Sprawozdanie laboratorium lista 2 - obliczenia naukowe

Zofia Tarchalska

1 Zadanie 1

Celem zadania było powtórzenie zadania 5 z listy 1, ale z lekko zmodyfikowanymi danymi. Dokładnie chodziło o to aby usunąć ostatnią 9 z czwartej współrzędnej wektora x i ostatnią 7 z piątej współrzędnej. Wektory te prezentują się następująco:
$x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$
$y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]$

Po zmianie:
$x = [2.718281828, -3.141592654, 1.414213562, 0.577215664, 0.301029995]$

Po lewej: poprzednie uzyskane wyniki. Po prawej: nowe wyniki

```
Float32
real: -1.00657107000000e-11
forward: -0.4999443
backward: -0.4543457
biggest_to_smallest: -0.5
smallest_to_biggest: -0.5

Float64
real: -1.00657107000000e-11
forward: 1.0251881368296672e-10
backward: -1.5643308870494366e-10
biggest_to_smallest: 0.0
smallest_to_biggest: 0.0
```

```
Float32
----------------------------
forward: -0.4999443
backward: -0.4543457
biggest_to_smallest: -0.5
smallest_to_biggest: -0.5

Float64
----------------------------
forward: -0.004296342739891585
backward: -0.004296342998713953
biggest_to_smallest: -0.004296342842280865
smallest_to_biggest: -0.004296342842280865
```

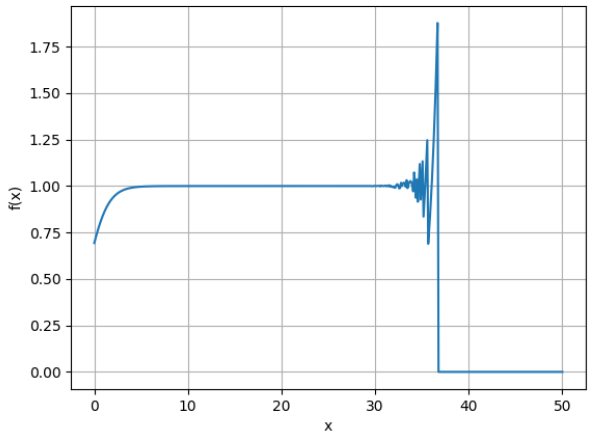
Można zauważyć, że jeśli chodzi o arytmetykę single wyniki w ogóle nie różnią się od tych uzyskanych w poprzedniej próbie. Dzieje się tak, ponieważ usuwane cyfry są na granicy precyzji. Inaczej jest w przypadku double, w tej arytmetyce otrzymujemy różne rezultaty.

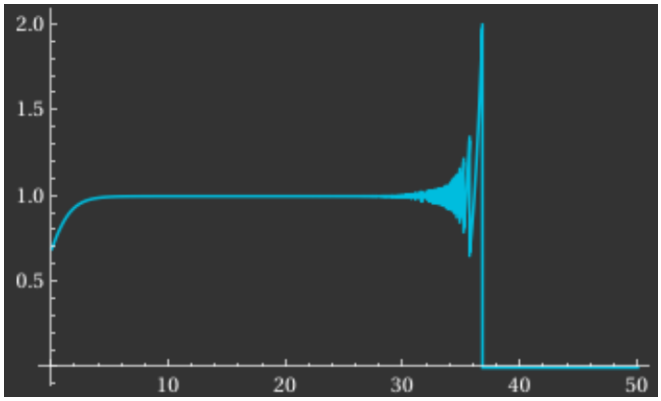
Wnioski

Okazuje się, że w przypadku arytmetyki Float64 niewielka zmiana na 10 miejscu po przecinku powoduje zmianę wyniku o 7 rzędów wielkości. Najpierw wynik był rzędu 10^{-10} żeby następnie wzrosnąć do rzędu wielokości 10^{-3} (dla sposobu forward i backward, ponieważ są to te bardziej precyzyjne sposoby). Widzimy zatem, że zadanie jest źle uwraunkowane (mała zmiana w danych powoduje duże zmiany wyniku).

2 Zadanie 2

W zadaniu należy narysować wykres funkcji $f(x) = e^x ln(1 + e^{-x})$ w co najmniej dwóch różnych programach do wizualizacji. Następnie trzeba policzyć jej granicę i porównać z otrzymanymi wykresami.





Po lewej widnieje wykres wygenerowany za pomocą PyPlot, a po prawej w WolframAlpha.
Teraz ręcznie policzmy granicę funkcji w nieskończoności:

$$\lim_{x\to\infty} e^x \cdot ln(1 + e^{-x}) = \lim_{x\to\infty} \frac{-e^{-x}}{(1 + e^{-x}) \cdot (-e^{-x})} = \lim_{x\to\infty} \frac{1}{1 + e^{-x}} = 1$$

Jak widać rzeczywisty przebieg funkcji różni się od tego co zwracają nam obydwa programy.

Wnioski

Znów zadanie charakteryzuje się silnym uwarunkowaniem numerycznym. Niedokładności wynikające z ograniczonej precyzji, prowadzą do odchyleń wartości funkcji od prawidłowego wyniku, szczególnie w zakresie $x \in [30, 36]$. Dla argumentów > 36 funkcja zbiega do 0, ponieważ zachodzi przybliżenie $1 + e^{-x} \approx 1$ zatem $ln(1 + e^{-x}) \approx 0$. Jest to sprzeczne z rzeczywistym przebiegiem funkcji i wyznaczoną granicą analityczną. Zaburzenie występuje w obydwu programach zewnętrznych, co oznacza złe uwarunkowanie zadania.

3 Zadanie 3

Zadanie polegało na rozwiązaniu układu równań liniowych dwoma różnymi sposobami oraz porównaniu ich pod kątem policzonych błędów względnych.

Mamy równanie liniowe postaci: $Ax = b$, gdzie:

- A to macierz współczynników. Generujemy ją na dwa sposoby:
  - $A = H_n$, gdzie H_n jest macierzą Hilberta stopnia n
  - $A = R_n$, gdzie R_n to losowa macierz stopnia n z podanym wskaźnikiem uwarunkowania c
- b to wektor prawych stron

Układ ten będziemy rozwiązywać dwoma metodami:

- metodą eliminacji Gauss'a - $x = A \backslash b$
- metodą z macierza odwrotną - $x = inv(A) * b$

Nasz dokładny x to $x = (1, ..., 1)^T$. Z jego pomocą będziemy obliczać błąd względny. Poniżej zamieszczone zostają wartości zwracane przez funkcje `rank(A)` i `cond(A)` oraz policzone błędy względne dla obu metod.

`Macierz Hilberta`

n	cond(A)	rank(A)	error Gauss	error inv
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	476607.2502422687	5	1.6828426299227195e-12	3.3544360584359632e-12
6	1.49510586424659e7	6	2.618913302311624e-10	2.0163759404347654e-10
7	4.753673565921816e8	7	1.2606867224171548e-8	4.713280397232037e-9
8	1.5257575538072489e10	8	6.124089555723088e-8	3.07748390309622e-7
9	4.931537556012197e11	9	3.8751634185032475e-6	4.541268303176643e-6
10	1.602441350036382e13	10	8.67039023709691e-5	0.0002501493411824886
11	5.222703245009594e14	10	0.00015827808158590435	0.007618304284315809
12	1.760619121841585e16	11	0.13396208372085344	0.258994120804705
13	3.1905581877988255e18	11	0.11039701117868264	5.331275639426837
14	9.27636978936766e17	11	1.4554087127659643	8.71499275104814
15	3.67568286586649e17	12	4.696668350857427	7.344641453111494
16	7.063115212292111e17	12	54.15518954564602	29.84884207073541
17	8.07124989431416e17	12	13.707236683836307	10.516942378369349
18	1.4135073701749765e18	12	10.257619124632317	24.762070989128866
19	5.190132496359103e18	13	102.15983486270827	109.94550732878284
20	1.3193976166344822e18	13	108.31777346206205	114.34403152557572
21	3.2903033202156175e18	13	44.089455838364245	34.52041154914292
22	8.482350008309597e18	13	17.003425713362045	102.60161611995359
23	6.101209031674573e17	13	25.842511917947366	22.272314298730727
24	1.8162451419244399e19	13	39.638573210187644	43.34914763015038
25	1.3309197553221074e18	13	7.095757204652332	21.04404299195525
26	7.779515179373411e18	14	63.80426636186403	100.78434642499187
27	4.28683702161786e18	14	27.43309009053957	35.68974530952139
28	5.937872779302876e18	14	276.91498822022265	290.1167291705239
29	8.277434084408434e18	14	60.095450394724104	43.40383683199056
30	3.8719824664564173e18	14	24.80615905441871	59.97231132227779
31	9.796434738176467e18	14	21.45662601984968	23.74575780277118
32	4.2803982785172644e18	14	36.582441571177284	67.4381226943068
33	1.1705168465593727e19	14	37.556822732776205	32.88969741379979
34	5.546235957952042e18	14	88.87380459381126	95.99116506490785
35	2.552419613144824e19	14	31.166902974731222	36.723963451169304
36	4.227992561870757e18	15	15.563379312608328	19.599011323097056
37	5.859007350289631e18	15	13.974714130452178	16.39248770656996
38	8.652991891691735e18	15	72.12122789133323	95.5655782183542
39	1.8383449979886094e19	15	118.2033650158989	263.5309838641091

40	6.581732387647914e18	15	23.926484807638683	140.97274594056717
41	1.5426903357896567e19	15	41.348771577098454	40.75749340255354
42	2.9056333619025285e19	15	229.6423260398746	333.75226335487844
43	1.4838416581923312e19	15	53.18930954995267	54.52704305417691
44	2.6895334840373182e19	15	124.67413636996756	94.88356401052424
45	1.214705872715781e19	15	244.58124814685374	179.92316617880468
46	1.5097027936171698e19	15	69.14584939886464	109.17112219679052
47	1.9943467382012723e20	15	41.43803149349301	83.82203728470039
48	1.0925283248003965e19	15	58.952689545073156	156.78973560359313
49	6.093374357739825e18	16	24.150620097509638	35.92139018094681
50	1.0993264246156683e19	16	63.36958239742337	69.99768122728986

Macierz losowa

n	c	rank(A)	error Gauss	error inv
5	1.000000000000001	5	1.4043333874306804e-16	2.2752801345137457e-16
5	9.999999999999996	5	2.482534153247273e-16	2.432376777795247e-16
5	1000.0000000000236	5	2.9790409838967276e-16	4.203627514058621e-15
5	9.999999998130372e6	5	5.799015982916193e-11	1.377444663806729e-10
5	9.999322361505425e11	5	8.767946479620035e-6	1.101328684658636e-5
5	4.3389446864305475e15	4	0.5593130377167642	0.5591916598984644
10	1.000000000000001	10	5.15985034193911e-16	3.3121136700345433e-16
10	9.999999999999993	10	2.8737410463596867e-16	2.4575834280036907e-16
10	999.9999999999911	10	2.293995889930822e-14	2.7609708695270473e-14
10	9.999999990916912e6	10	2.0955792024649492e-10	1.81840723377465e-10
10	9.99976991136977e11	10	4.196755515487614e-5	3.614790502971321e-5
10	1.7803773341590864e16	9	0.14037179426228394	0.13931683203224224
20	1.0000000000000016	20	5.382005793715205e-16	3.9409007944299576e-16
20	10.0	20	7.854386543748146e-16	6.309740391678007e-16
20	1000.0000000000028	20	3.120208680502288e-14	3.6192861929420846e-14
20	9.99999999564815e6	20	1.6033423624713666e-10	1.5697362145029217e-10
20	1.0000134366492958e12	20	1.6168699871775732e-5	1.4716047607150038e-5
20	8.563256185085127e15	19	0.3290975485091718	0.32066957992953976

Wnioski

Możemy zauważyć, że macierze Hilberta osiągają bardzo duże wartości współczynnika uwarunkowania (parametr `cond(A)`) już przy stosunkowo niewielkich rozmiarach n. Wysoki wskaźnik uwarunkowania oznacza, że układ $Ax = b$ jest źle uwarunkowany, czyli nawet niewielkie błędy w danych (w macierzy A lub w b) mogą prowadzić do dużych błędów w rozwiązaniu. Eksperymenty pokazują, że możemy to zaobserowować zarówno dla metody eliminacji Gauss'a jak i metody z użyciem odwrotności macierzy. Oznacza to, że numeryczne rozwiązanie układu Hilberta jest trudne do uzyskania z dużą dokładnością.
Dla losowych macierzy z ustalonym współczynnikiem uwarunkowania c, błędy względne są małe i podobne dla obu metod. Jednak współczynnik ten musi być mały. Wiadomo, że wysoki wskaźnik uwarunkowania w przypadku losowych macierzy od razu sprawia, że zadanie jest źle uwarunkowane. Oznacza to, że algorytmy są stabilne numerycznie jedynie dla macierzy dobrze uwarunkowanych.

4 Zadanie 4

Podpunkt a

W zadaniu 4 mamy do czynienia z dwoma różnymi postaciami tego samego wielomianu. Mowa tutaj o wielomianie Wilkinsona. Jego postać iloczynowa wygląda tak:

$$p(x) = (x-20)\cdot(x-19)\cdot(x-18)\cdot...\cdot(x-3)\cdot(x-2)\cdot(x-1)$$

Z tej postaci od razu łatwo policzyć wszystkie miejsca zerowe:

$$x_0 \in 1, 2, 3, ..., 18, 19, 20$$

Druga postać tego wielomianu to postać naturalna (współczynnikowa):

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

Ta postać wynika bezpośrednio z wymnożenia wszystkich nawiasów z poprzedniej postaci. Liczenie miejsc zerowych jest w jej przypadku znacznie bardziej skomplikowane. Poniżej znajduje się porównanie rzeczywistych pierwiastków równania wraz z wyliczonymi pierwiastkami z wielomianu $P(x)$, wartością wielomianu p i P dla obliczonego miesjca zerowego, oraz różnicę między rzeczywistym a wyliczonym m. zer.

k	z_k	$\lvert P(z_k)\rvert$	$\lvert p(z_k)\rvert$	$\lvert z_k - k\rvert$
1	0.9999999999998084	23323.616390897252	23310.180819556477	1.9162449405030202e-13
2	2.0000000000114264	64613.550791712885	73156.18130995684	1.1426415369442111e-11
3	2.9999999998168487	18851.098984644806	130289.20977428104	1.8315127192636282e-10
4	3.999999983818672	2.6359390809003003e6	2.0313511850413054e6	1.6181327833209025e-8
5	5.000000688670983	2.3709842874839526e7	2.1613359049100634e7	6.88670983350903e-7
6	5.999988371602095	1.2641076289358065e8	1.2165063267640364e8	1.162839790502801e-5
7	7.000112910766231	5.2301629899144447e8	5.061885949319773e8	0.00011291076623098917
8	7.999279406281878	1.798432141726085e9	1.7402728325721796e9	0.0007205937181220534
9	9.003273831140774	5.1218815526720 67e9	5.263758084354485e9	0.003273831140774064
10	9.989265687778465	1.4157542666785017e10	1.4147808356077827e10	0.010734312221535092
11	11.027997558569794	3.586354765112257e10	3.692632803664625e10	0.027997558569794023
12	11.94827395840048	8.510931555828575e10	8.162184753413098e10	0.051726041599520656
13	13.082031971969954	2.2136146301419052e11	2.04374354285492e11	0.08203197196995404
14	13.906800565193148	3.812024574451268e11	3.8519295444834576e11	0.09319943480685211
15	15.081439299377482	8.809029239560208e11	9.126025022282091e11	0.0814392993774824
16	15.942404318674466	1.6747434633806333e12	1.6751148968378867e12	0.05759568132553383
17	17.026861831476396	3.3067827086376123e12	3.5115331717998135e12	0.026861831476395537
18	17.99048462339055	6.166202940769282e12	6.644365795060319e12	0.009515376609449788
19	19.001981084996206	1.406783619602919e13	1.274643243051272e13	0.001981084996206306
20	19.999803908064397	3.284992217648231e13	2.3837033672895914e13	0.00019609193560299332

Wnioski

Wyliczone miejsca zerowe są bardzo zbliżone do rzeczywistych miejsc zerowych. Jednak po podstawieniu ich z powrotem do wzoru otrzymujemy bardzo rozbieżne wyniki. Jak sama nazwa wskazuje miejsce zerowe to taki argument, dla którego funkcja osiąga wartość 0. Żadna z policzonych $P(z_k), p(z_k)$ nie jest nawet bliska 0. Otrzymujemy wartości od rzędu 10^4 aż do 10^{13}. Dzieje się tak, ponieważ wielomian w postaci naturalnej przechowywany jest niedokładnie z uwagi na ograniczenia arytmetyki Float64. Niektóre współczynniki wielomianu przy niższych potęgach x są dużymi liczbami, których zapis pomija killka cyfr znaczących. Nie bez powodu wielomian jest nazwany złośliwym. Nawet bardzo małe oddalnie od precyzyjnej wartości miejsca zerowego, powoduje ogromne zaburzenia wyniku. Sugeruje to złe uwarunkowanie sposobu obliczania pierwiastków wielomianu.

Podpunkt b

Celem eksperymentu Wilkinsona, będącego tematem podpunktu b, było zaburzenie jednego ze współczynników i obserwacja zachodzących zmian. Współczynnikiem, który uległ zmianie był -210. Teraz przyjmuje wartość $-210-2^{-23}$. Analogiczna tabela jak w przykładzie powyżej tym razem dla wielomianu o zaburzonym współczynniku.

k	z_k	$\|P(z_k)\|$	$\|p(z_k)\|$	$\|z_k - k\|$
1	0.9999999999999805 + 0.0im	2168.9361669986724	2376.9361669985137	1.9539925233402755e-14
2	1.9999999999985736 + 0.0im	29948.438957395843	9132.438957447212	1.4264145420384011e-12
3	3.000000000105087 + 0.0im	239010.53520956426	74756.28518912957	1.0508705017286957e-10
4	3.9999999950066143 + 0.0im	939293.8049425513	626853.3644463811	4.993385704921138e-9
5	5.000000034712704 + 0.0im	7.44868039679552e6	1.0894298660834588e6	3.4712703822492585e-8
6	6.000005852511414 + 0.0im	1.4689332508961653e7	6.12250863773296e7	5.852511414161654e-6
7	6.999704466216799 + 0.0im	5.817946400915084e7	1.3252981745917735e9	0.00029553378320112955
8	8.007226654064777 + 0.0im	1.3954205929609105e8	1.7380734718418133e10	0.0072266540647767386
9	8.917396943382494 + 0.0im	2.459617755654851e8	1.3487291517349089e11	0.082603056617506
10	10.09529034477879 - 0.6432770896263527im	2.291018560461982e9	1.4824347490765474e12	0.6502965968281023
11	10.09529034477879 + 0.6432770896263527im	2.291018560461982e9	1.4824347490765474e12	1.110092326920887
12	11.793588728372308 - 1.6522535463872843im	2.077690789102519e10	3.2939582416936758e13	1.6650968123818863
13	11.793588728372308 + 1.6522535463872843im	2.077690789102519e10	3.2939582416936758e13	2.0458176697496047
14	13.99233053734825 - 2.5188196443048287im	9.390730597798799e10	9.545412818924855e14	2.5188313205122075
15	13.99233053734825 + 2.5188196443048287im	9.390730597798799e10	9.545412818924855e14	2.7129043747424584
16	16.73073008036981 - 2.8126272986972136im	9.592356563898315e11	2.7420705762218132e16	2.906000476898456
17	16.73073008036981 + 2.8126272986972136im	9.592356563898315e11	2.7420705762218132e16	2.8254873227453055
18	19.50243895868367 - 1.9403320231930836im	5.050467401799687e12	4.2524547099295936e17	2.4540193937292005
19	19.50243895868367 + 1.9403320231930836im	5.050467401799687e12	4.2524547099295936e17	2.004328632592893
20	20.84690887410499 + 0.0im	4.858653129933677e12	1.374367214153433e18	0.8469088741049902

Wnioski

Wykonany powyżej eksperyment potwierdza tezę, że zadanie jest źle uwarunkowane. Tym razem, niewielkie zaburzenie danych, powoduje pojawienie się pierwiastków zespolonych. Jest to coś kompletnie innego niż spodziewane wyniki. Wartości funkcji dla tych rzekomych pierwiastków są jeszcze większe niż w przypadku podpunktu a.

5 Zadanie 5

W tym zadaniu mamy rozważyć problem rekurencyjny dotyczący wzrotu populacji. Mamy równanie:

$$p_{n+1} := p_n + r \cdot p_n \cdot (1 - p_n)$$

określone dla n = 0, 1, 2, 3, ... oraz r będącej pewną stałą. Należało wykonać doświadczenie, w którym $p_0 = 0.01$ a $r = 3.0$. Liczba iteracji w każdym przypadku wynosiła 40, z tym, że rekurencję należało policzyć na dwa sposoby:

- sposób 1 zakładał policzenie reukrencji klasycznie, w pętli 40-krotnie
- sposób drugi zakładał obcięcie wyniku do trzech cyfr po przecinku po wykonaniu 10 iteracji i nastęnie policzenie już klasycznie do końca.

Wyniki dla arytmetyki `Float32` zawarte są w dwóch pierwszych kolumnach. Pierwsza to sposób 1, druga sposób 2. W kolumnie trzeciej znajdują się wyniki dla sposobu 1 i arytmetyki `Float64`

Iter	p	Iter	p (modified)	Iter	p
1	0.0397	1	0.0397	1	0.0397
2	0.15407173	2	0.15407173	2	0.15407173000000002
3	0.5450726	3	0.5450726	3	0.5450726260444213
4	1.2889781	4	1.2889781	4	1.2889780011888006
5	0.1715188	5	0.1715188	5	0.17151914210917552
6	0.5978191	6	0.5978191	6	0.5978201201070994
7	1.3191134	7	1.3191134	7	1.3191137924137974
8	0.056273222	8	0.056273222	8	0.056271577646256565
9	0.21559286	9	0.21559286	9	0.21558683923263022
10	0.7229306	10	0.7229306	10	0.722914301179573
		cut	0.722		
11	1.3238364	11	1.3241479	11	1.3238419441684408
12	0.037716985	12	0.036488414	12	0.03769529725473175
13	0.14660022	13	0.14195944	13	0.14651838271355924
14	0.521926	14	0.50738037	14	0.521670621435246
15	1.2704837	15	1.2572169	15	1.2702617739350768
16	0.2395482	16	0.28708452	16	0.24035217277824272
17	0.7860428	17	0.9010855	17	0.7881011902353041
18	1.2905813	18	1.1684768	18	1.2890943027903075
19	0.16552472	19	0.577893	19	0.17108484670194324
20	0.5799036	20	1.3096911	20	0.5965293124946907
21	1.3107498	21	0.09289217	21	1.3185755879825978
22	0.088804245	22	0.34568182	22	0.058377608259430724
23	0.3315584	23	1.0242395	23	0.22328659759944824
24	0.9964407	24	0.94975823	24	0.7435756763951792
25	1.0070806	25	1.0929108	25	1.315588346001072
26	0.9856885	26	0.7882812	26	0.07003529560277899
27	1.0280086	27	1.2889631	27	0.26542635452061003
28	0.9416294	28	0.17157483	28	0.8503519690601384
29	1.1065198	29	0.59798557	29	1.2321124623871897
30	0.7529209	30	1.3191822	30	0.37414648963928676
31	1.3110139	31	0.05600393	31	1.0766291714289444
32	0.0877831	32	0.21460639	32	0.8291255674004515
33	0.3280148	33	0.7202578	33	1.2541546500504441
34	0.9892781	34	1.3247173	34	0.29790694147232066
35	1.021099	35	0.034241438	35	0.9253821285571046
36	0.95646656	36	0.13344833	36	1.1325322626697856
37	1.0813814	37	0.48036796	37	0.6822410727153098
38	0.81736827	38	1.2292118	38	1.3326056469620293
39	1.2652004	39	0.3839622	39	0.0029091569028512065
40	0.25860548	40	1.093568	40	0.011611238029748606

Wnioski

Możemy zauważyć, że wyniki dla sposobu 1 pokrywają się dość dokładnie w obydwu arytmatykach. Dopiero w iteracji 18 występuje różnica na 2 miejscu znaczącym , a w iteracji 23 na pierwszym. W okolicach 19 iteracji model na którym zastosowaliśmy sposób 2, czyli sposób z obcięciem, zaczyna mocno odstawać od wyników w pozostałych dwóch kolumnach. Jest to spowodowane faktem, że nasze rygorystyczne obcięcie (do trzeciego miejsca po przecinku) kumuluje się w kolejnych iteracjach. Program działa rekurencyjnie, a więc nasze uproszczenie zastosowane po kroku 10 nie daje o sobie zapomnieć w kolejnych iteracjach. Zwłaszcza, że we wzorze wartość p_n jest podnoszona do kwadratu. Nasza utrata precyzji jest przez to jeszcze większa.
Wobec tego zauważamy, że zarówno zastosowane uproszczenie w sposobie 2 jak i stosowana arytmetyka mają znaczenie w precyzji obliczeń. Najbardziej godne zaufania i zapewne bliskie prawdy są obliczenia we `Float64` bez obcięć, jednak nawet ta aryemetyka i ten konkretny spsób nie gwarantują nam 100-procentowej dokładności.

6 Zadanie 6

W zadaniu 6 znów spotykamy się z rekurencją:

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

dla n = 0, 1, 2, 3, ..., gdzie c jest pewną daną stałą. Eksperymenty mamy przeprowadzić dla danych:

- $c = -2$ i $x_0 \in \{1, 2, 1.99999999999999\}$
- $c = -1$ i $x_0 \in \{1, -1, 0.75, 0.25\}$

Poniżej wartości otrzymane dla c = -2

x = 1.0	
Iter	x
1	-1.0
2	-1.0
3	-1.0
4	-1.0
5	-1.0
6	-1.0
7	-1.0
8	-1.0
9	-1.0
10	-1.0
11	-1.0
12	-1.0
13	-1.0
14	-1.0
15	-1.0
16	-1.0
17	-1.0
18	-1.0
19	-1.0
20	-1.0
21	-1.0
22	-1.0
23	-1.0
24	-1.0
25	-1.0
26	-1.0
27	-1.0
28	-1.0
29	-1.0
30	-1.0
31	-1.0
32	-1.0
33	-1.0
34	-1.0
35	-1.0
36	-1.0
37	-1.0
38	-1.0
39	-1.0
40	-1.0

x = 2.0	
Iter	x
1	2.0
2	2.0
3	2.0
4	2.0
5	2.0
6	2.0
7	2.0
8	2.0
9	2.0
10	2.0
11	2.0
12	2.0
13	2.0
14	2.0
15	2.0
16	2.0
17	2.0
18	2.0
19	2.0
20	2.0
21	2.0
22	2.0
23	2.0
24	2.0
25	2.0
26	2.0
27	2.0
28	2.0
29	2.0
30	2.0
31	2.0
32	2.0
33	2.0
34	2.0
35	2.0
36	2.0
37	2.0
38	2.0
39	2.0
40	2.0

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

Teraz wartości dla c = -1

x = 1.0

Iter	x
1	0.0
2	-1.0
3	0.0
4	-1.0
5	0.0
6	-1.0
7	0.0
8	-1.0
9	0.0
10	-1.0
11	0.0
12	-1.0
13	0.0
14	-1.0
15	0.0
16	-1.0
17	0.0
18	-1.0
19	0.0
20	-1.0
21	0.0
22	-1.0
23	0.0
24	-1.0
25	0.0
26	-1.0
27	0.0
28	-1.0
29	0.0
30	-1.0
31	0.0
32	-1.0
33	0.0
34	-1.0
35	0.0
36	-1.0
37	0.0
38	-1.0
39	0.0
40	-1.0

x = -1.0

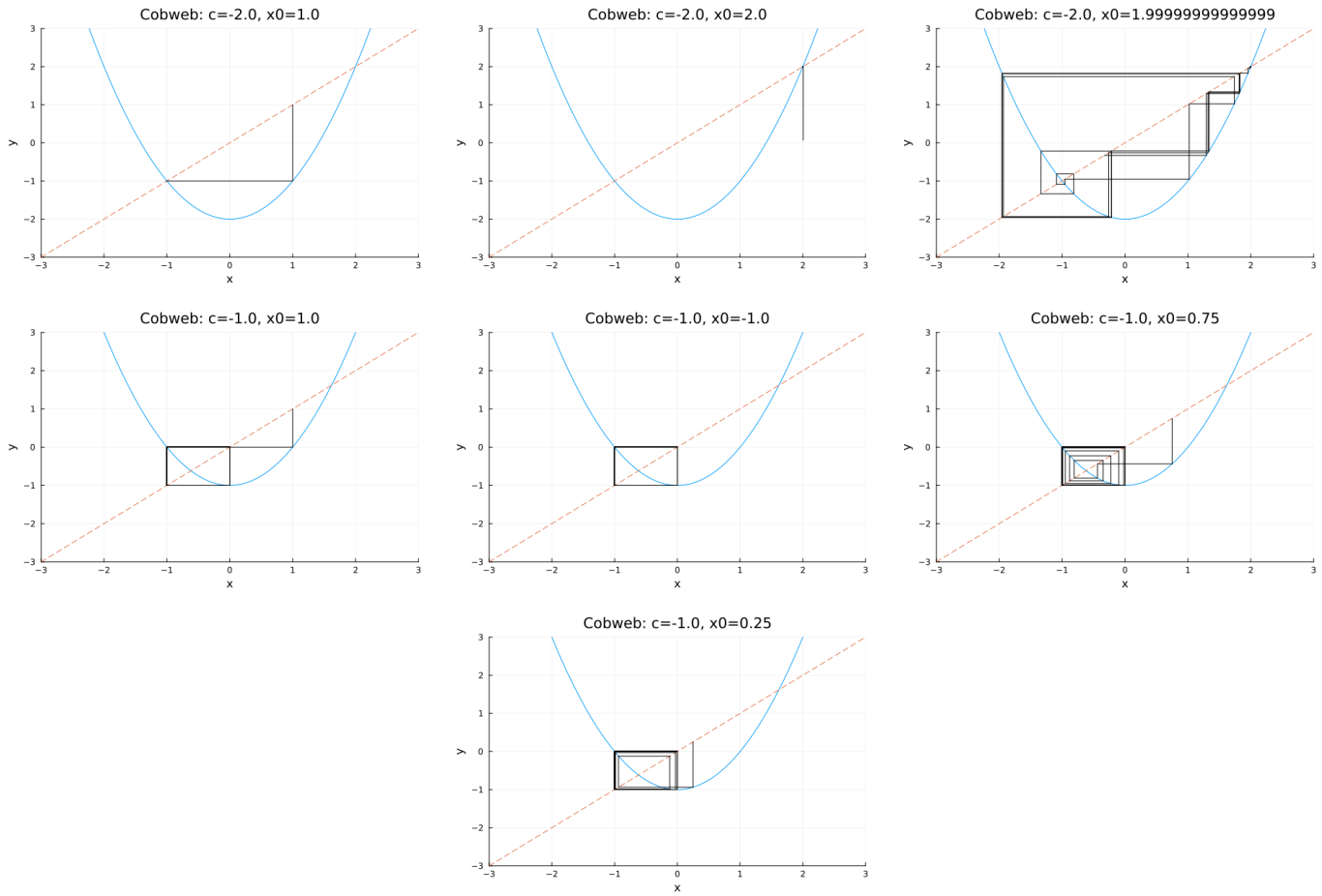
Iter	x
1	0.0
2	-1.0
3	0.0
4	-1.0
5	0.0
6	-1.0
7	0.0
8	-1.0
9	0.0
10	-1.0
11	0.0
12	-1.0
13	0.0
14	-1.0
15	0.0
16	-1.0
17	0.0
18	-1.0
19	0.0
20	-1.0
21	0.0
22	-1.0
23	0.0
24	-1.0
25	0.0
26	-1.0
27	0.0
28	-1.0
29	0.0
30	-1.0
31	0.0
32	-1.0
33	0.0
34	-1.0
35	0.0
36	-1.0
37	0.0
38	-1.0
39	0.0
40	-1.0

x = 0.75

Iter	x
1	-0.4375
2	-0.80859375
3	-0.3461761474609375
4	-0.8801620749291033
5	-0.2253147218564956
6	-0.9492332761147301
7	-0.0989561875164966
8	-0.9902076729521999
9	-0.01948876442658909
10	-0.999620188061125
11	-0.0007594796206411569
12	-0.9999994231907058
13	-1.1536182557003727e-6
14	-0.9999999999986692
15	-2.6616486792363503e-12
16	-1.0
17	0.0
18	-1.0
19	0.0
20	-1.0
21	0.0
22	-1.0
23	0.0
24	-1.0
25	0.0
26	-1.0
27	0.0
28	-1.0
29	0.0
30	-1.0
31	0.0
32	-1.0
33	0.0
34	-1.0
35	0.0
36	-1.0
37	0.0
38	-1.0
39	0.0
40	-1.0

x = 0.25

Iter	x
1	-0.9375
2	-0.12109375
3	-0.9853363037109375
4	-0.029112368589267135
5	-0.9991524699951226
6	-0.0016943417026455965
7	-0.9999971292061947
8	-5.741579369278327e-6
9	-0.9999999999670343
10	-6.593148249578462e-11
11	-1.0
12	0.0
13	-1.0
14	0.0
15	-1.0
16	0.0
17	-1.0
18	0.0
19	-1.0
20	0.0
21	-1.0
22	0.0
23	-1.0
24	0.0
25	-1.0
26	0.0
27	-1.0
28	0.0
29	-1.0
30	0.0
31	-1.0
32	0.0
33	-1.0
34	0.0
35	-1.0
36	0.0
37	-1.0
38	0.0
39	-1.0
40	0.0



Rysunek 1: Reprezentacje iteracji graficznej rekurencji $x_{n+1} = x_n^2 + c$

Wnioski

Precyzja wykonywanych obliczeń zależy od przyjętych parametrów początkowych. Np. przyjęcie $x_0 = 1.0$ dla obydwu c daje w każdej iteracji dobry wynik. Tak samo $x_0 = 2.0$ dla $c = -1.0$ oraz $x_0 = 2.0$ dla $c = -2.0$. Natmoiast dla $x_0 = 0.75$ oraz $x_0 = 0.25$ i $c = -1.0$ wyniki stabilizują się dopiero po czasie (z czego dla $c = -1.0$ dzieje się to szybciej niż w drugim przypadku) Dla $x_0 = 1.99999999999999$ wynik zawsze jest inny (niestabilny). Oznacza to, że aby otrzymać wiarygodny i dokładny wynik należy niemałą uwagę poświęcić na wybór odpowiednich wartości parametrów.

Czarne linie na wykresach reprezentują nam zbieganie do punktów stałych. Po uważnym prześledzeniu ich przebiegu dochodzimy do wniosku, że pokrywają się z danymi przedstawionymi w kolumnach nad wykresami. Tzn. np. dla $x = 1.0$ oraz $c = -2.0$ widzimy natychmaistową stablizację w punkcie -1.0. (Dla wszystkch iteracji otzrymujemy zawsze -1.0). Natmoiast dla $x = 1.99999999999999$ oraz $c = -2.0$ zauważamy ciągłe krążenie między wartościami ≈ 2 a ≈ -2 i brak jednego punktu stałego, co również zgadza się z wcześniej otrzymanymi danymi.