OBLICZENIA NAUKOWE – SPRAWOZDANIE 2

autor: Jan Sieradzki
nr indeksu: 236441

Zadanie 1

I. Krótki opis problemu :

Opisać jaki wpływ na wyniki programu realizujących obliczenie iloczynu skalarnego dwóch wektorów -

x = [2.718281828, −3.141592654, 1.414213562, 0.5772156649, 0.3010299957]
y = [1486.2497, 878366.9879, −22.37492, 4773714.647, 0.000185049].

- mają niewielkie zmiany danych (usunięcie ostatniej 9 z x4 i ostatniej 7 z x5) -

x = [2.718281828, −3.141592654, 1.414213562, 0.577215664, 0.301029995]
y = [1486.2497, 878366.9879, −22.37492, 4773714.647, 0.000185049].

II. Rozwiązanie :

Dodaję iteracyjnie odpowiadające sobie wartośći wektorów i tworzę małe sumy,
następnie te sumy dodaję na cztery sposoby:
A. Po kolei je dodaję.
B. Dodaję je od tyłu.
C. Ustawiam sumy od najmniejszego do największego i je kolejno dodaję.
D. Ustawiam sumy od największego do najmniejszego i je kolejno dodaję.

Po wszystkim wyświetlam wyniki dla każdego sposobu rozwiązania.
Powtarzam proces dla wektorów ze zmianami i bez nich.

III. Wyniki oraz ich interpretacja :

FLOAT32

Sposób	A	B	C	D
przed modyfikacją	-0.4999443	-0.4543457	-0.5	-0.5
po modyfikacji	-0.4999443	-0.4543457	-0.5	-0.5

FLOAT64

Sposób	A	B	C	D
przed modyfikacją	1.0251881368296672e-10	-1.5643308870494366e-10	0.0	0.0
po modyfikacji	-0.004296342739891585	-0.004296342998713953	-0.004296342842280865	-0.004296342842280865

Jak widać, modyfikacja nie wpłyneła na wyniki Float32, natomiast odmieniła rezultaty we wszystkich algorytmach Float64.

IV. Wnioski :

Usunięcie odległych cyfr w arytmetyce Float32 nie zmieniło wyników, co jest spowodowane tym, iż arytmetyka single, pozwalająca na dokładny zapis 7 cyfr po przecinku w systemie dziesiętnym, nie jest wystarczajco dokładna.

Odmienną rzecz obserwujemy we Float64, która jest już dokładniejsza i modyfikacje powodują, że we wszystkich czterech algorytmach dostajemy podobne, inne niż wcześniej, wyniki. Wnioskuję z tego, iż zmiana dziesiątej liczby po przecinku w arytmetyce Float64, istotnie wpływa na wyniki obliczeń. Zadanie nie jest dobrze uwarunkowane.

Zadanie 2

I. Krótki opis problemu :

Narysować wykres funkcji $f(x)=e^x*\ln(1+e^{-x})$ w co najmniej dwóch dowolnych programach do wizualizacji. Następnie policzyć granicę funkcji $\lim_{x\to\infty} f(x)$ Porównać wykres funkcji z policzoną granicą. Wyjaśnić zjawisko.

II. Rozwiązanie :

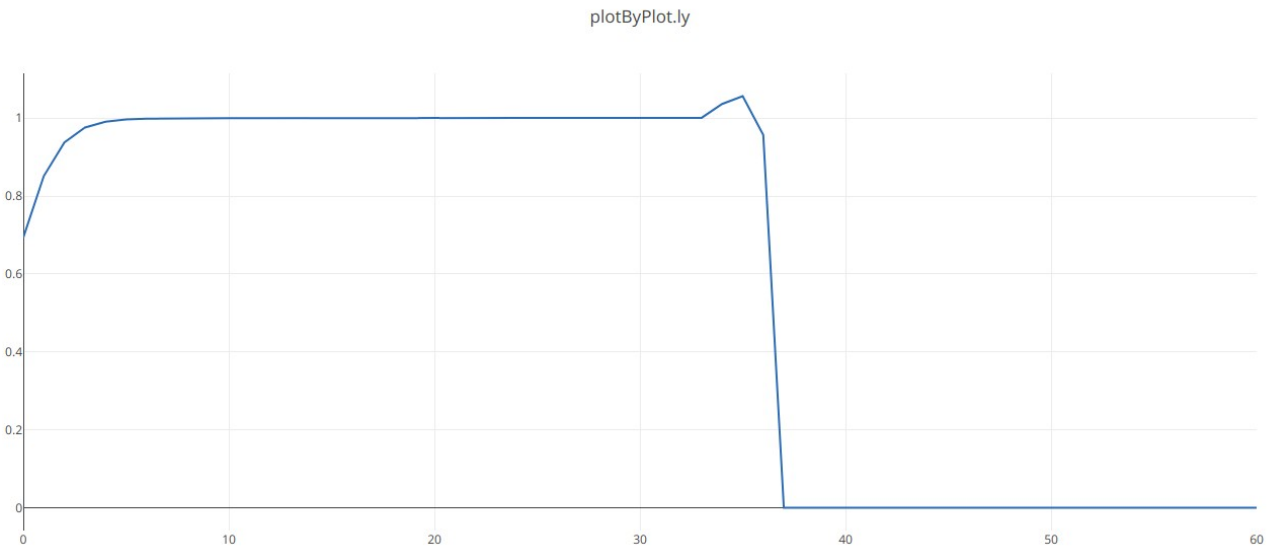
Najpierw policzę granicę f(x)

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

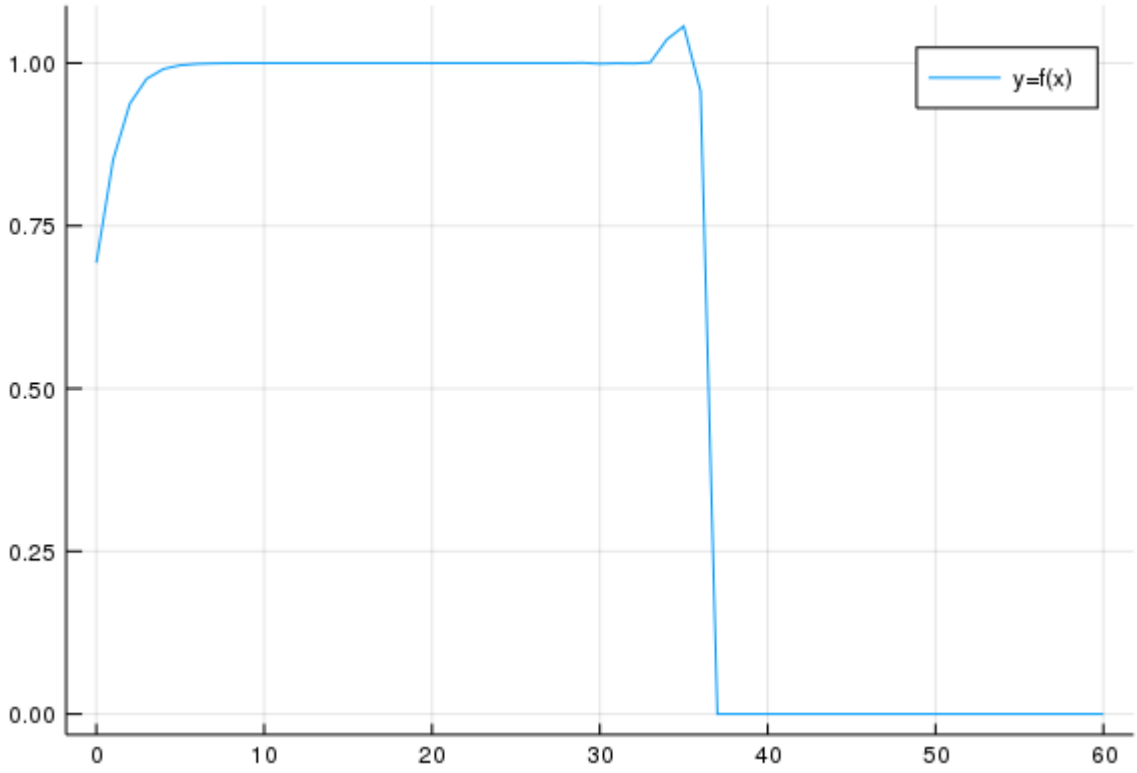
Dwa wykresy wygenerowałem za pomocą biblioteki Plot w języku Julia i strony plot.ly. Zakres argumentów to 0-60 dla funkcji f(x).

III. Wyniki oraz ich interpretacja :

Wykres wygenerowany przez stronę plot.ly:



Wykres wygenerowany przez bibliotekę Plots w języku Julia:



IV. Wnioski :

Na wykresach widać, iż funkcja w okolicach x=35 ma wysokie odchylenia od wcześniejszych wartości i nagle zaczyna przyjmować wartość 0, pomimo tego, że funkcja f(x) powinna zbiegać do 1. Nieprawidłowości na wykresie spowodowane są operacjami e^x , która , dla większych x, jest liczbą zbyt dużą, aby komputer mógł sobie z nią porawdzić.

Zadanie 3

I. Krótki opis problemu :

Należy rozwiązać układ równań Ax = b, używając algorytmu eliminacji Gaussa oraz iloczynu $x = A^{-1}b$, policzyć błędy względne. Zadanie wykonać dla dwóch macierzy A (pierwsza będzie macierzą Hilberta dla n > 1, druga macierzą losową dla n = 5,10,20 z rosnącymi wskaźnikami uwarunkowania : c = $1,10^1 ,10^3 ,10^7 ,10^{12} ,10^{16}$), i b, które należy wyliczyć za pomocą dokładnego $X = (1,1,1 \dots 1)^T$.

II. Rozwiązanie :

Użyłem funkcji hib(n) oraz matcond(n,c) do generowania macierzy A, metodę Gaussa wyliczałem za pomocą (x=A/b), a macierz odwrotną otrzymywałem funkcją inv(A). Błędy względne wyliczałem ze wzoru $\frac{\|\tilde{x}-x\|}{\|x\|}$.

III. Wyniki oraz ich interpretacja :

Tabela dla macierzy Hilberta:

n	COND	GAUSS (błąd względny)	INVERSION (błąd względny)
2	19.28147006790397	5.661048867003676e-16	1.1240151438116956e-15
3	524.0567775860644	8.022593772267726e-15	9.825526038180824e-15
4	15513.73873892924	4.4515459601812086e-13	2.950477637286781e-13
5	476607.25024259434	1.6828426299227195e-12	8.500055777753297e-12
6	1.4951058642254665e7	2.618913302311624e-10	3.3474135070361745e-10
7	4.75367356583129e8	1.2606867224171548e-8	5.163959183577243e-9
8	1.5257575538060041e10	1.026543065687064e-7	2.698715074276819e-7
9	4.931537564468762e11	4.83235712050215e-6	9.175846868614517e-6
10	1.6024416992541715e13	0.0006329153722983848	0.00045521422517408853
11	5.222677939280335e14	0.011543958596122112	0.00804446677343116
12	1.7514731907091464e16	0.2975640310734787	0.34392937091205217
13	3.344143497338461e18	2.375017867706776	5.585796893150773
14	6.200786263161444e17	5.281004646755168	4.800641929017436
15	3.674392953467974e17	1.177294734836712	4.8273577212576475
16	7.865467778431645e17	20.564655823804095	31.736467496266126
17	1.263684342666052e18	17.742214635179074	15.910335962604142
18	2.2446309929189128e18	4.2764564411159425	6.281223433472033
19	6.471953976541591e18	22.119937292648906	22.92561401563632
20	1.3553657908688225e18	14.930069669294001	21.53949860251383

Tabela dla macierzy losowej:

n	COND	GAUSS (błąd względny)	INV(błąd względny)
5	1.000000000000001	1.2161883888976234e-16	1.7901808365247238e-16
5	9.999999999999993	1.2161883888976234e-16	1.7901808365247238e-16
5	1000.0000000000335	2.8324398327409688e-14	3.0269981866124735e-14
5	9.999999999481412e6	6.20113040736834e-10	5.537752609916851e-10
5	1.0001097350542089e12	3.40470356634444e-5	3.2178120030661425e-5
5	2.2571973897822424e16	0.1823281302482371	0.185449744719112
10	1.0000000000000007	3.6821932062951477e-16	2.0471501066083614e-16
10	10.000000000000002	2.895107444979072e-16	2.742048596011341e-16
10	999.9999999999536	3.2274968148892794e-14	3.249964338447743e-14
10	9.999999995976405e6	1.3189620987453363e-10	7.908344474213749e-11
10	1.000015333598099e12	3.223499966946541e-5	2.9806855315317547e-5
10	8.631200072234053e15	0.35851581680816896	0.3914825733365866
20	1.000000000000001	5.433291284991773e-16	4.2638924323926724e-16
20	10.000000000000004	4.6510262453954645e-16	5.606351459942525e-16
20	999.9999999999453	1.8382423646844778e-14	2.4382883775710686e-14
20	9.999999997925162e6	2.8779800448015716e-11	4.230079314947545e-11
20	1.0000493888800762e12	8.129107860562902e-6	5.4215986473478304e-6
20	9.743444039037246e15	0.15857741097400516	0.14649218729167776

oznaczenia w tabelach :

n	– rozmiar macierzy
COND	– wskaźnik uwarunkowania
Gauss(błąd względny)	– błąd względny metody Gaussa
inv(błąd względny)	– błąd względny metody z macierzą odwrotną

Można zaobserwować, iż błędy względne w macierzy Hilerta są dużo większe od tych w macierzy losowej. Oprócz tego metoda Gaussa daje mniejsze błędy niż metoda z odwróceniem macierzy. Kolejnym ważnym faktem jest, iż w macierzy losowej, im większy wskaźnik uwarunkowania, tym większy błąd.

IV. Wnioski :

Macierz Hilberta (jej komórki wylicza się za pomocą wzoru : $h_{i,j}=\frac{1}{i+j-1}$) jest bardzo źle uwarunkowana, czego skutkiem są bardzo wysokie błędy względne dla większych n (im wyższe n, tym błędy względne są większe). Spowodowane jest to sposobem wyliczania macierzy Hilberta, który zwraca wiele wartości, których nie da się zapisać w sposób skończony, z czego wynikają późniejsze błędy.

Dowodem na to, jakie faktycznie znaczenie ma wskaźnik uwarunkowania jest macierz losowa, której błędy są tym większe, im większy jest wskaźnik. Co więcej metoda Gaussa okazuje się dokładniejsza obliczeniowo dla komputera, niż metoda obliczeniowa z odwróceniem macierzy.

Zadanie 4

I. Krótki opis problemu :

Mamy wielomian Wilkinsona p:

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)

a) Za pomocą funkcji roots (pakiet Polynomials) do obliczenia 20 zer wielomianu P w postaci naturalnej. Sprawdzić obliczone pierwiastki z_k, 1 <= k <= 20, obliczając $|P(z_k)|$, $|p(z_k)|$ i $|z_k - k|$. Wyjaśnić rozbieżności. Zapoznać się z funkcjami: Poly, poly, polyval (z pakietu Polynomials).

b) Powtórzyć eksperyment Wilkinsona, tj. zmienić współczynnik −210 na −210−2 −23 . Wyjaśnić zjawisko

II. Rozwiązanie :

Do rozwiązania użyłem następujących funkcji z biblioteki Polynomials:
Poly(x) – generuje wielomian za pomocą podanych współczynników
poly(x) – generuje wielomian na podstawie podanych pierwiastków wielomianu
polyval(p,x) – liczy wartość wielomianu p dla argumentu x

Algorytm działa w jednej funkcji, która jest uruchomiana dwa razy, dla współczynników z podpunktów a) i b).

III. Wyniki oraz ich interpretacja :

Tabela wielomanu przed modyfikacją

Actual roots	Roots found by roots()	P(z) result	p(z) result	z-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

Tabela wielomanu po modyfikacji

Roots z found by roots()	P(z) result	p(z) result
1.000000000000162 + 0.0im	19456.0	19456.0
2.0000000000076628 + 0.0im	54272.0	70656.0
2.9999999989085295 + 0.0im	782848.0	875008.0
4.000000027770936 + 0.0im	2.555392e6	2.817536e6
4.999999764014904 + 0.0im	6.717952e6	7.485952e6
5.9999998706066755 + 0.0im	1.9252736e7	1.7925632e7
7.000014538322563 + 0.0im	1.2139264e8	1.18582784e8
7.999908286011571 + 0.0im	3.96893184e8	3.92699392e8
9.000098361279779 + 0.0im	7.4291456e8	4.60736e8
10.001944800435727 + 0.0im	6.253824e8	2.56111616e9
10.985205958817525 + 0.0im	9.511332864e9	1.828041984e10
12.07012341290856 + 0.0im	2.434616064e10	1.10285019136e11
12.835059644003604 + 0.0im	8.770431488e10	3.4704067328e11
14.4431091456307 - 0.3130586828605012im	4.1416595000481635e11	3.066962265995617e12
14.4431091456307 + 0.3130586828605012im	4.1416595000481635e11	3.066962265995617e12
16.540360760741656 - 0.3907143161956746im	3.3752053852365005e12	3.677478938782452e13
16.540360760741656 + 0.3907143161956746im	3.3752053852365005e12	3.677478938782452e13
18.217849595179974 + 0.0im	1.277755372544e13	2.20104343106048e14
18.91267187876307 + 0.0im	2.3017536954368e13	4.451884076032e14
20.0101840504568 + 0.0im	5.2019032684032e13	1.2844306462848e15

Objaśnienie tabeli:

Roots z found by roots() - Pierwiastki „z” wielomianu P(x) znalezione przez funkcję roots(P(x))
P(z) result – wartość jaką zwróci wielomian utworzony przez funkcję Poly(x)
, argumentem jest piewiastek „z” znaleziony przez roots().
p(z) result – wartość jaką zwróci wielomian utworzony przez funkcję poly(x)
, argumentem jest piewiastek „z” znaleziony przez roots().
z-k – różnica między pierwiastkiem „z” a faktycznym pierwiastkiem wielomianu k

IV. Wnioski :

Ważnym wnioskiem jest, iż program nie liczy dokładnie miejsc zerowych wielomianu, a oprócz tego wartości zwracane przez P(z) i p(z) się różnią, co nasuwa myśl iż są niedokładne, co jest spowodowane niedostateczną precyzją (podane współczynniki mają nawet 20 cyfr znaczących, a Float64 ma od 15-17 cyfr znaczących). p(z) jest dokładniejsza, gdyż zwraca mniejsze wartości (bliższe 0).

Interesująca rzecz stała się po modyfikacji współczynnika, funkcja szukając miejsc zerowych zwórciła liczby zespolone. Udowadnia to, że zadanie znalezienia miejsc zerowych w wielomianie Wilinsona jest źle uwarunkowane, gdyż nawet niewielkie zaburzenie współczynników powoduje duże zmiany.

Zadanie 5

I. Krótki opis problemu :

Rozważmy równanie rekurencyjne (model logistyczny, model wzrostu populacji):
$p_{n+1} := p_n + rp_n(1 - p_n)$, dla n = 0, 1, . . . ,

r – pewna dana stała;
$r(1-p_n)$ – czynnik wzrostu populacji;
p_n – wielkość populacji stanowiąca procent maksymalnej wielkości populacji dla danego stanu środowiska.

1. Dla danych p0 = 0.01 i r = 3 wykonać 40 iteracji wyrażenia , a następnie wykonać ponownie 40 iteracji wyrażenia z niewielką modyfikacją tj. wykonać 10 iteracji, zatrzymać, zastosować obcięcie wyniku odrzucając cyfry po trzecim miejscu po przecinku (daje to liczbę 0.722) i kontynuować dalej obliczenia (do 40- stej iteracji) tak, jak gdyby był to ostatni wynik na wyjściu. Porównać otrzymane wyniki. Obliczenia wykonać w arytmetyce Float32 (w języku Julia).

2. Dla danych p0 = 0.01 i r = 3 wykonać 40 iteracji wyrażenia w arytmetyce Float32 i Float64 (w języku Julia). Porównać otrzymane wyniki.

II. Rozwiązanie :

Utworzyłem funkcję przyjmującą za argumenty wartość początkową rekurencji, stałą rekurencji i iterator, który pozwala wykonywać kolejne iteracje. Funkcję liczę dla Float32 i Float64, w sposób, jaki nakazuje zadanie.

III. Wyniki oraz ich interpretacja :

Wyniki 40 iteracji bez ucięcia i po ucięciu na 10 pozycji, arytmetyka Float32

N	**Wartość X_n przed ucięciem**	**Wartość X_n po ucięciu (N=10)**
0	0.01	0.01
1	0.0397	0.0397
2	0.15407173	0.15407173
3	0.5450726	0.5450726
4	1.2889781	1.2889781
5	0.1715188	0.1715188
6	0.5978191	0.5978191
7	1.3191134	1.3191134
8	0.056273222	0.056273222
9	0.21559286	0.21559286
10	0.7229306	**0.722**
11	1.3238364	1.3241479
12	0.037716985	0.036488414
13	0.14660022	0.14195944
14	0.521926	0.50738037
15	1.2704837	1.2572169
16	0.2395482	0.28708452
17	0.7860428	0.9010855
18	1.2905813	1.1684768
19	0.16552472	0.577893
20	0.5799036	1.3096911
21	1.3107498	0.09289217
22	0.088804245	0.34568182
23	0.3315584	1.0242395
24	0.9964407	0.94975823
25	1.0070806	1.0929108
26	0.9856885	0.7882812
27	1.0280086	1.2889631
28	0.9416294	0.17157483
29	1.1065198	0.59798557
30	0.7529209	1.3191822
31	1.3110139	0.05600393
32	0.0877831	0.21460639
33	0.3280148	0.7202578
34	0.9892781	1.3247173
35	1.021099	0.034241438
36	0.95646656	0.13344833
37	1.0813814	0.48036796
38	0.81736827	1.2292118
39	1.2652004	0.3839622
40	0.25860548	1.093568

Wyniki 40 iteracji w arytmetykach Float32 i Float64

N	**Wartość X_n FLOAT32**	**Wartość X_n FLOAT64**
0	0.01	0.01
1	0.0397	0.0397
2	0.15407173	0.15407173000000002
3	0.5450726	0.5450726260444213
4	1.2889781	1.2889780011888006
5	0.1715188	0.17151914210917552
6	0.5978191	0.5978201201070994
7	1.3191134	1.3191137924137974
8	0.056273222	0.056271577646256565
9	0.21559286	0.21558683923263022
10	0.7229306	0.722914301179573
11	1.3238364	1.3238419441684408
12	0.037716985	0.03769529725473175
13	0.14660022	0.14651838271355924
14	0.521926	0.521670621435246
15	1.2704837	1.2702617739350768
16	0.2395482	0.24035217277824272
17	0.7860428	0.7881011902353041
18	1.2905813	1.2890943027903075
19	0.16552472	0.17108484670194324
20	0.5799036	0.5965293124946907
21	1.3107498	1.3185755879825978
22	0.088804245	0.058377608259430724
23	0.3315584	0.22328659759944824
24	0.9964407	0.7435756763951792
25	1.0070806	1.315588346001072
26	0.9856885	0.07003529560277899
27	1.0280086	0.26542635452061003
28	0.9416294	0.8503519690601384
29	1.1065198	1.2321124623871897
30	0.7529209	0.37414648963928676
31	1.3110139	1.0766291714289444
32	0.0877831	0.8291255674004515
33	0.3280148	1.2541546500504441
34	0.9892781	0.29790694147232066
35	1.021099	0.9253821285571046
36	0.95646656	1.1325322626697856
37	1.0813814	0.6822410727153098
38	0.81736827	1.3326056469620293
39	1.2652004	0.0029091569028512065
40	0.25860548	0.011611238029748606

N – oznacza numer iteracji

IV. Wnioski :

Główną obserwacją jest, iż komputer nie radzi sobie z dokładnym obliczeniem wartości tego wzoru. Mnożenie powoduje bardzo szybki przyrost cyfr po przecinku, a odejmowanie małych, przybliżanych cyfr powoduje wielką niedokładność obliczeniową. Cały czas ma miejsce utrata cyfr znaczących. Różnice pomiędzy ostatnimi iteracjami Float32, a Float64 są około 25-krotne. Analogicznie, obcięcie w 10 kroku do trzech cyfr spowodowało znaczne odchylenia wynikowe, w 40 iteracji otrzymaliśmy wynik 4 krotnie większy niż bez obcięcia.

Zadanie 6

I. Krótki opis problemu :

Mamy równanie rekurencyjne : $x_{n+1} := x^2_n + c$ dla n = 0, 1, . . . , gdzie c jest pewną stałą Przeprowadzić następujące eksperymenty. Dla 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 $x_0 = 0.75$
7. $c = -1$ i $x_0 = 0.25$

wykonać, w języku Julia w arytmetyce Float64, 40 iteracji wyrażenia. Zaobserwować zachowanie generowanych ciągów.

II. Rozwiązanie :

Utworzyłem funkcję przyjmującą za argumenty wartość początkową rekurencji, stałą rekurencji i iterator, który pozwala wykonywać kolejne iteracje. Funkcję liczę dla wszystkich danych eksperymentalnych. Całość w arytmetyce Float64.

III. Wyniki oraz ich interpretacja :

Wyniki 40 iteracji w arytmetykach Float32 i Float64

N	**Wartość X_n (1)**	**Wartość X_n (2)**	**Wartość X_n (3)**	**Wartość X_n (4)**
0	1.0	2.0	1.99999999999999	1.0
1	-1.0	2.0	1.99999999999996	0.0
2	-1.0	2.0	1.9999999999998401	-1.0
3	-1.0	2.0	1.9999999999993605	0.0
4	-1.0	2.0	1.999999999997442	-1.0
5	-1.0	2.0	1.9999999999897682	0.0
6	-1.0	2.0	1.9999999999590727	-1.0
7	-1.0	2.0	1.999999999836291	0.0
8	-1.0	2.0	1.9999999993451638	-1.0
9	-1.0	2.0	1.9999999973806553	0.0
10	-1.0	2.0	1.999999989522621	-1.0
11	-1.0	2.0	1.9999999580904841	0.0
12	-1.0	2.0	1.9999998323619383	-1.0
13	-1.0	2.0	1.9999993294477814	0.0
14	-1.0	2.0	1.9999973177915749	-1.0
15	-1.0	2.0	1.9999892711734937	0.0
16	-1.0	2.0	1.9999570848090826	-1.0
17	-1.0	2.0	1.999828341078044	0.0
18	-1.0	2.0	1.9993133937789613	-1.0
19	-1.0	2.0	1.9972540465439481	0.0
20	-1.0	2.0	1.9890237264361752	-1.0
21	-1.0	2.0	1.9562153843260486	0.0
22	-1.0	2.0	1.82677862987391	-1.0
23	-1.0	2.0	1.3371201625639997	0.0
24	-1.0	2.0	-0.21210967086482313	-1.0
25	-1.0	2.0	-1.9550094875256163	0.0
26	-1.0	2.0	1.822062096315173	-1.0
27	-1.0	2.0	1.319910282828443	0.0
28	-1.0	2.0	-0.2578368452837396	-1.0
29	-1.0	2.0	-1.9335201612141288	0.0
30	-1.0	2.0	1.7385002138215109	-1.0
31	-1.0	2.0	1.0223829934574389	0.0
32	-1.0	2.0	-0.9547330146890065	-1.0
33	-1.0	2.0	-1.0884848706628412	0.0
34	-1.0	2.0	-0.8152006863380978	-1.0
35	-1.0	2.0	-1.3354478409938944	0.0
36	-1.0	2.0	-0.21657906398474625	-1.0
37	-1.0	2.0	-1.953093509043491	0.0
38	-1.0	2.0	1.8145742550678174	-1.0
39	-1.0	2.0	1.2926797271549244	0.0
40	-1.0	2.0	-0.3289791230026702	-1.0

N – oznacza numer iteracji

Wyniki 40 iteracji w arytmetykach Float32 i Float64

N	**Wartość X_n (5)**	**Wartość X_n (6)**	**Wartość X_n (7)**
0	1.0	0.75	0.25
1	0.0	-0.4375	-0.9375
2	-1.0	-0.80859375	-0.12109375
3	0.0	-0.3461761474609375	-0.9853363037109375
4	-1.0	-0.8801620749291033	-0.029112368589267135
5	0.0	-0.2253147218564956	-0.9991524699951226
6	-1.0	-0.9492332761147301	-0.0016943417026455965
7	0.0	-0.0989561875164966	-0.9999971292061947
8	-1.0	-0.9902076729521999	-5.741579369278327e-6
9	0.0	-0.01948876442658909	-0.9999999999670343
10	-1.0	-0.999620188061125	-6.593148249578462e-11
11	0.0	-0.0007594796206411569	-1.0
12	-1.0	-0.9999994231907058	0.0
13	0.0	-1.1536182557003727e-6	-1.0
14	-1.0	-0.9999999999986692	0.0
15	0.0	-2.6616486792363503e-12	-1.0
16	-1.0	-1.0	0.0
17	0.0	0.0	-1.0
18	-1.0	-1.0	0.0
19	0.0	0.0	-1.0
20	-1.0	-1.0	0.0
21	0.0	0.0	-1.0
22	-1.0	-1.0	0.0
23	0.0	0.0	-1.0
24	-1.0	-1.0	0.0
25	0.0	0.0	-1.0
26	-1.0	-1.0	0.0
27	0.0	0.0	-1.0
28	-1.0	-1.0	0.0
29	0.0	0.0	-1.0
30	-1.0	-1.0	0.0
31	0.0	0.0	-1.0
32	-1.0	-1.0	0.0
33	0.0	0.0	-1.0
34	-1.0	-1.0	0.0
35	0.0	0.0	-1.0
36	-1.0	-1.0	0.0
37	0.0	0.0	-1.0
38	-1.0	-1.0	0.0
39	0.0	0.0	-1.0
40	-1.0	-1.0	0.0

N – oznacza numer iteracji

IV. Wnioski :

Po eksperymentach mogę stwierdzić, że ciągi 1,2,4,5 są stabilne, a 3,6,7 są niestabilne. Ciągi niestabilne, czyli kumulujące się błędy w kolejnych iteracjach powodują poważną utratę dokładności obliczeń. Ciągi niestabilne są powodowane przez niezdolność do dokładnego obliczenia wyrażenia przez arytmetykę Float, która w konsekwencji dokonuje przybliżenia, które się nawarstwia.