Obliczenia naukowe

Laboratorium

Lista 2

Kinga Majcher
272354

Listopad 2024

1 Zadanie 1

1.1 Opis problemu

Problemem zadania jest powtórzenie zadania 5 z listy 1, tj. wykonanie iloczynu skalarnego wektorów, tym razem dla wektorów z usuniętą ostatnią 9 z x_4 i usuniętą ostatnią 7 zx_5, czyli dla wektorów

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

$$y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]$$

Prawidłowa wartość sumy wynosi -0.004296343192495245.

1.2 Rozwiązanie

Metody rozwiązania są tożsame z metodami z zadania 5 z listy 1.

1.2.1 Metoda 1

Inicjujemy wartość zmiennej *sum* na 0.0. Następnie w pętli *for i in 1:length(x)* zwiększamy wartość sumy o iloczyn $x[i] \cdot y[i]$ dla kolejnych i.

1.2.2 Metoda 2

Inicjujemy wartość zmiennej *sum* na 0.0. Następnie w pętli *for i = length(x):-1:1* zwiększamy wartość sumy o iloczyn $x[i] \cdot y[i]$ dla kolejnych i.

1.2.3 Metoda 3

Inicjujemy wartości zmiennych *positive_sum* oraz *negative_sum* na 0.0. Tworzymy tablicę, w której obliczamy poszczególne iloczyny $x[i] \cdot y[i]$ dla każdego i. Tworzymy tablicę *positive_products*, w której przechowujemy tylko iloczyny dodatnie posortowane malejąco oraz tablicę *negative_products*, w której przechowujemy tylko iloczyny ujemne posortowane rosnąco. Następnie w pętli zwiększamy *positive_sum* o kolejne wartości dodatnich iloczynów, a następnie w drugiej pętli zwiększamy *negative_sum* o kolejne wartości ujemnych iloczynów. Na samym końcu wartość *sum* ustawiamy na sumę *positive_sum* oraz *negative_sum*.

1.2.4 Metoda 4

Inicjujemy wartości zmiennych *positive_sum* oraz *negative_sum* na 0.0. Tworzymy tablicę, w której obliczamy poszczególne iloczyny $x[i] \cdot y[i]$ dla każdego i. Tworzymy tablicę *positive_products*, w której przechowujemy tylko iloczyny dodatnie posortowane rosnąco oraz tablicę *negative_products*, w której przechowujemy tylko iloczyny ujemne posortowane malejąco. Następnie w pętli zwiększamy *positive_sum* o kolejne wartości dodatnich iloczynów, a następnie w drugiej pętli zwiększamy *negative_sum* o kolejne wartości ujemnych iloczynów. Na samym końcu wartość *sum* ustawiamy na sumę *positive_sum* oraz *negative_sum*.

1.3 Wyniki

Typ	Wartość dla metody 1	Wartość dla metody 2	Wartość dla metody 3	Wartość dla metody 4
`Float32`	-0.4999443	-0.4543457	-0.5	-0.5
`Float64`	-0.004296342739891585	-0.004296342998713953	-0.004296342842280865	-0.004296342842280865

Tabela 1: Wyniki obliczania wartości iloczynu skalarnego wektorów $\tilde{x}$ i y dla czterech różnych metod i dla typów `Float32` i `Float64`

Typ	Wartość dla metody 1	Wartość dla metody 2	Wartość dla metody 3	Wartość dla metody 4
`Float32`	-0.4999443	-0.4543457	-0.5	-0.5
`Float64`	1.0251881368296672e-10	-1.5643308870494366e-10	0.0	0.0

Tabela 2: Wyniki obliczania wartości iloczynu skalarnego wektorów x i y z poprzedniego zadania dla czterech różnych metod i dla typów `Float32` i `Float64`

Typ	Wartość błędu względnego dla metody 1	Wartość błędu względnego dla metody 2	Wartość błędu względnego dla metody 3	Wartość błędu względnego dla metody 4
`Float32`	-11536.507538148926%	-10475.17248432669%	-11537.804002096218%	-11537.804002096218%
`Float64`	$-1.0534625357739804 \cdot 10^{-5}\%$	$-4.510377376253089 \cdot 10^{-6}\%$	$-8.151452627834422 \cdot 10^{-6}\%$	$-8.151452627834422 \cdot 10^{-6}\%$

Tabela 3: Wartości błędów względnych dla wektorów $\tilde{x}$ i y dla wszystkich metod i typów obliczone z wzoru $\frac{\lvert s-\tilde{s}\rvert}{s} \cdot 100\%$, gdzie s to rzeczywista wartość iloczynu, a $\tilde{s}$ to obliczona daną metodą wartość iloczynu skalarnego

Typ	Wartość błędu względnego dla metody 1	Wartość błędu względnego dla metody 2	Wartość błędu względnego dla metody 3	Wartość błędu względnego dla metody 4
`Float32`	-4.9668057e12%	-4.5137967e12%	-4.9673593e12%	-4.9673593e12%
`Float64`	-1118.4955313981627%	-1454.1186645165915%	-100.0%	-100.0%

Tabela 4: Wartości błędów względnych dla wektorów x i y z poprzedniego zadania dla wszystkich metod i typów obliczone z wzoru $\frac{\lvert s-\tilde{s}\rvert}{s} \cdot 100\%$, gdzie s to rzeczywista wartość iloczynu, a $\tilde{s}$ to obliczona daną metodą wartość iloczynu skalarnego

1.4 Interpretacja wyników oraz wnioski

Dla typu `Float32` wartości obliczonego iloczynu skalarnego zarówno dla wartości x i y z zadania 5 z listy 1 jak i dla wartości z tego zadania są takie same. Wynika to z małej precyzji tej arytmetyki i faktu, że obliczanie iloczynu skalarnego jest obarczone dużym błędem przybliżeń. Wartości błędów względnych dla obu typów zdecydowanie zmalały, względem wartości z poprzedniej listy. Niewielkie zmiany wartości rzędu 10^{-10} spowodowały, że rząd błędu względnego wyników zmienił się z 10^3 do 10^{-6} dla `Float64` oraz z 10^{12} do 10^4 dla `Float32`. Skoro tak mała zmiana danych ma tak ogromny wpływ na ostateczne wyniki i ich zgodność co do realnego wyniku to można powiedzieć, że zadanie jest źle uwarunkowane.

2 Zadanie 2

2.1 Opis problemu

Problemem zadania jest narysowanie wykresu funkcji $f(x) = e^x \cdot ln(1 + e^{-x})$ w co najmniej dwóch programach do wizualizacji, następnie obliczenie granicy $lim_{x\to\infty} f(x)$ i wyjaśnienie dlaczego wartości na wykresie funkcji odbiegają od wartości policzonej granicy.

2.2 Rozwiązanie

Wygenerowałam wykresy funkcji $f(x) = e^x \cdot ln(1 + e^{-x})$ na trzech stronach:

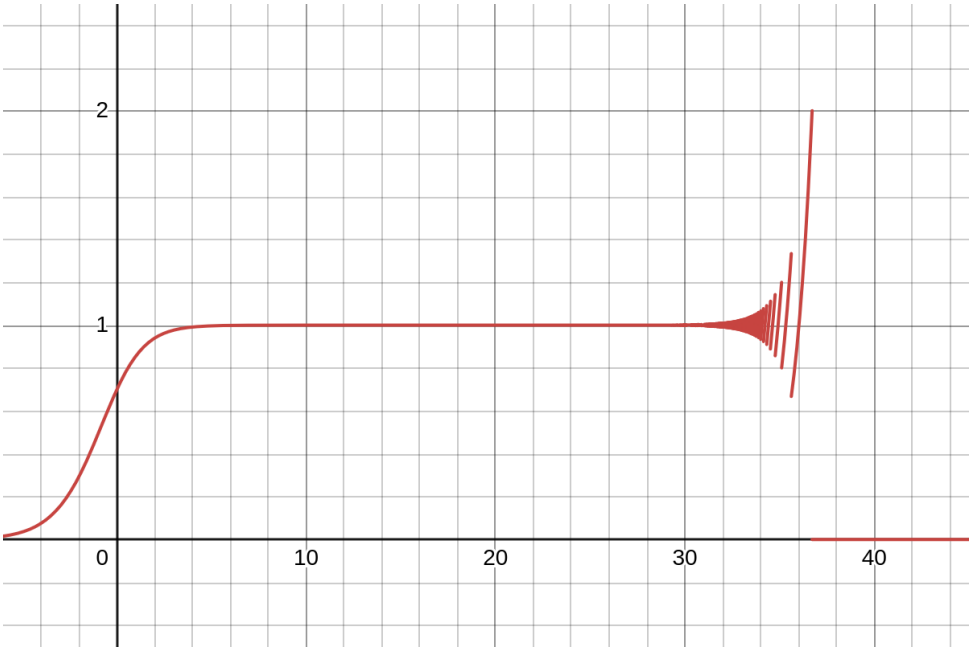
- desmos.com
- geogebra.org
- wolframalpha.com

Obliczenie wartości granicy funkcji:

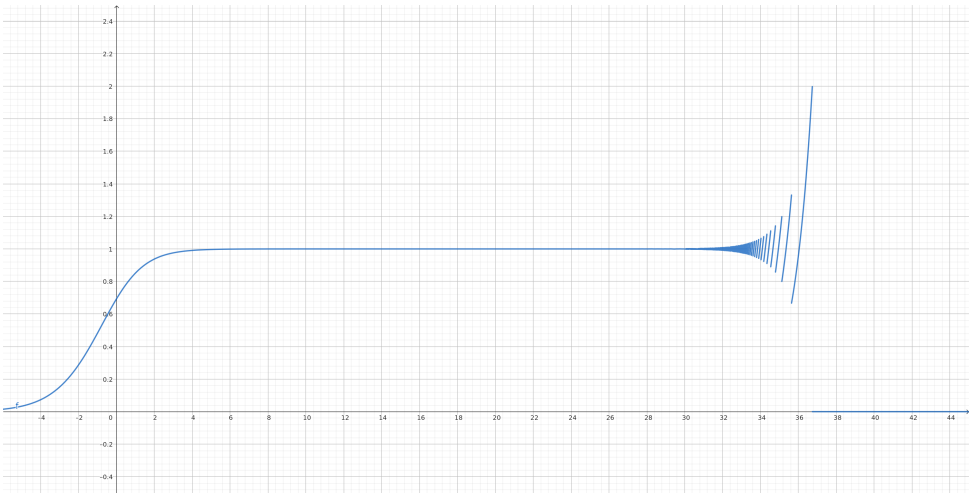
$$lim_{x\to\infty} f(x) = lim_{x\to\infty} e^x \cdot ln(1 + e^{-x}) = lim_{x\to\infty} \frac{ln(1 + e^{-x})}{e^{-x}} \overset{\text{L'Hôpital}}{=} lim_{x\to\infty} \frac{\frac{1}{e^x+1}}{e^{-x}} =$$

$$= lim_{x\to\infty} \frac{e^x}{e^x + 1} \overset{\text{L'Hôpital}}{=} lim_{x\to\infty} \frac{e^x}{e^x} = lim_{x\to\infty} 1 = 1$$

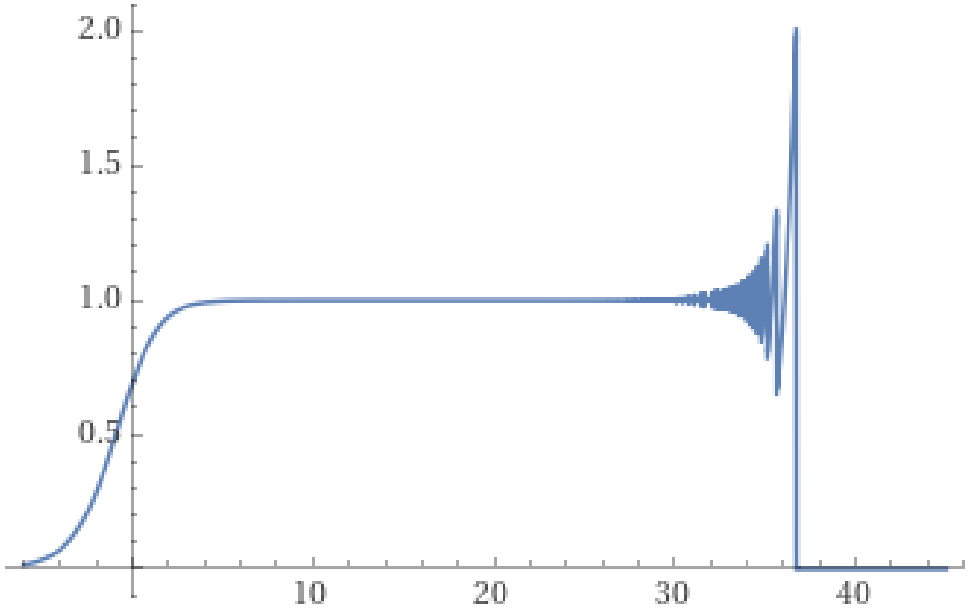
2.3 Wyniki



Rysunek 1: Wykres wygenerowany przez desmos.com



Rysunek 2: Wykres wygenerowany przez geogebra.org



Rysunek 3: Wykres wygenerowany przez wolframalpha.com

2.4 Interpretacja wyników oraz wnioski

Jak można zauważyć, wyniki przedstawione przez programy graficzne nie pokrywają się z obliczoną wartością granicy. Na wykresach już od wartości nieco powyżej 36 wartość funkcji zdaje się zbiegać do 0, podczas gdy w rzeczywistości powinna ona zbiegać do 1. Już w przedziale $(30, 37)$ funkcja zaczyna zachowywać się nieprawidłowo. Dla dużych x wartość e^x staje się bardzo duża, natomiast wartość $ln(1 + e^{-x})$ bardzo mała. Mnożenie wartości o tak dalekich od siebie rzędach wielkości jest obarczone bardzo dużym błędem obliczeń, stąd tak duże oscylacje wyniku. Dla jeszcze większych wartości x przez precyzję arytmetyki $e^{-x} \approx 0$ przez co wartość czynnika $ln(1 + e^{-x}) \approx 0$, co wpływa na całą wartość funkcji i powoduje, że od wartości nieco ponad 36 jest ona równa 0, co nie jest zgodne z wartością oczekiwaną. Wyniki otrzymane w programach do wizualizacji graficznej nie są wiarygodne. Jest to spowodowane precyzją arytmetyki i błędami obliczeń.

3 Zadanie 3

3.1 Opis problemu

Problemem zadania jest rozwiązanie układu równań liniowych postaci

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

dla danej macierzy współczynników $\mathbf{A} \in \mathbb{R}^{n \times n}$ i wektora prawych stron $\mathbf{b} \in \mathbb{R}^n$.
Macierz $\mathbf{A}$ należy generować w następujący sposób:

(a) $\mathbf{A} = \mathbf{H_n}$, gdzie $\mathbf{H_n}$ jest macierzą Hilberta stopnia n wygenerowaną za pomocą funkcji `A=hilb(n)`,

(b) $\mathbf{A} = \mathbf{R_n}$, gdzie $\mathbf{R_n}$ jest losową macierzą stopnia n z zadanym wskaźnikiem uwarunkowania c wygenerowaną za pomocą funkcji `A=matcond(n,c)`.

Wektor $\mathbf{b}$ zadany jest następująco $\mathbf{b} = \mathbf{Ax}$, gdzie $\mathbf{A}$ jest wygenerowaną macierzą, a $\mathbf{x} = (1, \ldots, 1)^T$. Zatem znane jest rozwiązanie dokładne $\mathbf{Ax} = \mathbf{b}$ dla $\mathbf{A}$ i $\mathbf{b}$.
Należy rozwiązać $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ dwoma metodami:

- metodą eliminacji Gaussa ($\mathbf{x} = \frac{\mathbf{A}}{\mathbf{b}}$)
- korzystając z wzoru $\mathbf{x} = \mathbf{A^{-1}b}$

Eksperymenty wykonać dla macierzy Hilberta $\mathbf{H_n}$ z rosnącym stopniem $n > 1$ oraz dla macierzy losowej $\mathbf{R_n}$, $n = 5, 10, 20$ z rosnącym wskaźnikiem uwarunkowania $c = 1, 10, 10^3, 10^7, 10^{12}, 10^{16}$. Porównać obliczony $\tilde{\mathbf{x}}$ z rozwiązaniem dokładnym $x = (1, \ldots, 1)^T$, tj. policzyć błędy względne.

3.2 Rozwiązanie

Do rozwiązania zostały wykorzystane metody `hilb(n)` do generowania macierzy Hilberta oraz `matcond(n, c)` do generowania macierzy losowej o podanym stopniu uwarunkowania podane na stronie przedmiotu. Do wyświetlania wskaźnika uwarunkowania macierzy i rzędu macierzy użyłam funkcji `cond(A)` oraz `rank(A)` z pakietu `Linear Algebra`. Ponadto zaimplementowałam dwie funkcje. Funkcja `solveHilbertsMatrices(n)` rozwiązuje dwoma metodami równania dla macierzy Hilberta od 1 do n, a także oblicza błędy względne obu metod i drukuje wyniki wraz z wartościami *cond* i *rank* w tabeli dla łatwości porównywania. Funkcja `solveRandomMatrices(n, c)` przyjmuje tablice n wielkości macierzy oraz tablicę c wartości wskaźnika uwarunkowania macierzy oraz oblicza i drukuje treści jak w pierwszej funkcji.

3.3 Wyniki

Otrzymane błędy dla macierzy Hilberta:

Wielkość	**cond(A)**	**rank(A)**	**Błąd względny eliminacji Gaussa**	**Błąd względny metody z inwersją**
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

Tabela 5: Wartości błędów względnych uzyskanych dla obu metod dla macierzy Hilberta od $n = 1$ do $n = 13$. Dla wartości > 13 precyzja arytmetyki jest niewystaczająca do dokładnych obliczeń.

Otrzymane błędy dla macierzy losowych:

Wielkość	**cond(A)**	**rank(A)**	**Błąd względny eliminacji Gaussa**	**Błąd względny metody z inwersją**
5	1.0	5	1.3136335981433191e-16	1.4895204919483638e-16
10	1.0	10	3.877842313165343e-16	2.3551386880256624e-16
20	1.0	20	4.1317602818954126e-16	4.2783218896497593e-16
5	10.0	5	7.021666937153402e-17	1.3136335981433191e-16
10	10.0	10	8.777083671441752e-16	1.0349517852878004e-15
20	10.0	20	5.312855095420853e-16	4.4130492403375385e-16
5	1000.0	5	4.522029829349417e-15	3.83051078018465e-15
10	1000.0	10	3.1556632090514235e-14	3.150209657129784e-14
20	1000.0	20	1.5623879529414442e-14	2.2516680567649047e-14
5	1.0e7	5	1.5466968819522093e-10	1.456645987490448e-10
10	1.0e7	10	6.303725399516712e-11	9.122553758007403e-11
20	1.0e7	20	4.393361955269142e-10	4.460976482430807e-10
5	1.0e12	5	5.8431079053373395e-5	6.051437405458738e-5
10	1.0e12	10	4.605832495908304e-6	4.63444893609002e-6
20	1.0e12	20	9.72489595903949e-6	1.3874240650581215e-5
5	1.0e16	4	0.27753044664571114	0.3090433828769029
10	1.0e16	9	0.002593527376343342	0.022810627063525674
20	1.0e16	19	0.20092868134059785	0.19241602267287927

Tabela 6: Wartości błędów względnych uzyskanych dla obu metod dla macierzy losowych o wielkościach $n \in \{5, 10, 20\}$ oraz wskaźnikach uwarunkowania $c \in \{1, 10, 10^3, 10^7, 10^{12}, 10^{16}\}$

3.4 Interpretacja wyników oraz wnioski

Dla macierzy Hilberta wraz z wzrostem wielkości bardzo szybko rosną zarówno wskaźnik uwarunkowania macierzy jak i błędy względne obu metod rozwiązywania. Dla obu metod błędy względne są podobnych rzędów więc nie można powiedzieć, że któraś z nich jest lepsza. Zadanie obliczania układu równań $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$, gdzie $\mathbf{A}$ jest macierzą Hilberta jest źle uwarunkowane nawet dla macierzy o niewielkich rozmiarach.

Dla macierzy losowych można zauważyć, że błędy względne w obu metodach zależą od wielkości wskaźnika uwarunkowania. Błędy są podobnych rzędów dla różnych wielkości macierzy o takim samym wskaźniku uwarunkowania. Podobnie jak dla macierzy Hilberta, nie można stwierdzić czy któraś z metod osiąga bardziej zbliżone do poprawnych wyniki, gdyż błędy są zbliżonych wielkości.

Jak można zauważyć, w obu przypadkach im większy wskaźnik uwarunkowania macierzy, tym większe błędy względne obliczeń. Dla dużych jego wartości zadanie staje się źle uwarunkowane.

4 Zadanie 4

4.1 Opis problemu

Dany jest wielomian Wilkinsona w postaci naturalnej:

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

oraz w postaci iloczynowej:

$$\begin{aligned} p(x) = {} & (x-20)(x-19)(x-18)(x-17)(x-16)(x-15)(x-14)(x-13)(x-12)(x-11) \\ & (x-10)(x-9)(x-8)(x-7)(x-6)(x-5)(x-4)(x-3)(x-2)(x-1) \end{aligned}$$

Należy użyć funkcji `roots` z pakietu `Polynomials` do policzenia zer wielomianu $P(x)$. Ponadto sprawdzić obliczone pierwiastki z_k obliczając $|P(z_k)|, |p(z_k)|$ oraz $|z_k - k|$. Wyjaśnić pojawiające się rozbieżności.
Następnie należy powtórzyć eksperyment Wilkinsona, czyli współczynnik -210 zamienić na $-210 - 2^{-23}$ i wyjaśnić zjawisko.

4.2 Rozwiązanie

Do rozwiązania zostały użyte metody `roots(polynomial)`, `polynomial(x)`, `fromroots(roots)` oraz konstruktor `Polynomial(coefficients)` z pakietu `Polynomials`. Ponadto zaimplementowałam trzy funkcje. Funkcja `calculateRoots(coefficients)` oblicza pierwiastki wielomianu dla podanych współczynników przy x. Funkcja `calculateValueGeneralForm(coefficients, x)` oblicza wartość wielomianu w postaci naturalnej dla podanych jego współczynników dla podanego x. Funkcja `calculateValueFactoredForm(roots, x)` oblicza wartość wielomianu w postaci iloczynowej dla podanych jego pierwiastków dla danego x. Na końcu program wyświetla wartości k, z_k, $|z_k - k|$, $|P(z_k)|$, $|p(z_k)|$ dla $k \in \{1, \ldots, 20\}$ w tabeli.

4.3 Wyniki

k	z_k	$\lvert z_k - k\rvert$	$\lvert P(z_k)\rvert$	$\lvert p(z_k)\rvert$
1	0.9999999999996989	3.0109248427834245e-13	35696.50964788257	5.518479490350445e6
2	2.0000000000283182	2.8318236644508943e-11	176252.60026668405	7.37869762990174e19
3	2.9999999995920965	4.0790348876384996e-10	279157.6968824087	3.3204139316875795e20
4	3.9999999837375317	1.626246826091915e-8	3.0271092988991085e6	8.854437035384718e20
5	5.000000665769791	6.657697912970661e-7	2.2917473756567076e7	1.8446752056545688e21
6	5.999989245824773	1.0754175226779239e-5	1.2902417284205095e8	3.320394888870117e21
7	7.000102002793008	0.00010200279300764947	4.805112754602064e8	5.423593016891273e21
8	7.999355829607762	0.0006441703922384079	1.6379520218961136e9	8.262050140110275e21
9	9.002915294362053	0.002915294362052734	4.877071372550003e9	1.196559421646277e22
10	9.990413042481725	0.009586957518274986	1.3638638195458128e10	1.655260133520688e22
11	11.025022932909318	0.025022932909317674	3.585631295130865e10	2.24783329792479e22
12	11.953283253846857	0.04671674615314281	7.533332360358197e10	2.886944688412679e22
13	13.07431403244734	0.07431403244734014	1.9605988124330817e11	3.807325552826988e22
14	13.914755591802127	0.08524440819787316	3.5751347823104315e11	4.612719853150334e22
15	15.075493799699476	0.07549379969947623	8.21627123645597e11	5.901011420218566e22
16	15.946286716607972	0.05371328339202819	1.5514978880494067e12	7.010874106897764e22
17	17.025427146237412	0.025427146237412046	3.694735918486229e12	8.568905825736165e22
18	17.99092135271648	0.009078647283519814	7.650109016515867e12	1.0144799361044434e23
19	19.00190981829944	0.0019098182994383706	1.1435273749721195e13	1.1990376202371257e23
20	19.999809291236637	0.00019070876336257925	2.792410639368072 7e13	1.4019117414318134e23

Tabela 7: Wartości k, z_k, $|z_k - k|$, $|P(z_k)|$, $|p(z_k)|$ dla $k \in \{1, \ldots, 20\}$ dla wielomianu Wilkinsona

k	$\lvert P(k)\rvert$	$\lvert p(k)\rvert$
1	0.0	0
2	8192.0	0
3	27648.0	0
4	622592.0	0
5	2.176e6	0
6	8.84736e6	0
7	2.4410624e7	0
8	5.89824e7	0
9	1.45753344e8	0
10	2.27328e8	0
11	4.79074816e8	0
12	8.75003904e8	0
13	1.483133184e9	0
14	2.457219072e9	0
15	3.905712e9	0
16	6.029312e9	0
17	9.116641408e9	0
18	1.333988352e10	0
19	1.9213101568e10	0
20	2.7193344e10	0

Tabela 8: Wartości wielomianu Wilkinsona dla dokładnych pierwiastków dla formy naturalnej i iloczynowej

k	$\mathbf{z_k}$	$\lvert\mathbf{P}(\mathbf{z_k})\rvert$
1	0.9999999999998357 + 0.0im	20259.872313418207
2	2.0000000000550373 + 0.0im	346541.4137593836
3	2.99999999660342 + 0.0im	2.2580597001197007e6
4	4.000000089724362 + 0.0im	1.0542631790395478e7
5	4.99999857388791 + 0.0im	3.757830916585153e7
6	6.000020476673031 + 0.0im	1.3140943325569446e8
7	6.99960207042242 + 0.0im	3.939355874647618e8
8	8.007772029099446 + 0.0im	1.184986961371896e9
9	8.915816367932559 + 0.0im	2.2255221233077707e9
10	10.095455630535774 - 0.6449328236240688im	1.0677921232930157e10
11	10.095455630535774 + 0.6449328236240688im	1.0677921232930157e10
12	11.793890586174369 - 1.6524771364075785im	3.1401962344429485e10
13	11.793890586174369 + 1.6524771364075785im	3.1401962344429485e10
14	13.992406684487216 - 2.5188244257108443im	2.157665405951858e11
15	13.992406684487216 + 2.5188244257108443im	2.157665405951858e11
16	16.73074487979267 - 2.812624896721978im	4.850110893921027e11
17	16.73074487979267 + 2.812624896721978im	4.850110893921027e11
18	19.5024423688181 - 1.940331978642903im	4.557199223869993e12
19	19.5024423688181 + 1.940331978642903im	4.557199223869993e12
20	20.84691021519479 + 0.0im	8.756386551865696e12

Tabela 9: Wartości k, z_k, $|z_k - k|$, $|P(z_k)|$, $|p(z_k)|$ dla $k \in \{1, \ldots, 20\}$ dla wielomianu Wilkinsona ze zmienionym a_{19} z -210 na $-210 - 2^{-23}$.

4.4 Interpretacja wyników oraz wnioski

Jak można zauważyć, żaden z pierwiastków wielomianu Wilkinsona nie został wyliczony dokładnie. Wraz z wzrostem wartości pierwiastka rosną błędy bezwzględne wartości obliczonego pierwiastka. Błędy te wydają się być stosunkowo małe, gdyż dla żadnego pierwiastka nie przekraczają rzędu 10^{-1}, jednak po policzeniu wartości wielomianu w postaci naturalnej i iloczynowej dla tych niedokładnie obliczonych pierwiastków otrzymujemy wartości nawet rzędu odpowiednio 10^{13} oraz 10^{23}. Świadczy to o tym, że problem obliczania pierwiastków wielomianu Wilkinsona jest źle uwarunkowany - niewielkie zmiany w wartości obliczonego pierwiastka powodują ogromne zmiany w wartości wielomianu.

Sprawdziłam również eksperymentalnie jakie wartości będzie przyjmował wielomian dla dokładnych wartości pierwiastków w postaci naturalnej i iloczynowej. Dla postaci iloczynowej wartość wynosi 0 dla każdego z pierwiastków, a dla naturalnej wartości rosną wraz z wzrostem wartości pierwiastka podobnie jak w dla wartości wyznaczonych. Jest to spowodowane tym, że arytmetyka `Float64` ma w systemie dziesiętnym od 15 do 17 miejsc znaczących, a niektóre współczynniki wielomianu, zwłaszcza te przy niższych potęgach są na tyle dużymi liczbami, że nie da się ich zapisać bez utraty kilku miejsc znaczących. Nie możemy więc przechowywać wielomianu Wilkinsona w postaci naturalnej w sposób dokładny.

Dzięki eksperymentowi Wilkinsona, czyli nieznacznemu zmienieniu wartości jednego z współczynników wielomianu również można zauważyć jak bardzo źle uwarunkowane jest zadanie. Bardzo niewielka zmiana współczynnika a_{19} sprawiła, że wartości pierwiastków wielomianu zyskały część urojoną. Podobnie jak w niezmienionym wielomianie, policzone wartości $|P(x_k)|$ znacząco odbiegają od realnej wartości jaką powinien on przyjąć, czyli 0 i dla wartości o największym module sięgają rzędu 10^{12}.

5 Zadanie 5

5.1 Opis problemu

Dane jest równanie rekurencyjne (modelu logistycznego, modelu wzrostu populacji):

$$p_{n+1} := p_n + rp_n(1 - p_n) \text{ dla } n = 0, 1, \ldots$$

gdzie r jest pewną daną stałą, $r(1 - p_n)$ jest czynnikiem wzrostu populacji, a p_0 jest wielkością populacji stanowiącą procent maksymalnej wielkości populacji dla danego środowiska.
Należy przeprowadzić następujące eksperymenty:

1. Dla danych $p_0 = 0.01$ i $r = 3$ wykonać 40 iteracji wyrażenia, a następnie wykonać ponownie 40 iteracji wyrażenia z niewielką modyfikacją, tj. wykonać 10 iteracji, zatrzymać, zastosować obcięcie wyniku odrzu-

cając cyfry po trzecim miejscu po przecinku i kontynuować obliczenia dalej obliczenia do 40 iteracji tak, jakby był to ostatni wynik na wyjściu. Porównać otrzymane wyniki. Obliczenia wykonywać w arytmetyce `Float32`.

2. Dla danych $p_0 = 0.01$ i $r = 3$ wykonać 40 iteracji wyrażenia w arytmetyce `Float32` i `Float64`. Porównać otrzymane wyniki.

5.2 Rozwiązanie

Zaimplementowałam dwie funkcje: `nextPFloat32(p, r)` oraz `nextPFloat64(p, r)`, które zwracają wartość wyrażenia $p + rp(1 - p)$ odpowiednio w arytmetyce `Float32` i `Float64`. Ponadto zaimplementowałam dwie funkcje `experiment1()` oraz `experiment2()`, które przeprowadzają eksperymenty zgodnie z opisem powyżej i drukują wyniki w tabelach dla łatwej interpretacji.

5.3 Wyniki

n	**p_n w `Float32`**	**p_n z obcięciem po n = 10 w `Float32`**
1	0.0397	0.0397
2	0.15407173	0.15407173
3	0.5450726	0.5450726
4	1.2889781	1.2889781
5	0.1715188	0.1715188
6	0.5978191	0.5978191
7	1.3191134	1.3191134
8	0.056273222	0.056273222
9	0.21559286	0.21559286
10	0.7229306	0.722
11	1.3238364	1.3241479
12	0.037716985	0.036488414
13	0.14660022	0.14195944
14	0.521926	0.50738037
15	1.2704837	1.2572169
16	0.2395482	0.28708452
17	0.7860428	0.9010855
18	1.2905813	1.1684768
19	0.16552472	0.577893
20	0.5799036	1.3096911
21	1.3107498	0.09289217
22	0.088804245	0.34568182
23	0.3315584	1.0242395
24	0.9964407	0.94975823
25	1.0070806	1.0929108
26	0.9856885	0.7882812
27	1.0280086	1.2889631
28	0.9416294	0.17157483
29	1.1065198	0.59798557
30	0.7529209	1.3191822
31	1.3110139	0.05600393
32	0.0877831	0.21460639
33	0.3280148	0.7202578
34	0.9892781	1.3247173
35	1.021099	0.034241438
36	0.95646656	0.13344833
37	1.0813814	0.48036796
38	0.81736827	1.2292118
39	1.2652004	0.3839622
40	0.25860548	1.093568

Tabela 10: Wyniki p_n dla eksperymentu 1

n	p_n w Float32	p_n w Float64
1	0.0397	0.0397
2	0.15407173	0.15407173000000002
3	0.5450726	0.5450726260444213
4	1.2889781	1.2889780011888006
5	0.1715188	0.17151914210917552
6	0.5978191	0.5978201201070994
7	1.3191134	1.3191137924137974
8	0.056273222	0.056271577646256565
9	0.21559286	0.21558683923263022
10	0.7229306	0.722914301179573
11	1.3238364	1.3238419441684408
12	0.037716985	0.03769529725473175
13	0.14660022	0.14651838271355924
14	0.521926	0.521670621435246
15	1.2704837	1.2702617739350768
16	0.2395482	0.24035217277824272
17	0.7860428	0.7881011902353041
18	1.2905813	1.2890943027903075
19	0.16552472	0.17108484670194324
20	0.5799036	0.5965293124946907
21	1.3107498	1.3185755879825978
22	0.088804245	0.058377608259430724
23	0.3315584	0.22328659759944824
24	0.9964407	0.7435756763951792
25	1.0070806	1.315588346001072
26	0.9856885	0.07003529560277899
27	1.0280086	0.26542635452061003
28	0.9416294	0.8503519690601384
29	1.1065198	1.2321124623871897
30	0.7529209	0.37414648963928676
31	1.3110139	1.0766291714289444
32	0.0877831	0.8291255674004515
33	0.3280148	1.2541546500504441
34	0.9892781	0.29790694147232066
35	1.021099	0.9253821285571046
36	0.95646656	1.1325322626697856
37	1.0813814	0.6822410727153098
38	0.81736827	1.3326056469620293
39	1.2652004	0.0029091569028512065
40	0.25860548	0.011611238029748606

Tabela 11: Wyniki p_n dla eksperymentu 2

5.4 Interpretacja wyników oraz wnioski

W pierwszym eksperymencie wartości po kilku iteracjach od iteracji 10, kiedy to zaszła zmiana, są do siebie dość zbliżone, jednakże z każdą kolejną iteracją wyniki te stają się coraz bardziej rozbieżne w sposób dość nieprzewidywalny. Niewielka zmiana, jaką było obcięcie wartości p_{10} do trzech miejsc po przecinku ma znaczny wpływ na wyniki p_n w końcowych iteracjach, które mają ze sobą niewiele wspólnego.

W drugim eksperymencie możemy zauważyć, że wartości p_n dla początkowych n są bardzo zbliżone do siebie dla obu arytmetyk. Od wartości $n = 22$ zaczynają nieco bardziej od siebie odbiegać, by dla końcowych iteracji, podobnie jak w eksperymencie 1, nie mieć ze sobą zbyt wiele wspólnego.

Można powiedzieć, że metoda obliczania wartości na arytmetyce `Float64` jest bardziej dokładna, jednakże patrząc na to jak różne są wartości p_n dla `Float32` i `Float64` oraz jak duży wpływ na wartości końcowych p_n miała niewielka zmiana w trakcie obliczeń lepszym stwierdzeniem jest to, że żadna z metod obliczania p_n nie jest zbytnio wiarygodna.

Powodem, dla którego wartości nie są wiarygodne jest fakt, że do obliczania każdego p_k potrzebujemy obli-

czenia każdej wartości od p_{k-1} do p_1. Występuje więc tutaj kumulowanie się błędów obliczeń. Dodatkowo nie pomaga fakt, że w obliczeniach musimy podnosić wartość do kwadratu, co jest operacją szczególnie podatną na niewystarczającą precyzję arytmetyki.

6 Zadanie 6

6.1 Opis problemu

Dane jest równanie rekurencyjne

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

gdzie c jest pewną stałą. Należy przeprowadzić następujące eksperymenty. Dla danych:

1. $c = -2$ i $x_0 = 1$
2. $c = -2$ i $x_0 = 2$
3. $c = -2$ i $x_0 = 1.99999999999999$
4. $c = -1$ i $x_0 = 1$
5. $c = -1$ i $x_0 = -1$
6. $c = -1$ i $x_0 = 0.75$
7. $c = -1$ i $x_0 = 0.25$

należy wykonać 40 iteracji wyrażenia w arytmetyce `Float64`. Zaobserwować zachowanie generowanych ciągów.

6.2 Rozwiązanie

Zaimplementowałam funkcję `nextX(c, x)`, która zwraca wartość wyrażenia $x_n^2 + c$ działając w arytmetyce `Float64`. Ponadto zaimplementowałam funkcję `experiment()`, która przeprowadza eksperyment wyznaczania kolejnych wyrazów ciągu x_n dla odpowiednich c i x_0 zgodnych z opisem powyżej i drukuje wyniki w tabelach dla łatwej interpretacji.

6.3 Wyniki

6.3.1 Wyniki iteracyjne

Wyniki działania programu do wyznaczania kolejnych wyrazów ciągu x_n.

n	$\mathbf{x_0 = 1}$	$\mathbf{x_0 = 2}$	$\mathbf{x_0 = 1.99999999999999}$
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

n	$\mathbf{x_0 = 1}$	$\mathbf{x_0 = 2}$	$\mathbf{x_0 = 1.99999999999999}$
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

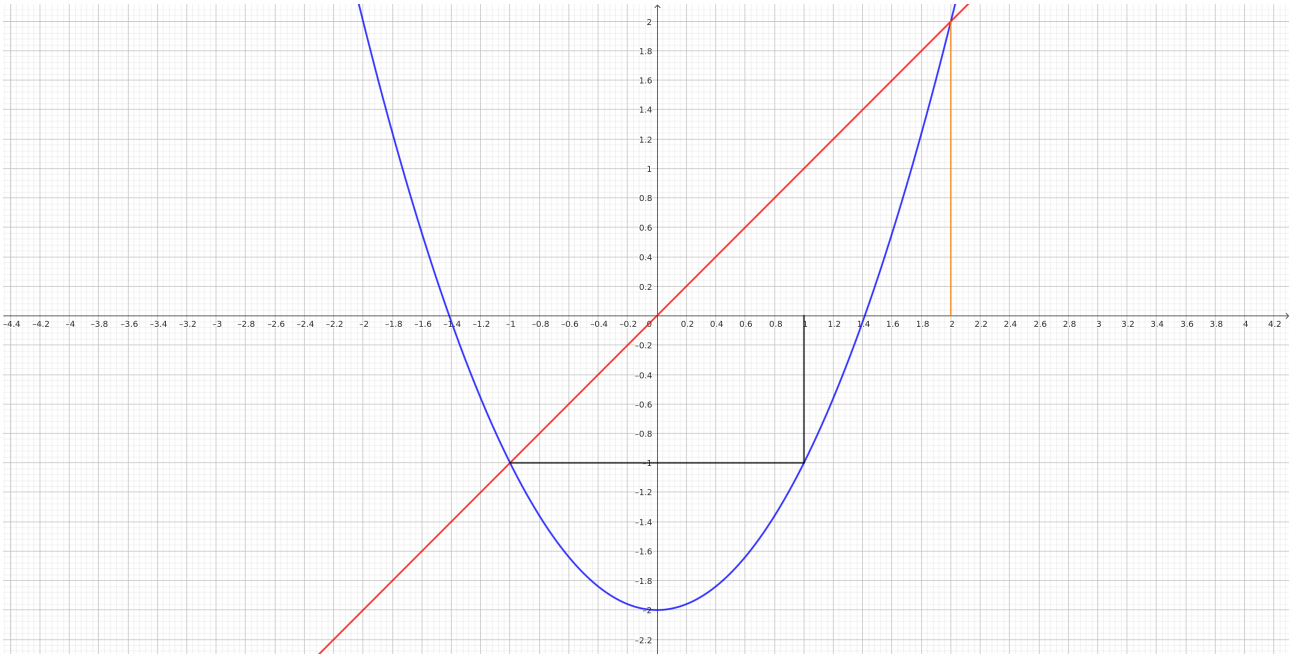
Tabela 12: Wyniki 40 iteracji $x_{n+1} := x_n^2 + c$ dla $c = -2$ i $x_0 \in \{1, 2, 1.99999999999999\}$.

n	$\mathbf{x_0 = 1}$	$\mathbf{x_0 = -1}$	$\mathbf{x_0 = 0.75}$	$\mathbf{x_0 = 0.25}$
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

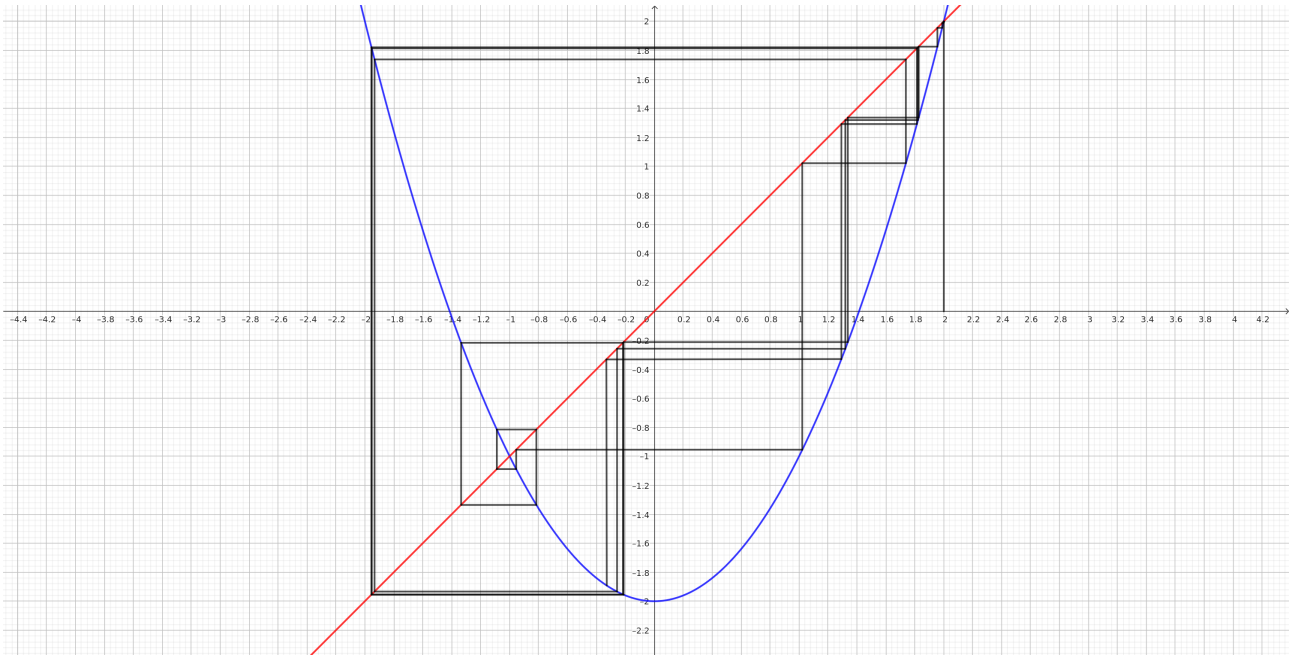
n	$\mathbf{x_0 = 1}$	$\mathbf{x_0 = -1}$	$\mathbf{x_0 = 0.75}$	$\mathbf{x_0 = 0.25}$
35	0.0	0.0	0.0	-1.0
36	-1.0	-1.0	-1.0	0.0
37	0.0	0.0	0.0	-1.0
38	-1.0	-1.0	-1.0	0.0
39	0.0	0.0	0.0	-1.0
40	-1.0	-1.0	-1.0	0.0

Tabela 13: Wyniki 40 iteracji $x_{n+1} := x_n^2 + c$ dla $c = -1$ i $x_0 \in \{1, -1, 0.75, 0.25\}$.

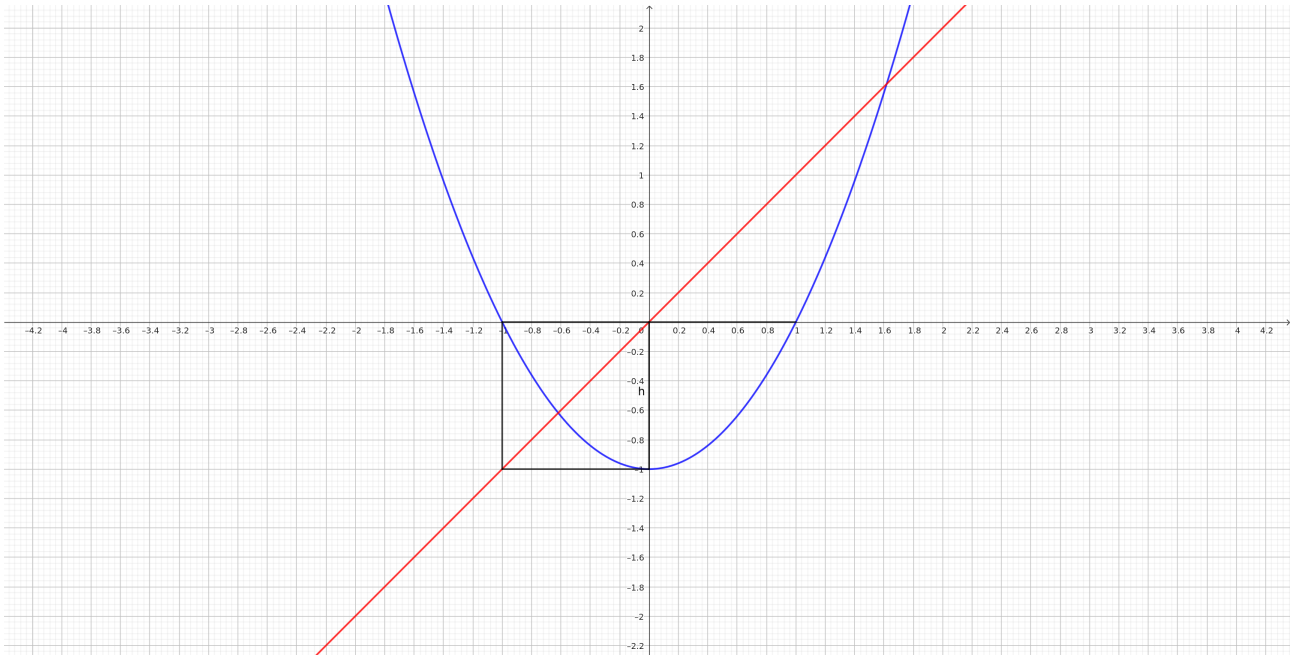
6.3.2 Interpretacja graficzna wyników



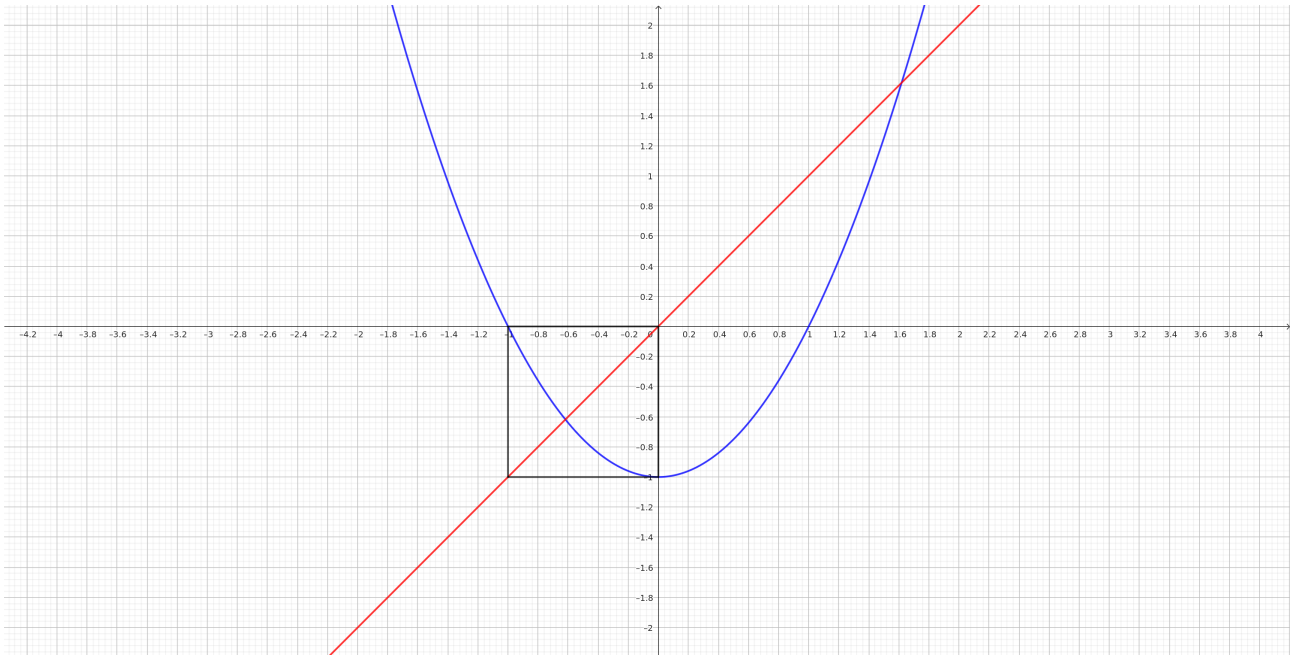
Rysunek 4: Przedstawienie graficzne iteracyjnego wyznaczania x_n z równania rekurencyjnego $x_{n+1} := x_n^2 - 2$ dla $x_0 = 1$ (czarna łamana) i $x_0 = 2$ (pomarańczowa łamana)



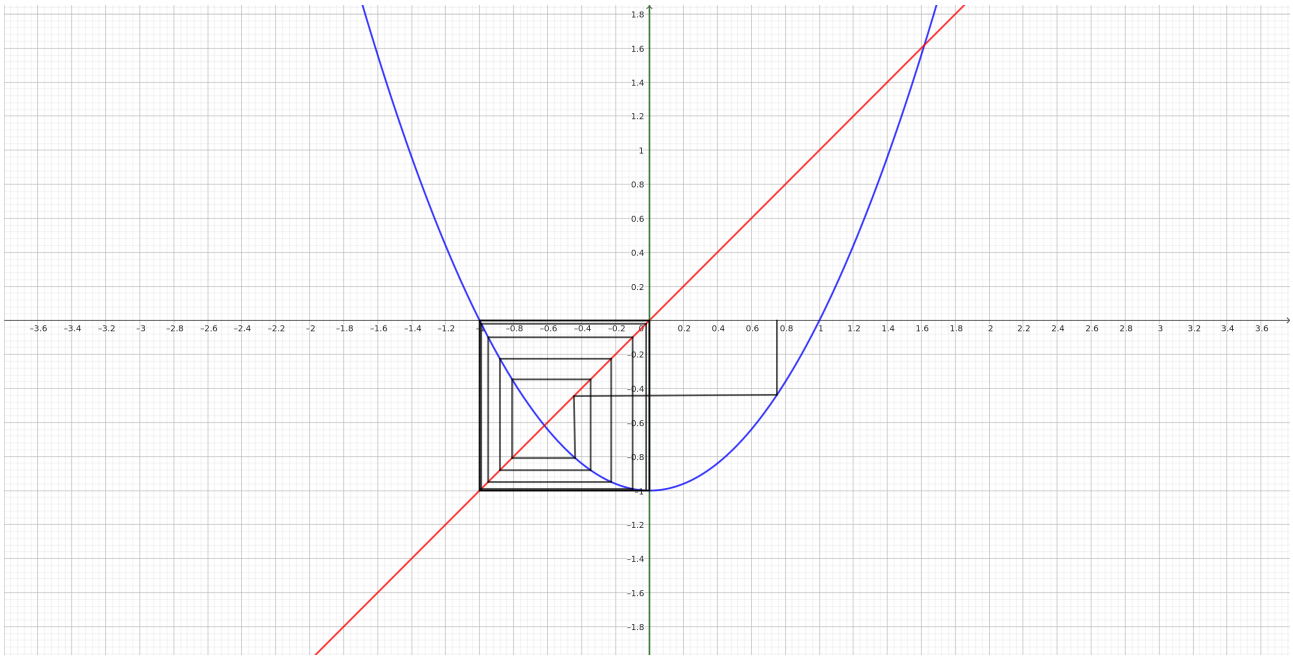
Rysunek 5: Przedstawienie graficzne iteracyjnego wyznaczania x_n z równania rekurencyjnego $x_{n+1} := x_n^2 - 2$ dla $x_0 = 1.99999999999999$ (czarna łamana)



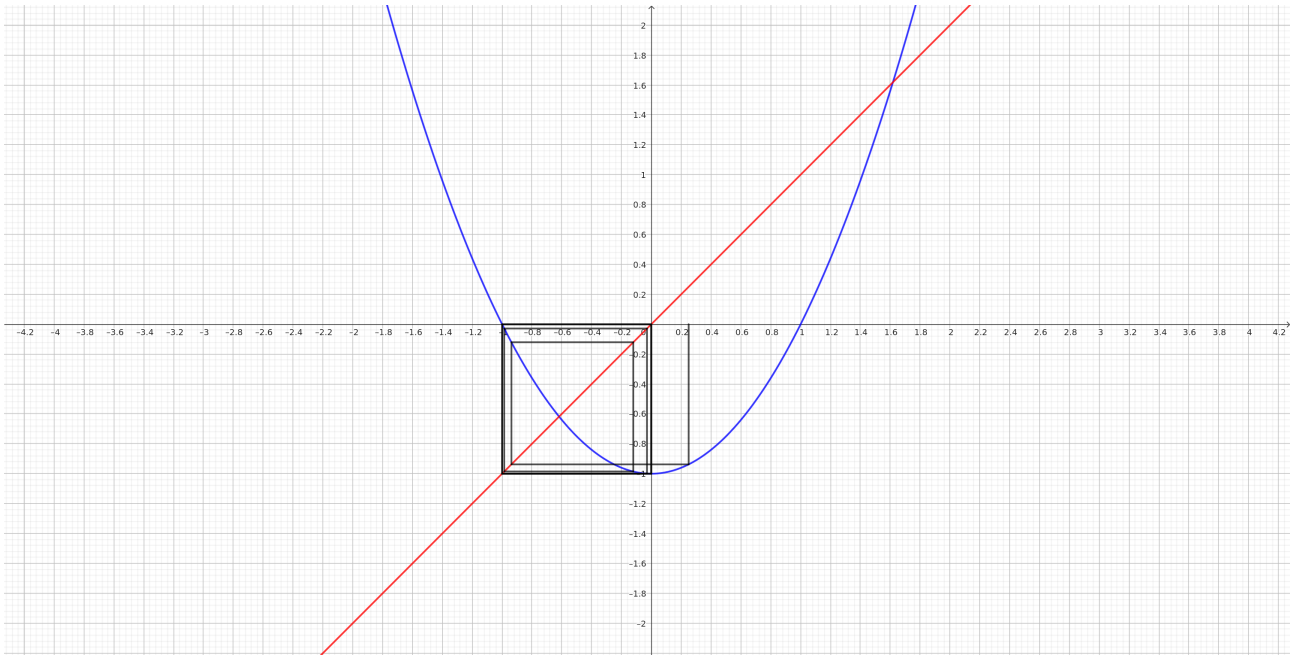
Rysunek 6: Przedstawienie graficzne iteracyjnego wyznaczania x_n z równania rekurencyjnego $x_{n+1} := x_n^2 - 1$ dla $x_0 = 1$ (czarna łamana)



Rysunek 7: Przedstawienie graficzne iteracyjnego wyznaczania x_n z równania rekurencyjnego $x_{n+1} := x_n^2 - 1$ dla $x_0 = -1$ (czarna łamana)



Rysunek 8: Przedstawienie graficzne iteracyjnego wyznaczania x_n z równania rekurencyjnego $x_{n+1} := x_n^2 - 1$ dla $x_0 = 0.75$ (czarna łamana)



Rysunek 9: Przedstawienie graficzne iteracyjnego wyznaczania x_n z równania rekurencyjnego $x_{n+1} := x_n^2 - 1$ dla $x_0 = 0.25$ (czarna łamana)

6.4 Interpretacja wyników oraz wnioski

1. $x_{n+1} := x_n^2 - 2$, gdzie $x_0 = 1$: łamana iteracji graficznej już na samym początku działania programu trafia do punktu stałego, skąd nie ma już wyjścia, co powoduje ze kolejne iteracje nie mają wpływu na wartość x_n, która stale wynosi -1

2. $x_{n+1} := x_n^2 - 2$, gdzie $x_0 = 2$: podobnie jak w podpunkcie poprzednim, łamana iteracji graficznej już na samym początku działania programu trafia do punktu stałego, skąd nie ma już wyjścia, co powoduje ze kolejne iteracje nie mają wpływu na wartość x_n, która stale wynosi 2

3. $x_{n+1} := x_n^2 - 2$, gdzie $x_0 = 1.99999999999999$: bardzo niewielka zmiana wartości x_0 względem wartości z poprzedniego przykładu, sprawia, że nawet po 40 iteracjach wartość ciągu się nie stabilizuje

4. $x_{n+1} := x_n^2 - 1$, gdzie $x_0 = 1$: dla tych danych od razu wartości ciągu stabilizują się i wchodzą w cykl $(0, -1, 0, -1, \dots)$

5. $x_{n+1} := x_n^2 - 1$, gdzie $x_0 = -1$: zachodzi taka sama sytuacja jak w podpunkcie poprzednim, wartości ciągu od razu stabilizują się i występują w cyklu $(0, -1, 0, -1, \dots)$

6. $x_{n+1} := x_n^2 - 1$, gdzie $x_0 = 0.75$: wartości ciągu stabilizują się dopiero po kilkunastu iteracjach i osiągają cykl $(0, -1, 0, -1, \dots)$

7. $x_{n+1} := x_n^2 - 1$, gdzie $x_0 = 0.25$: podobnie jak w podpunkcie poprzednim wartości ciągu stabilizują się i osiągają cykl $(0, -1, 0, -1, \dots)$, jednak dzieje się to po mniejszej liczbie iteracji niż dla $x_0 = 0.75$

Proces rekurencyjnego wyznaczania kolejnych wyrazów ciągu może mieć różną stabilność zależną od parametrów początkowych. Może być on całkowicie stabilny (jak w przykładach dla $c = -2$ i $x_0 = 1$, $c = -2$ i $x_0 = 2$, $c = -1$ i $x_0 = 1$ czy $c = -1$ i $x_0 = -1$), stabilizujący się (jak w przykładach dla $c = -1$ i $x_0 = 0.75$ oraz $c = -1$ i $x_0 = 0.25$), czy też całkowicie niestabilny (dla $c = -2$ i $x_0 = 1.99999999999999$). Od parametrów wejściowych zależy nie tylko to, czy ciąg jest stabilny, ale też to, z jaką prędkością się stabilizuje (dla $c = -1$ i $x_0 = 0.25$ dzieje się to szybciej niż dla $c = -1$ - $x_0 = 0.75$).

Dobór odpowiednich parametrów i liczba iteracji okazują się więc kluczowe w kwestii stabilności.