# Obliczenia Naukowe - Laboratorium 2

Michał Waluś 279695

Listopad 2025

## 1 Zadanie 1

**Opis problemu:** Używając języka Julia i typów `Float32` i `Float64`, obliczenie iloczynu skalarnego wektorów $x$ i $y$ na cztery sposoby:

1. "w przód od najmniejszego do największego indeksu
2. "w tył w odwrotnej kolejności niż "w przód"
3. od największego do najmniejszego, dodając dodatnie i ujemne składniki osobno
4. odwrotnie niż w 3.

oraz porównanie uzyskanych wartości z wynikami uzyskanymi zastępując $x$, wektorem $x_1$ o zbliżonej wartości.

$$x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$$

$$x_1 = [2.718281828, -3.141592654, 1.414213562, 0.577215664, 0.301029995]$$

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

**Rozwiązanie**: Dla 1. i 2. policzenie sumy z użyciem pętli, dla 3. i 4. wyliczenie najpierw $x_i y_i$ dla wszystkich współrzędnych, podzielenie na dwie tablice, posortowanie ich (odpowiednio, w zależności od punktu), obliczenie sum tablic i dodanie ich.

**Otrzymane wyniki:**

Wyniki zwrócone przez program:

```
Float32 forward: -0.4999443
Float32 backward: -0.4543457
Float32 from biggest: -0.5
Float32 from smallest: -0.5

Float64 forward: 1.0251881368296672e-10
Float64 backward: -1.5643308870494366e-10
Float64 from biggest: 0.0
Float64 from smallest: 0.0

Results for x1:

Float32 forward: -0.4999443
Float32 backward: -0.4543457
Float32 from biggest: -0.5
```

```
Float32 from smallest: -0.5

Float64 forward: -0.004296342739891585
Float64 backward: -0.004296342998713953
Float64 from biggest: -0.004296342842280865
Float64 from smallest: -0.004296342842280865
```

| Metoda | (32 bit) $x \cdot y$ | (32 bit) $x_1 \cdot y$ | (64 bit) $x \cdot y$ | (64 bit) $x_1 \cdot y$ |
|---|---|---|---|---|
| 1. | $-0.49994$ | $-0.49994$ | $1.0252 \cdot 10^{-10}$ | $-4.2963 \cdot 10^{-3}$ |
| 2. | $-0.45435$ | $-0.45435$ | $-1.5643 \cdot 10^{-10}$ | $-4.2963 \cdot 10^{-3}$ |
| 3. | $-0.5$ | $-0.5$ | $0.0$ | $-4.2963 \cdot 10^{-3}$ |
| 4. | $-0.5$ | $-0.5$ | $0.0$ | $-4.2963 \cdot 10^{-3}$ |

Tabela 1: Wyniki zaokrąglone do 5 cyfr znaczących

Zmiana danych wejściowych nie jest widoczna w wynikach dla zmiennych 32-bitowy (prawdopodobnie, ponieważ nie mają one wystarczającej precyzji by dane były zapisane inaczej). Wyniki dla zmiennych 64-bitowych znacząco się różnią, dla pierwszych dwóch metod aż o 7 rzędów wielkości, za to dla pozostałych dwóch metod, o ile wyniki dla oryginalnych danych wyniosły 0, dla zmienionych danych pozostały takie same. Porównując z wartościami dokładnymi: $x \cdot y = -1.00657107000000 \cdot 10^{-11}$ oraz $x_1 \cdot y = -4.2963400 \cdot 10^{-3}$, widzimy że w drugim przypadku wyniki są bardzo zbliżone do prawdziwego.

**Wnioski:** Dla tych danych możemy zaobserwować, że zmiana, która może wydawać się pomijalna - zmniejszenie danych wejściowych o mniej niż $10^{-6}\%$, zmienia wynik o aż 8 rzędów wielkości. Dodatkowo, ponieważ nowa wartość jest znacznie bliższa danym wejściowym, nasz program jest w stanie dość dokładnie ją obliczyć używając zmiennych 64-bitowych. Możemy tutaj zaobserwować, że - dla odpowiednio dobranych danych - nawet ekstremalnie mała zmiana może spowodować bardzo dużą różnicę w wyniku oraz że nasz algorytm przestanie zwracać prawdziwe wyniki.

# 2 Zadanie 2

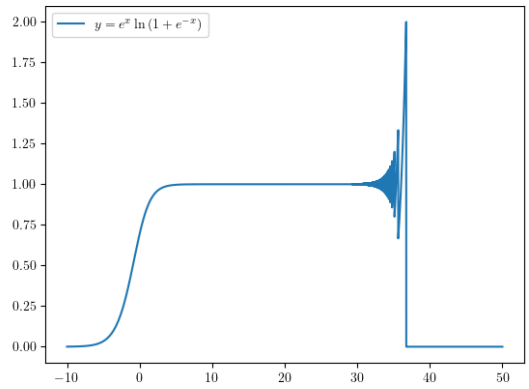
**Opis problemu:** Wygenerowanie wykresu funkcji $f(x) = e^x \ln(1 + e^{-x})$ z użyciem dwóch programów do wizualizacji oraz porównanie z granicą $\lim_{x\to\infty} f(x)$.

**Rozwiązanie:** Generujemy wykresy za pomocą dwóch programów, tutaj przedstawione zostały wykresy stworzone z użyciem biblioteki matplotlib w języku Python oraz dostępnego na stronie Desmos narzędzia Graphic Calculator (przedstawione na Rysunku 1). Następnie obliczamy granicę (poprzez $\overset{\text{H}}{=}$ oznaczamy użycie reguły l'Hopitala):

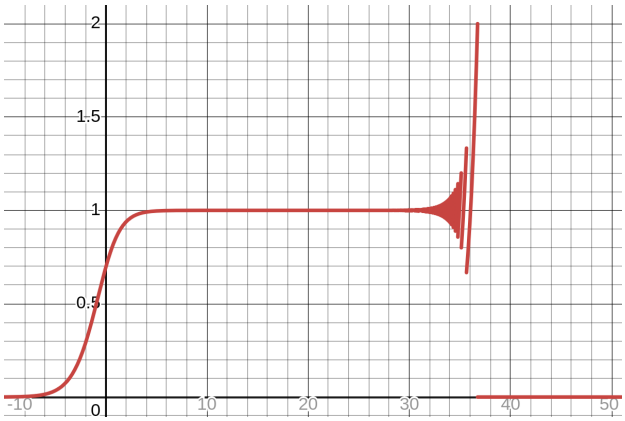
$$\lim_{x\to\infty} e^x \ln\left(1 + e^{-x}\right) = \lim_{x\to\infty} \frac{\ln(1 + e^{-x})}{e^{-x}} \overset{\text{H}}{=} \lim_{x\to\infty} \frac{\frac{-e^{-x}}{1+e^{-x}}}{-e^{-x}} = \lim_{x\to\infty} \frac{1}{1 + e^{-x}} = \frac{1}{1+0} = 1$$

**Interpretacja wyników:** Wraz ze wzrostem wartości $x$, wartość $e^{-x}$ bardzo szybko maleje, aż w pewnym momencie (dla naszych wykresów około $x = 37$) staje się mniejsza od epsilona maszynowego i po dodaniu do niej 1 dostajemy po prostu 1, co powoduje, że wartość logarytmu zostaje obliczona jako 0, czyli też wartość całej funkcji. Wcześniej widoczna oscylacja wartości jest spowodowana utratą precyzji przy zaokrąglaniu $1 + e^{-x}$.

**Wnioski:** Programy do wizualizacji, mimo bycia bardzo wygodnymi narzędziami również korzystają z arytmetyki zmiennopozycyjnej i są podatne na jej wady, zatem zwracane przez nie




(a) Wykres wygenerowany używają biblioteki matplotlib z języka Python




(b) Wykres wygenerowany używając narzędzia Desmos Graphing Calculator

Rysunek 1: Wykresy funkcji $f(x)$

wyniki mogą się różnić od prawdy. Powinniśmy zatem weryfikować czy otrzymane wykresy mają sens, w szczególności po zaobserwowaniu dziwnych zachowań, takich jak oscylacja wartości.

# 3 Zadanie 3

**Opis problemu:** Z użyciem języka Julia, używając algorytmów eliminacji Gaussa oraz przemnożenia przez macierz odwrotną, obliczenie rozwiązań układów równań w formie $\mathbf{Ax} = \mathbf{b}$, gdzie $\mathbf{b} = \mathbf{A}(1, 1, \ldots, 1)^T$ oraz

- a) $\mathbf{A}$ jest macierzą Hilberta stopnia $n \geqslant 1$
- b) $\mathbf{A}$ jest losową macierzą stopnia $n \in \{5, 10, 20\}$ ze wskaźnikiem uwarunkowania $c \in \{1, 10, 10^3, 10^7, 10^{12}, 10^{16}$.

Po wyznaczeniu rozwiązań, należy je porównać z rozwiązaniem dokładnym $(1, \ldots, 1)$ oraz wyznaczyć błąd względny.

**Rozwiązanie:** Używamy funkcji z biblioteki `LinearAlgebra`. Dla rozwiązania dokładnego $x$ oraz wyliczonej wartości $\tilde{x}$, błąd względny wyznaczamy za pomocą wzoru

$$\frac{||x - \tilde{x}||}{||x||}$$

gdzie $||x||$ to norma wektora $x$.

**Wyniki:** Przedstawione w poniższych tabelach (Tabela 2 i Tabela 3), gdzie "Gauss"oznacza wynik uzyskany poprzez metodę eliminacji Gaussa, a "Inverse"rozwiązanie uzyskane poprzez pomnożenie przez macierz odwrotną.

Dla losowych macierzy, tylko dla bardzo dużego wskaźnika uwarunkowania $10^16$ otrzymujemy duży błąd względny, w szczególności dla $n = 5$, gdzie wynosi on ponad 25%. Dla mniejszego wskaźnika otrzymane błędy są małe (aczkolwiek jak pokazało zadanie 1 mogą i tak znacząco wpłynąć na dalsze obliczenia).
Dla macierzy Hilberta, poza kilkoma najmniejszymi, otrzymujemy bardzo duże błędy względne, wynoszące nawet ponad 100, co pokazuje bardzo złe uwarunkowanie tych macierzy.
Dla prawie wszystkich przykładów, błędy wyników zwróconych przez obie metody były zbliżone

| $n$ | Gauss | Inverse |
|---|---|---|
| 1 | 0.0 | 0.0 |
| 2 | 5.661048867003676e-16 | 1.4043333874306803e-15 |
| 3 | 8.022593772267726e-15 | 0.0 |
| 4 | 4.137409622430382e-14 | 0.0 |
| 5 | 1.6828426299227195e-12 | 3.3544360584359632e-12 |
| 6 | 2.618913302311624e-10 | 2.0163759404347654e-10 |
| 7 | 1.2606867224171548e-08 | 4.713280397232037e-09 |
| 8 | 6.124089555723088e-08 | 3.07748390309622e-07 |
| 9 | 3.8751634185032475e-06 | 4.541268303176643e-06 |
| 10 | 8.67039023709691e-05 | 0.0002501493411824886 |
| 11 | 0.0001582708158590435 | 0.007618304284315809 |
| 12 | 0.13396208372085344 | 0.258994120804705 |
| 13 | 0.11039701117868264 | 5.331275639426837 |
| 14 | 1.4554087127659643 | 8.71499275104814 |
| 15 | 4.696668350857427 | 7.344641453111494 |
| 16 | 54.15518954564602 | 29.84884207073541 |
| 17 | 13.707236683836307 | 10.516942378369349 |
| 18 | 10.257619124632317 | 24.762070989128866 |
| 19 | 102.15983486270827 | 109.94550732878284 |
| 20 | 108.31777346206205 | 114.34403152557572 |

Tabela 2: Błędy względne wyników dla **A** będącego macierzą Hilberta.

i z powyższych danych nie można zdecydować o tym czy któraś metoda jest lepsza. Znaczącą różnicę można zaobserwować dla kilku macierzy Hilberta, głównie $n = 13, 14, 16, 18$. W szczególności dla $n = 13$, gdzie błąd dla eliminacji Gaussa jest prawie 50 razy mniejszy.

| $n$ | $c$ | Gauss | Inverse |
|---|---|---|---|
| 5 | 1 | 1.2161883888976234e-16 | 1.4043333874306804e-16 |
| 5 | 10 | 2.220446049250313e-16 | 1.9860273225978183e-16 |
| 5 | $10^3$ | 6.976230176211131e-15 | 5.5750443644609555e-15 |
| 5 | $10^7$ | 1.625566119241207e-10 | 1.7510737474648423e-10 |
| 5 | $10^{12}$ | 1.7301870942413447e-05 | 2.2337161712998105e-05 |
| 5 | $10^{16}$ | 0.20126112355425005 | 0.25155764746872633 |
| 10 | 1 | 2.0770370905276122e-16 | 1.5303368297126222e-16 |
| 10 | 10 | 3.6821932062951477e-16 | 2.3022074639253675e-16 |
| 10 | $10^3$ | 4.304459375041018e-15 | 4.992795269684505e-15 |
| 10 | $10^7$ | 2.0223348082855609e-10 | 2.520255604739961e-10 |
| 10 | $10^{12}$ | 4.197356775363785e-06 | 8.683150970527678e-06 |
| 10 | $10^{16}$ | 0.146056540226733 | 0.12950853615688812 |
| 20 | 1 | 3.519599608236525e-16 | 5.628294210973806e-16 |
| 20 | 10 | 4.710277376051325e-16 | 5.081620675959989e-16 |
| 20 | $10^3$ | 5.117634419664772e-15 | 2.0008670855398256e-15 |
| 20 | $10^7$ | 4.81292693683237e-11 | 6.025771333914663e-11 |
| 20 | $10^{12}$ | 1.066153516185104e-05 | 1.2384462554759176e-05 |
| 20 | $10^{16}$ | 0.03263118522141365 | 0.027607544506382215 |

Tabela 3: Błędy względne wyników dla **A** będącego losową macierzą o wskaźniku uwarunkowania $c$.

**Wnioski:** Dla niektórych danych, nawet dość dokładnie reprezentowalnych jako liczba zmiennopozycyjna, niezależnie od algorytmu dostajemy wyniki bardzo dalekie od prawdy. Powinniśmy zwarać uwagę na wskaźnik uwarunkowania macierzy i jeśli jest rzędu $10^{18}$, jak dla macierzy Hilberta 20 stopnia, to otrzymane wyniki będą całkowicie niepoprawne i problem powinnien zostać rozwiązany inną metodą, albo z użyciem znacznie większej precyzji.

# 4 Zadanie 4

**Opis problemu:** Używając języka Julia oraz biblioteki Polynomials obliczenie pierwiastków wielomianu Wilkinsona

$$P(x) = \prod_{j=1}^{20}(x - j)$$

a następnie sprawdzenie wyliczonych wartości $z_k$, $k \in \{1, \ldots, 20\}$ obliczając $|P(z_k)|$, $|p(z_k)|$ oraz $|z_k - k|$, gdzie $P(z_k)$ oznacza bezpośrednie podstawienie wartości do postaci naturalnej wielomianu, a $p(z_k)$ wymnożenie powyższego iloczynu podstawiając $z_k$ zamiast $x$.

**Rozwiązanie:** Używamy funkcji `roots` do wyznaczenia pierwiastków oraz pętli do wyliczenia $p(x)$.

**Wyniki:**

| $k$ | $z_k$ | $\lvert P(z_k)\rvert$ | $\lvert p(z_k)\rvert$ | $\lvert z_k - k\rvert$ |
|---|---|---|---|---|
| 1 | 0.9999999999996989 | 35696.5096479 | 36626.4254824 | 3.01092484278e-13 |
| 2 | 2.0000000000283182 | 176252.600267 | 181303.933673 | 2.83182366445e-11 |
| 3 | 2.9999999995920965 | 279157.696882 | 290172.285889 | 4.07903488764e-10 |
| 4 | 3.9999999837375317 | 3.0271092989e6 | 2.04153729028e6 | 1.62624682609e-8 |
| 5 | 5.000000665769791 | 2.29174737566e7 | 2.0894625007e7 | 6.65769791297e-7 |
| 6 | 5.999989245824773 | 1.29024172842e8 | 1.12504845776e8 | 1.07541752268e-5 |
| 7 | 7.000102002793008 | 4.8051127546e8 | 4.57290864273e8 | 0.000102002793008 |
| 8 | 7.999355829607762 | 1.6379520219e9 | 1.55564593774e9 | 0.000644170392238 |
| 9 | 9.002915294362053 | 4.87707137255e9 | 4.68781617565e9 | 0.00291529436205 |
| 10 | 9.990413042481725 | 1.36386381955e10 | 1.26346018969e10 | 0.00958695751827 |
| 11 | 11.025022932909318 | 3.58563129513e10 | 3.3001284745e10 | 0.0250229329093 |
| 12 | 11.953283253846857 | 7.53333236036e10 | 7.3885256654e10 | 0.0467167461531 |
| 13 | 13.07431403244734 | 1.96059881243e11 | 1.84762150931e11 | 0.0743140324473 |
| 14 | 13.914755591802127 | 3.57513478231e11 | 3.55142775284e11 | 0.0852444081979 |
| 15 | 15.075493799699476 | 8.21627123646e11 | 8.42320155896e11 | 0.0754937996995 |
| 16 | 15.946286716607972 | 1.55149788805e12 | 1.57072873663e12 | 0.053713283392 |
| 17 | 17.025427146237412 | 3.69473591849e12 | 3.31697822389e12 | 0.0254271462374 |
| 18 | 17.99092135271648 | 7.65010901652e12 | 6.34485314179e12 | 0.00907864728352 |
| 19 | 19.00190981829944 | 1.14352737497e13 | 1.22857173667e13 | 0.00190981829944 |
| 20 | 19.999809291236637 | 2.79241063937e13 | 2.31830953527e13 | 0.000190708763363 |

Tabela 4: Otrzymane pierwiastki wielomianu Wilkinsona, wartości w trzech kolumnach po prawej zostały zaokrąglone do 12 cyfr znaczących.

Dla żadnego pierwiastka otrzymana wartość nie jest równa rzeczywistej wartości oraz dla większych pierwiastków różnica jest co raz większa. Dodatkowo wartości wielomianu dla tych pierwiastków znacząco się różnią od 0, bo nawet najmniejsza uzyskana wartość to około 36 tysięcy, aczkolwiek jest to niewielka liczba w porównaniu do współczynników wielomianu, które są na-

wet rzędu $10^{20}$, co uniemożliwia ich dokładne zapisanie jako 64-bitowa liczba zmiennopozycyjna i powoduje niedokładności wynikające z błędów reprezentacji.

## 4.1 Eksperyment Wilkinsona

**Opis problemu:** Powtórzenie eksperymentu Wilkinsona, czyli obliczenie pierwiastków wielomianu $P_1(x) = P(x) - 2^{-23}x^{19}$.

**Rozwiąznie:** Jak w przypadku wielomianu $P(x)$ wyliczamy pierwiastki z użyciem funkcji `roots` z biblioteki Polynomials.

**Wyniki:**

| $k$ | $z_k$ |
|---|---|
| 1 | 0.9999999999998357 + 0.0i |
| 2 | 2.0000000000550373 + 0.0i |
| 3 | 2.99999999660342 + 0.0i |
| 4 | 4.000000089724362 + 0.0i |
| 5 | 4.99999857388791 + 0.0i |
| 6 | 6.000020476673031 + 0.0i |
| 7 | 6.99960207042242 + 0.0i |
| 8 | 8.007772029099446 + 0.0i |
| 9 | 8.915816367932559 + 0.0i |
| 10 | 10.095455630535774 - 0.6449328236240688i |
| 11 | 10.095455630535774 + 0.6449328236240688i |
| 12 | 11.793890586174369 - 1.6524771364075785i |
| 13 | 11.793890586174369 + 1.6524771364075785i |
| 14 | 13.992406684487216 - 2.5188244257108443i |
| 15 | 13.992406684487216 + 2.5188244257108443i |
| 16 | 16.73074487979267 - 2.812624896721978i |
| 17 | 16.73074487979267 + 2.812624896721978i |
| 18 | 19.5024423688181 - 1.940331978642903i |
| 19 | 19.5024423688181 + 1.940331978642903i |
| 20 | 20.84691021519479 + 0.0i |

Tabela 5: Wyliczone pierwiastki wielomianu $P_1(x)$.

Po niewielkiej zmianie oryginalnego wielomianu mającego wszystkie pierwiastki rzeczywiste, otrzymaliśmy wielomian, którego połowa pierwiastkóœ jest zespolona (o niezerowej części urojonej). Dodatkowo ich część urojona jest względnie spora, bo jej wartość 5-20% wartości rzeczywistej.

## 4.2 Wnioski

Oba te wielomiany pokazują, że nawet dla bardzo malej zmiany w wartościach początkowych, może nastąpić wyraźna zmiana w wyniku. Dodatkowo widzimy, że nawet dla wielomianu Wilkinsona, którego ani stopień ani pierwiastki nie są bardzo duże lub bardzo male, nie jesteśmy go w stanie dokładnie reprezentować i ciężko wykonywać na nim dokładne obliczenia, co sugeruje że wykorzystując tylko 64-bitową arytmetykę zmiennopozycyjną może być trudno pracować z wielomianami stopnia wyższego niż 10 czy może nawet 5.

# 5 Zadanie 5

**Opis problemu:** Rozważamy równanie rekurencyjne

$$p_{n+1} = p_n + rp_n(1 - p_n)$$

gdzie $p_0 = 0.01$ i $r = 3$.

1. Używając `Float32` w języku Julia obliczamy $p_{40}$ z powyższego równania i otrzymany wynik porównujemy wynik z wynikiem otrzymanym poprzez zaokrąglenie $p_{10}$ w dół do 3 cyfr po przecinku i wykonanie kolejnych 30 iteracji z nową wartością.

2. W języku Julia obliczamy $p_{40}$ wykorzystując arytmetki `Float32` i `Float64`.

**Rozwiązanie:** Tworzymy funkcję, która dla danego $p$ i $r$ ewaluuje ww. równanie rekurencyjne oraz iteracyjnie obliczamy $p_{40}$. Dla zmodyfikowanej wersji dodajemy sprawdzenie czy wykonaliśmy właśnie 10 iterację, po której zaokrąglamy aktualną wartość $p$.

**Wyniki:**
W tabeli 6 przedstawione zostały wartości kolejnych iteracji dla wszystkich wariantów. `Float32` i `Float64` oznaczają obliczenia wykonywane w danej arytmetyce, a `Float32`* oznacza wariant z zaokrągleniem po 10 iteracji.
Widzimy, że do 11 iteracji wartości we wszystkich wariantach są bardzo zbliżone, ale do kolejnego wartości w wariancie z zaokrągleniem zaczynają się wyraźnie różnić. Od 19 iteracji widoczna staje się różnica pomiędzy wariantem 32, a 64 bitowym, a po 30 iteracji w każdej kolumine jest całkowicie inna wartość.

**Wnioski:** Podczas liczenia równań rekurencyjnych, które zawierają wyraz większy niż liniowy (jak tutaj składnik kwadratowy) błędy reprezentacji bardzo szybko zwiększają swój wpływ na otrzymane wyniki, zatem obliczenia tego typu powinniśmy wykonywać co najmniej w precyzji 64-bitowej.

# 6 Zadanie 6

**Opis problemu:** Używając arytmetyki `Float64` w języku Julia obliczyć 40 iteracji wyrażenia

$$x_{n+1} = x_n^2 + c$$

dla następujących 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. $c = -1$, $x_0 = 0.25$

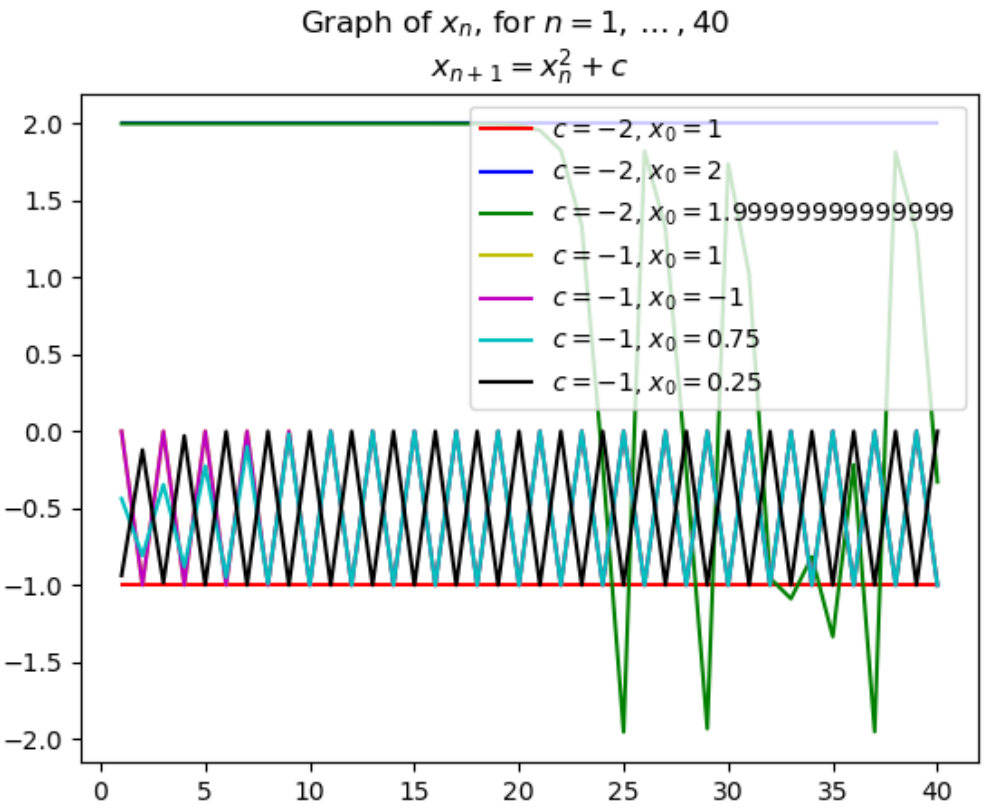
**Rozwiązanie:** Iteracyjnie wyznaczamy kolejne wartości.

**Wyniki:**

| n | Float32 | Float32* | Float64 |
|---|---|---|---|
| 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 | 0.722 | 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 | 0.09289217 | 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 |

Tabela 6: Uzyskane wyniki dla wszystkich 3 wariantów

```
c=-2 x0=1.0 x40=-1.0
c=-2 x0=2.0 x40=2.0
c=-2 x0=1.99999999999999 x40=-0.3289791230026702
c=-1 x0=1.0 x40=-1.0
c=-1 x0=-1.0 x40=-1.0
c=-1 x0=0.75 x40=-1.0
c=-1 x0=0.25 x40=0.0
```




Rysunek 2: Wykres wartości wszystkich ciągów

Na rysunku 2 zostało przedstawione jak zmieniają się wartości ciągi dla poszeczególnych wartości startowych. Widzimy, że przypadek 1 oraz 2 to punkty stałe tego ciągu, pierwszy równy -1, a drugi 2. Wartości w przypadku 3 po około 23 iteracjach zaczynają co iterację zmienieć znak i ich wartości się powoli zmiejszają. Dla pozostałych przypadków, Po co najwyżej kilkunastu iteracjach zaczynają oscylować między 0 i -1, dla przypadku 6 widzimy, że od 10 iteracji, jego wartości całkowicie się pokrywają z tymi dla przypadku 5. Ponieważ $(-1)^2 = 1^2$, to niezależnie które z nich wybierzemy jako wyraz startowy, wszystkie pozostałe wartości będą takie same (o ile $c$ jest takie samo), więc wykresy 4 i 5 przypadku całkowicie się pokrywają.

**Wnioski:** Dla powyższego równania rekurencyjnego (które również jest używane do wyznaczenie zbioru Mandelbrota) widzimy, że pracując z liczbami o skończonej precyzji, jeśli ich wartość nie rozbiega do nieskończoności, to zbiega do któregoś punktu stałego, albo oscyluje między kilkoma wartościami. Mimo, że matematycznie przypadki 5 i 6 mają różne wartości, to od pewnego momentu te różnice nie są możliwe do przedstawienia w skończonej precyzji. Pokazuje to też, że dla odpowiednio dobranych wartości początkowych równanie rekurencyjne, które zdaje się być "rosnące", może mieć wszystkie wartości skończone niezależnie od liczby wykonanych iteracji.