OBLICZENIA NAUKOWE

Lista nr 2.

Patrycja Paradowska
nr indeksu: 244952
Prowadzący laboratoria: Mgr inż. Marta Słowik

09.11.2019r.

1 Zadanie 1. - Iloczyn skalarny

Zadanie obejmujące eksperymentalne przeprowadzenie badania wpływu niewielkich zmian danych na wyniki przeprowadzanych obliczeń dla algorytmów obliczania iloczynu skalarnego dwóch wektorów.

1.1 Przedstawienie problemu

Należało sprawdzić, jaki wpływ na wyniki otrzymywanych przez nas obliczeń mogą mieć nawet relatywnie niewielkie zmiany w danych wejściowych, a więc zaburzono wartości dwóch współrzędnych wektora z zadania 5. z listy 1. Wektory X oraz Y z poprzedniej listy wyglądały następująco:

$$X = [2.718281828, -3.141592654, 1.414213562, 0.577215664\mathbf{9}, 0.301029995\mathbf{7}]$$

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

Ostatnie cyfry (9 i 7) we współrzędnych x_4 oraz x_5 zostały usunięte i przy ponownym obliczaniu iloczynu skalarnego $X' \cdot Y$ został użyty wektor X':

$$X' = [2.718281828, -3.141592654, 1.414213562, 0.577215664, 0.301029995]$$

1.2 Algorytm

Zastosowano cztery algorytmy z poprzedniej listy, które obliczały na różne sposoby iloczyn skalarny zgodnie ze specyfikacją zadania:

1. "w przód": $\sum_{i=1}^{n} x_i y_i$; czyli zaczynamy obliczanie iloczynu skalarnego od pierwszych współrzędnych
2. "w tył": $\sum_{i=n}^{1} x_i y_i$; czyli zaczynamy obliczanie iloczynu skalarnego od ostatnich współrzędnych
3. dodanie dodatnich liczb w porządku od największej do najmniejszej oraz ujemnych w porządku od najmniejszej do największej, a następnie dodanie do siebie obliczonych sum częściowych; zostało to wykonane za pomocą sortowania i odpowiedniego dodania elementów tablicy sum częściowych;
4. od najmniejszego do największego - metoda przeciwna do sposobu *3*.

1.3 Uzyskane wyniki

Obliczenia wykonano najpierw na parze wektorów X i Y, a następnie X' i Y dla typów `Float32` i `Float64`. Otrzymano następujące wyniki:

Zadanie 5, Lista 1		
Sposób	Float32	Float64
1.	-0.4999443	1.0251881368296672e-10
2.	-0.4543457	-1.5643308870494366e-10
3.	-0.5	0.0
4.	-0.5	0.0
Bieżące wyniki		
Sposób	Float32	Float64
1.	-0.4999443	-0.004296342739891585
2.	-0.4543457	-0.004296342998713953
3.	-0.5	-0.004296342842280865
4.	-0.5	-0.004296342842280865

1.4 Obserwacje i wnioski

Analiza wyników wprowadzonych do tabel pokazuje, że wykonana modyfikacja danych nie spowodowała zmiany wyników obliczeń dla par wektorów X,Y oraz X',Y w arytmetyce `Float32`, nadal są one dalekie od poprawnych. Wprowadzone zaburzenie jest niewielkie - bład względny rzędu 10^{-9}. A więc takie same rezultaty w tej arytmetyce mogą być spowodowane niedużą wartością zaburzenia wobec precyzji obliczeń. Ze stosunkowo małą precyzją związana jest niedokładność w przechowywaniu poszczególnych składowych wektorów, która spowodowała, iż usunięcie 9 z x_4 nie wprowadziło zmian w zapisie bitowym i wartości x_4 oraz x_4' mają identyczną reprezentację bitową we `Float32`. x_5 i x_5' różnią się jedynie dopiero na najmniej znaczącym bicie.

Jeżeli chodzi o arytmetykę `Float64`, to patrząc na tabelę, zauważamy wyraźne różnice w wynikach obliczeń przed i po zastosowaniu niewielkiego zaburzenia danych wejściowych. Tym razem wyniki różnią się od tych uzyskanych w zadaniu 5. z listy 1. nawet od 3 do 7 rzędów wielkości. Jest to spowodowane podwójną precyzją, która umożliwia przechowywanie dokładniejszego wyniku. Zaskakujące jest, że wprowadzenie z pozoru niewielkich modyfikacji rzędu 10^{-9} przyczyni się do aż tak wielkich zmian w wynikach. Warto zwrócić uwagę, że żaden z nowo uzyskanych wyników nie jest zerem. Bieżące wartości iloczynów skalarnych uzyskanych wskutek działania poszczególnych algorytmów są zdecydowanie bardziej zbliżone do siebie niż dla wcześniejszych danych, a nawet w granicach dopuszczalnego błędu takie same. Usunięcie ostatnich cyfr z x_4 i x_5 spowodowało dokładniejsze zapisanie wektorów i można by wyciągnąć wniosek, że arytmetyka okazała się wystarczająca do obliczenia iloczynu skalarnego.

Algorytmy użyte do obliczania iloczynu skalarnego są bardzo wrażliwe na niewielkie zmiany w danych, zatem mamy do czynienia z zadaniem źle uwarunkowanym (niewielkie względne zmiany danych powodują duże względne odkształcenia wyników). Na błędy w uzyskiwanych wartościach wpływa też fakt, że wektory są prawie ortogonalne(prostopadłe). Precyzja arytmetyki odgrywa kluczową rolę w zadaniach źle uwarunkowanych, wskazane jest zastosowanie maksymalnej precyzji.

2 Zadanie 2. - Wykres funkcji

2.1 Przedstawienie problemu

W zadaniu 2. należało narysować wykres funkcji:

$$f(x) = e^x \ln(1 + e^{-x})$$

w co najmniej dwóch programach do wizualizacji oraz obliczyć granicę tej funkcji $\lim\limits_{x\to\infty} f(x)$. Następnym celem było dokonanie porównania i wyjaśnienie zaistniałego zjawiska.

2.2 Rozwiązanie

Wykresy funkcji narysowano przy użyciu:

1. środowiska `Wolfram Alpha`,
2. biblioteki `Plotly` w języku `Julia`,

Obliczono także granicę $\lim\limits_{x\to\infty} f(x)$ analitycznie oraz za pomocą biblioteki *SymPy* w języku *Julia*, a uzyskany wynik porównano ze zwracanym przez program `WolframAlpha`.

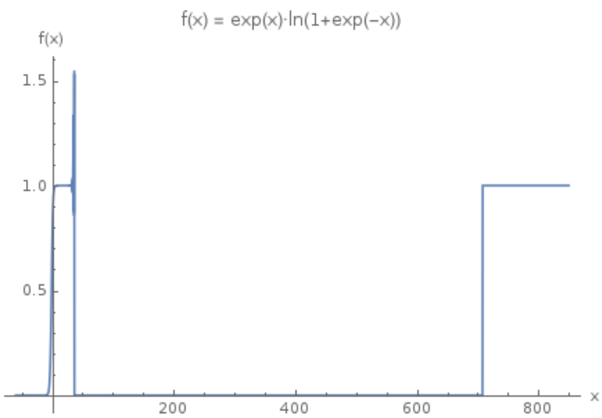
2.3 Uzyskane wyniki

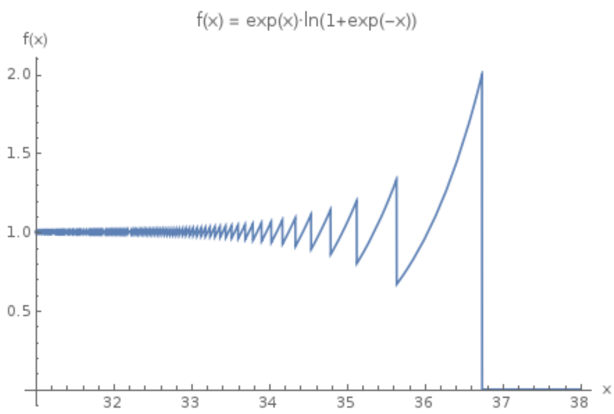
Wynik obliczonej granicy $\lim\limits_{x\to\infty} f(x)$ przez napisany program wynosi 1. Identyczny wynik został osiągnięty poprzez przeprowadzenie rozwiązania analitycznego, używając reguły de l'Hospitala:

$$\lim_{x\to\infty} f(x) = \lim_{x\to\infty} e^x \ln(1+e^{-x}) = \lim_{x\to\infty} \frac{\ln(1+e^{-x})}{e^{-x}} \underset{\left[\frac{0}{0}\right]}{\overset{H}{=}} \lim_{x\to\infty} \frac{(\ln(1+e^{-x}))'}{(e^{-x})'} =$$
$$= \lim_{x\to\infty} \frac{\frac{1}{1+e^{-x}}\cdot(-e^{-x})}{-e^{-x}} = \lim_{x\to\infty} \frac{1}{1+e^{-x}} = 1.$$

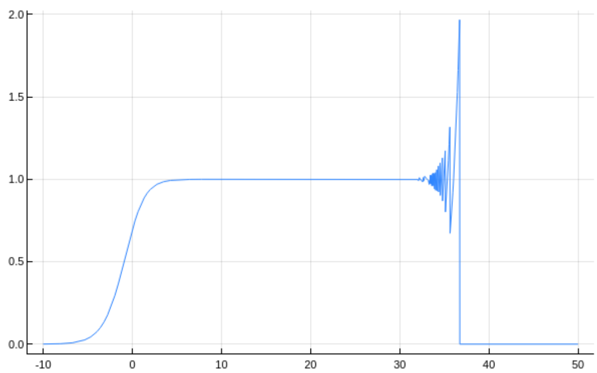
`Wolfram Alpha` również zwrócił wynik $= 1$.

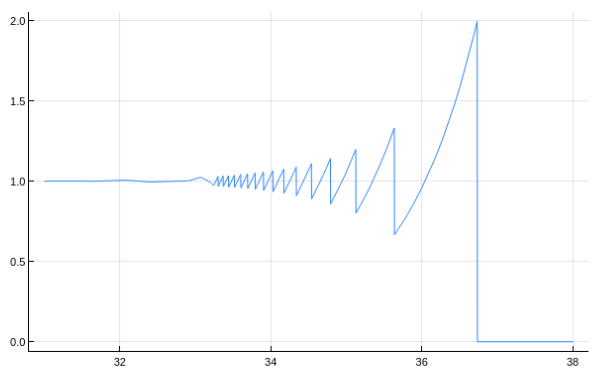
Poniżej znajdują się wykresy funkcji $f(x) = e^x \ln(1+e^{-x})$ narysowane w środowisku `WolframAlpha`:





Następnie skorzystano z biblioteki `Plotly` w języku *Julia*:





2.4 Obserwacje i wnioski

Patrząc na wygenerowane wykresy, trudno dostrzec obliczonej granicy funkcji równej 1. Można natomiast zauważyć, że w okolicy $x > 31$ wykres funkcji zaczyna oscylować w nietypowy sposób. Jest to związane z numerycznymi błędami spowodowanymi dodaniem wewnatrz logarytmu bardzo małej wartości e^{-x} do stosunkowo dużej jedynki. Powoduje to utratę cyfr znaczących, a następnie pomnożenie tak zaburzonej wartości, czyli $\ln(1 + e^{-x})$ przez bardzo dużą liczbę e^x, co wpływa na znaczne narastanie błędu. Obliczanie funkcji f można więc uznać za przykład kolejnego zadania niezbyt dobrze uwarunkowanego, gdyż bardzo małe zmiany wartości argumentu x powodują znaczące zmiany wyników, duże odchylenia. Są one widoczne na wykresie jako oscylacje.

W chwili, gdy $1 + e^{-x} = 1$ (e^{-x} zostaje całkowicie pochłonięte przez 1), to $\ln(1 + e^{-x}) = 0$, co tłumaczy pojawienie się wartości 0 na wykresie. W okolicy $x = 37$ funkcja na wykresach przyjjmuje wartość stale równą zero. Ma to związek z bardzo szybko malejącą wartością funkcji e^{-x}. Dla $x = 37$ jest ona rzędu 10^{-17}, co jest wartością mniejszą od epsilona maszynowego dla `Float64`. W związku z tym $\ln(1 + e^{-37}) = \ln 1 = 0$.

Przykuwać uwagę może zachowanie środowiska `Wolfram`, w którym w okolicy $x = 750$ funkcja przeskakuje nagle do wartości 1, czyli swojej wartości granicy. Wynikać to może z faktu przepełnienia (overflow) dla e^x. Program musiałby policzyć wartość symbolu nieoznaczonego $0 \cdot \infty$, co daje wynik NaN, więc zamiast tego dokonuje pewnego rodzaju oszustwa i przyjęcia wartości funkcji w granicy równej 1.

3 Zadanie 3. - Rozwiązywanie układu równań

3.1 Przedstawienie problemu

W kolejnym zadaniu należało rozwiązać układ równań liniowych w postaci

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

dla danej macierzy współczynników $\mathbf{A} \in \mathbb{R}^{n\times n}$ i wektora prawych stron $\mathbf{b} \in \mathbb{R}^n$. Macierz $\mathbf{A}$ została generowana na dwa sposoby:

(a) $\mathbf{A} = \mathbf{H}_n$, gdzie $\mathbf{H}_n$ jest macierzą Hilberta stopnia n wygenerowaną za pomocą funkcji `A = hilb(n)` zdefiniowanej w dołączonym do zadania pliku. Macierz Hilberta to macierz kwadratowa z elementami danymi wzorem:

$$h_{ij} = \frac{1}{i + j - 1}$$

Na przykład macierz Hilberta 5x5 wygląda następująco:

$$H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{bmatrix}$$

(b) $\mathbf{A} = \mathbf{R}_n$, gdzie $\mathbf{R}_n$ jest macierzą stopnia n z zadanym wskaźnikiem uwarunkowania c wygenerowaną za pomocą funkcji `A = matcond(n,c)` zdefiniowanej w dołączonym do zadania pliku.

Zgodnie z poleceniem zadania, zadany układ równań należało rozwiązać za pomocą dwóch różnych metod:

(1) eliminacji Gaussa, czyli `x = A \ b`

(2) używając wzoru $x = \mathbf{A}^{-1}\mathbf{b}$, czyli, w języku Julia: `x = inv(A)*b`.

Ponadto, opisane wyżej eksperymenty należało przeprowadzić: dla macierzy Hilberta $\mathbf{H}_n$ z rosnącym stopniem $n > 1$ oraz dla macierzy losowej $\mathbf{R}_n$ dla $n \in \{5, 10, 20\}$ z rosnącym wskaźnikiem uwarunkowania $c \in \{1, 10, 10^3, 10^7, 10^{12}, 10^{16}\}$.

3.2 Rozwiązanie

Otrzymane układy równań rozwiązano metodą eliminacji Gaussa oraz metodą macierzy odwrotnej i dla uzyskanych wyników obliczono błędy względne:

$$\Delta\tilde{\mathbf{x}} = \frac{||\mathbf{x}-\tilde{\mathbf{x}}||}{||\mathbf{x}||}$$

Funkcje, które zostały zaimplementowane do rozwiązania równania liniowego umieszczone w plikach źródłowych to: metodą Gaussa (**methodOfGaussianElimination()**), metodą inwersji (**methodOfInversion()**), a także występuje funkcja licząca błąd względny uzyskanych rozwiązań - **countingRelativeErrors()**.

W języku *Julia* za pomocą funkcji `cond(A)` mogłam również sprawdzić, jaki jest wskaźnik uwarunkowania wygenerowanej macierzy oraz za pomocą funkcji `rank(A)`, jaki jest rząd macierzy.

3.3 Uzyskane wyniki

W wyniku działania algorytmów otrzymano następujące wyniki, które zaprezentowano w poniższych tabelach (są to błędy obliczone dla każdej z metod).

Wyniki dla macierzy Hilberta H_n :

n	Rząd	`cond(A)`	Błąd względny Gaussa	Błąd względny inwersji
1	1	1.0	0.0	0.0
2	2	19.28147006790397	$5.66104886700367 \cdot 10^{-16}$	$1.404333387430680 \cdot 10^{-15}$
3	3	524.0567775860644	$8.02259377226772 \cdot 10^{-15}$	0.0
4	4	15513.73873892924	$4.13740962243038 \cdot 10^{-14}$	0.0
5	5	476607.25024259434	$1.682842629922719 \cdot 10^{-12}$	$3.354436058435963 \cdot 10^{-12}$
6	6	$1.495105864225466 \cdot 10^{7}$	$2.61891330231162 \cdot 10^{-10}$	$2.016375940434765 \cdot 10^{-10}$
7	7	$4.7536735658312 \cdot 10^{8}$	$1.260686722417154 \cdot 10^{-8}$	$4.71328039723203 \cdot 10^{-9}$
8	8	$1.525757553806004 \cdot 10^{10}$	$6.12408955572308 \cdot 10^{-8}$	$3.0774839030962 \cdot 10^{-7}$
9	9	$4.93153756446876 \cdot 10^{11}$	$3.875163418503247 \cdot 10^{-6}$	$4.54126830317664 \cdot 10^{-6}$
10	10	$1.602441699254171 \cdot 10^{13}$	$8.6703902370969 \cdot 10^{-5}$	0.0002501493411824886
11	11	$5.22267793928033 \cdot 10^{14}$	0.00015827808158590435	0.007618304284315809
12	11	$1.751473190709146 \cdot 10^{16}$	0.13396208372085344	0.258994120804705
13	11	$3.34414349733846 \cdot 10^{18}$	0.11039701117868264	5.331275639426837
14	12	$6.20078626316144 \cdot 10^{17}$	1.4554087127659643	8.71499275104814
15	12	$3.67439295346797 \cdot 10^{17}$	4.696668350857427	7.344641453111494
16	12	$7.86546777843164 \cdot 10^{17}$	54.15518954564602	29.84884207073541
17	12	$1.26368434266605 \cdot 10^{18}$	13.707236683836307	10.516942378369349
18	12	$2.244630992918912 \cdot 10^{18}$	9.134134521198485	7.575475905055309
19	13	$6.47195397654159 \cdot 10^{18}$	9.720589712655698	12.233761393757726
20	13	$1.355365790868822 \cdot 10^{18}$	7.549915039472976	22.062697257870493

Wyniki dla macierzy losowej R_n :

n	Rząd	c	Błąd dla metody eliminacji Gaussa	Błąd dla metody inwersji
5	5	1.0	$1.489520491948363 \cdot 10^{-16}$	$1.216188388897623 \cdot 10^{-16}$
5	5	10.0	$2.80866677486136 \cdot 10^{-16}$	$1.216188388897623 \cdot 10^{-16}$
5	5	1000.0	$8.90562200290678 \cdot 10^{-15}$	$1.32216329020653 \cdot 10^{-14}$
5	5	$1. \cdot 10^{7}$	$1.897153882444540 \cdot 10^{-10}$	$2.702114728380730 \cdot 10^{-10}$
5	5	$1. \cdot 10^{12}$	$3.149537434670237 \cdot 10^{-6}$	$1.092363057195766 \cdot 10^{-5}$
5	4	$1. \cdot 10^{16}$	0.08627458226764755	0.06987712429686843
10	10	1.0	$3.43990022795940 \cdot 10^{-16}$	$2.457583428003690 \cdot 10^{-16}$
10	10	10.0	$2.74204859601134 \cdot 10^{-16}$	$3.748544367384394 \cdot 10^{-16}$
10	10	1000.0	$9.55146748377680 \cdot 10^{-15}$	$1.456179286088797 \cdot 10^{-14}$
10	10	$1. \cdot 10^{7}$	$2.86819506455097 \cdot 10^{-10}$	$3.20919429481563 \cdot 10^{-10}$
10	10	$1. \cdot 10^{12}$	$7.8323491039955 \cdot 10^{-6}$	$6.008905030068726 \cdot 10^{-6}$
10	9	$1. \cdot 10^{16}$	0.06459023625268558	0.1128088209233879
20	20	1.0	$6.04536571471835 \cdot 10^{-16}$	$6.64280865943016 \cdot 10^{-16}$
20	20	10.0	$4.263892432392672 \cdot 10^{-16}$	$4.38502959679432 \cdot 10^{-16}$
20	20	1000.0	$4.28799551822986 \cdot 10^{-14}$	$4.01745365666494 \cdot 10^{-14}$
20	20	$1. \cdot 10^{7}$	$8.45568212845224 \cdot 10^{-12}$	$5.07788086257475 \cdot 10^{-11}$
20	20	$1. \cdot 10^{12}$	$2.423340464052032 \cdot 10^{-5}$	$2.17969222908066 \cdot 10^{-5}$
20	19	$1. \cdot 10^{16}$	0.21774410225511787	0.20868480280995097

3.4 Obserwacje i wnioski

Wyniki pokazują, że w przypadku macierzy Hilberta $\mathbf{H}_n$ błąd dla każdej z zastosowanych metod rośnie wraz ze wzrostem rozmiaru macierzy. Podobnie zachowuje się wskaźnik uwarunkowania - wraz ze wzrostem stopnia macierzy, rośnie wskaźnik uwarunkowania, a im większy wskaźnik uwarunkowania macierzy, tym większy błąd względny. Z dwóch zastosowanych metod – eliminacji Gaussa oraz odwrotności, można powiedzieć, że większymi błędami odznaczała się ta druga, zatem nieco lepszym algorytmem w przypadku macierzy Hilberta jest eliminacja Gaussa.

Jeśli chodzi o macierz losową $\mathbf{R}_n$, to dla danego ustalonego współczynnika uwarunkowania c, błędy nie zmieniają się jednak bardzo istotnie wraz ze wzrostem stopnia macierzy n. Dla danego c błędy dla różnych n (5,10 lub 20) nie różnią się więcej, niż o 1 rząd wielkości. Podobnie jak dla macierzy Hilberta zauważamy, że wzrost błędu względnego dla obu metod rozwiązywania układu równań jest bezpośrednio powiązany ze wzrostem wskaźnika uwarunkowania c (nawet jeśli macierz miała ten sam rozmiar, to im c był większy, tym większe generowały się błędy).

Na podstawie eksperymentów można wyciągnąć wnioski, że zadanie rozwiązywania układu równań liniowych dla macierzy Hilberta jest zadaniem bardzo źle uwarunkowanym, ponieważ wraz ze wzrostem stopnia macierzy gwałtownie rośnie jej wskaźnik uwarunkowania cond(A), a co za tym idzie - błąd względny przeprowadzanych obliczeń. Przykładowo cond($\mathbf{H}_5$) wynosi już około $1.5 \cdot 10^7$, natomiast Cond($\mathbf{H}_{10}$) aż $1.6 \cdot 10^{13}$.
Uzyskane rezultaty pokazują, że obliczenia prowadzone na macierzach o dużym współczynniku uwarunkowania generują duże błędy względne.

4 Zadanie 4. - "Złośliwy wielomian" Wilkinsona

4.1 Przedstawienie problemu

Celem zadania było obliczenie zer *wielomianu Wilkinsona* zapisanego w postaci iloczynowej p (1) oraz w postaci ogólnej P (2) oraz obliczenie wartości $|P(z_k)|$ i $|p(z_k)|$ wielomianu w obliczonych pierwiastkach oraz błędu bezwzględnego $|z_k - k|$ wyznaczonych pierwiastków.

$$
\begin{aligned}
p(x) &= (x-20)(x-19)(x-18)\cdots(x-2)(x-1) && (1)\\
P(x) &= x^{20} - 210x^{19} + 20615x^{18} - 1256850x^{17} + 53327946x^{16} - 1672280820x^{15} + && (2)\\
&\quad + 40171771630x^{14} - 756111184500x^{13} + 11310276995381x^{12} - 135585182899530x^{11} + \\
&\quad - 1307535010540395x^{10} - 10142299865511450x^{9} + 63030812099294896x^{8} + \\
&\quad - 311333643161390640x^{7} + 1206647803780373360x^{6} - 3599979517947607200x^{5} + \\
&\quad + 8037811822645051776x^{4} - 12870931245150988800x^{3} + 13803759753640704000x^{2} + \\
&\quad - 8752948036761600000x + 2432902008176640000
\end{aligned}
$$

Następnie należało nieco zaburzyć wielomian przez zmianę współczynnika przy x^{19} z -210 na $-210 - 2^{-23}$ i powtórzyć obliczenia oraz wyjaśnić zjawisko.

4.2 Rozwiązanie

Zadanie należało rozwiązać przy użyciu pakietu `Polynomials` w języku `Julia`. Znajdują się tam bardzo przydatne funkcje, na przykład:

1. **roots(P)** - obliczanie miejsc zerowych wielomianu P utworzonego z danych współczynników
2. **Poly(p)** - tworzenie wielomianu z tablicy p, zawierającej współczynniki wielomianu
3. **poly(p)** - stworznie wielomianu z tablicy p, zawierającej miejsca zerowie wielomianu
4. **polyval** - obliczanie wartości wielomianu P w zadanych punktach

Wykorzystanie powyższych funkcji pomogło nam stworzyć wielomiany P (na podstawie współczynników wielomianu Wilkinsona - postać kanoniczna) oraz p (na podstawie miejsc zerowych - postać iloczynowa). Następnie obliczono miejsca zerowe wielomianu P za pomocą funkcji `roots` oraz wykorzystując funkcję `polyval`: $|P(z_k)|$, $|p(z_k)|$ oraz $|z_k - k|$.

4.3 Uzyskane wyniki

Wyniki dla danych z podpunktu **(a)**:

k	$\lvert P(z_k)\rvert$	$\lvert p(z_k)\rvert$	$\lvert z_k - k\rvert$
1	36352.0	38400.0	$3.010924842783424 \cdot 10^{-13}$
2	181760.0	198144.0	$2.831823664450894 \cdot 10^{-11}$
3	209408.0	301568.0	$4.079034887638499 \cdot 10^{-10}$
4	$3.10681 \cdot 10^{6}$	$2.84467 \cdot 10^{6}$	$1.62624682609191 \cdot 10^{-8}$
5	$2.411468 \cdot 10^{7}$	$2.334668 \cdot 10^{7}$	$6.65769791297066 \cdot 10^{-7}$
6	$1.2015206 \cdot 10^{8}$	$1.188249 \cdot 10^{8}$	$1.075417522677923 \cdot 10^{-5}$
7	$4.8039833 \cdot 10^{8}$	$4.7829094 \cdot 10^{8}$	0.00010200279300764947
8	$1.68269107 \cdot 10^{9}$	$1.6784972 \cdot 10^{9}$	0.0006441703922384079
9	$4.46532659 \cdot 10^{9}$	$4.45785958 \cdot 10^{9}$	0.002915294362052734
10	$1.270712678 \cdot 10^{10}$	$1.269690726 \cdot 10^{10}$	0.009586957518274986
11	$3.575989555 \cdot 10^{10}$	$3.574346905 \cdot 10^{10}$	0.025022932909317674
12	$7.21677158 \cdot 10^{10}$	$7.214665062 \cdot 10^{10}$	0.04671674615314281
13	$2.1572362905 \cdot 10^{11}$	$2.1569633075 \cdot 10^{11}$	0.07431403244734014
14	$3.6538325094 \cdot 10^{11}$	$3.65344793 \cdot 10^{11}$	0.08524440819787316
15	$6.1398775347 \cdot 10^{11}$	$6.1393841561 \cdot 10^{11}$	0.07549379969947623
16	$1.55502775193 \cdot 10^{12}$	$1.55496109721 \cdot 10^{12}$	0.05371328339202819
17	$3.77762377830 \cdot 10^{12}$	$3.77753294694 \cdot 10^{12}$	0.025427146237412046
18	$7.19955486105 \cdot 10^{12}$	$7.199447475 \cdot 10^{12}$	0.009078647283519814
19	$1.027837616281 \cdot 10^{13}$	$1.027823565670 \cdot 10^{13}$	0.0019098182994383706
20	$2.746295274547 \cdot 10^{13}$	$2.746278890700 \cdot 10^{13}$	0.00019070876336257925

Wyniki dla danych z podpunktu **(b)** (powtórzenie eksperymentu dla wielomianu z zaburzonym współczynnikiem):

k	z_k	$\lvert P(z_k)\rvert$	$\lvert p(z_k)\rvert$
1	$0.9999999999998357 + 0.0im$	20992.0	22016.0
2	$2.0000000000550373 + 0.0im$	349184.0	365568.0
3	$2.99999999660342 + 0.0im$	$2.22156 \cdot 10^6$	$2.29529 \cdot 10^6$
4	$4.000000089724362 + 0.0im$	$1.04678 \cdot 10^7$	$1.072998 \cdot 10^7$
5	$4.99999857388791 + 0.0im$	$3.946393 \cdot 10^7$	$4.330393 \cdot 10^7$
6	$6.000020476673031 + 0.0im$	$1.2914841 \cdot 10^8$	$2.0612044 \cdot 10^8$
7	$6.99960207042242 + 0.0im$	$3.8812313 \cdot 10^8$	$1.75767091 \cdot 10^9$
8	$8.007772029099446 + 0.0im$	$1.07254732 \cdot 10^9$	$1.852548659 \cdot 10^{10}$
9	$8.915816367932559 + 0.0im$	$3.06557542 \cdot 10^9$	$1.3717431705 \cdot 10^{11}$
10	$10.095455630535774 - 0.6449328236240688im$	$7.14311363803582 \cdot 10^9$	$1.491263381675401 \cdot 10^{12}$
11	$10.095455630535774 + 0.6449328236240688im$	$7.14311363803582 \cdot 10^9$	$1.491263381675401 \cdot 10^{12}$
12	$11.793890586174369 - 1.6524771364075785im$	$3.35775611317185 \cdot 10^{10}$	$3.296021414130166 \cdot 10^{13}$
13	$11.793890586174369 + 1.6524771364075785im$	$3.35775611317185 \cdot 10^{10}$	$3.296021414130166 \cdot 10^{13}$
14	$13.992406684487216 - 2.5188244257108443im$	$1.061206453308197 \cdot 10^{11}$	$9.54594159518366 \cdot 10^{14}$
15	$13.992406684487216 + 2.5188244257108443im$	$1.061206453308197 \cdot 10^{11}$	$9.54594159518366 \cdot 10^{14}$
16	$16.73074487979267 - 2.812624896721978im$	$3.31510347598176 \cdot 10^{11}$	$2.742089401676406 \cdot 10^{16}$
17	$16.73074487979267 + 2.812624896721978im$	$3.31510347598176 \cdot 10^{11}$	$2.742089401676406 \cdot 10^{16}$
18	$19.5024423688181 - 1.940331978642903im$	$9.53942460981782 \cdot 10^{12}$	$4.252502487993469 \cdot 10^{17}$
19	$19.5024423688181 + 1.940331978642903im$	$9.53942460981782 \cdot 10^{12}$	$4.252502487993469 \cdot 10^{17}$
20	$20.84691021519479 + 0.0im$	$1.11445350451 \cdot 10^{13}$	$1.374373319724971 \cdot 10^{18}$

k	z_k	$\lvert z_k - k\rvert$
1	$0.9999999999998357 + 0.0im$	$1.643130076445231 \cdot 10^{-13}$
2	$2.0000000000550373 + 0.0im$	$5.50373080443478 \cdot 10^{-11}$
3	$2.99999999660342 + 0.0im$	$3.396579906222996 \cdot 10^{-9}$
4	$4.000000089724362 + 0.0im$	$8.97243621622578 \cdot 10^{-8}$
5	$4.99999857388791 + 0.0im$	$1.426112089752962 \cdot 10^{-6}$
6	$6.000020476673031 + 0.0im$	$2.047667303095579 \cdot 10^{-5}$
7	$6.99960207042242 + 0.0im$	0.00039792957757978087
8	$8.007772029099446 + 0.0im$	0.007772029099445632
9	$8.915816367932559 + 0.0im$	0.0841836320674414
10	$10.095455630535774 - 0.6449328236240688im$	0.6519586830380406
11	$10.095455630535774 + 0.6449328236240688im$	1.1109180272716561
12	$11.793890586174369 - 1.6524771364075785im$	1.665281290598479
13	$11.793890586174369 + 1.6524771364075785im$	2.045820276678428
14	$13.992406684487216 - 2.5188244257108443im$	2.5188358711909045
15	$13.992406684487216 + 2.5188244257108443im$	2.7128805312847097
16	$16.73074487979267 - 2.812624896721978im$	2.9060018735375106
17	$16.73074487979267 + 2.812624896721978im$	2.825483521349608
18	$19.5024423688181 - 1.940331978642903im$	2.454021446312976
19	$19.5024423688181 + 1.940331978642903im$	2.004329444309949
20	$20.84691021519479 + 0.0im$	0.8469102151947894

4.4 Obserwacje i wnioski

Dla uzyskanych pierwiastków wartości wielomianów $P(x)$ i $p(x)$ dla podpunktu **(a)** oraz **(b)** są bardzo dalekie od 0 - dla ostatniego pierwiastka wartość wielomianu jest aż rzędu 10^{13}. Stało się

to, chociaż uzyskane miejsca zerowe nie różnią się wiele od tych, które powinny zostać otrzymane.
Pomimo, że na pierwszy rzut oka wartości dla tych pierwiastków $|z_k - k|$ nie wydają się takie duże (np odchylenie 10^{-13} może sprawiać pozór nieznaczącego), to jednak przy obliczaniu wartości wielomianu Wilkinsona te niedokładnie obliczone wyniki zostają pomnożone przez duże współczynniki. W tym konkretnym wielomianie drobny błąd w wartości pierwiastka mnożony jest przez czynnik rzędu 19!. Można zauważyć, że wartości obliczonych błędów rosną wraz ze wzrostem wartości dla danego pierwiastka.
Wpływające na błędy w trakcie obliczania miejsc zerowych współczynniki wielomianu P nie są dokładnie reprezentowane w arytmetyce `Float64`. Jest to widoczne przy powtórzeniu eksperymentu Wilkinsona - w przypadku zaburzenia jednego ze współczynników wielomianu, funkcja `roots` zwróciła pierwiastki zespolone.
Zatem z pierwszej części zadania wnioskujemy, że nieprawidłowe wartości wynikają z ograniczeń zastosowanej arytmetyki - wartości poszczególnych współczynników nie mogą być dokładnie przechowywane. Jak podaje wskazówka w poleceniu zadania - arytmetyka `Float64` w języku Julia ma od 15 do 17 cyfr znaczących w systemie dziesiętnym. Ma to znaczący wpływ na jakość otrzymanych wyników.
Z drugiej części zadania można wywnioskować, że rozwiązywany przez nas problem jest zadaniem bardzo źle uwarunkowanym pod względem szukania pierwiastka zadanego wielomianu. Wprowadzenie zaburzenia jednego ze współczynników o jedynie 2^{-23} wpłynęło na to, iż w wyniku uzyskaliśmy pierwiastki zespolone.

5 Zadanie 5. - Model logistyczny, model wzrostu populacji

5.1 Przedstawienie problemu

W przedostatnim zadnaiu należało zbadać model wzrostu populacji (model logistyczny). Musieliśmy rozważyć równanie rekurenecyjne:

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

w którym r oznaczało pewną daną stałą, $r(1-p_n)$ to czynnik wzrostu populacji, a p_0 było wielkością populacji stanowiącą procent maksymalnej wielkości populacji dla danego stanu środowiska.

Żeby wykonać to zadanie, wykonano kolejne eksperymenty:

1. Dla danych $p_0 = 0.01$ i $r = 3$ przeprowadzić 40 iteracji powyższego wyrażenia w arytmetyce `Float32`. Następnie ponownie wykonać 40 iteracji wyrażenia z małą modyfikacją - przeprowadzeniem 10 iteracji, zatrzymaniem, zastosowaniem obcięcia wyniku odrzucając cyfry po trzecim miejscu po przecinku (uzyskano liczbę 0.722) i kontynuowaniem dalej obliczeń do 40-stej iteracji traktując to tak, jak gdyby był to ostatni wynik na wyjściu. Oczywiście należało w końcowej fazie dokonać porównania obu rezulatatów.

2. Dla danych $p_0 = 0.01$ i $r = 3$ przeprowadzić 40 iteracji powyższego wyrażenia w arytmetyce `Float32` i `Float64`. Następnie porównać rezultaty iteracji dla obu arytmetyk.

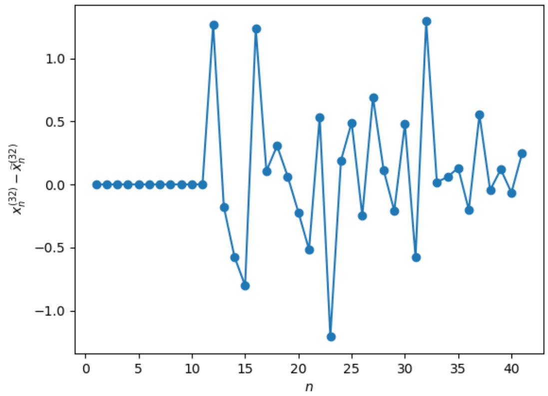
5.2 Rozwiązanie

Zaimplementowany został program w języku `Julia`, który oblicza kolejne wartości wyrażenia rekurencyjnego w pierwszych 40 iteracjach w arytmetykach `Float32`, `Float64` oraz `Float32` z obcięciem wyniku 10. iteracji do 3 cyfr po przecinku. Obliczono także różnice między wartościami w `Float32` i `Float64` oraz `Float32` i `Float32` z obcięciem.

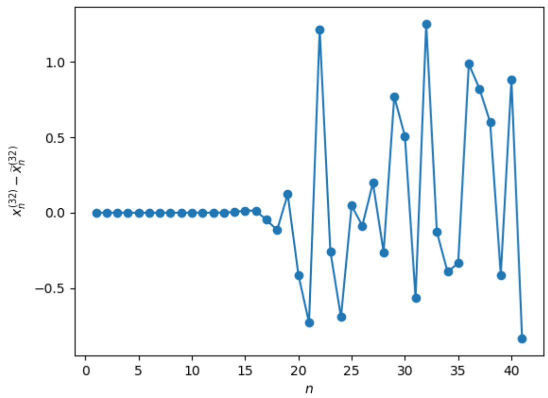
5.3 Uzyskane wyniki

Obliczenia w arytmetyce `Float64` przeprowadzono bez wprowadzania intencjonalnego zaburzeina danych, aby porównać wcześniej otrzymane dane z możliwie dokładnymi wynikami przeprowadzonych obliczeń.
Wyniki n-tej iteracji wyrażenia w poszczególnych eksperymentach (i arytmetykach) przedstawione są poniżej. Dla lepszego pokazania rozbieżności pomiędzy kolejnymi iteracjammi, załączono wykresy przedstawiające różnice otrzymanych wyników.

Arytmetyka `Float32` oraz `Float32` z obcięciem do 3 cyfr po przecinku po 10 iteracjach:



Arytmetyka `Float32` oraz `Float64`:



n	`Float32 bez modyfikacji`	`Float32 z modyfikacja`	`Float64 bez modyfikacji`
1	0.0397	0.0397	0.0397
2	0.15407173	0.15407173	0.15407173000000002
3	0.5450726	0.5450726	0.5450726260444213
4	1.2889781	1.2889781	1.2889780011888006
5	0.1715188	0.1715188	0.17151914210917552
6	0.5978191	0.5978191	0.5978201201070994
7	1.3191134	1.3191134	1.3191137924137974
8	0.056273222	0.056273222	0.056271577646256565
9	0.21559286	0.21559286	0.21558683923263022
10	0.7229306	0.7229306	0.722914301179573
11	1.3238364	1.3241479	1.3238419441684408
12	0.037716985	0.036488414	0.03769529725473175
13	0.14660022	0.14195944	0.14651838271355924
14	0.521926	0.50738037	0.521670621435246
15	1.2704837	1.2572169	1.2702617739350768
16	0.2395482	0.28708452	0.24035217277824272
17	0.7860428	0.9010855	0.7881011902353041
18	1.2905813	1.1684768	1.2890943027903075
19	0.16552472	0.577893	0.17108484670194324
20	0.5799036	1.3096911	0.5965293124946907
21	1.3107498	0.9289217	1.3185755879825978
22	0.088804245	0.34568182	0.058377608259430724
23	0.3315584	1.0242395	0.22328659759944824
24	0.9964407	0.94975823	0.7435756763951792
25	1.0070806	1.0929108	1.315588346001072
26	0.9856885	0.7882812	0.07003529560277899
27	1.0280086	1.2889631	0.26542635452061003
28	0.9416294	0.17157483	0.8503519690601384
29	1.1065198	0.59798557	1.2321124623871897
30	0.7529209	1.3191822	0.37414648963928676
31	1.3110139	0.05600393	1.0766291714289444
32	0.0877831	0.21460639	0.8291255674004515
33	0.3280148	0.7202578	1.2541546500504441
34	0.9892781	1.3247173	0.29790694147232066
35	1.021099	0.034241438	0.9253821285571046
36	0.95646656	0.13344833	1.1325322626697856
37	1.0813814	0.48036796	0.6822410727153098
38	0.81736827	1.2292118	1.3326056469620293
39	1.2652004	0.3839622	0.0029091569028512065
40	0.25860548	1.093568	0.011611238029748606

5.4 Obserwacje i wnioski

Wprowadzenie niewielkiej zmiany w p_{10} wpływa na poprawność otrzymywanych wyników. Zmodyfikowana wartość p_{40} jest równa aż 1.093568. Porównujemy ją z p_{40} bez wprowadzonego zaburzenia w arytmetyce `Float32` - wartość ta wynosi zaledwie 0.25860548 i jest ponad czterokrotnie mniejsza. W arytmetyce `Float64`, w której możemy oczekiwać dokładniejszych wyników - wartość p_{40} wynosi 0.011611238029748606 - jest to ok. sto razy mniejsza wartość, niż p_{40} uzyskana w wyniku zaburzenia. Obcięcie pewnej liczby cyfr znaczących ma wpływ na pozostałe wyniki. Raz popełniony błąd niedokładności kumuluje się i propaguje na kolejne wartości dla ciągu, dlatego

że każda następna wartość jest zależna od poprzedniej - nazywamy to sprzężeniem zwrotnym. Obcięcie wyniku 10. iteracji do 3 cyfr po przecinku tworzy błąd względny rzędu zaledwie 0.13%, jednak wyniki kolejnych iteracji zaczynają znacznie się rozbiegać.
W drugiej części eksperymentu chcieliśmy pokazać, jak precyzja zastosowanej arytmetyki wpływa na uzyskiwane wyniki. Dane nie zostały w żaden sposób zaburzone, jednak dostrzegamy, że od pewnego momentu znacznie się one różnią, w zależności od tego, czy obliczenia wykonywane są w arytmetyce `Float32`, czy `Float64` (zauważalne różnice pojawiają się w poblizu 20. iteracji). Dziwić może fakt, że wyniki wyższych iteracji są nieskolerowane - obrazuje to istnienie pewnego chaosu w systemie, tzw. *chaosu deterministycznego*.
W zadaniu występuje zjawisko czułej zależności od warunków początkowych. Zwiększając precyzję obliczeń, możemy jedynie opóźnić zjawisko niemożności przewidywania, nie możemy jednak całkowicie go uniknąć. W nastepnych iteracjach wymagana jest coraz większa liczba cyfr znaczących, żeby dokładnie zapisać wynik, przez co od pewnego momentu niewystarczająco dokładne stają się obliczenia nawet we `Float64`. W obliczeniach komputerowych nie dysponujemy jednak arytmetyką o nieskończonej precyzji, zatem błędy zawsze będą występować, a następnie powiększać się przez przeniesienie zaburzonych danych jako wejścia kolejnych iteracji. Zastosowana precyzja powinna być tym większa, im dalszej iteracji wyniku potrzebujemy.
Mówiąc nieformalnie, proces numeryczny jest niestabilny, jeśli niewielkie błędy, popełnione w początkowym stadium procesu kumulują się w kolejnych stadiach, powodując poważną utratę dokładności obliczeń. Zaobserowowana w tym zadaniu numeryczna niestabilność jest trudna do uniknięcia.

6 Zadanie 6. - Iterowanie funkcji kwadratowej

6.1 Przedstawienie problemu

Ostatnie zadanie polegało na przeprowadzeniu w arytmetyce `Float64` iteracji równania rekurencyjnego:

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

gdzie c jest pewną stałą, dla danych wejściowych:

1. $c = -2$, $x_0 = 1$,
2. $c = -2$, $x_0 = 2$,
3. $c = -2$, $x_0 = 1.99999999999999$,
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

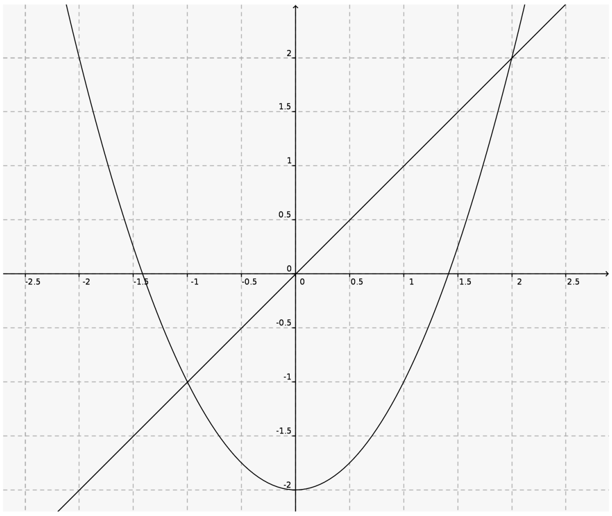
Kolejne wartości wyrażenia obliczane są przy pomocy funkcji w języku `Julia`, która przyjmuje jako argumenty warunki początkowe x_0 i c i wykonuje zadaną liczbę iteracji. Zaimplementowana funkcja liczy rekurencyjnie podane wyrażenie.

6.3 Wyniki

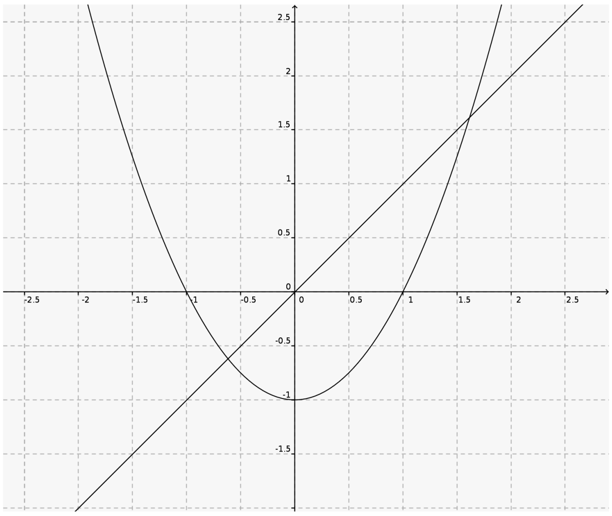
W poniższej tabeli zaprezentowane są wyniki dla każdej z 40 iteracji w zależności od wartości x_0 i c. Numery kolumn, podane w 1. wierszu tabeli, oznaczają konkretny z zestawów danych, które zostały opisane w podpunkcie **6.1**.

n	1	2	3	4	5	6	7
1	−1.0	2.0	1.99999999999996	0.0	0.0	−0.4375	−0.9375
2	−1.0	2.0	1.9999999999998401	−1.0	−1.0	−0.80859375	−0.12109375
3	−1.0	2.0	1.9999999999993605	0.0	0.0	−0.3461761474609375	−0.9853363037109375
4	−1.0	2.0	1.999999999997442	−1.0	−1.0	−0.8801620749291033	−0.029112368589267135
5	−1.0	2.0	1.9999999999897682	0.0	0.0	−0.2253147218564956	−0.9991524699951226
6	−1.0	2.0	1.9999999999590727	−1.0	−1.0	−0.9492332761147301	−0.0016943417026455965
7	−1.0	2.0	1.999999999836291	0.0	0.0	−0.0989561875164966	−0.9999971292061947
8	−1.0	2.0	1.9999999993451638	−1.0	−1.0	−0.9902076729521999	$-5.741579369278327e-6$
9	−1.0	2.0	1.9999999973806553	0.0	0.0	−0.01948876442658909	−0.9999999999670343
10	−1.0	2.0	1.999999989522621	−1.0	−1.0	−0.999620188061125	$-6.593148249578462e-11$
11	−1.0	2.0	1.9999999580904841	0.0	0.0	−0.0007594796206411569	−1.0
12	−1.0	2.0	1.9999998323619383	−1.0	−1.0	−0.9999994231907058	0.0
13	−1.0	2.0	1.9999993294477814	0.0	0.0	$-1.1536182557003727e-6$	−1.0
14	−1.0	2.0	1.9999973177915749	−1.0	−1.0	−0.9999999999986692	0.0
15	−1.0	2.0	1.9999892711734937	0.0	0.0	$-2.6616486792363503e-12$	−1.0
16	−1.0	2.0	1.9999570848090826	−1.0	−1.0	−1.0	0.0
17	−1.0	2.0	1.999828341078044	0.0	0.0	0.0	−1.0
18	−1.0	2.0	1.9993133937789613	−1.0	−1.0	−1.0	0.0
19	−1.0	2.0	1.9972540465439481	0.0	0.0	0.0	−1.0
20	−1.0	2.0	1.9890237264361752	−1.0	−1.0	−1.0	0.0
21	−1.0	2.0	1.9562153843260486	0.0	0.0	0.0	−1.0
22	−1.0	2.0	1.82677862987391	−1.0	−1.0	−1.0	0.0
23	−1.0	2.0	1.3371201625639997	0.0	0.0	0.0	−1.0
24	−1.0	2.0	−0.21210967086482313	−1.0	−1.0	−1.0	0.0
25	−1.0	2.0	−1.9550094875256163	0.0	0.0	0.0	−1.0
26	−1.0	2.0	1.822062096315173	−1.0	−1.0	−1.0	0.0
27	−1.0	2.0	1.319910282828443	0.0	0.0	0.0	−1.0
28	−1.0	2.0	−0.2578368452837396	−1.0	−1.0	−1.0	0.0
29	−1.0	2.0	−1.9335201612141288	0.0	0.0	0.0	−1.0
30	−1.0	2.0	1.7385002138215109	−1.0	−1.0	−1.0	0.0
31	−1.0	2.0	1.0223829934574389	0.0	0.0	0.0	−1.0
32	−1.0	2.0	−0.9547330146890065	−1.0	−1.0	−1.0	0.0
33	−1.0	2.0	−1.0884848706628412	0.0	0.0	0.0	−1.0
34	−1.0	2.0	−0.8152006863380978	−1.0	−1.0	−1.0	0.0
35	−1.0	2.0	−1.3354478409938944	0.0	0.0	0.0	−1.0
36	−1.0	2.0	−0.21657906398474625	−1.0	−1.0	−1.0	0.0
37	−1.0	2.0	−1.953093509043491	0.0	0.0	0.0	−1.0
38	−1.0	2.0	1.8145742550678174	−1.0	−1.0	−1.0	0.0
39	−1.0	2.0	1.2926797271549244	0.0	0.0	0.0	−1.0
40	−1.0	2.0	−0.3289791230026702	−1.0	−1.0	−1.0	0.0

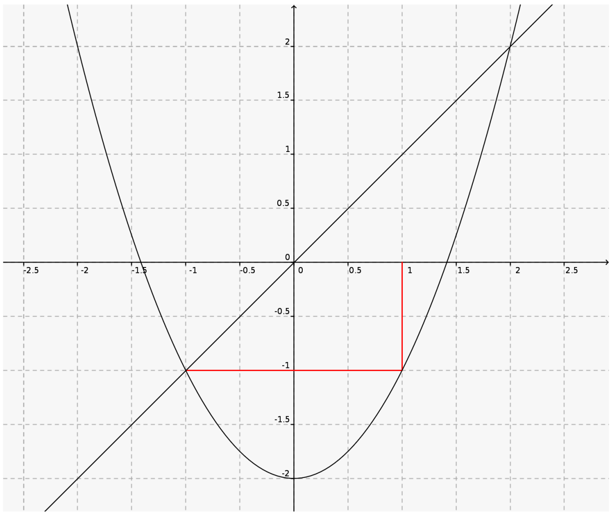
Następnie możemy obejrzeć wygenerowane wykresy, przedstawiające iteracje graficzne wyrażenia $x_{n+1} = x_n^2 + c$ dla wybranych x_0 i c:



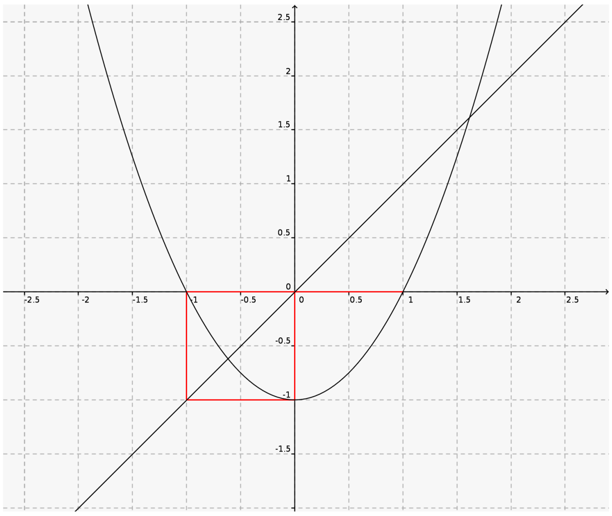
$A) x_{n+1} = x_n^2 - 2$



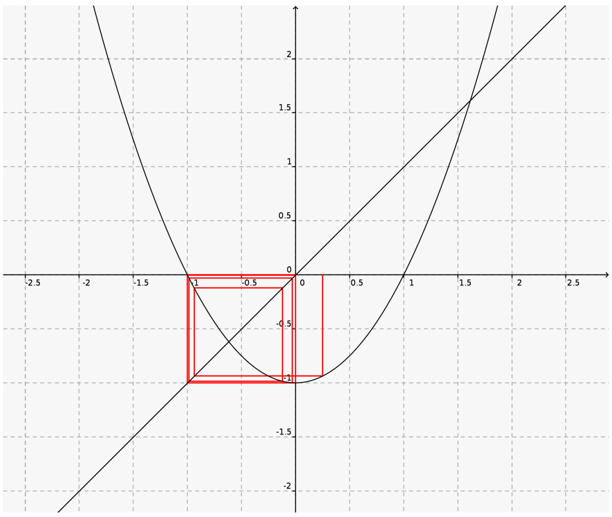
$B) x_{n+1} = x_n^2 - 1$



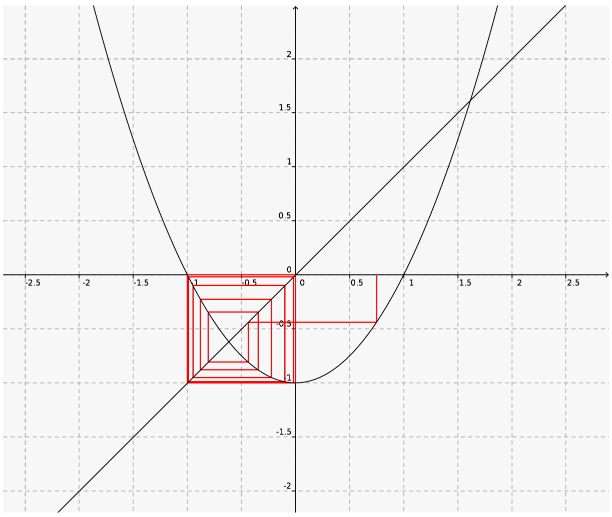
$C) x_0 = 1, c = -2$



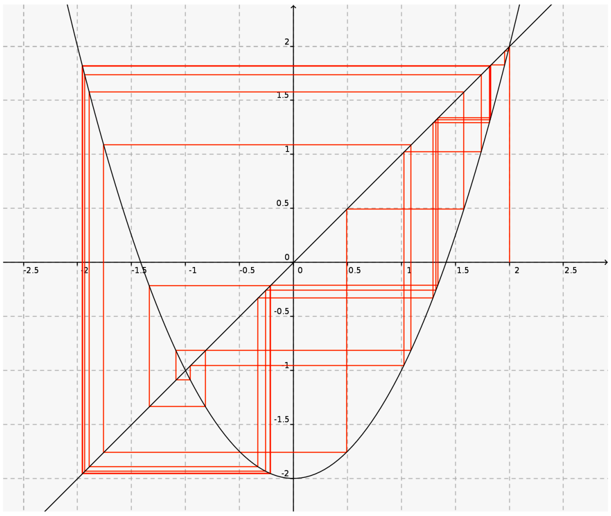
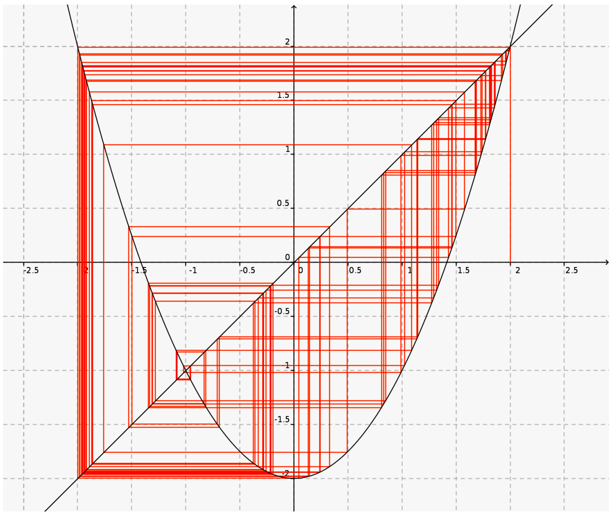
$D) x_0 = 1, c = -1$



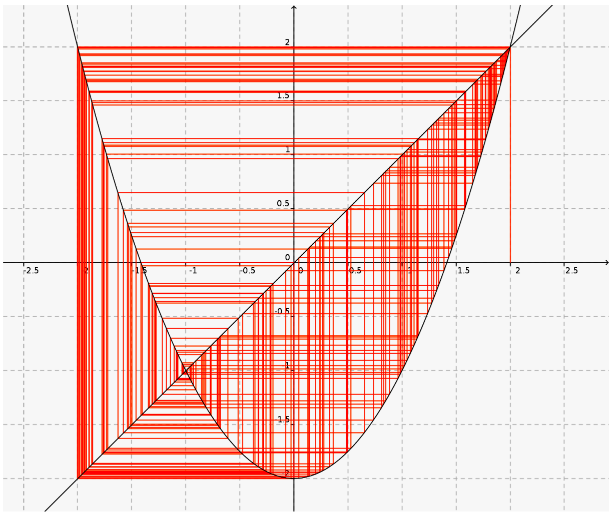
$E) x_0 = 0.25, c = -1$



$F) x_0 = 0,75, c = -1$

$G) x_0 = 1.99999999999999, c = -2, 40 iteracji$
$H) x_0 = 1.99999999999999, c = -2, 100 iteracji$



$I) x_0 = 1.99999999999999, c = -2, 200 iteracji$

6.4 Obserwacje i wnioski

Wyniki zawarte w tabeli pokazują, iż kolejne iteracje wyrażenia $x_{n+1} := x_n^2 + c$ mogą zachowywać się w zupełnie odmienny sposób dla różnych warunków początkowych. W dużej mierze zależne są przede wszystkim od danych wejściowych c i x_0. Wyrażenie ma stałą wartość dla warunków początkowych a) i b), jednak nawet minimalna zmiana punktu startowego c) wystarcza, aby ciąg stał się rozbieżny, co może świadczyć o złym uwarunkowaniu tego zadania. Dla warunków początkowych d) i e) wyrażenie ma charakter naprzemienny - dla parzystych argumentów przyjmuje wartość -1, dla nieparzystych 0. Dla warunków początkowych f) i g) wyrażenie początkowo zachowuje się niestabilnie, jednak stabilizuje się już po kilkunastu iteracjach, również naprzemiennie na wartościach -1 i 0. Zadanie to jest dosyć podobne do poprzedniego zadania o modelu wzrostu populacji.
Czy obserwowane zachowanie chaotyczne jest spowodowane operacją podnoszenia liczb do potęgi 2, której dokonujemy? W zadaniu 5. można było stwierdzić, iż za niestabilność numeryczną odpowiedzialna jest jedynie czynność podnoszenia do kwadratu, jednak iteracje przeprowadzone w zadaniu 6. pokazują, że jest to błędny wniosek. Stabilność układu określa to, jak pewien

układ matematyczny jest wrażliwy na zmiany w danych. Patrząc na przedstawioną wyżej tabelę dostrzegamy, że dla danych, gdzie $x_0 = 1$ lub $x_0 = 2$ a $c = -2$ iteracja zachowuje się w sposób stabilny. Otrzymaliśmy w pierwszej kolumnie same wartości -1, a w drugiej 2. Jednak kiedy wartość x_0 przyjmuje niewielkie odchylenie od 2, czyli podaną w trzecim przykładzie $x_0 = 1.99999999999999$, obserwujemy niestabilność zobrazowaną także na przedstawionych wykresach. Z każdą iteracją otrzymujemy wtedy coraz bardziej niepoprawne wyniki, podczas gdy moglibyśmy się spodziewać, iż ponieważ właściwie $x_0 \approx 2$, to otrzymane rezultaty będą zbliżone do $[2.0, 2.0, 2.0, \ldots, 2.0]$.
Wnioskujemy, iż niektóre początkowe dane skutkują stabilnym zachowaniem, a inne tworzą niestabilność wyników. Jeśli spojrzymy na zadany przedział $[-2, 2]$, to występuje w nim niewiele wartości prowadzących do rozwiązań stabilnych. Funkcja $\phi(x) = x^2 - 2$ ma 2 stałe punkty, czyli takie wartości x, że $\phi(x) = x$. Są to wartości dobrze widoczne na wykresie: $x = -1$ oraz $x = 2$. Wartości początkowe, dla których funkcja $\phi(x)$ jest zbieżna do tych punktów, doprowadzają do rozwiązań stabilnych. Metoda iteracji graficznej pozwoliła dostrzec zbieżność jednak tylko dla pojedynczych wartości x_0, a mianowicie: $-2, -1, 0, 1, 2$, a praktycznie $\phi(x)$ jest rozbieżna. Analizowanie błędu jest trudnym zadaniem, a zwłaszcza widzimy to, gdy wartość $c = -2$ zastąpiono $c = -1$. Dla takiej funkcji $\phi(x)$ uzystaliśmy punkty stałe $\frac{1-\sqrt{5}}{2}$ oraz $\frac{1+\sqrt{5}}{2}$. Wystąpiło również interesujące zjawisko, że kiedy zaczynamy od x_0 równego $1, -1, 0.75$ czy 0.25, po jakiejś liczbie iteracji proces ustala się i powtarzają się tylko dwie wartości: 0 i -1. Taki rezultat możemy również uzyskać poprzez wybranie wielu innych wartości początkowych. Spoglądając na wykresy, widzimy, iż układ sprzężenia zwrotnego dla takich wartości x_0 jest w stanie idealnie stabilnym, tzn. jest przewidywalny, a niewielkie błędy powstające w kolejnych iteracjach zanikają albo ulegają redukcji, zatem istnieje możliwość ich pominięcia. W takich sytuacjach arytmetyka o skończonej precyzji wystarcza do analizy i daje wiarygodne rezultaty.