e18	13	39.638573210187644	43.34914763015038
25	1.3309197502927598e18	13	7.095757204652332	21.04404299195525
26	5.825077809111695e18	14	63.80426636186403	100.78434642499187
27	4.414670357556845e18	14	27.43309009053957	35.68974530952139
28	5.889050001520599e18	14	276.91498822022265	290.1167291705239
29	8.060274133463556e18	14	60.095450394724104	43.40383683199056
30	5.512691295390715e18	14	24.80615905441871	59.97231132227779

Tabela 2: Wyniki dla macierzy Hilberta **hilb(n)**

Zacznijmy od wyników dla macierzy Hilberta. Patrząc na kolumnę "cond(A)"można od razu stwierdzić, że tak generowane macierze są bardzo źle uwarunkowane i wraz ze wzrostem n szybko rośnie wartość niestabilności numerycznej macierzy. Spośród dwóch algorytmów wykorzystanych do obliczenia rozwiązania układu równań, w tym przypadku metoda Gaussa osiąga wyniki trochę lepsze niż te zdobyte metodą odwracania macierzy. Natomiast warto zauważyć, że otrzymane błędy względne nadal są duże dla obu tych metod i że rosną wraz z rozmiarem macierzy Hilberta.

Jeśli chodzi o rozwiązania znalezione dla macierzy wygenerowanych losowo z odpowiednim wskaźnikiem uwarunkowania, to obie metody zwracają bardzo przybliżone do siebie błędy względne. Dla coraz większego wskaźnika uwarunkowania te błędy również rosną i tracona jest precyzja rozwiązania. Można wnioskować, że obie metody są poprawne i generują dobre wyniki, lecz nie są w stanie dobrze działać dla macierzy źle uwarunkowanych.

4 Wielomian Wilkinsona

Zadanie polegało na obliczeniu 20 pierwiastków dla wielomianu Wilkinsona, który przyjmuje formy:

- **Postaci normalnej:**

$$P(x) = x^{20} - 210x^{19} + 20615x^{18} - 1256850x^{17} + 53327946x^{16} - 1672280820x^{15} + 40171771630x^{14} - 756111184500x^{13} + 11310276995381x^{12} - 135585182899530x^{11} + 1307535010540395x^{10} - 10142299865511450x^{9} + 63030812099294896x^{8} - 311333643161390640x^{7} + 1206647803780373360x^{6} - 3599979517947607200x^{5} + 8037811822645051776x^{4} - 12870931245150988800x^{3} + 13803759753640704000x^{2} - 8752948036761600000x + 2432902008176640000$$

- **Postaci kanonicznej:**

$$p(x) = (x-20)(x-19)(x-18)(x-17)(x-16)(x-15)(x-14)(x-13)(x-12)(x-11)(x-10)(x-9)(x-8)(x-7)(x-6)(x-5)(x-4)(x-3)(x-2)(x-1)$$

Do wyliczenia pierwiastków powyższego wielomianu skorzystałem z funkcji **roots** dla postaci normalnej wielomianu Wilkinsona i otrzymałem poniższe wyniki:

k	z_k	$\lvert P(z_k)\rvert$	$\lvert p(z_k)\rvert$	$\lvert z_k$ - k$\rvert$
1	0.9999999999996989	35696.50964788257	36626.425482422805	3.0109248427834245e-13
2	2.0000000000283182	176252.60026668405	181303.93367257662	2.8318236644508943e-11
3	2.9999999995920965	279157.6968824087	290172.2858891686	4.0790348876384996e-10
4	3.9999999837375317	3.0271092988991085e6	2.0415372902750901e6	1.626246826091915e-8
5	5.000000665769791	2.2917473756567076e7	2.0894625006962188e7	6.657697912970661e-7
6	5.999989245824773	1.2902417284205095e8	1.1250484577562995e8	1.0754175226779239e-5
7	7.000102002793008	4.805112754602064e8	4.572908642730946e8	0.00010200279300764947
8	7.999355829607762	1.6379520218961136e9	1.5556459377357383e9	0.0006441703922384079
9	9.002915294362053	4.877071372550003e9	4.687816175648389e9	0.002915294362052734
10	9.990413042481725	1.3638638195458128e10	1.2634601896949205e10	0.009586957518274986
11	11.025022932909318	3.585631295130865e10	3.300128474498415e10	0.025022932909317674
12	11.953283253846857	7.533332360358197e10	7.388525665404988e10	0.04671674615314281
13	13.07431403244734	1.9605988124330817e11	1.8476215093144193e11	0.07431403244734014
14	13.914755591802127	3.5751347823104315e11	3.5514277528420844e11	0.08524440819787316
15	15.075493799699476	8.21627123645597e11	8.423201558964254e11	0.07549379969947623
16	15.946286716607972	1.5514978880494067e12	1.570728736625802e12	0.05371328339202819
17	17.025427146237412	3.694735918486229e12	3.3169782238892363e12	0.025427146237412046
18	17.99092135271648	7.650109016515867e12	6.34485314179128e12	0.009078647283519814
19	19.00190981829944	1.1435273749721195e13	1.228571736671966e13	0.0019098182994383706
20	19.999809291236637	2.7924106393680727e13	2.318309535271638e13	0.00019070876336257925

Tabela 3: Wyniki dla wielomianu Wilkinsona

Pierwiastki wielomianu Wilkinsona obliczone za pomocą funkcji **roots** różnią się od rzeczywistych wartości maksymalnie o około rząd setnych. Mimo że wyniki mogą się wydawać bardzo zbliżone do faktycznych pierwiastków, to te różnice znacząco wpływają na działanie programu i podstawienie obliczonych pierwiastków do postaci normalnej P(x) oraz do postaci kanonicznej p(x) skutkuje ogromnymi wartościami, zamiast samymi zerami.

Na podstawie tych wyników można wysunąć wniosek, iż problem wyznaczania pierwiastków dla wielomianu Wilkinsona jest zadaniem źle uwarunkowanym. Powodem jest wpływ niewielkich różnic w wartościach pierwiastków na ostateczne wyniki, które różnią się od faktycznych wartości o rząd wielkości sięgający nawet 10^{13}, a przypomnijmy, że precyzja arytmetyki Float64 ogranicza się do około 15 cyfr znaczących.

k	z_k	|P(z_k)|	|p(z_k)|	|z_k - k|
1	0.9999999999998357 + 0.0im	20259.872313418207	19987.872313406835	1.6431300764452317e-13
2	2.0000000000550373 + 0.0im	346541.4137593836	352369.4138087958	5.503730804434781e-11
3	2.99999999660342 + 0.0im	2.2580597001197007e6	2.4162415582518433e6	3.3965799062229962e-9
4	4.000000089724362 + 0.0im	1.0542631790395478e7	1.1263702300292023e7	8.972436216225788e-8
5	4.99999857388791 + 0.0im	3.757830916585153e7	4.475744423806908e7	1.4261120897529622e-6
6	6.000020476673031 + 0.0im	1.3140943325569446e8	2.1421031658039317e8	2.0476673030955794e-5
7	6.99960207042242 + 0.0im	3.939355874647618e8	1.7846173427860644e9	0.00039792957757978087
8	8.007772029099446 + 0.0im	1.184986961371896e9	1.8686972170009857e10	0.007772029099445632
9	8.915816367932559 + 0.0im	2.2255221233077707e9	1.3746309775142993e11	0.0841836320674414
10	10.095455630535774 - 0.6449328236240688im	1.0677921232930157e10	1.490069535200058e12	0.6519586830380407
11	10.095455630535774 + 0.6449328236240688im	1.0677921232930157e10	1.490069535200058e12	1.1109180272716561
12	11.793890586174369 - 1.6524771364075785im	3.1401962344429485e10	3.2962792355717145e13	1.665281290598479
13	11.793890586174369 + 1.6524771364075785im	3.1401962344429485e10	3.2962792355717145e13	2.0458202766784277
14	13.992406684487216 - 2.5188244257108443im	2.157665405951858e11	9.546022365750216e14	2.518835871190904
15	13.992406684487216 + 2.5188244257108443im	2.157665405951858e11	9.546022365750216e14	2.7128805312847097
16	16.73074487979267 - 2.812624896721978im	4.850110893921027e11	2.742106076928478e16	2.9060018735375106
17	16.73074487979267 + 2.812624896721978im	4.850110893921027e11	2.742106076928478e16	2.825483521349608
18	19.5024423688181 - 1.940331978642903im	4.557199223869993e12	4.2524858765203725e17	2.4540214463129764
19	19.5024423688181 + 1.940331978642903im	4.557199223869993e12	4.2524858765203725e17	2.0043294443099486
20	20.84691021519479 + 0.0im	8.756386551865696e12	1.37437435599976e18	0.8469102151947894

Tabela 4: Wyniki dla **zmodyfikowanego** wielomianu Wilkinsona

Eksperyment został powtórzony dla lekko zmodyfikowanego wielomianu Wilkinsona, w którym to zmieniłem współczynnik przy x^{19} z -210 na $-210 - 2^{-23}$. Sam proces generowania wyników nie zmienił się, natomiast patrząc na powyższą tabelę można zauważyć, że wprowadzenie nawet tak (mogłoby się wydawać) minimalnej modyfikacji drastycznie wpływa na przebieg działania programu.

Dla postaci kanonicznej wyniki różnią się od zera o wielkości rzędu aż 10^{18}, czyli o wielkości około 10 tysięcy razy większe niż te uzyskane dla niezmodyfikowanego wielomianu Wilkinsona! Dla arytmetyki Float64 współczynniki wielomianu Wilkinsona są zbyt mało precyzyjnie zapisywane. Wprowadzenie małych zmian w danych wejściowych znacząco wpłynęło na wartości danych wyjściowych, czyli zadanie to jest źle uwarunkowane.

5 Model wzrostu populacji

Rozważamy równanie rekurencyjne w postaci:

$$p_{n+1} := p_n + rp_n(1 - p_n), \text{ gdzie:}$$

r - pewna dana stała; $rp_n(1 - p_n)$ - czynnik wzrostu populacji; p_0 - wielkość populacji stanowiąca procent maksymalnej wielkości populacji dla danego stanu środowiska.

W ramach eksperymentu należało wykonać 40 iteracji tego równania dla $r = 3$ oraz $p = 0.01$ i przeprowadzić obliczenia w arytmetykach Float32 oraz Float64. Dodatkowo, w arytmetyce Float32 w 10. iteracji trzeba było dokonać "obcięcia"do 3 cyfr znaczących i porównać otrzymane wyniki.

Wyniki moich eksperymentów umieściłem w poniższej tabeli:

n	Float32	obcięty Float32	Float64
1	0.0397	0.0397	0.039700000000000006
2	0.15407173	0.15407173	0.15407173000000005
3	0.5450726	0.5450726	0.5450726260444214
4	1.2889781	1.2889781	1.2889780011888008
5	0.1715188	0.1715188	0.17151914210917463
6	0.5978191	0.5978191	0.5978201201070967
7	1.3191134	1.3191134	1.3191137924137961
8	0.056273222	0.056273222	0.05627157764626167
9	0.21559286	0.21559286	0.21558683923264893
10	**0.7229306**	**0.722**	**0.7229143011796237**
11	1.3238364	1.3241479	1.3238419441684237
12	0.037716985	0.036488533	0.037695297254799254
13	0.14660023	0.1419599	0.14651838271381398
14	0.52192605	0.5073818	0.5216706214360409
15	1.2704837	1.2572184	1.2702617739357684
16	0.2395482	0.28707945	0.24035217277573784
17	0.7860428	0.901074	0.788101190228897
18	1.2905813	1.168493	1.2890943027949757
19	0.16552472	0.57784426	0.17108484668450918
20	0.5799036	1.3096651	0.5965293124428507
21	1.3107499	0.092992425	1.3185755879607823
22	0.08880377	0.34602693	0.05837760834476091
23	0.33155674	1.0249038	0.22328659791088076
24	0.9964373	0.94833183	0.7435756772236771
25	1.0070873	1.0953275	1.3155883456187583
26	0.9856746	0.78208303	0.07003529709132894
27	1.0280352	1.2933705	0.26542635984930363
28	0.9415718	0.15506029	0.8503519818886585
29	1.1066148	0.54811007	1.2321124482487258
30	0.75267017	1.2911663	0.3741465376064961
31	1.3111435	0.1633339	1.0766292556171968
32	0.08728206	0.5733017	0.8291253603162694
33	0.32627377	1.3071823	1.254154851906327
34	0.98573136	0.102552414	0.297906229944765
35	1.0279264	0.37865865	0.9253805542593504
36	0.94180745	1.0844874	1.132534706433374
37	1.106226	0.8096107	0.6822342419051102
38	0.7536962	1.2720344	1.3326062851369196
39	1.3106109	0.23392308	0.0029066069884153833
40	0.089340806	0.7715323	0.011601082861106218

Jak można zauważyć w powyższej tabeli - ostateczne wyniki w każdej kolumnie znacząco się od siebie różnią. Dokonanie obcięcia w 10. iteracji (czyli zmienienie wartości o około 0.001) skutkuje diametralnie innymi danymi wyjściowymi, co może świadczyć o złym uwarunkowaniu zadania.

Dodatkowo, obliczając równanie rekurencyjne w kontekście modelu wzrostu populacji, wykonujemy działanie potęgowania, co bardzo szybko zmusza komputer do wykonywania zaokrągleń z uwagi na ograniczoną precyzję obliczeń. Tracone są przez to cyfry znaczące, co skutkuje dużą i błyskawiczną kumulacją błędów przy obliczaniu kolejnych wyrazów równania.

6 Kolejne równanie rekurencyjne i iteracje graficzne

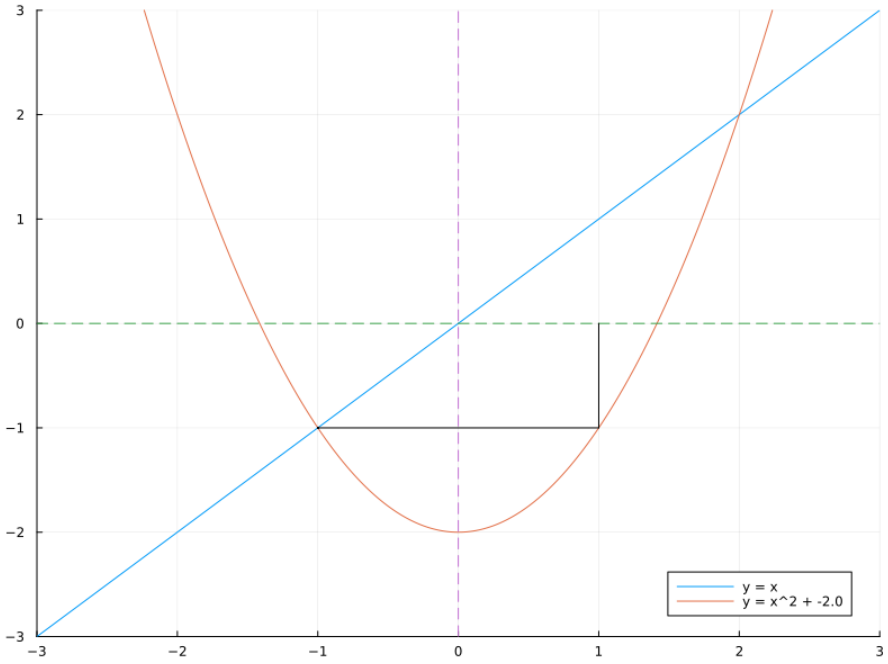
W zadaniu należało rozważyć równanie rekurencyjne w postaci:

$$x_{n+1} := x_n^2 + c, \text{ dla } n = 0, 1, ...,$$

gdzie:
c - pewna dana stała.

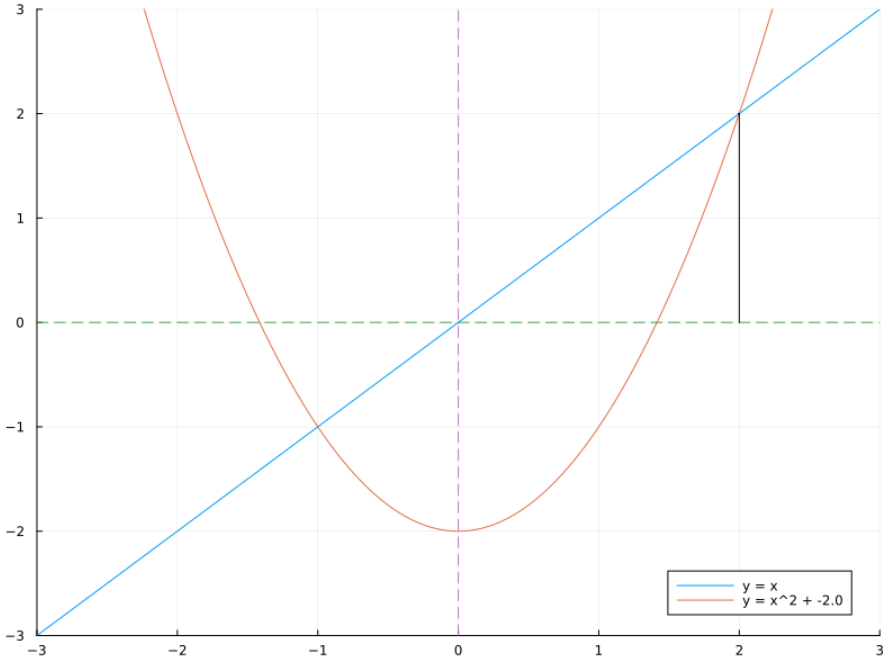
Dla podanych na liście wartości dla stałej c oraz x_0 trzeba było wykonać w języku Julia , w arytmetyce Float64, dokładnie 40 iteracji powyższego wyrażenia oraz wyciągnąć odpowiednie wnioski z wyników. Dodatkowo, napisałem program o nazwie **graphic_iterations.jl**, w którym wizualizuję iteracje graficzne powyższego równania rekurencyjnego dla wszystkich podpunktów dla 40 iteracji. Oprócz wykresów, stworzyłem dwie tabele z konkretnymi wartościami po 40 iteracjach dla określonych danych wejściowych.

- $c = -2$:
  - ▶ $x_0 = 1$:



Jak można zauważyć na wykresie, dla każdej iteracji wartość x wyniesie -1.0, co zgadza się z danymi umieszczonymi w Tabeli 5.

  - ▶ $x_0 = 2$



W tym przypadku dla każdej iteracji wartość x osiągnie 2.0, co również zgadza się z otrzymanymi wynikami.

- $x_0 = 1.99999999999999$

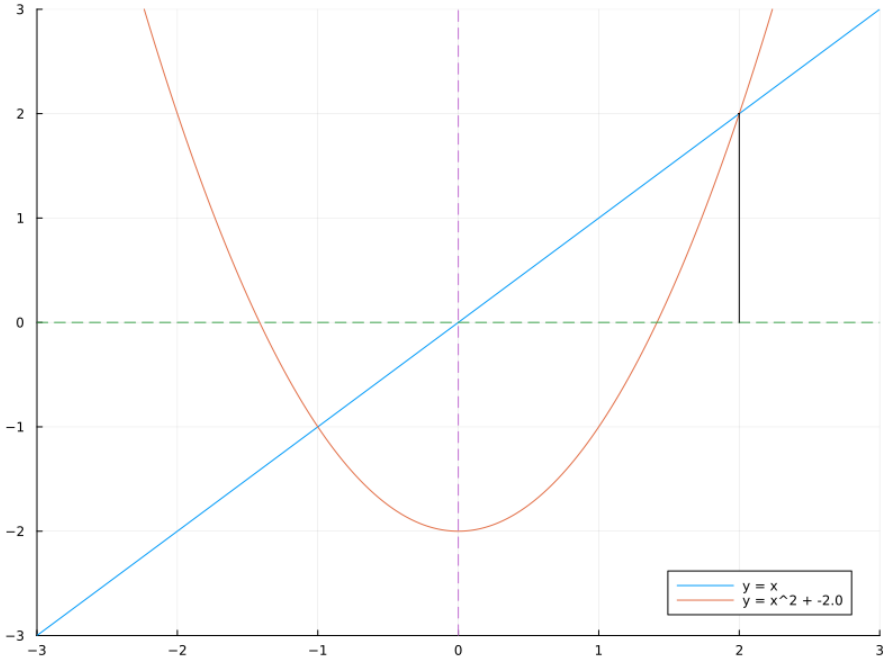
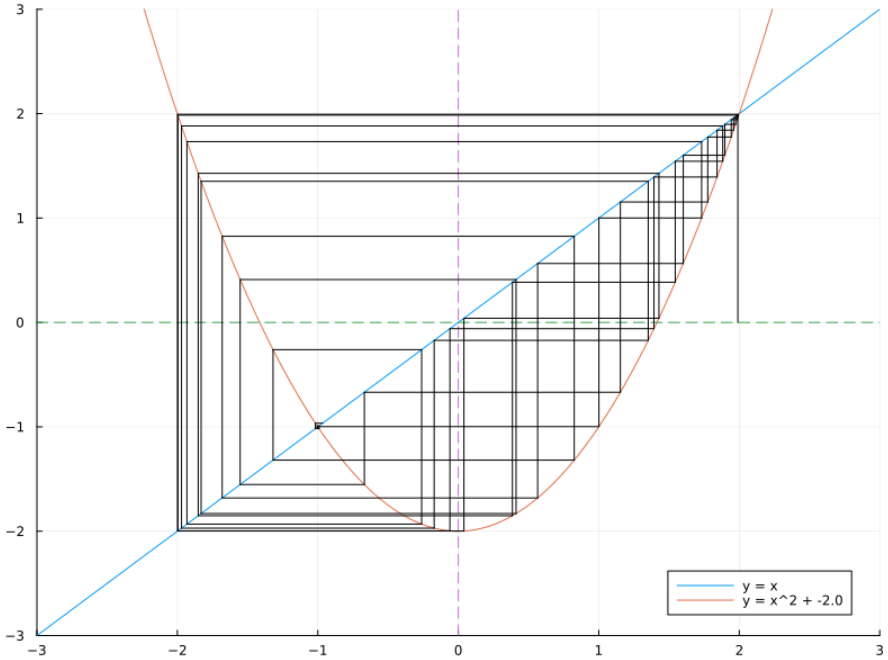


Tabela 5. wskazuje na to, że dla $x_0 = 1.99999999999999$ wartości znacznie różnią się od $x_0 = 2$, lecz oba wykresy są identyczne. Z matematycznego punktu widzenia, osiągniemy punkt stały $\alpha = 2.0 \iff x_0 = 2.0$. Moją tezą jest, iż punkt x_0 w tym podpunkcie zawiera za dużo cyfr znaczących, przez co podczas tworzenia wykresu dochodzi do zaokrągleń i w efekcie - takich samych wykresów.

- $x_0 = 1.99$



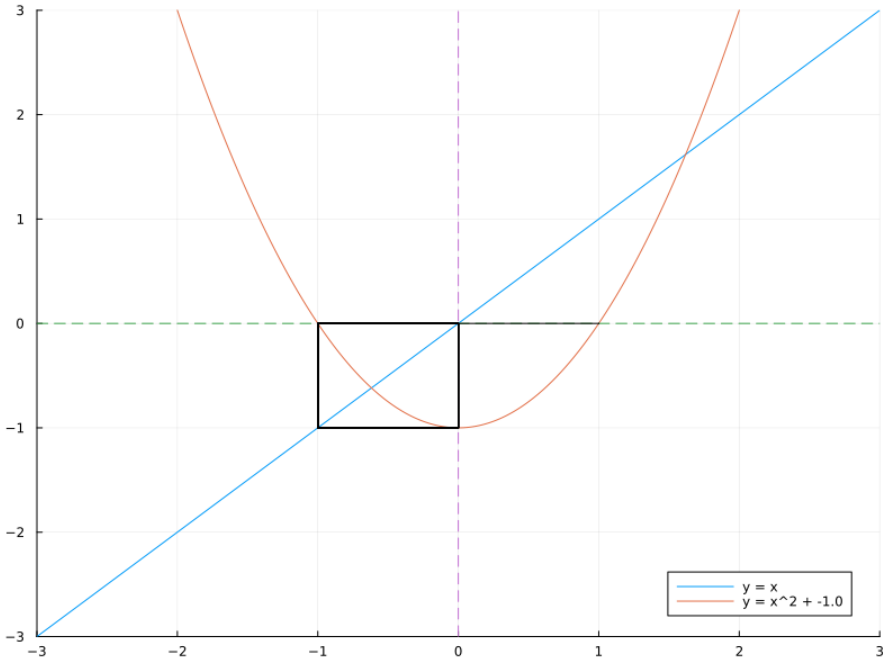
Z uwagi na poprzedni podpunkt postanowiłem zmniejszyć liczbę x_0 i przyrównać ją do liczby 1.99, co okazało się dobrym pomysłem w kontekście udowodnienia mojej tezy. Jak można wnioskować po powyższym wykresie dla $x_0 = 1.99$, funkcja dąży do punktu stałego $\alpha = -1.0$!

n	$x_0 = 1$	$x_0 = 2$	$x_0 = 1.99999999999999$
1	-1.0	2.0	1.99999999999996
2	-1.0	2.0	1.9999999999998401
3	-1.0	2.0	1.9999999999993605
4	-1.0	2.0	1.999999999997442
5	-1.0	2.0	1.9999999999897682
6	-1.0	2.0	1.9999999999590727
7	-1.0	2.0	1.999999999836291
8	-1.0	2.0	1.9999999993451638
9	-1.0	2.0	1.9999999973806553
10	-1.0	2.0	1.999999989522621
11	-1.0	2.0	1.9999999580904841
12	-1.0	2.0	1.9999998323619383
13	-1.0	2.0	1.9999993294477814
14	-1.0	2.0	1.9999973177915749
15	-1.0	2.0	1.9999892711734937
16	-1.0	2.0	1.9999570848090826
17	-1.0	2.0	1.999828341078044
18	-1.0	2.0	1.9993133937789613
19	-1.0	2.0	1.9972540465439481
20	-1.0	2.0	1.9890237264361752
21	-1.0	2.0	1.9562153843260486
22	-1.0	2.0	1.82677862987391
23	-1.0	2.0	1.3371201625639997
24	-1.0	2.0	-0.21210967086482313
25	-1.0	2.0	-1.9550094875256163
26	-1.0	2.0	1.822062096315173
27	-1.0	2.0	1.319910282828443
28	-1.0	2.0	-0.2578368452837396
29	-1.0	2.0	-1.9335201612141288
30	-1.0	2.0	1.7385002138215109
31	-1.0	2.0	1.0223829934574389
32	-1.0	2.0	-0.9547330146890065
33	-1.0	2.0	-1.0884848706628412
34	-1.0	2.0	-0.8152006863380978
35	-1.0	2.0	-1.3354478409938944
36	-1.0	2.0	-0.21657906398474625
37	-1.0	2.0	-1.953093509043491
38	-1.0	2.0	1.8145742550678174
39	-1.0	2.0	1.2926797271549244
40	-1.0	2.0	-0.3289791230026702

Tabela 5: Wyniki dla $c = -2$ oraz odpowiednich wartości x_0

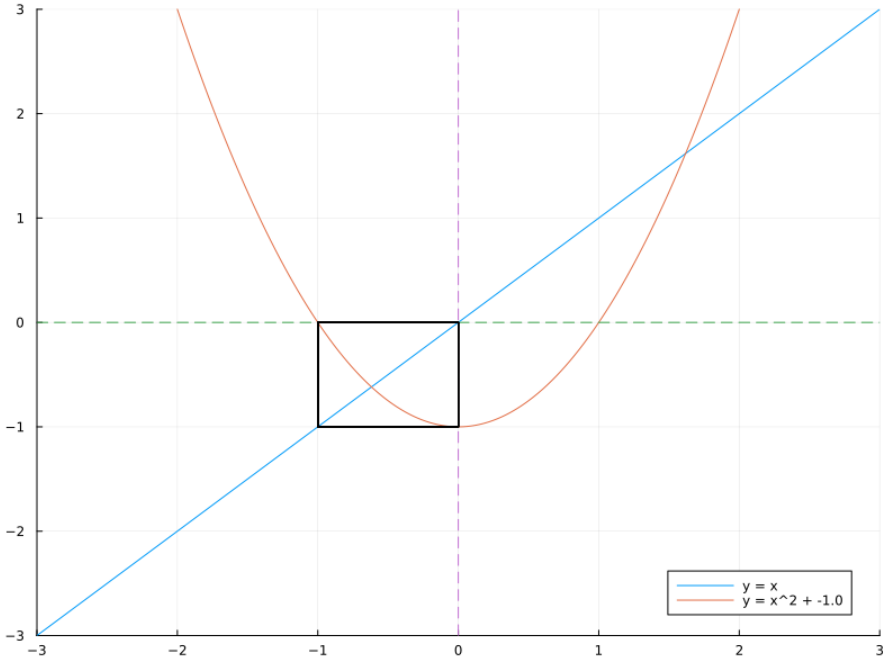
- $c = -1$

  - $x_0 = 1$



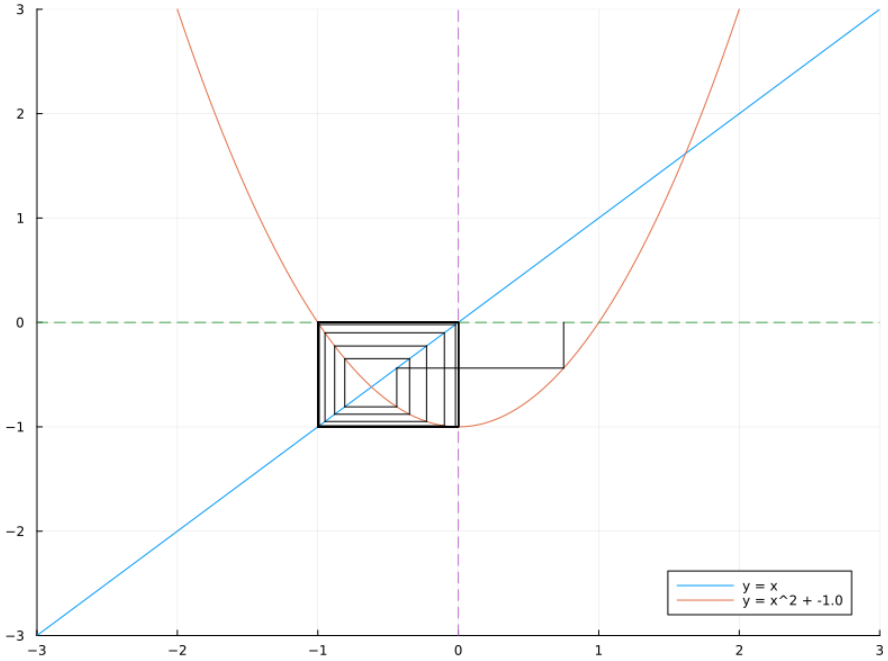
Łatwo jest zauważyć, że dla $x_0 = 1$ przybliżenia przyjmują na zmianę wartości 0.0 oraz -1.0, co zgadza się z danymi uzyskanymi w Tabeli 6.

  - $x_0 = -1$



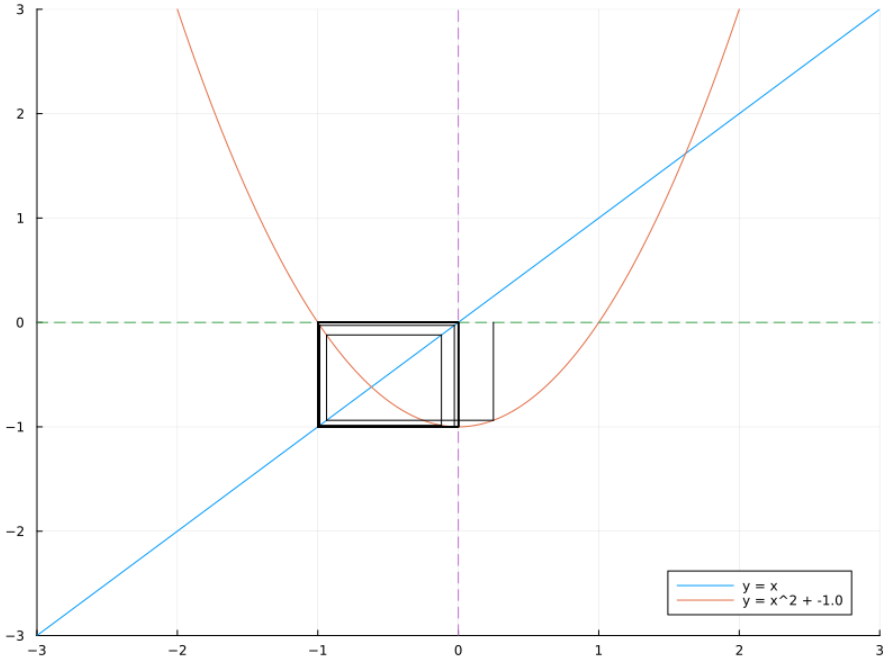
Dokładnie identyczna sytuacja występuje dla $x_0 = -1$ (zgadza się to również z wynikami w Tabeli 6.)

- $x_0 = 0.75$



Dla $x_0 = 0.75$ zachodzi ciekawa sytuacja: na początku (jak można zobaczyć na wykresie) wartości oscylują w zakresie -1 oraz 0. Później, z uwagi na ograniczoną precyzję arytmetyki, uzyskiwane wartości ulegają zaokrągleniu, przez co przykład ten staje się identyczny jak dla np. $x_0 = 1$.

- $x_0 = 0.25$



Jest to identyczny przypadek jak podpunkt wyżej, lecz tutaj ta transformacja zachodzi trochę szybciej.

n	$x_0 = 1$	$x_0 = -1$	$x_0 = 0.75$	$x_0 = 0.25$
1	0.0	0.0	-0.4375	-0.9375
2	-1.0	-1.0	-0.80859375	-0.12109375
3	0.0	0.0	-0.3461761474609375	-0.9853363037109375
4	-1.0	-1.0	-0.8801620749291033	-0.029112368589267135
5	0.0	0.0	-0.2253147218564956	-0.9991524699951226
6	-1.0	-1.0	-0.9492332761147301	-0.0016943417026455965
7	0.0	0.0	-0.0989561875164966	-0.9999971292061947
8	-1.0	-1.0	-0.9902076729521999	-5.741579369278327e-6
9	0.0	0.0	-0.01948876442658909	-0.9999999999670343
10	-1.0	-1.0	-0.999620188061125	-6.593148249578462e-11
11	0.0	0.0	-0.0007594796206411569	-1.0
12	-1.0	-1.0	-0.9999994231907058	0.0
13	0.0	0.0	-1.1536182557003727e-6	-1.0
14	-1.0	-1.0	-0.9999999999986692	0.0
15	0.0	0.0	-2.6616486792363503e-12	-1.0
16	-1.0	-1.0	-1.0	0.0
17	0.0	0.0	0.0	-1.0
18	-1.0	-1.0	-1.0	0.0
19	0.0	0.0	0.0	-1.0
20	-1.0	-1.0	-1.0	0.0
21	0.0	0.0	0.0	-1.0
22	-1.0	-1.0	-1.0	0.0
23	0.0	0.0	0.0	-1.0
24	-1.0	-1.0	-1.0	0.0
25	0.0	0.0	0.0	-1.0
26	-1.0	-1.0	-1.0	0.0
27	0.0	0.0	0.0	-1.0
28	-1.0	-1.0	-1.0	0.0
29	0.0	0.0	0.0	-1.0
30	-1.0	-1.0	-1.0	0.0
31	0.0	0.0	0.0	-1.0
32	-1.0	-1.0	-1.0	0.0
33	0.0	0.0	0.0	-1.0
34	-1.0	-1.0	-1.0	0.0
35	0.0	0.0	0.0	-1.0
36	-1.0	-1.0	-1.0	0.0
37	0.0	0.0	0.0	-1.0
38	-1.0	-1.0	-1.0	0.0
39	0.0	0.0	0.0	-1.0
40	-1.0	-1.0	-1.0	0.0

Tabela 6: Wyniki dla $c = -1$ oraz odpowiednich wartości x_0

To zadanie dowodzi, że iteracje graficzne stanowią silne i przydatne narzędzie do analizy zachowań pewnych funkcji matematycznych, zwłaszcza tych, które mają iteracyjne struktury. Iteracje graficzne są silnym narzędziem wizualizacyjnym, które pomagają intuicyjnie zrozumieć i analizować złożone struktury matematyczne i dynamiczne procesy.