Obliczenia naukowe. Sprawozdanie z listy 2.

Maciej Bazela 261743

11 listopad 2022

Spis treści

1 Zadanie 1

1.1 Opis problemu

Zadanie 1. polegało na powtórzeniu zadania 5. z listy 1, ze zmodyfikowanymi danymi.
Z wektora x należało usunąć ostatnie cyfry z dwóch elementów.

1.2 Rozwiązanie

Kod źródłowy rozwiązań do tego zadania znajduje się w pliku zad1.jl.

Rozwiązanie niczym nie odbiega od rozwiązania zadania 5. z listy 1. Zmieniony wektor oznaczyłem w opisach tabelek jako $\tilde{x}$ (w kodzie jako mx).

1.3 Wyniki

Algorytm sumowania	**Float32 (zad 5. lista 1.)**	**Float32 (zmienione dane)**	**Różnica (stare - zmienione)**
W przód	-0.4999443	-0.4999443	0.0
W tył	-0.4543457	-0.4543457	0.0
Od najw. do najm.	-0.5	-0.5	0.0
Od najm. do najw.	-0.5	-0.5	0.0

Tabela 1: Porównanie wyników iloczynów skalarnych x i y oraz $\tilde{x}$ i y dla *Float32* i różnych algorytmów sumowania.

Algorytm sumowania	**Float64 (zad 5. lista 1.)**	**Float64 (zmienione dane)**	**Różnica (stare - zmienione)**
W przód	$1.0251881368296672e-10$	-0.004296342739891585	0.004296342842410399
W tył	$-1.5643308870494366e-10$	-0.004296342998713953	0.004296342842280865
Od najw. do najm.	0.0	-0.004296342842280865	0.004296342842280865
Od najm. do najw.	0.0	-0.004296342842280865	0.004296342842280865

Tabela 2: Porównanie wyników iloczynów skalarnych x i y oraz $\tilde{x}$ i y dla *Float64* i różnych algorytmów sumowania.

1.4 Wnioski

Dla arytmetyki *Float32* nie zaobserwowałem żadnej zmiany. Zaburzenie x_4 i x_5 jest na tyle małe, że przez zaokrąglenie mantysy we *Float32* otrzymujemy to samo, co przed zaburzeniem.
Jednak dla arytmetyki *Float64* wystąpiła bardzo duża różnica pomiędzy wynikami z zaburzonymi danymi, a wynikami z zad 5. listy 1.
Zaburzenie danych w tym zadaniu było rzędu 10^{-10}, a różnica w wynikach jest rzędu 10^{-3}, co oznacza, że zadanie jest źle uwarunkowane (wskaźnik uwarunkowania jest wysoki). Niewielka zmiana argumentów powoduje znaczne odchylenie wyników końcowych.

2 Zadanie 2

2.1 Opis problemu

Zadanie 2. polegało na narysowaniu wykresu funkcji

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

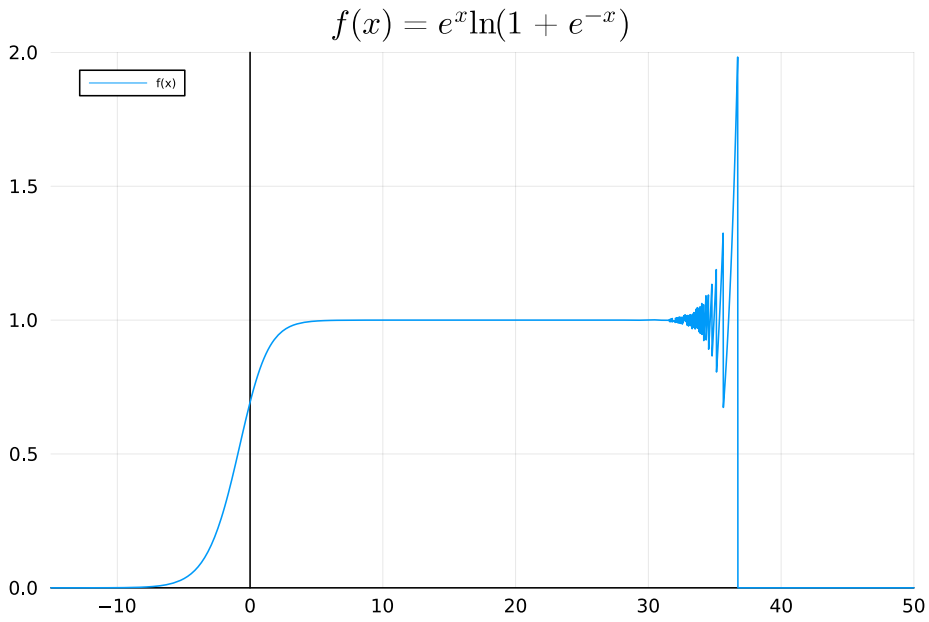
w co najmniej dwóch dowolnych programach do wizualzacji. Narysowane wykresy należało porównać z granicą funkcji $\lim_{x \to \infty} f(x)$.

2.2 Rozwiązanie

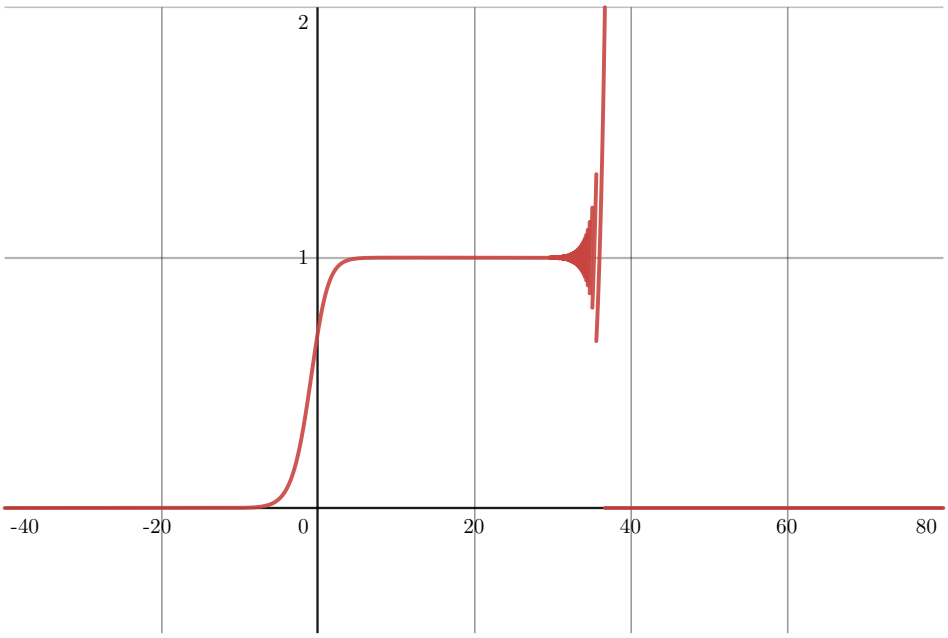
Do wizualzacji wykresu funkcji f wykorzystałem bibliotekę **Plots.jl** oraz aplikację **Desmos**.

2.3 Wyniki

Granica podanej funkcji to: $\lim_{x\to\infty} e^x \ln(1+e^{-x}) = 1$



Rysunek 1: Wykres funkcji $e^x \ln(1+e^{-x})$ przy użyciu biblioteki **Plots.jl**.



Rysunek 2: Wykres funkcji $e^x \ln(1+e^{-x})$ przy użyciu aplikacji **Desmos**.

2.4 Wnioski

Dla $x > 30$ można zaobserwować dużą fluktuację wartości $f(x)$. Od pewnego momentu wartości $f(x)$ spadają do 0 i takie pozostają. Powodem takiego stanu rzeczy jest obliczanie wartości $\ln(1+e^{-x})$. Dla $30 < x < 40$ wartość $\ln(1+e^{-x})$ zbliża się do wartości *macheps*. Dla $x = 36$ wynosi dokładnie *macheps*, a dla $x > 37$: $\ln(1+e^{-37}) = Float64(0)$.

3 Zadanie 3

3.1 Opis problemu

W zadaniu 3. należało porównać metody rozwiązywania układu równań liniowych (metoda eliminacji Gaussa, odwrotności macierzy) zadanych w poniższy sposób:

$$\boldsymbol{Ax} = \boldsymbol{b}$$
$$\boldsymbol{A} \in \mathbb{R}^{n \times n}$$
$$\boldsymbol{b} \in \mathbb{R}^{n}$$

gdzie $\boldsymbol{b}$ to wektor prawych stron, a $\boldsymbol{A}$ to macierz generowana przez kod źródłowy podany przez prowadzącego kursu (macierz Hilberta lub losowa). Za $\boldsymbol{x}$ mieliśmy przyjąć wektor jedynek ($\boldsymbol{x} = (1, \ldots, 1)$).

3.2 Rozwiązanie

Kod źródłowy rozwiązań do tego zadania znajduje się w pliku zad3.jl.

Do obliczania niektórych z właśności macierzy (rząd, wskaźnik uwarunkowania, norma) wykorzystałem funkcje zaimplementowane w bibliotece *LinearAlgebra*. Błąd względny metody Gaussa i macierzy odwrotnej wyliczyłem ze wzoru:

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

Dane do przeprowadzenia eksperymentów są podane w treści zadania na liście.

3.3 Wyniki

n	Rząd	Wskaźnik uwarunkowania	Błąd (eliminacja Gaussa)	Błąd (odwrotność macierzy)
1	1	1.0	0.0	0.0
3	3	524.0567775860644	$8.022593772267726e-15$	0.0
5	5	476607.2502425855	$1.6828426299227195e-12$	$3.3544360584359632e-12$
7	7	$4.753673567446793e8$	$1.2606867224171548e-8$	$4.713280397232037e-9$
9	9	$4.9315375594102344e11$	$3.8751634185032475e-6$	$4.541268303176643e-6$
11	10	$5.222701316549833e14$	0.00015827808158590435	0.007618304284315809
13	11	$3.1883950689209334e18$	0.11039701117868264	5.331275639426837
15	12	$3.67568286586649e17$	4.696668350857427	7.344641453111494
17	12	$1.249010044779401e18$	13.707236683836307	10.516942378369349
19	13	$6.472700911391398e18$	102.15983486270827	109.94550732878284
21	13	$3.2902428208431683e18$	44.089455838364245	34.52041154914292
23	13	$6.313779002744782e17$	25.842511917947366	22.272314298730727
25	13	$1.3309197502927598e18$	7.095757204652332	21.04404299195525
27	14	$4.414670357556845e18$	27.43309009053957	35.68974530952139
29	14	$8.060274133463556e18$	60.095450394724104	43.40383683199056
31	14	$2.4040817319121502e19$	21.45662601984968	23.74575780277118
33	14	$1.1705237268888885e19$	37.556822732776205	32.889697413799794
35	14	$2.5524196131500777e19$	31.166902974731222	36.723963451169304
37	15	$5.871718859396612e18$	13.974714130452178	16.39248770656996
39	15	$9.058525451393528e18$	118.20336501589891	263.5309838641091

Tabela 3: Wyliczone wartości wskaźniku uwarunkowania i błędów dla macierzy Hilberta.

n	Rząd	Wskaźnik uwarunkowania	Błąd (eliminacja Gaussa)	Błąd (odwrotność macierzy)
5	5	1.0	$1.9860273225978183e-16$	$1.2161883888976234e-16$
5	5	10.000000000000012	$1.2161883888976234e-16$	$2.2752801345137457e-16$
5	5	999.9999999999217	$3.273106189992482e-14$	$3.2452257702549463e-14$
5	5	$1.0000000009428933e7$	$7.63304645474209e-11$	$6.104870979733251e-11$
5	5	$9.999486794370239e11$	$2.777261821195042e-5$	$2.5072748909233066e-5$
5	4	$1.2592472135701236e16$	0.12170732606427752	0.08186366810893583
10	10	1.0000000000000016	$3.7155169009665004e-16$	$3.367473164340295e-16$
10	10	9.999999999999993	$3.080743865682491e-16$	$3.5631068175681536e-16$
10	10	1000.0000000000343	$3.719588468984917e-14$	$3.959423287482303e-14$
10	10	$9.999999992497459e6$	$1.7107394055125457e-10$	$1.738938648054569e-10$
10	10	$1.0000651356427539e12$	$3.949353995834508e-5$	$4.353662960739791e-5$
10	9	$1.1278884217635066e16$	0.12976134922183502	0.03314563036811941
20	20	1.0000000000000009	$5.573275351449751e-16$	$5.289603976864219e-16$
20	20	9.999999999999998	$3.1401849173675503e-16$	$3.050589040050974e-16$
20	20	1000.0000000000023	$2.24648903608456e-14$	$2.3636228402127114e-14$
20	20	$1.0000000006490897e7$	$2.2205650930536758e-12$	$9.334801253844997e-11$
20	20	$1.0000483258516975e12$	$8.048149779925011e-6$	$1.068849396437365e-5$
20	19	$2.0346925746691836e16$	0.11031450683130099	0.11087834029988115

Tabela 4: Wyliczone wartości wskaźniku uwarunkowania i błędów dla losowej macierzy.

3.4 Wnioski

Macierz Hilberta jest wyjątkowo źle uwarunkowana. Wraz ze wzrostem wielkości macierzy wskaźnik uwarunkowania bardzo szybko się powiększa.
Wskaźnik uwarunkowania był jednym z argumentów funkcji generującej macierz losową. Jak widać, jego wzrost mocno wpływa na powiększanie się błędów.
Z tych obserwacji można śmiało wywnioskować, że wskaźnik uwarunkowania macierzy ma bardzo duży wpływ na błąd względny w obu metodach wyliczających rozwiązanie układu liniowego.

Wielkość macierzy nie ma dużego wpływu na błędy względne obu metod. Różnica pomiędzy błędami obu metod dla losowej macierzy 20×20 i wskaźniku uwarunkowania 1.0, a losową macierzą 5×5 i wskaźnikiem 1.0 jest rzędu 10^{-16}

4 Zadanie 4

4.1 Opis problemu

W zadaniu 4. mieliśmy skorzystać z pakietu *Polynomials* i wyznaczyć pierwiastki wielomanu Wilkinsona, obliczyć wartości tego wielomianu (oraz wartości jego postaci iloczynowej) dla wyliczonych pierwiastków oraz wyznaczyć różnicę $|z_k - k|$, gdzie z_k to k-ty pierwiastek wielomianu.
Dodatkowo mieliśmy sprawdzić co się stanie, gdy lekko zaburzymy jeden ze współczynników wielomianu $(-210x^{19} \rightarrow (-210 - 2^{-23})x^{-19})$.

4.2 Rozwiązanie

Kod źródłowy rozwiązań do tego zadania znajduje się w pliku zad4.jl.

W podanym pliku źródłowym znajduje się dokładna implementacja tego zadania.
Warto też zaznaczyć, że przez zaburzenie współczynnika przy x^{19} pierwiastki wielomianu będą liczbami zespolonymi. Dlatego w poniższych wynikach znajduje się część urojona $(\pm im)$.

4.3 Wyniki

Przez p oznaczamy wielomian Wilkinsona w postaci iloczynowej $(x^{20} - 20)(x^{19} - 19) \ldots (x^1 - 1)$, a przez P wielomian Wilkinsona w postaci ogólnej $(a_{20}x^{20} + a_{19}x^{19} + \ldots + a_1x^1)$.

k	Z_k	$\lvert P(Z_k)\rvert$	$\lvert p(Z_k)\rvert$	$\lvert Z_k - k\rvert$
1	0.9999999999996989	35696.50964788257	36626.4254824228	$3.0109248427834245e-13$
2	2.0000000000283182	176252.60026668405	181303.9336725767	$2.8318236644508943e-11$
3	2.9999999995920965	279157.6968824087	290172.2858891687	$4.0790348876384996e-10$
4	3.9999999837375317	$3.0271092988991085e6$	$2.04153729027509e6$	$1.626246826091915e-8$
5	5.000000665769791	$2.2917473756567076e7$	$2.0894625006962176e7$	$6.657697912970661e-7$
6	5.999989245824773	$1.2902417284205095e8$	$1.1250484577562997e8$	$1.0754175226779239e-5$
7	7.000102002793008	$4.805112754602064e8$	$4.5729086427309465e8$	0.00010200279300764947
8	7.999355829607762	$1.6379520218961136e9$	$1.5556459377357383e9$	0.0006441703922384079
9	9.002915294362053	$4.877071372550003e9$	$4.68781617564839e9$	0.002915294362052734
10	9.990413042481725	$1.3638638195458128e10$	$1.2634601896949207e10$	0.009586957518274986
11	11.025022932909318	$3.585631295130865e10$	$3.3001284744984142e10$	0.025022932909317674
12	11.953283253846857	$7.533332360358197e10$	$7.388525665404987e10$	0.04671674615314281
13	13.07431403244734	$1.9605988124330817e11$	$1.84762150931442e11$	0.07431403244734014
14	13.914755591802127	$3.5751347823104315e11$	$3.551427752842085e11$	0.08524440819787316
15	15.075493799699476	$8.21627123645597e11$	$8.423201558964255e11$	0.07549379969947623
16	15.946286716607972	$1.5514978880494067e12$	$1.5707287366258018e12$	0.05371328339202819
17	17.025427146237412	$3.694735918486229e12$	$3.3169782238892354e12$	0.025427146237412046
18	17.99092135271648	$7.650109016515867e12$	$6.344853141791281e12$	0.009078647283519814
19	19.00190981829944	$1.1435273749721195e13$	$1.2285717366719662e13$	0.0019098182994383706
20	19.999809291236637	$2.7924106393680727e13$	$2.318309535271639e13$	0.00019070876336257925

Tabela 5: Wyliczone wartości wielomianu Wilkinsona dla k-tego pierwiastka Z_k.

k	Z_k	$\lvert Z_k - k\rvert$
1	$0.9999999999998357 + 0.0im$	$1.6431300764452317e-13$
2	$2.0000000000550373 + 0.0im$	$5.503730804434781e-11$
3	$2.99999999660342 + 0.0im$	$3.3965799062229962e-9$
4	$4.000000089724362 + 0.0im$	$8.972436216225788e-8$
5	$4.99999857388791 + 0.0im$	$1.4261120897529622e-6$
6	$6.000020476673031 + 0.0im$	$2.0476673030955794e-5$
7	$6.99960207042242 + 0.0im$	0.00039792957757978087
8	$8.007772029099446 + 0.0im$	0.007772029099445632
9	$8.915816367932559 + 0.0im$	0.0841836320674414
10	$10.095455630535774 - 0.6449328236240688im$	0.6519586830380407
11	$10.095455630535774 + 0.6449328236240688im$	1.1109180272716561
12	$11.793890586174369 - 1.6524771364075785im$	1.665281290598479
13	$11.793890586174369 + 1.6524771364075785im$	2.0458202766784277
14	$13.992406684487216 - 2.5188244257108443im$	2.518835871190904
15	$13.992406684487216 + 2.5188244257108443im$	2.7128805312847097
16	$16.73074487979267 - 2.812624896721978im$	2.9060018735375106
17	$16.73074487979267 + 2.812624896721978im$	2.825483521349608
18	$19.5024423688181 - 1.940331978642903im$	2.4540214463129764
19	$19.5024423688181 + 1.940331978642903im$	2.0043294443099486
20	$20.84691021519479 + 0.0im$	0.8469102151947894

Tabela 6: Pierwiastki wielomianu Wilkinsona z zaburzonym współczynnikiem x^{19}.

$\lvert P(Z_k)\rvert$	$\lvert p(Z_k)\rvert$
20259.872313418207	19987.872313406842
346541.4137593836	352369.413808796
2.2580597001197007e6	2.416241558251844e6
1.0542631790395478e7	1.126370230029202e7
3.757830916585153e7	4.475744423806907e7
1.3140943325569446e8	2.1421031658039317e8
3.939355874647618e8	1.7846173427860644e9
1.184986961371896e9	1.868697217000985e10
2.2255221233077707e9	1.3746309775142996e11
1.0677921232930157e10	1.4900695352000583e12
1.0677921232930157e10	1.4900695352000583e12
3.1401962344429485e10	3.2962792355717137e13
3.1401962344429485e10	3.2962792355717137e13
2.157665405951858e11	9.546022365750218e14
2.157665405951858e11	9.546022365750218e14
4.850110893921027e11	2.742106076928478e16
4.850110893921027e11	2.742106076928478e16
4.557199223869993e12	4.25248587652037e17
4.557199223869993e12	4.25248587652037e17
8.756386551865696e12	1.3743743559997601e18

Tabela 7: Wyliczone wartości wielomianu Wilkinsona z zaburzonym współczynnikiem x^{19} dla k-tego pierwiastka Z_k.

4.4 Wnioski

Nawet małe niedokładności w wyliczeniu pierwiastków wielomianu Wilkinsona za pomocą funkcji *roots* z pakietu *Polynomials* sprawiły, że wyliczając wartości wielomianu dla tych pierwiastków dostaliśmy wyniki mocno odbiegające od zera. Już dla 4-tego z kolei pierwiastka różnica względem zera jest rzędu 10^6! Oznacza to, że wskaźnik uwarunkowania dla wielomianu Wilkinsona jest duży.
Po zaburzeniu współczynniku przy x^{19}, pierwiastki wielomianu weszły w liczby zespolone, a wyliczone wartości odbiegły od wartości przed zaburzeniem.

Współczynniki tego wielomianu są bardzo duże, dlatego nie wystarcza liczb znaczących do reprezentacji wyniku wyliczania wartości wielomianu w systemie dziesiętnym.

5 Zadanie 5

5.1 Opis problemu

Zadanie 5. polegało na implementacji rekurencyjnego modelu wzrostu populacji (modelu logistycznego) zadanego wzorem:

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

oraz przeprowadzeniu prostych eksperymentów.

5.2 Rozwiązanie

Kod źródłowy rozwiązań do tego zadania znajduje się w pliku zad5.jl.

Podany program definiuje zadaną funkcję i drukuje wyniki eksperymentów do konsoli.
Trochę źle zinterpretowałem treść zadania, ponieważ myślałem, że eksperyment z obicnaniem do 3 cyfr znaczących mamy wykonać *co* 10. *iteracji*, więc dodatkowo jako 5. i 6. wynik podałem wartości takiego eksperymentu dla *Float32* i *Float64*. Co więcej, powtórzyłem eksperyment 3. dla arytmetyki *Float64*.

5.3 Wyniki

Wyniki zadanych eksperymentów:

1. $p_0 = 0.01,\ r = 3,\ 40$ iteracji w arytmetyce *Float32*: **0.25860548**
2. $p_0 = 0.01,\ r = 3,\ 40$ iteracji w arytmetyce *Float64*: **0.011611238029748606**
3. $p_0 = 0.01,\ r = 3,\ 40$ iteracji w arytmetyce *Float32* z obcinaniem do 3 cyfr znaczących po 10. iteracji: **1.093568**
4. (*) $p_0 = 0.01,\ r = 3,\ 40$ iteracji w arytmetyce *Float64* z obcinaniem do 3 cyfr znaczących po 10. iteracji: **0.7305550338104317**
5. (*) $p_0 = 0.01,\ r = 3,\ 40$ iteracji w arytmetyce *Float32* z obcinaniem do 3 cyfr znaczących co 10. iteracji: **0.71587336**
6. (*) $p_0 = 0.01,\ r = 3,\ 40$ iteracji w arytmetyce *Float64* z obcinaniem do 3 cyfr znaczących co 10. iteracji: **0.7159416017707503**

(*) - eksperymenty dodatkowe

5.4 Wnioski

Precyzja obliczeń oraz zaburzanie aktualnej wartości p_n mocno wpływa na ostateczny wynik po 40 iteracjach. Po kilkunastu krokach iteracji brakuje liczb znaczących do reprezentacji kolejnych stanów rekurencji. Oczywiście dla arytmetyki *double* obliczenia są dokładniejsze, ale podana rekurencja jest na tyle źle uwarunkowana, że nawet bardzo mała zmiana danych wejściowych daje kompletnie inny wynik po takiej samej liczbie iteracji:

population_growth_model(40 iterations, Float64(3), Float64(0.0100000000001)) = 0.1875235400264368

Co ciekawe, eksperyment z obcinaniem do 3 cyfr znaczących co 10 iteracji dał podobne wyniki dla *Float32* i *Float64*. Pośrednie wyniki też są do siebie zbliżone.

6 Zadanie 6

6.1 Opis problemu

W zadaniu 6. mieliśmy przeprowadzić eksperymenty nad prostym równaniem rekurencyjnym:

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

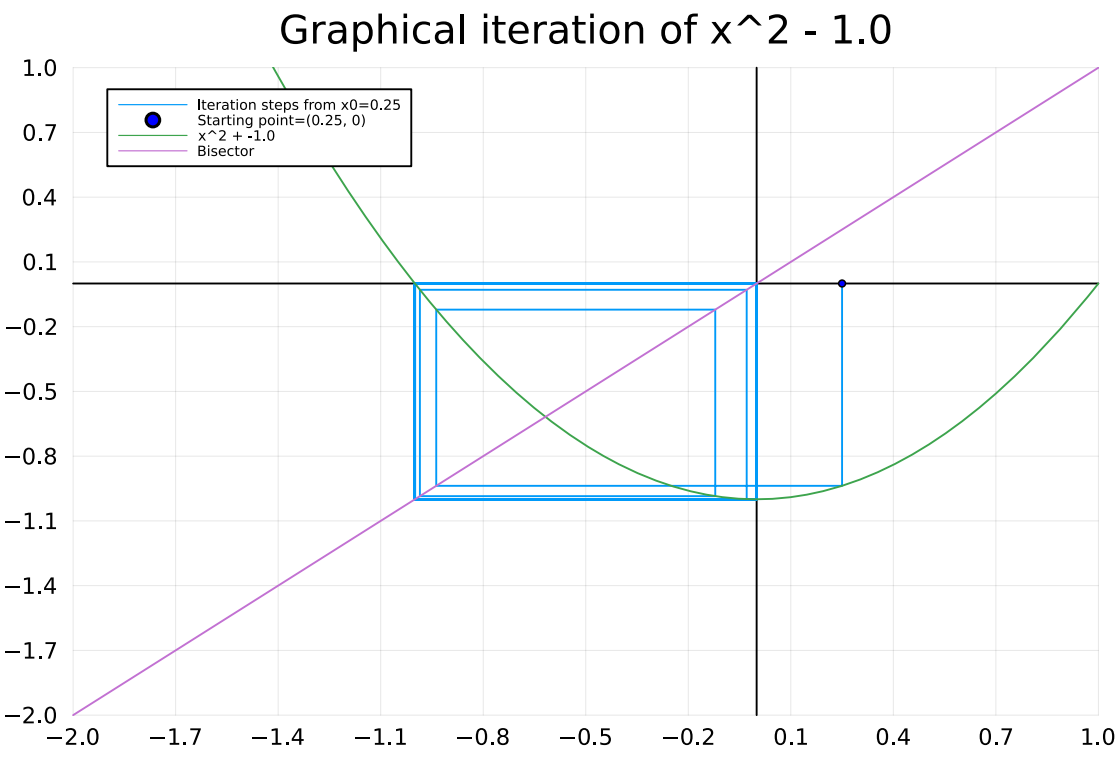
dla podanych stałych c i początkowych wartości x_0. Obliczenia miały zostać przeprowadzone przez 40 iteracji w arytmetyce *Float64*.

6.2 Rozwiązanie

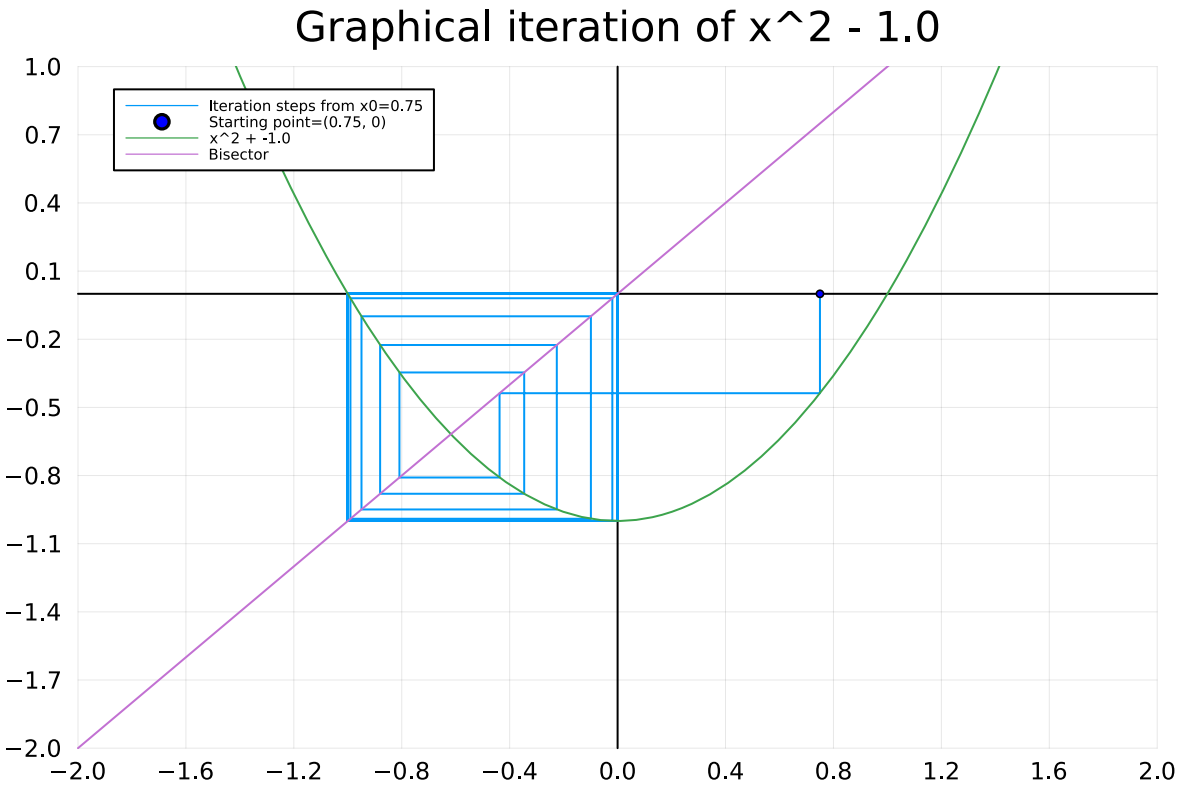
Kod źródłowy rozwiązań do tego zadania znajduje się w pliku zad6.jl.

Kod źródłowy zawiera literalną implementację tego zadania, wraz z drukowaniem do konsoli. Do przeprowadzenia iteracji graficznej zadanej rekurencji skorzystałem z biblioteki *Plots.jl*.

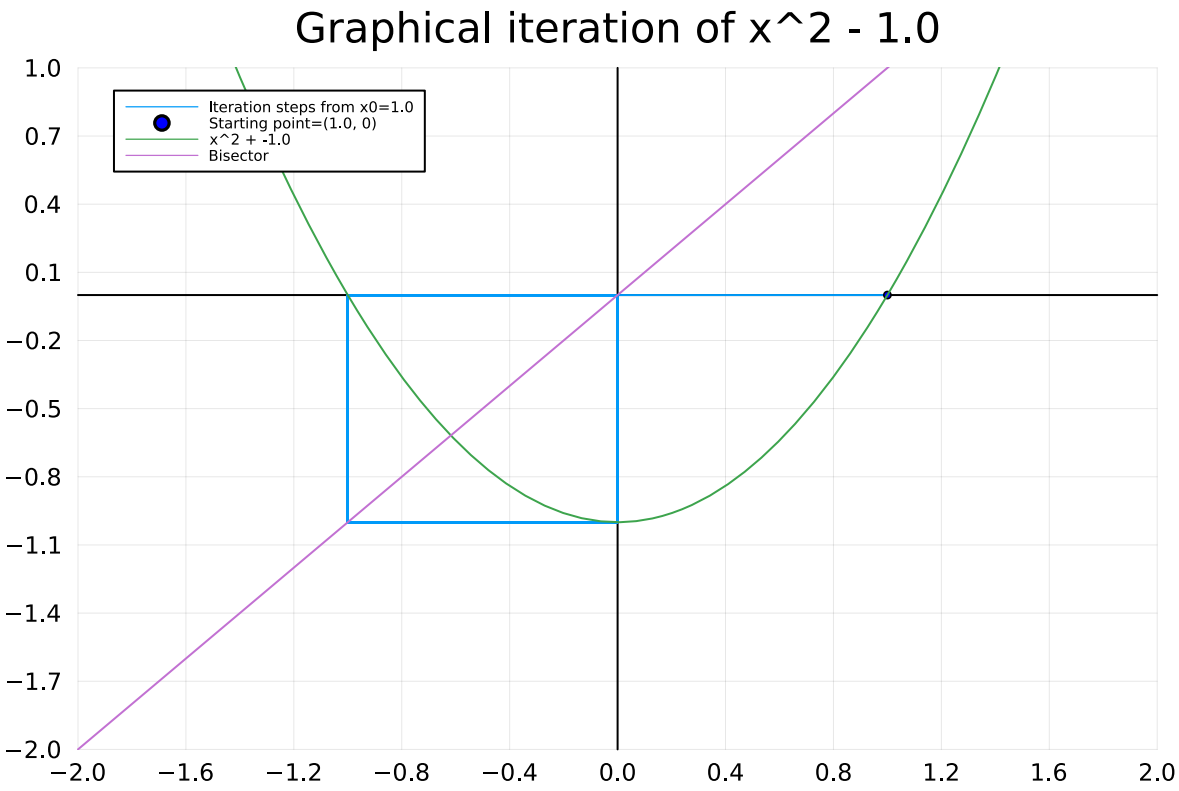
6.3 Wyniki



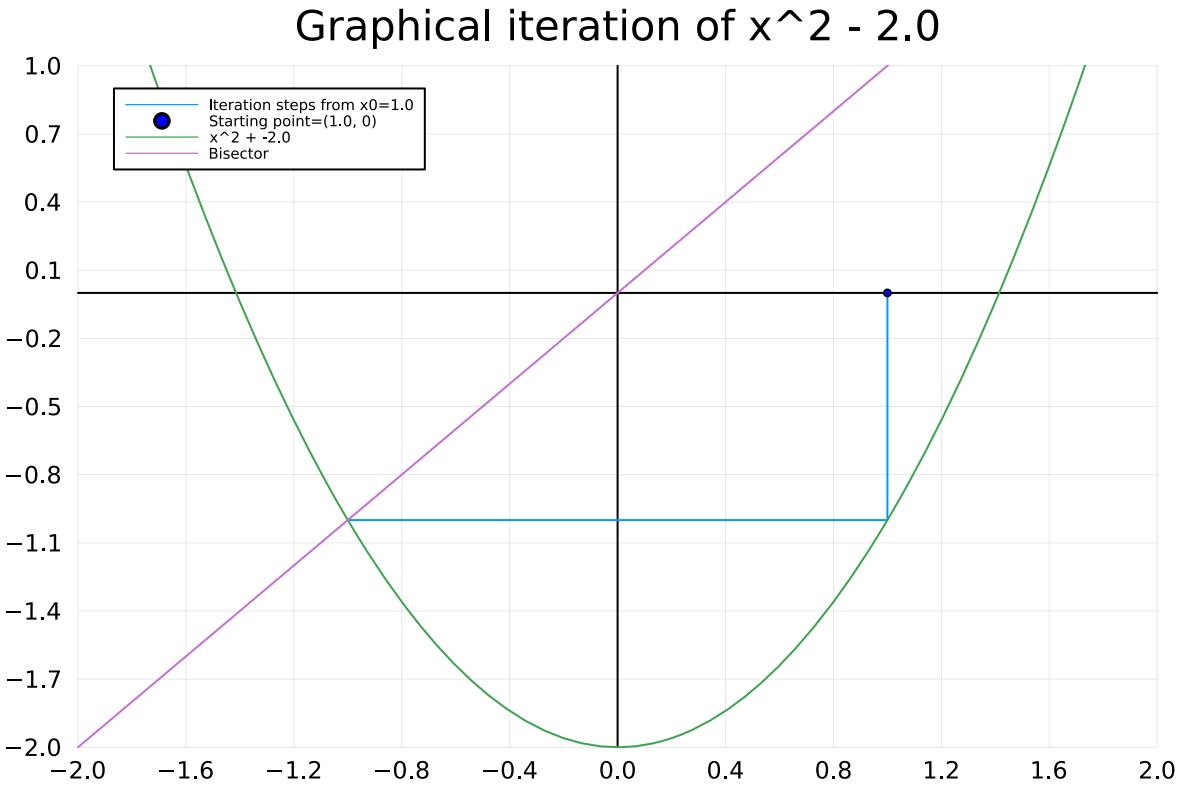
Rysunek 3: Iteracja graficzna $x_{n+1} = x_n^2 + c$ dla $c = -1$, $x_0 = 0.25$.



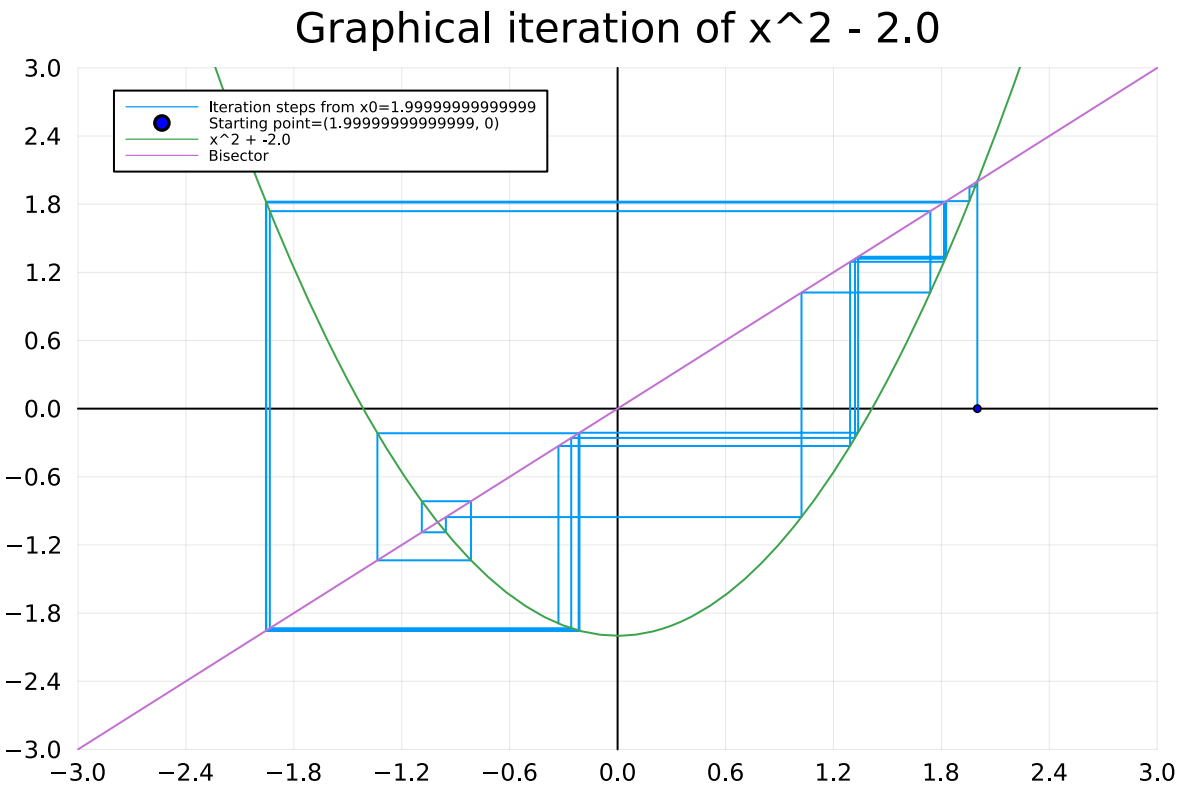
Rysunek 4: Iteracja graficzna $x_{n+1} = x_n^2 + c$ dla $c = -1$, $x_0 = 0.75$.



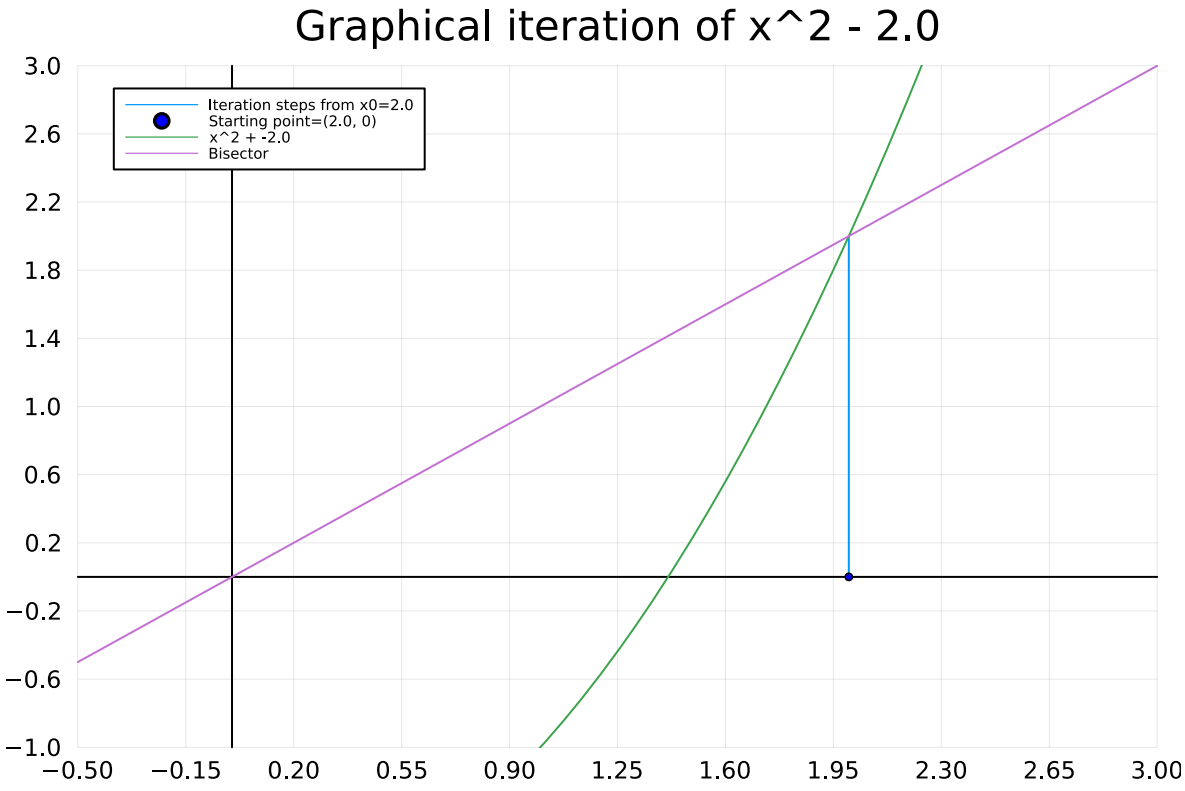
Rysunek 5: Iteracja graficzna $x_{n+1} = x_n^2 + c$ dla $c = -1$, $x_0 = 1.0$.



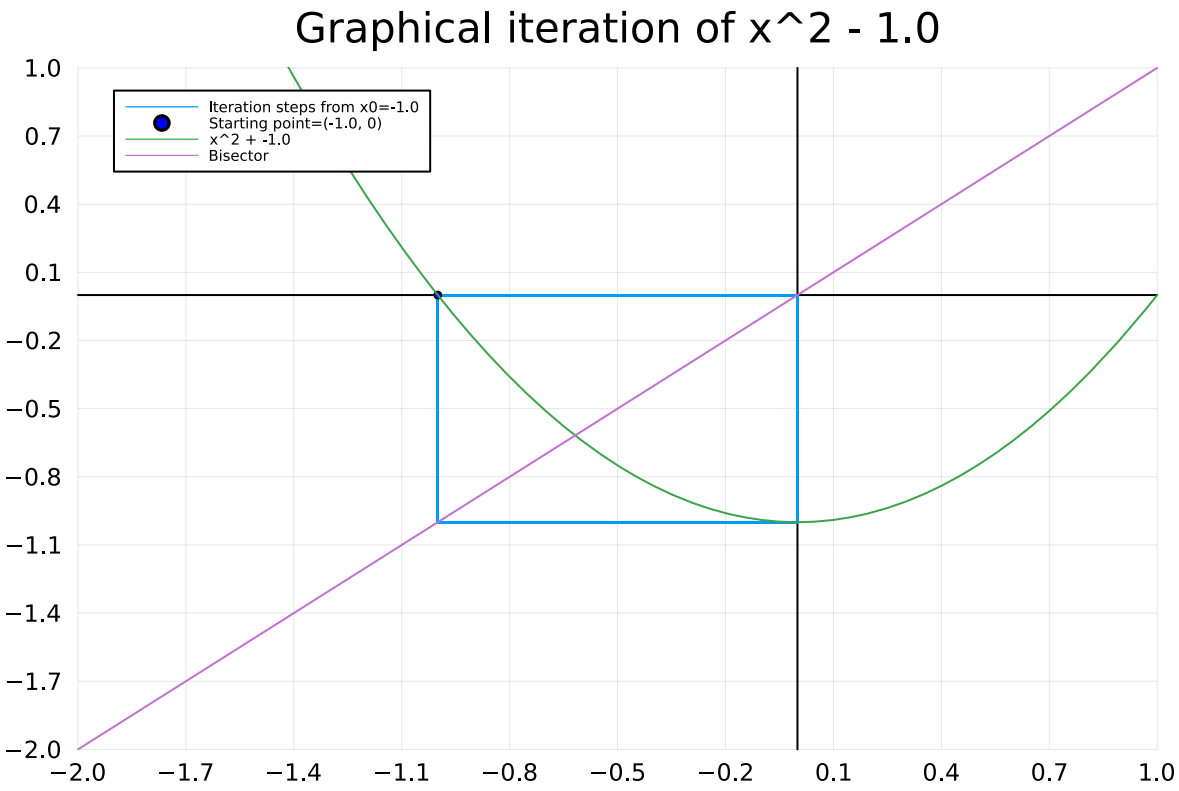
Rysunek 6: Iteracja graficzna $x_{n+1} = x_n^2 + c$ dla $c = -2$, $x_0 = 1.0$.



Rysunek 7: Iteracja graficzna $x_{n+1} = x_n^2 + c$ dla $c = -2$, $x_0 = 1.99999999999999$.



Rysunek 8: Iteracja graficzna $x_{n+1} = x_n^2 + c$ dla $c = -2$, $x_0 = 2$.



Rysunek 9: Iteracja graficzna $x_{n+1} = x_n^2 + c$ dla $c = -1$, $x_0 = -1$.

6.4 Wnioski

Dla $c = -2$ i wartości $x_0 = 1.0$ i $x_0 = 2.0$ funkcja od razu zbiega do wartości całkowitej. Dla $x_0 = 1.0$ jest to -1, a dla $x_0 = 2.0$ jest to 2

Dla $c = -1$ i wartości $x_0 = 1.0$ i $x_0 = -1.0$ wartości x_n przyjmują naprzemiennie -1.0 i 0.0. Łatwo to zauważyć po zobaczeniu samego wzoru rekurencji. Na wykresie iteracji graficznej jest to ładnie zobrazowane poprzez tworzenie się kwadratu o wierzchołkach $(0.0,\ 0.0)$, $(-1.0,\ 0.0)$ $(-1.0,\ -1.0)$, $(0,\ -1.0)$.

Dla $x_0 = 0.25$ i $x_0 = 0.75$ wartości na początku wahają się pomiędzy -1.0 a 0.0, ale po ok. 25 iteracjach także wpadają w naprzemienne zamiany pomiędzy -1.0 i 0.0.
Dzieje się tak, ponieważ po kilkunastu iteracjach błędy w reprezentacji zmiennopozycyjnej kumulują się na tyle, że wartość x_n^2 zbliża się do *macheps*. Kiedy odejmiemy od dostateczniej małej liczby 1.0, wynik zostanie zaokrąglony do -1.0, co potem powoduje zamianę pomiędzy dwoma wartościami.

Najciekawszy wynik zwróciła rekurencja dla $c = -2$ i $x_0 = 1.99999999999999$. Przez kilkanaście pierwszych iteracji wartość jest bliska 2.0, ale powoli nakładają się błędy odejmowania 2.0. Od pewnego momentu błąd jest na tyle duży, że reprezentacja x_n we *Float64* zaczyna znacznie odbiegać od rzeczywistej wartości, dlatego wykres staje się chaotyczny. Rekurancja dla 40 iteracji nie zbiega do żadnej wartości.
Z ciekawości sprawdziłem co się dzieje dla np. 400 iteracji i ku mojemu ździwieniu x_0 wraca czasami do wartości bliskich 2.0, ale potem znowu wpada w chaos.

Oczywistym jest, że podana rekurencja jest źle uwarunkowana.