Obliczenia naukowe

Lista 2

Jakub Brodziński
229781

1 Zadanie 1

1.1 Opis problemu

Celem zadania było powtórzenie obliczeń, a następnie porównanie ich z olbiczeniami wykonanymi na poprzedniej liście, tmym razem zaburzających w niewielkim stopniu dane wejściowe. Dane różniły się dla dwóch wspólrzędnych, których wartość dla poprzedniej listy wynosiła $x_4 = 0.5772156649$ oraz $x_5 = 0.3010299957$. Po zaburzeniu są one równe $x_4' = 0.5772156640$ oraz $x_5' = 0.3010299950$. Dokładna wartość liczonego przez nas iloczynu skalarnego jest równa $-1.00657107000000e - 11$.

1.2 Opis rozwiązania

Obliczenia zostały wykonane dla danych wejściowych przed i po zaburzeniu, dla arytmetyki *single* oraz *double*. Do obliczeń został wykorzystany program z Listy 1. Algorytm A był algorytmem, gdzie kolejno sumowaliśmy iloczyny współrzędnych. W algorytmie B iloczyny sumowaliśmy w odwrotnej kolejności niż miało to miejsce w algorytmie A . W przypadku algorytmu C na początku sumowaliśmy dodatnie iloczynu (od największego do najmniejszego), a następnie ujemne iloczyny (od najmniejszego do największego). Algorytm D był algorytmem, gdzie sumowaliśmy iloczynu od największego do najmniejszego.

1.3 Wyniki

Porównanie wyników przed i po zaburzeniu danych wejściowych zostało przedstawione w tabeli poniżej.

Algorytm	*Lista*1		*Lista*2	
	*Float*32	Float64	*Float*32	*Float*64
A	-0.4999443	$1.0251881368296672e - 10$	-0.4999443	-0.004296342739891585
B	-0.4543457	$-1.5643308870494366e - 10$	-0.4543457	-0.004296342998713953
C	-0.5	0.0	-0.5	-0.004296342842280865
D	-0.5	0.0	-0.5	-0.004296342842280865

1.4 Wnioski

Precyzja arytmetyki *Float*32 jest na tyle mała, że wprowadzone zaburzenia nie miały wpływu na wynik, niezależnie od wykorzystanego algorytmu. W przypadku *Float*64 precyzja była wystarczająca, aby zauważyć wyraźne różnice w wynikach przed i po zaburzeniu x_4 oraz x_5. Niewielkie względne zaburzenie danych spodowodowały znaczne odchylenia od wyników z czego możemy wnioskować, że zadanie jest zadaniem źle uwarynkowanym.

2 Zadanie 2

2.1 Opis problemu

Celem zadania jest przedstawienie graficzne wykresu funkcji $f(x) = e^x ln(1 + e^{-x})$ w dwóch dowolnych progamach do wizualizacji, a następnie porównanie otrzymanego rezultatu z granicą funkcji $\lim_{x\to\infty} f(x)$.

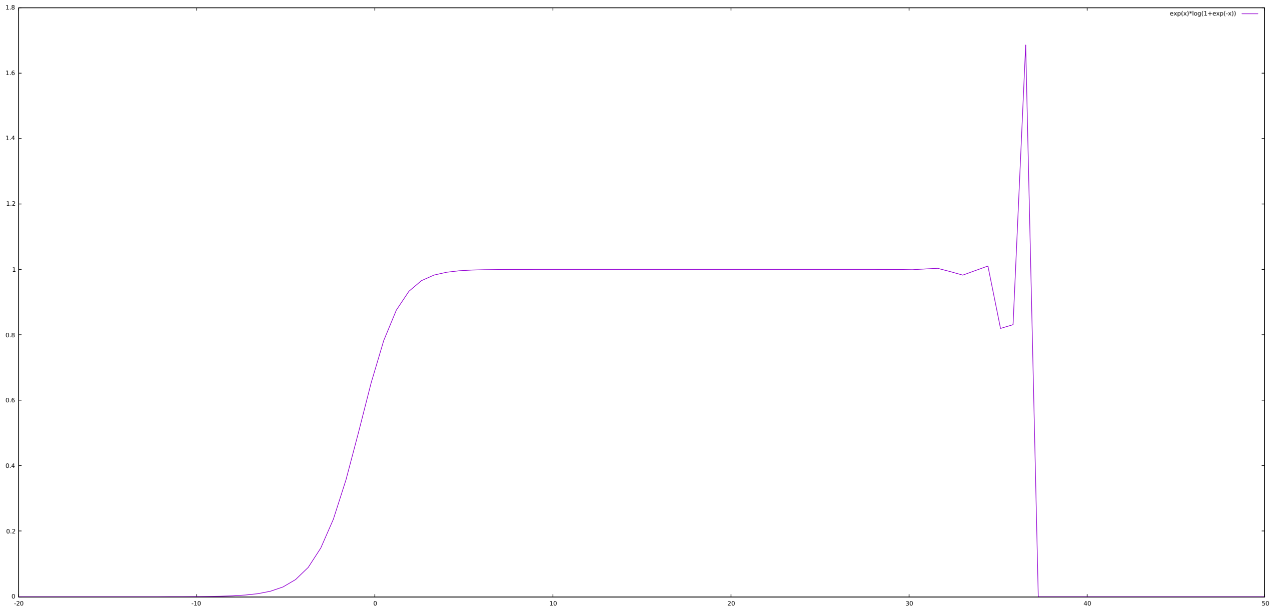
2.2 Opis rozwiązania

Do wizualizacji funkcji zostały wykorzystane dwa oprogramowania, *GNUPlot* oraz *Octave*.

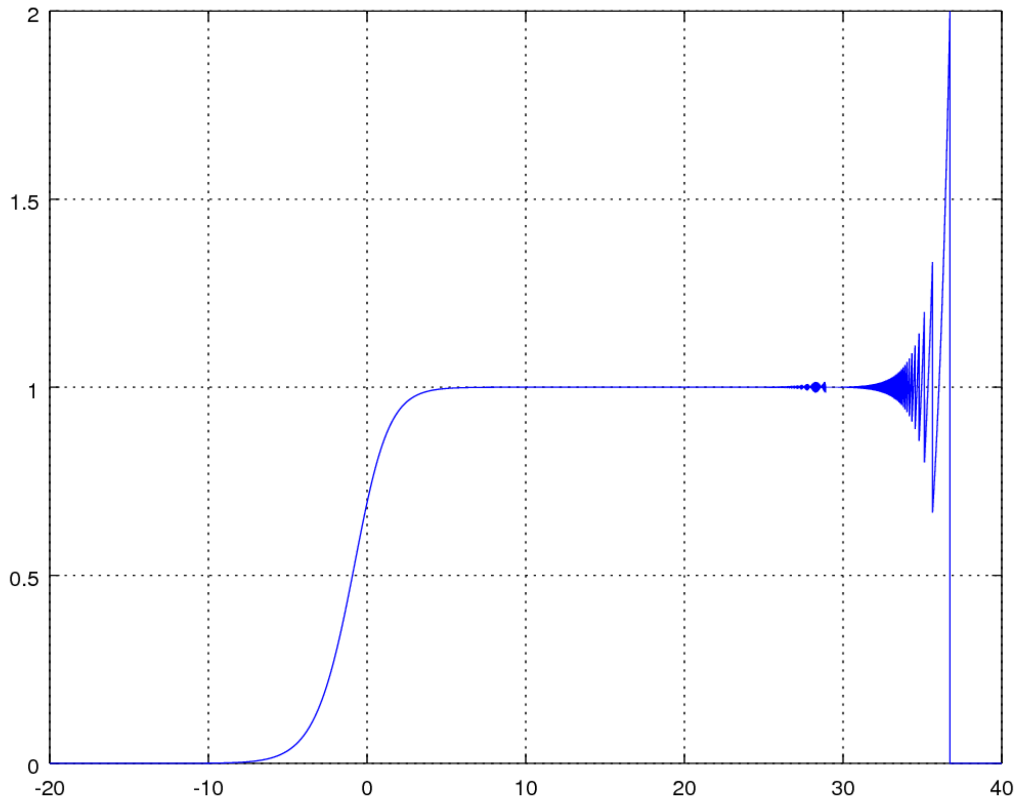
2.3 Wyniki

Policzona granica funkcji wynosi:

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



Rysunek 1: $f(x) = e^x ln(1 + e^{-x})$ przy użyciu $GNUPlot$



Rysunek 2: $f(x) = e^x ln(1 + e^{-x})$ przy użyciu $Oktave'a$

2.4 Wnioski

Przedstawione wykresy wyraźnie odbiegają od tego, do jakiego punktu zbiega funkcja $f(x)$. Powodem tego jest część funkcji f, a mianowice wyrażenie $ln(1 + e^{-x})$, które wraz ze wzrostem x jest bliższe 0. Funkcja f sprowadza się do mnożenia liczby bardzo dużej z liczbą bardzo małą, przez co występuje redukcja cyfr znaczących, która jest powodem widocznym anomalii.

3 Zadanie 3

3.1 Opis problemu

Celem zadania było rozwiązanie układu równań liniowych $Ax = b$ dla danej macierzy wspólczynników $A \in \mathbb{R}^{n \times n}$ i wektora prawych stron $b \in \mathbb{R}^n$, gdzie $x = (1, ..., 1)^T$ dzięki czemu znamy dokładny wynik równania. Macierz A początkowo jest macierzą $Hilberta$, a w drugiej części zadaniu jest to macierz losowa o zadanym rozmiarze oraz wskaźniku uwarunkowania. Układu równań liniowych mają być policzone na dwa sposoby: $x = A \backslash b$ oraz $x = A^{-1}b$.

3.2 Opis rozwiązania

Przed rozpoczęciem rozwiązania należy wygenerowac macierz A poprzez wywołanie funkcji $hilb(n)$ lub $mathcond(n, c)$, w zależności czy chcemy otrzymac macierz $Hilberta$ czy też macierz losową. Symbol n odpowiada rozmiarowi macierzy, którą otrzymamy, a c to zadany wskaźnik uwarunkowania. Następnie należy policzyc wektor b, ponieważ znana jest dokładna wartość x. Mając macierz A oraz b wykorzystujemy jeden z dwóch algorytmów i otrzymujmey wynik x, który porównujemy z dokładną wartością $x = (1, ..., 1)^T$. Dla każdego obliczonego x został policzony błąd względny liczony jako $\frac{\|x-x'\|}{\|x\|}$.

3.3 Wyniki

Wyniki dla macierzy $Hilberta$ zostały przedstawione w tabeli poniżej, gdzie n to rozmiar macierzy, $rank(A)$ to rząd macierzy A, $cond(A)$ to wskaźnik uwarunkowania macierzy A, natomiast σ_A oraz σ_B to kolejno błędy względne uzyskane przy korzystaniu z algorytmu $x = A \backslash b$ oraz $A^{-1}b$.

n	$rank(A)$	$cond(A)$	σ_A	σ_B
5	5	476607.25024331047	$1.6828426299227195e - 12$	$3.2543043465682462e - 12$
10	10	$1.6024868379056498e13$	0.00010722274297833791	0.0002552291736870214
20	13	$2.5382496382297713e18$	53.10656030160922	39.73518807228086

Wyniki dla macierzy loswoej macierzy o zadanym stopniu i zadanym wskaźniku uwarunkowania zostały przedstawione w tabeli poniżej, gdzie n to rozmiar macierzy, c to zadany wskaźnik uwarunkowania, $rank(A)$ to rząd macierzy A, $cond(A)$ to wskaźnik uwarunkowania macierzy A, natomiast σ_A oraz σ_B to kolejno błędy względne uzyskane przy korzystaniu z algorytmu $x = A\backslash b$ oraz $A^{-1}b$.

n	c	$rank(A)$	$cond(A)$	σ_A	σ_B
5	1.0	5	1.0000000000000009	$1.719950113979703e-16$	$8.599750569898515e-17$
	10.0	5	9.999999999999991	$6.040266291023252e-16$	$6.435464048854839e-16$
	1000.0	5	999.9999999999362	$5.656474633347035e-15$	$2.7196249460821258e-14$
	$1.0e7$	5	$9.999999997580813e6$	$4.353848285956683e-10$	$3.134578470568233e-10$
	$1.0e12$	5	$1.0000105793255767e12$	$4.128851217311691e-5$	$1.3214498959113138e-5$
	$1.0e16$	4	$6.481294592978624e15$	0.19633161431395538	0.05970304117044624
10	1.0	10	1.0000000000000009	$1.954749347017227e-16$	$2.6272671962866383e-16$
	10.0	10	10.000000000000004	$1.570092458683775e-16$	$3.5975337699988616e-16$
	1000.0	10	999.9999999999977	$2.1561663055773042e-14$	$1.903480532519339e-14$
	$1.0e7$	10	$9.99999999955629e6$	$3.383135053154978e-10$	$1.871102722732891e-10$
	$1.0e12$	10	$9.999323449014872e11$	$6.480277315743716e-6$	$4.696495023674256e-5$
	$1.0e16$	9	$1.1637263023277446e16$	0.06898243005213321	0.08572416702715752
20	1.0	20	1.0000000000000016	$7.893521746395322e-16$	$4.852068387831067e-16$
	10.0	20	9.999999999999998	$4.482332113961174e-16$	$4.677452743560217e-16$
	1000.0	20	1000.0000000000415	$1.1146771862425775e-14$	$1.5668783297916933e-14$
	$1.0e7$	20	$9.999999995339684e6$	$9.407735316287122e-11$	$4.6081575480516405e-10$
	$1.0e12$	20	$9.999963233670859e11$	$1.635090958420586e-5$	$3.8355169779333314e-5$
	$1.0e16$	19	$1.972765083298713e16$	0.18551475473714543	0.18958597146458878

3.4 Wnioski

Na podstawie przedstawionych wyników możemy zaobserwować w praktyce jak wskaźnik uwarunkowania macierzy wpływa na prowadzone przez nas obliczenia. Macierz Hilberta to wyjątkowo złośliwa"macierz o bardzo wysokim wskaźniku uwarnkowania oraz ogromnym błedzie względem. Dla macierzy *Hilberta* rozmiaru 20 wskaźnik uwarunkowania macierzy to aż $cond(A) = 2.5382496382297713e18$, a błędy względne to kolejno $\sigma_A = 53.10656030160922$ oraz $\sigma_B = 39.73518807228086$

4 Zadanie 4

4.1 Opis problemu

Celem zadania było zaznajomienie się z złośliwym wielomianem"Wilkinson'a oraz dostępnym w Julii pakietem *Polynomials*, poprzez wykorzystanie pakietu w obliczaniu pierwiastków zadanego wielomianu oraz policzenie $|P(z_k)|$,$|p(z_k)|$,$|z_k - k|$. Przez z_k oznaczamy policzony przy pomocy pakietu *Polynomials* pierwiastek,k to dokładna wartość tego pieriwastka, a $|P(z_k)|$ oraz $|p(z_k)|$ to wartość wielomianu P oraz p dla $x = z_k$. Wielomiany zadane są wzorami:

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

$$p(x) = (x-20)(x-19)(x-18)(x-17)(x-16)(x-15)(x-14)(x-13)(x-12)(x-11)(x-10)(x-9)(x-8)(x-7)(x-6)(x-5)(x-4)(x-3)(x-2)(x-1)$$

W drugiej części zadania powtórzony został eksperyment Wilkinson'a, tj. przed obliczaniem pierwiastków wielomianu jeden ze współczynników wielomianu został zmieniony (a_{19}) z -210 na $-210 - 2^{-23}$.

4.2 Opis rozwiązania

Rozwiazywanie tego zadania zostało rozpoczęte od stworzenia dwóch zmiennych wykorzystując funkcje $Poly(ArrayFloat64, 1)$ oraz $poly(ArrayFloat64, 1)$, które zwracały strukture danych reprezentującą kolejno wielomian $P(x)$ oraz $p(x)$. Funkcja $Poly()$ jako argument brała tablice współczynników wielomianu, natomiast funkcja $poly()$ tablice pierwiastków wielomianu. Następnie uzyskane zostały pierwiastki wielomianu $P(x)$ poprzez wywołanie funkcji $roots(P)$. Zaimplementowany został rownież wielomian wykorzystanu do eksperymentu Wilkinson'a $P'(x)$. Końcowym etapem rozwiązania było policzenie $|P(z_k)|$,$|p(z_k)|$,$|z_k - k|$, dla każdego z_k otrzymanego z $roots()$.

4.3 Wyniki

Wyniki obliczeń zostały przedstawione w tabeli poniżej, wszystkie obliczenia były wykonywane w arytmetyce $Float64$.
W pierwszej tabeli zostały zawarte wyniki dla $P(x)$ oraz $p(x)$, czyli wielomianu Wilkinson'a. Natomiast w drugiej tabeli zostały zawarte wyniki dla $P'(x)$ (wielomianu Wilkinson'a z zaburzonym a_{19}) oraz $p(x)$.

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

k	z'_k	$P'(z'_k)$	$p(z_k)$	$\lvert z'_k - k\rvert$
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

4.4 Wnioski

W pierwszej tabeli widoczna różnica pomiędzy pierwiastkami wyliczonymi z_k oraz faktycznymi wartościami pierwiastków. Na podstawie wyników można wywnioskować, że wielomian Wilkinsona jest bardzo złośliwym"wielomianem i jest bardzo czuły na drobne odchylenia argumnetów, mimo że z_1 od $k = 1$ różnią się dopiero na 13 miejscu po przecinku to wartość wielomianu różni się o $\approx 3.6e4$. Dla pierwiastka $k = 20$ błąd bezwzględny pierwiastka jest równy 0.00019070876336257925, natomiast wartość $P(z_k)$ różni się od wartości $P(k)$ aż o $2.7462952745472e13$.
Na drugiej tabeli widzimy analogiczne obliczenia, tym razem dla zaburzonego wielomianu $P'(x)$. Mimo tak drobnego zaburzenia $a'_{19} = a_{19} - 2^{-23}$ wyniki znaczne różnią się od wyników uzyskanych dla $P(x)$. Tym razem część otrzymanych przez nas pierwiastków to liczby zespolone. Po raz kolejny w bardzo wyraźne sposób widzimy, jak bardzo wielomian Wilkinson'a jest wrażliwy na odchylenie danych wejściowych, z czego możemy wywnioskować, że jest źle uwarunkowany. Warto rownież zauważyć, że dwa równe sobie wielomiany, przechowywane przez pakiet *Polynomials* w dwa różne sposoby dla tych samych argumentów podają wartości odbiegające od siebie.

5 Zadanie 5

5.1 Opis problemu

Celem zadania było przeprowadzenie eksperymentów na równaniu rekurencyjnym (modelu logistycznym, wzrostu populacji):

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

W równaniu r to pewna stała. Obliczenia należy wykonać dla $Float32$ oraz $Float64$. Dodatkowo dla $Float32$ wykonać dodatkowo eksperyment, gdzie po 10 iteracjach zastosować obcięcie do 3 miejsc po przecinku dla p_{10}. Powyższe równanie przedstawia sprzężenie zwrotne, gdzie dane wyjściowe stają się danymi wejściowymi dla kolejnej iteracji.

5.2 Opis rozwiązania

Została przeprowadzona symulacji dla $p_0 = 0.01$ oraz $r = 3$. Symulacja została przeprowadzona w pętli for. W celu ułatwienia interpretacji wyników zapoznałem się z eksperymentem przeprowadzonym przez $Lorenza$.

5.3 Wyniki

Wyniki symulacji zostały przedstawione w tabeli poniżej. W $Float32$ przez p_i' została oznaczona symulacja, gdzie przy p_{10} wartość została obcięta do 3 miejsc po przecinku.

i	$Float64$	$Float32$	
	p_i	p_i	p'_i
0	0.01	0.01	0.01
1	0.0397	0.0397	0.0397
2	0.15407173000000002	0.15407173	0.15407173
3	0.5450726260444213	0.5450726	0.5450726
4	1.2889780011888006	1.2889781	1.2889781
5	0.17151914210917552	0.1715188	0.1715188
6	0.5978201201070994	0.5978191	0.5978191
7	1.3191137924137974	1.3191134	1.3191134
8	0.056271577646256565	0.056273222	0.056273222
9	0.21558683923263022	0.21559286	0.21559286
10	0.722914301179573	0.7229306	0.722
11	1.3238419441684408	1.3238364	1.3241479
12	0.03769529725473175	0.037716985	0.036488414
13	0.14651838271355924	0.14660022	0.14195944
14	0.521670621435246	0.521926	0.50738037
15	1.2702617739350768	1.2704837	1.2572169
16	0.24035217277824272	0.2395482	0.28708452
17	0.7881011902353041	0.7860428	0.9010855
18	1.2890943027903075	1.2905813	1.1684768
19	0.17108484670194324	0.16552472	0.577893
20	0.5965293124946907	0.5799036	1.3096911
21	1.3185755879825978	1.3107498	0.09289217
22	0.058377608259430724	0.088804245	0.34568182
23	0.22328659759944824	0.3315584	1.0242395
24	0.7435756763951792	0.9964407	0.94975823
25	1.315588346001072	1.0070806	1.0929108
26	0.07003529560277899	0.9856885	0.7882812
27	0.26542635452061003	1.0280086	1.2889631
28	0.8503519690601384	0.9416294	0.17157483
29	1.2321124623871897	1.1065198	0.59798557
30	0.37414648963928676	0.7529209	1.3191822
31	1.0766291714289444	1.3110139	0.05600393
32	0.8291255674004515	0.0877831	0.21460639
33	1.2541546500504441	0.3280148	0.7202578
34	0.29790694147232066	0.9892781	1.3247173
35	0.9253821285571046	1.021099	0.034241438
36	1.1325322626697856	0.95646656	0.13344833
37	0.6822410727153098	1.0813814	0.48036796
38	1.3326056469620293	0.81736827	1.2292118
39	0.0029091569028512065	1.2652004	0.3839622
40	0.011611238029748606	0.25860548	1.093568

5.4 Wnioski

Pierwsze 10 iteracji wygląda identycznie dla $Float32$ co jest faktem oczywistym. Przed kolejną iteracją w jednej z symulacji wynik został obcięty co poskutkowało propagacją błędu. Początkowo róznica między wynikami p_i oraz p'_i nie jest znaczna, lecz w poźniejszych iteracjach błąd się nawarstwia aż do takiego stopnia, że w czterdziestej iteracji wartość p_{40} jest prawie 4 razy mniejsza niż wartość p'_{40}. Jako, że powyższa symulacja jest przykładem sprzężenia zwrotnego, to nawet drobny błąd nawarstwia się co raz bardziej z każdą iteracją. Zjawisko to zostało nazwane przez Lorena jako ćhaos deterministyczny". Wartość końcowa symulacji dla $Float64$ oraz $Float32$ różnią się prawie 20-krotnie, powodem tego jest zbyt mała dokładność $Float32$. Od około 11 iteracji wyniki p_i dla $Float32$ oraz $Float64$ co raz bardziej zaczynają od siebie odbiegać.

6 Zadanie 6

6.1 Opis problemu

Celem zadania było przeprowadzenie serii eksperymentów na równaniu rekurencyjnym:

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

Eksperymenty polegały na róznym doborze stałych c oraz x_0. Podane równanie jest układem sprzężonym zwrotnie.

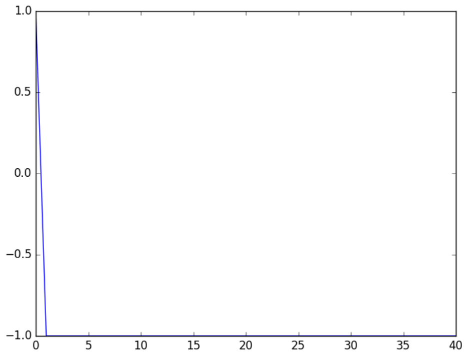
6.2 Opis rozwiązania

Iteracje były wykonywane przy użyciu pętli, dla każdej zadanej pary c oraz x_0 został wygenerowany wykres przy użyciu pakietu *PyPlot*.

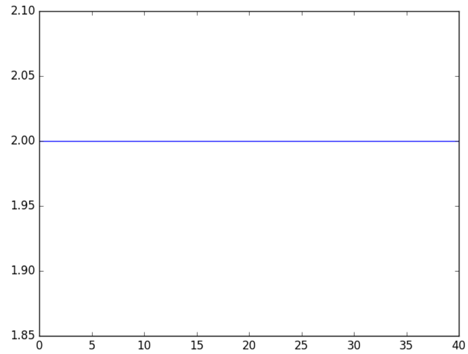
6.3 Wyniki

Wyniki eksperymentów zostały przedstawione w tabeli poniżej, wszystkie obliczenia były wykonywane dla *Float*64. Zostały również przedstawione wykresy prezentujące graficznie wszystkie eksperymenty.

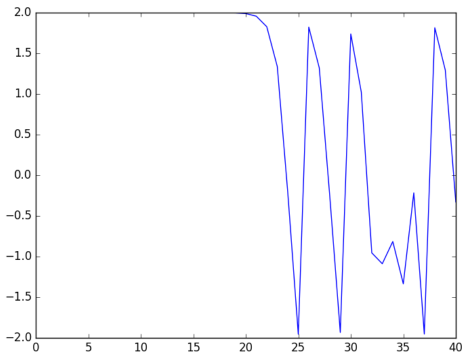
x_i \ c	-2.0	-2.0	-2.0	-1.0	-1.0	-1.0	-1.0
x_0	1.0	2.0	1.99999999999999	1.0	-1.0	0.75	0.25
x_5	-1.0	2.0	1.99999999999996	0.0	0.0	-0.4375	-0.9375
x_{10}	-1.0	2.0	1.9999999999998401	-1.0	-1.0	-0.80859375	-0.12109375
x_{15}	-1.0	2.0	1.9999999999993605	0.0	0.0	-0.3461761474609375	-0.9853363037109375
x_{20}	-1.0	2.0	1.999999999997442	-1.0	-1.0	-0.8801620749291033	-0.029112368589267135
x_{25}	-1.0	2.0	1.9999999999897682	0.0	0.0	-0.2253147218564956	-0.9991524699951226
x_{30}	-1.0	2.0	1.9999999999590727	-1.0	-1.0	-0.9492332761147301	-0.0016943417026455965



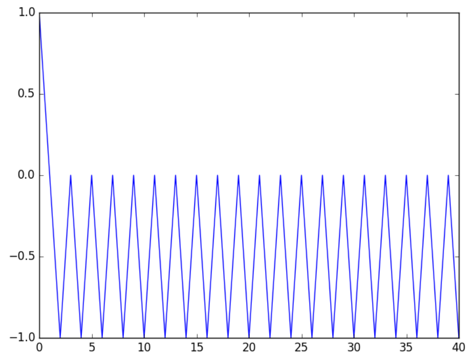
Rysunek 3: $x_{n+1} = x_n - 2$
gdzie $x_0 = 1.0$



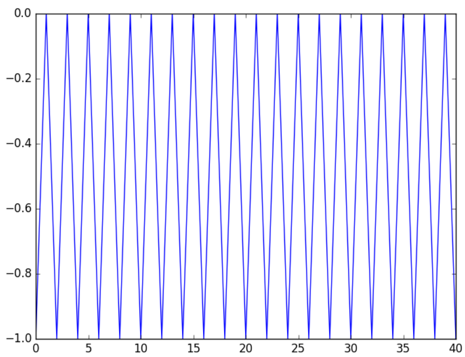
Rysunek 4: $x_{n+1} = x_n - 2$
gdzie $x_0 = 2.0$



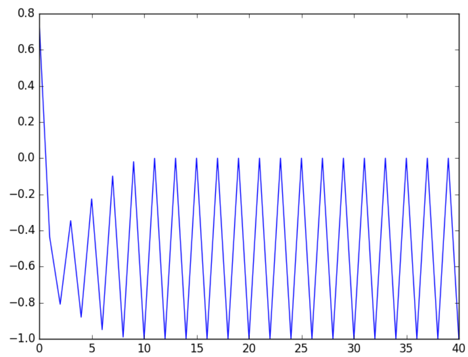
Rysunek 5: $x_{n+1} = x_n - 2$
gdzie $x_0 = 1.99999..$



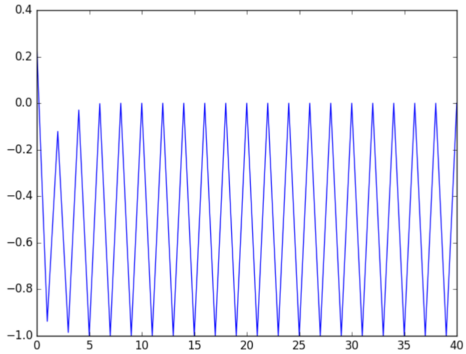
Rysunek 6: $x_{n+1} = x_n - 1$
gdzie $x_0 = 1.0$



Rysunek 7: $x_{n+1} = x_n - 1$
gdzie $x_0 = -1.0$



Rysunek 8: $x_{n+1} = x_n - 1$
gdzie $x_0 = 0.75$



Rysunek 9: $x_{n+1} = x_n - 1$
gdzie $x_0 = 0.25$

6.4 Wnioski

Na wykresach prezentujących eksperymenty na równaniach rekurencyjnych możemy zaobserwować dwa zjawiska, niestabilność oraz stabilizacje. Oba są związane z układami sprzężonymi zwrotnie. Dla rysnku 3 stabilizacja układu sprzężenia zwrotnego była zauważalna dopiero po pewnej liczbie iteracji. Natomiast dla rysunku 4 stabilizacja była widoczna od samego początku iteracji. W przypadku rysnku 5 od pewnej liczby iteracji wartości przestają być uporządkowane, pojawia się niestabilność układu. Natomiast na rysunku 9 widzimy zupełnie coś innego, stabilność układu objawia się jego regulanością. Wartości powtarzają się w stałych interwałach. Podobne zjawisko ma miejsce na rysnku 8 oraz rysunku 6, gdzie to stabilność na stałych interwałach jest widoczna, ale dopiero po pewnej liczbie iteracji.
Podobnie jak w zadaniu 5 niestabilność pojawiała się z powodu zbyt małej precyzji arytmetyki $Float64$, z każdą iteracyjną konieczna dokładność wzrastała, co musiało skończyć się tym, że po pewnej liczbie iteracji, arytmetyka $Float64$ stawała sie za mało dokładna (podbonie jak w zadaniu 5).