Obliczenia naukowe - Lista nr 2

Paweł Narolski

6 listopada 2018 r.

1 Badanie uwarunkowania zadań

1.1 Eksperymentalne badanie wpływu niewielkich zmian danych na wyniki przeprowadzanych obliczeń dla algorytmów obliczania iloczynu skalarnego dwóch wektorów

Pierwszy eksperyment przeprowadzony w ramach drugiej listy laboratoryjnej miał na celu zbadanie, jaki wpływ na wyniki otrzymywanych przez nas obliczeń mogą mieć nawet relatywnie niewielkie zmiany w danych wejściowych.

Aby przeprowadzić doświadczenie skorzystaliśmy z zaimplementowanych na potrzeby pierwszej listy laboratoryjnej z *Obliczeń naukowych* czterech algorytmów obliczania iloczynu skalarnego:

1. Algorytmu obliczający sumę "w przód": $\sum_{i=1}^{n} x_i * y_i$
2. Algorytmu obliczający sumę "w tył": $\sum_{i=n}^{1} x_i * y_i$
3. Algorytmu "od największego do najmniejszego" [malejący]: (1) dodaj dodatnie liczby w porządku od największego do najmniejszego, (2) dodaj ujemne liczby w porządku od najmniejszego do największego, (3) dodaj do siebie obliczone sumy częściowe
4. Algorytmu "od najmniejszego do największego" [rosnący]: (1) dodaj dodatnie liczby w porządku od najmniejszego do największego, (2) dodaj ujemne liczby w porządku od największego do najmniejszego, (3) dodaj do siebie obliczone sumy częściowe

Obliczenia wykonamy na danych wejściowych, które będą stanowiły dwie pary wektorów:

$$x = [2.718281828, 3.141592654, 1.414213562, 0.57721566\mathbf{9}, 0.301029995\mathbf{7}]$$
$$y = [1486.2497, 878366.9879, 22.37492, 4773714.647, 0.000185049]$$

oraz

$$a = [2.718281828, 3.141592654, 1.414213562, 0.577215664, 0.301029995]$$
$$b = [1486.2497, 878366.9879, 22.37492, 4773714.647, 0.000185049]$$

które różnią się jedynie brakiem ostatniej cyfry 9 w a_4 oraz brakiem ostatniej 7 w b_5.

Porównanie wyników działania algorytmów podczas operowania na liczbach zmiennoprzecinkowych w arytmetyce *Float32* oraz *Float64* dla obu par wektorów x, y i a, b prezentuje się następująco:

Algorytm	$x \cdot y$, **Float32**	$a \cdot b$, **Float32**	$x \cdot y$, **Float64**	$a \cdot b$, **Float64**
w przód	-0.4999443	-0.4999443	1.0251881368296672e-10	-0.004296342739891585
w tył	-0.4543457	-0.4543457	-1.5643308870494366e-10	-0.004296342998713953
malejący	-0.5	-0.5	0.0	-0.004296342842280865
rosnący	-0.5	-0.5	0.0	-0.004296342842280865

Otrzymane wyniki przeprowadzonych operacji dla liczb zmiennoprzecinkowych w arytmetyce *Float32* i *Float64* następnie mieliśmy porównać z prawidłową wartością iloczynu skalarnego wektorów x i y, która z dokładnością do 15 cyfr wynosi:

$$1.0065710700000010 * 10^{-}11$$

Zauważamy, że wyniki obliczeń przeprowadzonych dla par wektorów x, y i a, b w arytmetyce *Float32* są identyczne. Dzieje się tak z powodu zbyt małej precyzji zapisu liczb zmiennopozycyjnych potrzebnej do prawidłowej reprezentacji wyniku.

Co więcej, dostrzegamy rażącą różnicę pomiędzy wynikami przeprowadzonych obliczeń w arytmetyce *Float64* dla par wektorów x, y i a, b pomimo dokonania - jak mogłoby się wydawać - relatywnie niewielkiej zmiany w danych wejściowych. Otrzymane wyniki dla pary a, b znacznie odbiegają od prawidłowego, oczekiwanego rezultatu obliczeń.

Wobec powyższych dochodzimy do wniosku, że mamy do czynienia z zadaniem *źle uwarunkowanym*, ponieważ małe zmiany danych początkowych wywołują duże zmiany wyników końcowych.

1.2 Wykorzystanie dostępnych programów do wizualizacji w celu narysowania wykresu funkcji $f(x) = e^x ln(1 + e^{-x})$

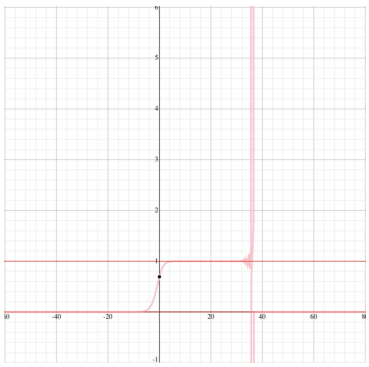
Naszym kolejny zadaniem było narysowanie wykresu funkcji danej wzorem

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

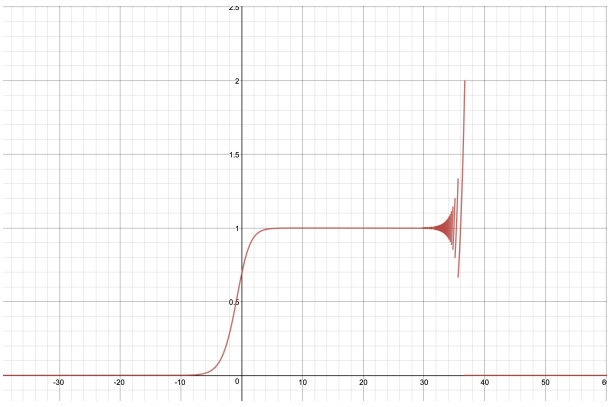
za pomocą dostępnych programów do wizualizacji.

W tym celu skorzystano z darmowych aplikacji *Desmos*, *GeoGebra* oraz *Symbolab*, z których można skorzystać za pośrednictwem dowolnej, nowoczesnej przeglądarki internetowej.

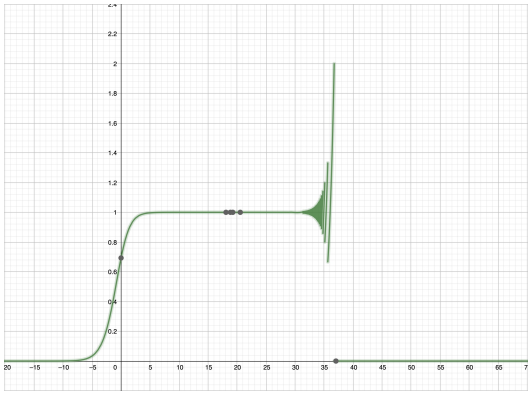
Wyniki działania wyżej wymienionych programów prezentują się następująco:



Rysunek 1. Wykres funkcji $f(x) = e^x ln(1 + e^{-x})$ w programie Symbolab



Rysunek 2. Wykres funkcji $f(x) = e^x ln(1 + e^{-x})$ w programie Desmos



Rysunek 3. Wykres funkcji $f(x) = e^x ln(1 + e^{-x})$ w programie Geogebra

Następnie otrzymane wykresy porównujemy z obliczoną granicą funkcji $f(x)$ dla $x \to \infty$, która wynosi:

$$\lim_{x \to \infty} e^x ln(1 + e^{-x}) = 1 \tag{2}$$

Zauważamy, że dla $x \geqslant 32$ wykresy generowane przez programy *Symbolab*, *Desmos* i *GeoGebra* są błędne. Wartości na wykresie funkcji oscylują znacznie powyżej oraz poniżej ustalonej granicy funkcji, by następnie osiągnąć zero.

Dzieje się tak, ponieważ dla $x \geqslant 32$ dokonujemy mnożenia bardzo dużej liczby, jaką jest e^x z bardzo małą liczbą $ln(1 + e^{-x})$, przez co otrzymujemy skrajnie błędne wyniki. Dla $x38$ zaś funkcja przyjmuje wartość równą 0, ponieważ dla tych argumentów dochodzi już do tak znacznych błędów przybliżenia e^{-x}, że $ln(1 + e^{-x}) = ln(1) = 0$, a więc $f(x) = 0$ dla $x \geqslant 39$.

Na podstawie przeprowadzonych eksperymentów możemy zatem stwierdzić, że algorytm obliczający wartość funkcji $f(x)$ jest niestabilny numerycznie, ponieważ małe błędy popełnione na poszczególnych etapach obliczeń nawarstwiają się istotnie zniekształcając otrzymany wynik.

1.3 Rozwiązywanie układu równań liniowych przy użyciu metody eliminacji Gaussa oraz inwersji

Otrzymaliśmy do rozwiązania układ równań liniowych:

$$Ax = b \tag{3}$$

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

1. $A = H_n$, gdzie H_n jest macierzą Hilberta stopnia n wygenerowaną za pomocą funkcji $A = hilb(n)$ której implementacja w języku Julia została dołączona do listy zadań
2. $A = R_n$, gdzie R_n jest losową macierzą stopnia n z zadanym wskaźnikiem uwarunkowania c wygenerowanym za pomocą funkcji $A = matcond(n, c)$ której implementacja w języku Julia została dołączona do listy zadań

Naszym zadaniem było rozwiązać powyższy układ równań za pomocą dwóch algorytmów:

1. Algorytmu eliminacji Gaussa ($x = A \div b$)
2. Algorytmu wykorzystującego inwersję macierzy A ($x = A^{-1} \cdot b$)

dla macierzy H_n rosnącego stopnia $n1$ oraz dla macierzy losowej R_n z rosnącym wskaźnikiem uwarunkowania $c = 1, 10, 10^3, 10^7, 10^{12}, 10^{16}$, a następnie porównać obliczoną przybliżoną wartość $\tilde{x}$ z dokładnym $x = (1, \ldots, 1)^T$.

Przy użyciu funkcji *cond()* dostępnej dla programistów języka Julia dokonano także obliczeń wskaźnika uwarunkowania macierzy, który pozwala na oszacowanie, z dokładnością do ilu miejsc po przecinku jesteśmy w stanie podać prawidłowy wynik przeprowadzanych obliczeń.

Wyniki przeprowadzonych obliczeń dla macierzy Hilberta rozmiaru n x n zaprezentowano w poniższej tabeli.

n	**Błąd dla metody eliminacji Gaussa**	**Błąd dla metody inwersji**	**Wskaźnik *cond()***
2	5.661048867003676e-16	1.4043333874306803e-15	19.28147006790397
3	8.022593772267726e-15	0.0	524.0567775860644
4	4.137409622430382e-14	0.0	15513.73873892924
5	1.6828426299227195e-12	3.3544360584359632e-12	476607.25024259434
6	2.618913302311624e-10	2.0163759404347654e-10	1.4951058642254665e7
7	1.2606867224171548e-8	4.713280397232037e-9	4.75367356583129e8
8	6.124089555723088e-8	3.07748390309622e-7	1.5257575538060041e10
9	3.8751634185032475e-6	4.541268303176643e-6	4.9315375644687 62e11
10	8.67039023709691e-5	0.0002501493411824886	1.6024416992541715e13
11	0.00015827808158590435	0.007618304284315809	5.222677939280335e14
12	0.1339620837208 5344	0.258994120804705	1.7514731907091464e16
13	0.11039701117868264	5.331275639426837	3.344143497338461e18
14	1.4554087127659643	8.71499275104814	6.200786263161444e17
15	4.696668350857427	7.344641453111494	3.674392953467974e17
16	54.15518954564602	29.84884207073541	7.865467778431645e17
17	13.707236683836307	10.516942378369349	1.2636843426660 52e18
18	9.134134521198485	7.575475905055309	2.2446309929189128e18
19	9.720589712655698	12.233761393757726	6.471953976541591e18
20	7.549915039472976	22.062697257870493	1.3553657908688225e18

Zauważamy, że błąd względny przeprowadzanych obliczeń na macierzach Hilberta rośnie wraz ze wzrostem wskaźnika uwarunkowania *cond()*. Wzorst błędu względnego dla obu metod rozwiązywania układu równań jest także bezpośrednio związany ze wzrostem stopnia macierzy Hilberta.

Następnie przeprowadzono obliczenia dla macierzy losowej R_n dla $n = 5, 10, 20$ wraz ze zdefiniowanym wskaźnikiem uwarunkowania $c = 1, 10, 10^3, 10^7, 10^{12}, 10^{16}$, wyznaczając przy tym błędy względne obliczeń. Wyniki przeprowadzonych eksperymentów prezentują się następująco:

n	**Błąd dla metody eliminacji Gaussa**	**Błąd dla metody inwersji**	**Wskaźnik *cond()***
5	1.4043333874306804e-16	1.1102230246251565e-16	1
5	1.9860273225978183e-16	1.1102230246251565e-16	10
5	1.9078787780978257e-14	2.177033834961652e-14	10^3
5	1.0714543741857892e-10	1.3712815695197491e-10	10^7
5	2.298881931334716e-5	2.4515180989165693e-5	10^{12}
5	0.35101316036292596	0.3283481384140924	10^{16}
10	3.14018491736755e-16	2.673771110915334e-16	1
10	2.979040983896727e-16	3.9094986940344 54e-16	10
10	6.191123632071186e-15	9.61532620829385e-15	10^3
10	3.6527737590885094e-10	3.418292007084761e-10	10^7
10	3.191299578170427e-5	2.5151521620334627e-5	10^{12}
10	0.18403101082992243	0.13184389443580616	10^{16}
20	8.0174707872164015e-16	4.124295487574583e-16	1
20	1.0999055204272255e-15	9.729507111180987e-16	10
20	2.6466841381137368e-14	2.7135267990813953e-14	10^3
20	2.652234173595074e-10	2.3615498860455003e-10	10^7
20	2.4714645794708757e-6	3.983802523461423e-6	10^{12}
20	0.060092404386669206	0.06048541487497342	10^{16}

Podobnie jak dla macierzy Hilberta zauważamy, że wzrost błędu względnego dla obu metod rozwiązywania układu równań jest bezpośrednio powiązany ze wzrostem wskaźnika uwarunkowania *cond()*.

Na podstawie przeprowadzonych eksperymentów zauważamy, że zadanie rozwiązywania układu równań liniowych dla macierzy Hilberta jest zadaniem *źle uwarunkowanym*, ponieważ wraz ze wzrostem stopnia macierzy gwałtownie rośnie jej wskaźnik uwarunkowania *cond()*, a co za tym idzie - błąd względny przeprowadzanych obliczeń.

W wypadku przeprowadzanych obliczeń dla macierzy losowych błąd względny również jest powiązany z rosnącym wskaźnikiem uwarunkowania *cond()* danej macierzy. Błędy względne odpowiadające obu użytym metodom rosną, jednak zdecydowanie wolniej, niż dla macierzy Hilberta.

1.4 Złośliwy wielomian Wilkinson'a

Obliczanie pierwiastków tzw. *złośliwego wielomianu Wilkinson'a* stanowi kolejny przykład zadania źle uwarunkowanego, które przebadaliśmy w ramach niniejszej listy laboratoryjnej.

Problem *złośliwego wielomianu Wilkinsona* pojawił się w analizie numerycznej podczas badania algorytmu obliczającego pierwiastki wielomianu postaci:

$$p(x) = \sum_{i=0}^{n} c_i x^i. \tag{4}$$

Okazało się, że zadanie znalezienia pierwiastków wielomianu postaci (1) ze współczynnikami c_i jest zadaniem źle uwarunkowanym, kiedy mamy do czynienia z następującymi sytuacjami:

1. Wielomian ma pierwiastek o krotności 1 (np. $w(x) = x^2$ ma pierwiastek $x = 0$ krotności 2
2. Wielomian ma zbliżone do siebie miejsca zerowe
3. Wielomian ma równo oddalone miejsca zerowe

Złośliwy wielomian Wilkinsona to przykład wielomianu, którego miejsca zerowe są równo od siebie oddalone (3). Okazuje się, że problem ten jest **bardzo** źle uwarunkowany, na tyle, że jego odkrycie Wilkinson uznał za najbardziej traumatyczne przeżycie w swojej karierze naukowej[1]

Badanie złego uwarunkowania obliczania *złośliwego wielomianu Wilkinsowa* rozpoczęliśmy od obliczenia pierwiastków wielomianu $P(x)$o współczynnikach zapisanych w pliku *wielomian.txt* dołączonym do listy zadań za pomocą funkcji *roots()* dostępnej w pakiecie *Polynomials* dla języka Julia.

$P(x)$ jest postacią wielomianu Wilkinsona $p(x)$, tak więc możemy porównać otrzymane dane w wyniku wykonania funkcji *roots()* z oczekiwanymi, prawidłowymi wynikami:

Pierwiastek $p(x)$	Pierwiastek $P(x)$
1	0.9999999999996989
2	2.0000000000283182
3	2.9999999995920965
4	3.9999999837375317
5	5.000000665769791
6	5.999989245824773
⋮	⋮
18	17.99092135271648
19	19.00190981829944
20	19.999809291236637

[1]Wilkinson, James H. (1984). "The perfidious polynomial". In Gene H. Golub. Studies in Numerical Analysis. Mathematical Association of America. p. 3. ISBN 978-0-88385-126-5

Następnie obliczone pierwiastki z_k wielomianu $P(x)$ porównaliśmy z wartościami otrzymanymi po obliczeniu $|P(z_k)|$, $|p(z_k)|$ oraz $|z_k - k|$:

k	z_k	$\|P(z_k)\|$	$\|p(z_k)\|$	$\|z_k - k\|$
1	0.9999999999996989	36352.0	38400.0	3.0109248427834245e-13
2	2.0000000000283182	181760.0	198144.0	2.8318236644508943e-11
3	2.9999999995920965	209408.0	301568.0	4.0790348876384996e-10
4	3.9999999837375317	3.106816e6	2.844672e6	1.626246826091915e-8
5	5.000000665769791	2.4114688e7	2.3346688e7	6.657697912970661e-7
6	5.999989245824773	1.20152064e8	1.1882496e8	1.0754175226779239e-5
7	7.000102002793008	4.80398336e8	4.78290944e8	0.00010200279300764947
8	7.999355829607762	1.682691072e9	1.67849728e9	0.0006441703922384079
9	9.002915294362053	4.465326592e9	4.457859584e9	0.002915294362052734
10	9.990413042481725	1.2707126784e10	1.2696907264e10	0.009586957518274986
11	11.025022932909318	3.5759895552e10	3.5743469056e10	0.025022932909317674
12	11.953283253846857	7.216771584e10	7.2146650624e10	0.04671674615314281
13	13.07431403244734	2.15723629056e11	2.15696330752e11	0.07431403244734014
14	13.914755591802127	3.65383250944e11	3.653447936e11	0.08524440819787316
15	15.075493799699476	6.13987753472e11	6.13938415616e11	0.07549379969947623
16	15.946286716607972	1.555027751936e12	1.554961097216e12	0.05371328339202819
17	17.025427146237412	3.777623778304e12	3.777532946944e12	0.025427146237412046
18	17.99092135271648	7.199554861056e12	7.1994474752e12	0.009078647283519814
19	19.00190981829944	1.0278376162816e13	1.0278235656704e13	0.0019098182994383706
20	19.999809291236637	2.7462952745472e13	2.7462788907008e13	0.00019070876336257925

Zauważamy, że błędy przybliżenia powstałe podczas obliczania pierwiastków z_k przyczyniają się do otrzymania skrajnie błędnych wartości funkcji $P(x)$ i $p(x)$. Wyniki znacznie odbiegają od wartości oczekiwanych z powodu faktu, iż

Na podstawie powyższych obserwacji możemy potwierdzić, że zadanie obliczenia pierwiastków wielomianu Wilkinsona rzeczywście jest zadaniem skrajnie źle uwarunkowanym.

1.4.1 Eksperyment Wilkinsona

Zobaczymy teraz, co się stanie w sytuacji, kiedy zmniejszymy współczynnik x^{19} wynoszący -210.0 o 2^{-23}. Wówczas wartość wielomianu $P(20)$ zmieni się z 0 na $6,25 * 10^{17}$, a pierwiastek wielomianu $x = 20$ przybierze wartość $x \approx 20,8$. Wielomian w nowej postaci będzie posiadał także pierwiastki zespolone.

Pomimo dokonania stosunkowo niewielkiej zmiany pierwiastki wielomianu będą teraz znacznie inaczej porozmieszczane niż pierwotnie pomimo dokonania relatywnie niewielkiej zmiany w postaci wielomianu i faktu, że w wyjściowym wielomianie pierwiastki były rozmieszczone w sposób równomierny.

Wilkinson udowodnił, że takie zachowanie po dokonaniu zmiany wynika z faktu, iż niektóre pierwiastki, takie jak $a = 15$ będą miały wiele "bliskich" pierwiastków b takich, że moduł $|a - b|$ jest mniejszy od liczby a.[2].

Obliczając nowe wartości pierwiastków zmienionego wielomianu $P(x)$ zauważamy, jak duży wpływ na otrzymane wyniki ma odjęcie niewielkiej wartości 2^{-23} (wyniki zaprezentowano w tabeli niżej).

Zauważamy, że pierwiastki wielomianu $P(x)$ uległy przesunięciu. Część z nich reprezentowana jest teraz także przez liczby zespolone.

Powyższy eksperyment ponownie udowadnia, jak skrajnie źle uwarunkowanym zadaniem jest obliczanie pierwiastków wielomianu Wilkinson'a.

[2]Szczegółowa analiza jest dostępna w artykule https://en.wikipedia.org/wiki/Wilkinsons-polynomial na podstawie którego oparto niniejszy podrozdział sprawozdania

k	z_k	$\|P(z_k)\|$	$\|p(z_k)\|$	$\|z_k - k\|$
1	0.9999999999998357 + 0.0im	20992.0	22016.0	1.6431300764452317e-13
2	2.0000000000550373 + 0.0im	349184.0	365568.0	5.503730804434781e-11
3	2.99999999660342 + 0.0im	2.221568e6	2.295296e6	3.3965799062229962e-9
4	4.000000089724362 + 0.0im	1.046784e7	1.0729984e7	8.972436216225788e-8
5	4.99999857388791 + 0.0im	3.9463936e7	4.3303936e7	1.4261120897529622e-6
6	6.000020476673031 + 0.0im	1.29148416e8	2.06120448e8	2.0476673030955794e-5
7	6.99960207042242 + 0.0im	3.88123136e8	1.757670912e9	0.00039792957757978087
8	8.007772029099446 + 0.0im	1.072547328e9	1.8525486592e10	0.007772029099445632
9	8.915816367932559 + 0.0im	3.065575424e9	1.37174317056e11	0.0841836320674414
10	10.095455630535774 - 0.6449328236240688im	7.143113638035824e9	1.4912633816754019e12	0.6519586830380406
11	10.095455630535774 + 0.6449328236240688im	7.143113638035824e9	1.4912633816754019e12	1.1109180272716561
12	11.793890586174369 - 1.6524771364075785im	3.357756113171857e10	3.2960214141301664e13	1.665281290598479
13	11.793890586174369 + 1.6524771364075785im	3.357756113171857e10	3.2960214141301664e13	2.045820276678428
14	13.992406684487216 - 2.5188244257108443im	1.0612064533081976e11	9.545941595183662e14	2.5188358711909045
15	13.992406684487216 + 2.5188244257108443im	1.0612064533081976e11	9.545941595183662e14	2.7128805312847097
16	16.73074487979267 - 2.812624896721978im	3.315103475981763e11	2.7420894016764064e16	2.9060018735375106
17	16.73074487979267 + 2.812624896721978im	3.315103475981763e11	2.7420894016764064e16	2.825483521349608
18	19.5024423688181 - 1.940331978642903im	9.539424609817828e12	4.2525024879934694e17	2.454021446312976
19	19.5024423688181 + 1.940331978642903im	9.539424609817828e12	4.2525024879934694e17	2.004329444309949
20	20.84691021519479 + 0.0im	1.114453504512e13	1.3743733197249713e18	0.8469102151947894

Konkludujemy także, iż nasze intuicyjne pojmowanie znaczenia dokonywania niektórych operacji na liczbach zmiennoprzecinkowych (do których należy np. odejmowanie bardzo małych liczb) może okazać się skrajnie niezgodne ze stanem faktycznym, szczególnie, jeśli mamy do czynienia z zadaniem źle uwarunkowanym.

1.5 Badanie wzoru rekurencyjnego opisującego model rozwoju populacji Verhulste'a

Przebadamy teraz wzór rekurencyjny opisujący model rozwoju populacji Verhulste'a, nazywanego także modelem *logistycznym*[3]:

$$p_{n+1} = p_n + rp_n(1 - p_n) \tag{5}$$

gdzie r jest pewną stałą, $r(1 - p_n)$stanowi czynnik wzrostu populacji, a p_0 jest wielkością populacji stanowiącą procent maksymalnej wielkości populacji dla danego stanu środowiska.

Przeprowadzimy dwa eksperymenty w oparciu o powyższe równanie:

1. Sprawdzimy, co się stanie, kiedy intencjonalnie zaburzymy otrzymane wyniki poprzez obcięcie cyfr znaczących podczas jednej z iteracji programu obliczającego wartość wyrażenia dla $n = 40$
2. Porównamy otrzymane wyniki obliczeń bez intencjonalnego wprowadzenia zaburzenia wyników w arytmetyce *Float32* i *Float64*

1.5.1 Zaburzenie wyników obliczeń

Zaimplementowany został program w języku Julia (dołączony do niniejszego sprawozdania w pliku *ex5.jl*, który dla danych $p = 0.01$ i $r = 3$ wykonuje 40 iteracji pętli obliczającej kolejne wartości wyrażenia rekurencyjnego, a następnie powtarza obliczenia, z tą różnicą, że po 10. iteracji dokonujemy obcięcia otrzymanego wyniku p_{10} do trzech cyfr znaczących po przecinku (bez zaokrąglenia). W tym celu wykorzystujemy funkcję *trunc()* dostępną dla programistów języka *Julia*:

$$p_{10} = trunc(p_{10}, 3) = 0.722. \tag{6}$$

Po zaburzeniu danych w kolejnej iteracji algorytmu obliczenia dokonywane są na nowej wartości p_{10} zamiast na poprzedniej, wynoszącej w arytmetyce *Float32* 0.7229306.

[3]od logis (fr.) - dom, kwatera

Następnie przeprowadziliśmy obliczenia w arytmetyce *Float64* bez wprowadzania intencjonalnego zaburzenia danych, aby porównać wcześniej otrzymane dane z możliwe dokładnymi wynikami przeprowadzonych obliczeń. Otrzymane rezultaty przedstawiono w poniższej tabeli:

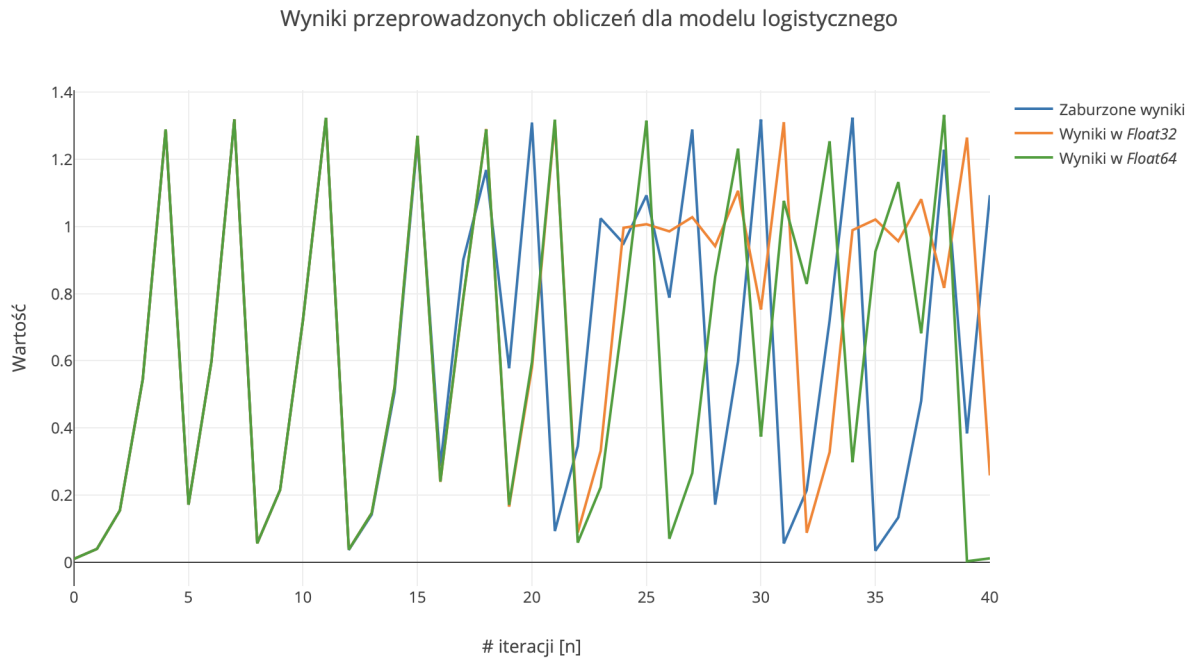
n	p_n z zaburzeniem	p_n bez zaburzenia	p_n bez zaburzenia w *Float64*
0	0.01	0.01	0.01
1	0.0397	0.0397	0.0397
2	0.15407173	0.15407173	0.15407173000000002
3	0.5450726	0.5450726	0.5450726260444213
4	1.2889781	1.2889781	1.2889780011888006
5	0.1715188	0.1715188	0.17151914210917552
6	0.5978191	0.5978191	0.5978201201070994
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	1.3241479	1.3238364	1.3238419441684408
12	0.036488414	0.037716985	0.03769529725473175
13	0.14195944	0.14660022	0.14651838271355924
14	0.50738037	0.521926	0.521670621435246
15	1.2572169	1.2704837	1.2702617739350768
16	0.28708452	0.2395482	0.24035217277824272
17	0.9010855	0.7860428	0.7881011902353041
18	1.1684768	1.2905813	1.2890943027903075
19	0.577893	0.16552472	0.17108484670194324
20	1.3096911	0.5799036	0.5965293124946907
21	0.09289217	1.3107498	1.3185755879825978
22	0.34568182	0.088804245	0.058377608259430724
23	1.0242395	0.3315584	0.22328659759944824
24	0.94975823	0.9964407	0.7435756763951792
25	1.0929108	1.0070806	1.315588346001072
26	0.7882812	0.9856885	0.07003529560277899
27	1.2889631	1.0280086	0.26542635452061003
28	0.17157483	0.9416294	0.8503519690601384
29	0.59798557	1.1065198	1.2321124623871897
30	1.3191822	0.7529209	0.37414648963928676
31	0.05600393	1.3110139	1.0766291714289444
32	0.21460639	0.0877831	0.8291255674004515
33	0.7202578	0.3280148	1.2541546500504441
34	1.3247173	0.9892781	0.29790694147232066
35	0.034241438	1.021099	0.9253821285571046
36	0.13344833	0.95646656	1.1325322626697856
37	0.48036796	1.0813814	0.6822410727153098
38	1.2292118	0.81736827	1.3326056469620293
39	0.3839622	1.2652004	0.0029091569028512065
40	1.093568	0.25860548	0.011611238029748606

Zauważamy, jak fundamentalny wpływ na poprawność otrzymanych wyników ma dokonanie niewielkiej zmiany w p_{10}. "Zaburzone" p_{40} wynosi aż 1.093568, podczas gdy p_{40} bez dokonanego zaburzenia w arytmetyce *Float32* wynosi zaledwie 0.25860548. To ponad czterokrotnie mniejsza wartość!

Dokonując dalszych porównań zauważamy, że w arytmetyce *Float64*, w której możemy spodziewać się otrzymania dokładniejszych wyników, wartość p_{40} wynosi 0.011611238029748606. Względem zaburzonej wartości p_{40} stanowi to blisko **sto razy mniejszą wartość**.

Aby zobrazować różnicę w otrzymanych wynikach, otrzymane dane zwizualizowano za pomocą wykresu

liniowego, który prezentuje się w następujący sposób:



Zauważamy, że wykresy dla trzech przebadanych przypadków przestają pokrywać się ze sobą dla $n \geqslant 16$.

Na podstawie przeprowadzonych obliczeń wnioskujemy, że zadanie obliczenia kolejnych wyrazów ciągu opisanego równaniem rekurencyjnym (5) również jest zadaniem źle uwarunkowanym. Zauważamy, że niewielkie zaburzenie jednej z wielu wartości liczbowych podczas wykonywania obliczeń może prowadzić do otrzymania skrajnie niepoprawnych wyników. Zwiększenie precyzji obliczeń, choć dla pewnych n może przyczynić się do uzyskania wyników zbliżonych do poprawnych, dla złożonych symulacji rzędu $n = 100$, $n = 500$ itd. nie pozwoli na uniknięcie błędów zaokrągleń.

1.6 Badanie wzoru rekurencyjnego opisanego funkcją kwadratową $x^2 + c$

Pozostało nam teraz przebadać równanie rekurencyjne opisane wzorem:

$$x_{n+1} := x_n^2 + c \tag{7}$$

dla $n = 0, 1, \ldots$, gdzie c jest pewną stałą. Równanie to, z dokładnością do zmiany układu współrzędnych, jest równoważne równaniu (5) przebadanemu w ramach poprzedniego zadania[4].

Otrzymaliśmy zadanie przeprowadzenia 40. iteracji wyrażenia (7) w arytmetyce *Float64* dla następujących danych:

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

[4]Na podstawie rozdziału 1. książki "Granice chaosu. Fraktale", H. - O. Peitgen, H. Jurgens, D. Saupe

7. c=1 i x_0 =0.25

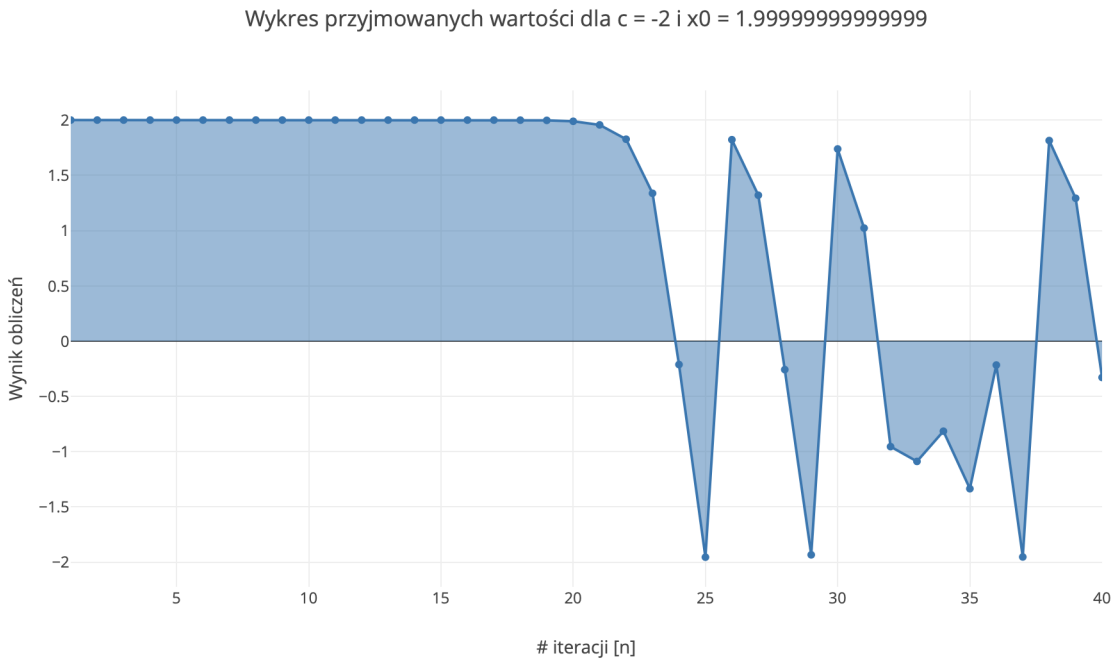
Zauważamy, że wyniki przeprowadzonych obliczeń dla niektórych wartości c i x_0 cechuje regularność opisana za pomocą tabelki:

c	x_0	Zwracane wartości
-2	1	$[-1, -1, -1, \ldots, -1]$
-2	2	$[2.0, 2.0, 2.0, \ldots, 2.0]$
-1	1	$[0.0, -1.0, \ldots, -1.0]$
-1	-1	$[0.0, -1.0, \ldots, -1.0]$

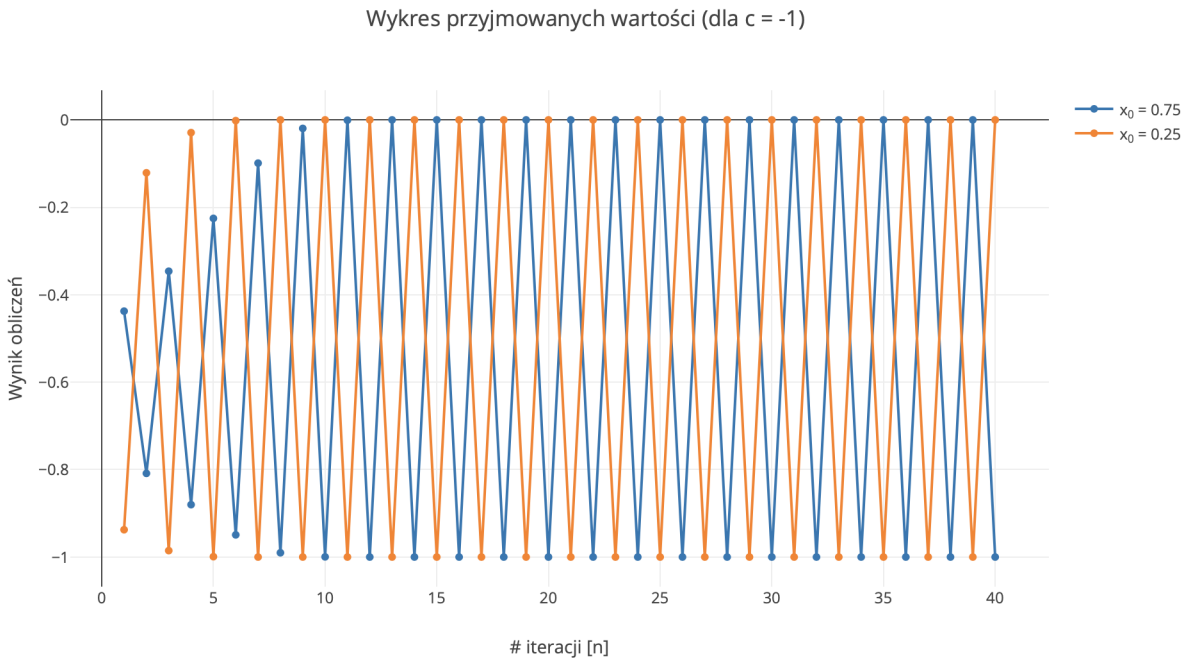
Dla pozostałych wartości liczbowych wyniki zaprezentowano poniżej:

n	$c = -2, x_0 = 1.9...9$	$c = -1, x_0 = 0.75$	$c = -1, x_0 = 0.25$
1	1.99999999999996	-0.4375	-0.9375
2	1.9999999999998401	-0.80859375	-0.12109375
3	1.9999999999993605	-0.3461761474609375	-0.9853363037109375
4	1.999999999997442	-0.8801620749291033	-0.029112368589267135
5	1.9999999999897682	-0.2253147218564956	-0.9991524699951226
6	1.9999999999590727	-0.9492332761147301	-0.0016943417026455965
7	1.999999999836291	-0.0989561875164966	-0.9999971292061947
8	1.9999999993451638	-0.9902076729521999	-5.741579369278327e-6
9	1.9999999973806553	-0.01948876442658909	-0.9999999999670343
10	1.999999989522621	-0.999620188061125	-6.593148249578462e-11
11	1.9999999580904841	-0.0007594796206411569	-1.0
12	1.9999998323619383	-0.9999994231907058	0.0
13	1.9999993294477814	-1.1536182557003727e-6	-1.0
14	1.9999973177915749	-0.9999999999986692	0.0
15	1.9999892711734937	-2.6616486792363503e-12	-1.0
16	1.9999570848090826	-1.0	0.0
17	1.999828341078044	0.0	-1.0
18	1.9993133937789613	-1.0	0.0
19	1.9972540465439481	0.0	-1.0
20	1.9890237264361752	-1.0	0.0
21	1.9562153843260486	0.0	-1.0
22	1.82677862987391	-1.0	0.0
23	1.3371201625639997	0.0	-1.0
24	-0.21210967086482313	-1.0	0.0
25	-1.9550094875256163	0.0	-1.0
26	1.822062096315173	-1.0	0.0
27	1.319910282828443	0.0	-1.0
28	-0.2578368452837396	-1.0	0.0
29	-1.9335201612141288	0.0	-1.0
30	1.7385002138215109	-1.0	0.0
31	1.0223829934574389	0.0	-1.0
32	-0.9547330146890065	-1.0	0.0
33	-1.0884848706628412	0.0	-1.0
34	-0.8152006863380978	-1.0	0.0
35	-1.3354478409938944	0.0	-1.0
36	-0.21657906398474625	-1.0	0.0
37	-1.953093509043491	0.0	-1.0
38	1.8145742550678174	-1.0	0.0
39	1.2926797271549244	0.0	-1.0
40	-0.3289791230026702	-1.0	0.0

Zauważamy, że dla wartości $x_0 = 1.99999999999999$ utrata precyzji wykonywanych obliczeń z każdą iteracją pętli prowadzi do uzyskania coraz większego "rozrzutu" pomiędzy przyjmowanymi przez ciąg wartościami, co ilustruje poniższy wykres:



Dla wartości $x_0 = 0.75$ oraz $x_0 = 0.25$ odpowiednio dla $n15$ i $n10$ obserwujemy zaokrąglenie liczb bliskich zeru do 0.0 oraz bliskich ujemnej jedności do -1.0, spowodowane utratą bitów znaczących podczas wykonywanych operacji, a co za tym idzie - ciąg staje się cykliczny. Możemy to zauważyć na wykresie:



1.6.1 Przyczyny chaosu

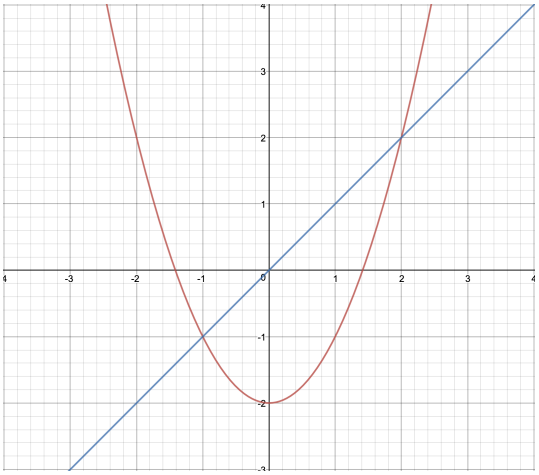
Czy obserwowane zachowanie chaotyczne jest spowodowane operacją podnoszenia liczb do kwadratu, której dokonujemy? Moglibyśmy się tego spodziewać, biorąc pod uwagę fakt, że jeśli dla pewnych n dojdzie do zaburzenia wyników obliczeń np. w wyniku błędu przybliżenia, to w następnych iteracjach $n+1, n+2, \ldots$ błąd zostanie zwielokrotniony za sprawą funkcji kwadratowej.

Okazuje się jednak, że sama operacja podnoszenia liczb do kwadratu nie wyjaśnia powstającego chaosu. Aby się o tym przekonać, przeanalizujmy poniższe przykłady[5]:

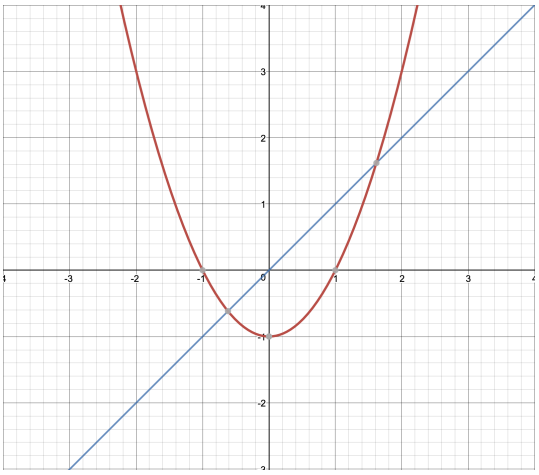
Zauważyliśmy że dla niektórych danych początkowych ($c \in \{-2, -1\}$ oraz $x_0 \in \{1, 2, -1\}$) otrzymujemy stałe, stabilne wyniki obliczeń.

Przeprowadzając doświadczenia dla $x_0 = 1.99999999999999$ zauważyliśmy także, że z każdą iteracją otrzymujemy coraz bardziej niepoprawne wyniki, podczas gdy moglibyśmy się spodziewać, że ponieważ $x_0 \approx 2$, otrzymane rezultaty będą zbliżone do $[2.0, 2.0, 2.0, \ldots, 2.0]$. Jak również spostrzegliśmy, po pewnej liczbie iteracji dla $c = -1$ i $x_0 \in \{0.75, 0.25\}$ proces ustala się i powtarzają się jedynie wartości 0 i -1.

Przeprowadzając analizę wykresu funkcji $f(x) = x^2 - 2$ oraz $f(x) = x^2 - 1$ możemy także wyciągnąć dodatkowe wnioski:



$x_0 = 1 \implies \phi(x)$ zbieżna do -1
$x_0 = 2 \implies \phi(x)$ zbieżna do 2
$x_0 = 1.99999999999999 \implies \phi(x)$ rozbieżna



$x_0 = 1 \implies$ wyrazy nieparzyste $\phi(x)$ zbieżne do 0, parzyste do -1

[5]Na podstawie rozdziału 1. książki "Granice chaosu. Fraktale", H. - O. Peitgen, H. Jurgens, D. Saupe

$$x_0 = -1 \implies \text{wyrazy nieparzyste } \phi(x) \text{ zbieżne do 0, parzyste do -1}$$
$$x_0 = 0.75 \implies \text{wyrazy nieparzyste } \phi(x) \text{ od pewnego } x \text{ zbieżne do 0, parzyste do -1}$$
$$x_0 = 0.25 \implies \text{wyrazy nieparzyste } \phi(x) \text{ od pewnego } x_1 x \text{ zbieżne do 0, parzyste do -1}$$

Zauważamy zatem, że otrzymywane wyniki w dużej mierze zależne są przede wszystkim od danych wejściowych c i x_0.

Co więcej, nieduże zmiany w tych pierwotnych danych mogą prowadzić do otrzymania bardzo różnych wyników (np. w wyniku utraty precyzji wykonywanych obliczeń) lub do stabilnych wyników obliczeń. Wobec powyższych stwierdzamy, że zadanie to jest źle uwarunkowane.