OBLICZENIA NAUKOWE

Lista 2

NIEBIESKI-KAPTUREK

INF-FIFA-PPT-TEAM®

LISTOPAD 2018

1. Zadanie 1 - "Iloczyn skalarny dwóch wektorów"

1.1. Opis problemu

Na poprzedniej liście mieliśmy napisać program w języku Julia realizujący następujący eksperyment obliczania 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].

Zaimplementować mamy cztery algorytmy i policzyć sumę na cztery sposoby:

(a) "w przód" $\sum_{i=1}^{n} x_i\, y_i$, tj. algorytm

S := 0
for *i* := 1 to *n* **do**
    S := *S* + $x_i * y_i$
end for

(b) "w tył" $\sum_{i=n}^{1} x_i\, y_i$, tj. algorytm

S := 0
for *i* := n **downto** *1* **do**
    S := *S* + $x_i * y_i$
end for

(c) od największego do najmniejszego (dodać dodatnie liczby w porządku od największego do najmniejszego, dodaj ujemne liczby w porządku od najmniejszego do największego, a następnie daj do siebie obliczone sumy częściowe),

(d) od najmniejszego do największego (przeciwnie do metody (c)).

Teraz mamy powtórzyć to zadanie, przy założeniu, że usuwamy ostatnią 9 z x_4 i ostatnią 7 z x_5, oraz sprawdzić, jak zmieniły się wyniki.

1.2. Rozwiązanie

Cała implementacja rozwiązania pozostaje niezmieniona, w stosunku do zadania piątego z poprzedniej listy, oprócz fragmentu, gdzie tworzymy wektory.
Tam usuwamy ostatnią cyfrę z x_4 i x_5, co wynika z treści polecenia zadania.

```
# Tworzenie wektorów
x = [2.718281828, -3.141592654, 1.414213562, 0.577215664, 0.301029995]
y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]
```

Listing1: Jedyna zmiana w kodzie w stosunku do zad5.jl

1.3. Wyniki

```
julia> zad5()
Dokładny wynik: S = -1.00657107000000e-11
Algorytm A:
 pojedyncza precyzja: S = -0.4999443
 podwójna precyzja: S = 1.0251881368296672e-10
Algorytm B:
 pojedyncza precyzja: S = -0.4543457
 podwójna precyzja: S = -1.5643308870494366e-10
Algorytm C:
 pojedyncza precyzja: S = -0.5
 podwójna precyzja: S = 0.0
Algorytm D:
 pojedyncza precyzja: S = -0.5
 podwójna precyzja: S = 0.0
```

Listing2: Wyniki z zad5.jl (poprzednia lista)

```
julia> zad1()
Algorytm A:
 pojedyncza precyzja: S = -0.4999443
 podwójna precyzja: S = -0.004296342739891585
Algorytm B:
 pojedyncza precyzja: S = -0.4543457
 podwójna precyzja: S = -0.004296342998713953
Algorytm C:
 pojedyncza precyzja: S = -0.5
 podwójna precyzja: S = -0.004296342842280865
Algorytm D:
 pojedyncza precyzja: S = -0.5
 podwójna precyzja: S = -0.004296342842280865
```

Listing3: Wyniki z zad1.jl, pogrubione zostały wyniki różne od tych uzyskanych w listingu2

1.4. Wnioski

W przypadku arytmetyki Float32 usunięcie ostatnich cyfr w x_4 i x_5 nie miało wpływu na zmianę wyniku, w stosunku do rozwiązania z poprzedniej listy. Spowodowane jest to tym, że ta arytmetyka nie jest wystarczająco dokładna. Liczby pojedynczej precyzji *single*, pozwalają na zapis siedmiu cyfr znaczących w systemie dziesiętnym. Modyfikacja składowych wektora dotyczyła cyfr na dziesiątych pozycjach, więc automatycznie nic to nie zmieniło.

W przypadku arytmetyki Float64 ta kosmetyczna zmiana przyczyniła się do zmiany ostatecznego wyniku w każdym z czterech używanych algorytmów. Otrzymane dane są do siebie bardzo podobne i wynoszą: S ≈ -0.004296342..., a z ostatnich dwóch są dokładnie sobie równe. Wynika z tego, że dodanie cyfry na dziesiątej pozycji w dwóch składowych wektora x, powoduje mocne rozbieżności w wynikach zależnych od wybranego algorytmu.

2. Zadanie 2 - "Wykres funkcji logarytmicznej"

2.1. Opis problemu

W zadaniu drugim musimy narysować wykres funkcji $f(x) = e^x \ln(1 + e^{-x})$, a następnie policzyć granicę funkcji $\lim_{x\to\infty} f(x)$, porównać wykres funkcji z policzoną granicą oraz wyjaśnić zjawisko.

2.2. Rozwiązanie

Zacznijmy od policzenia granicy funkcji f(x):

$$\lim_{x\to\infty} e^x \ln(1 + e^{-x}) = \lim_{x\to\infty} \frac{\ln(1+e^{-x})}{e^{-x}} \overset{Reguła\ de\ l'Hospitala}{\Longleftrightarrow} \lim_{x\to\infty} \frac{(\ln(1+e^{-x}))'}{(e^{-x})'}$$

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

Zatem:

$$\lim_{x\to\infty} f(x) = 1$$

```
# Stworzenie logarytmu o podstawie e
function ln(x) return log(exp(1), x) end
# Stworzenie funkcji z zadania
function f(x) return (exp(1)^x) * ln(1 + exp(-x)) end
# Wyświetlenie wyników
function zad2() for i=0:50 println("$i, ", f(i)) end end
```

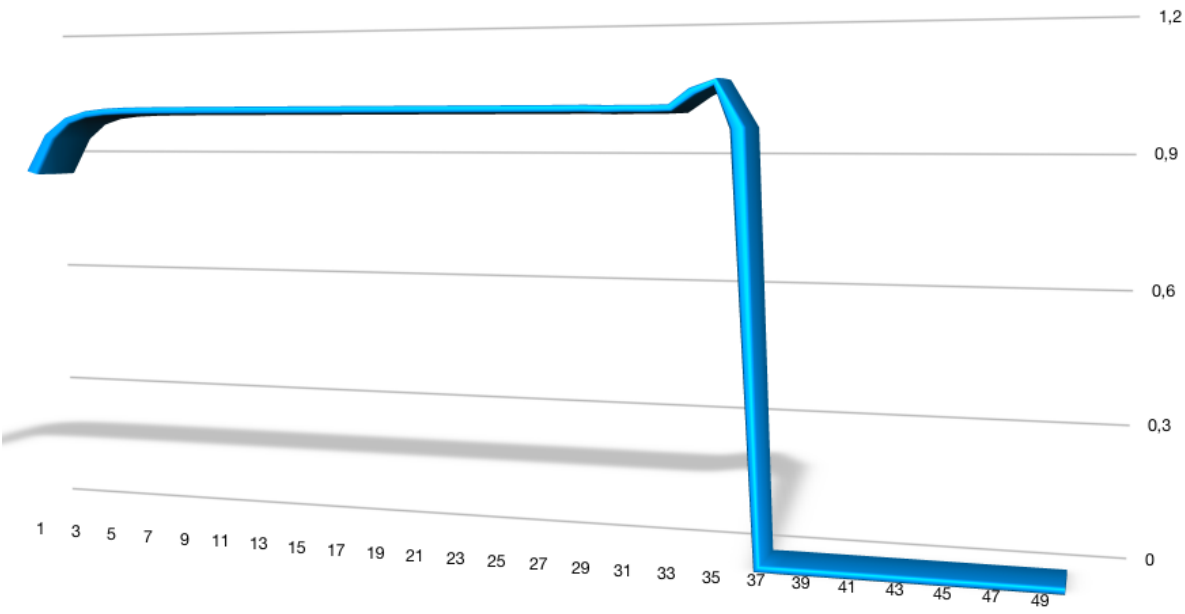
Listing4: Kod programu zad2.jl

Obraz1: Modelka z wilkiem

2.3. Wyniki

Wyniki przedstawiam w postaci dwóch wykresów, wygenerowanych odpowiednio w programie *MS Excel* oraz *Numbers*.

Rysunek1: Wykres funkcji f(x) wykonany w Excelu.



Rysunek2: Wykres funkcji f(x) wykonany w Numbers.

2.4. Wnioski

Na podanych wykresach widać, że dla argumentu $x = 33$, wartość przekracza 1.0, co jest już odchyleniem. Następnie aż $f(35) \approx 1.0565$ funkcja rośnie, dla $f(36)$ przyjmuje ona wartość około 0.957, a każdy następny wynik wynosi 0.0. Spowodowane jest to prawdopodobnie podnoszeniem do coraz wyższych potęg liczby e.

3. Zadanie 3 - "Układ równań liniowych"

3.1. Opis problemu

W zadaniu trzecim mamy do rozwiązania układ równań liniowych $\boldsymbol{Ax} = \boldsymbol{b}$
za pomocą dwóch algorytmów: eliminacji Gaussa ($x = A/b$)
oraz inwersji ($x = A^{-1}b$ ($x = inv(A) * b$)).
Eksperymenty mamy przeprowadzić dla macierzy Hilberta H_n
z rosnącym stopniem $n > 1$ oraz dla macierzy losowej R_n, $n = 5, 10, 20$
z rosnącym wskaźnikiem uwarunkowania $c = 10^0, 10^1, 10^3, 10^7, 10^{12}, 10^{16}$.
Musimy policzyć błędy względne i porównać z dokładnym rozwiązaniem.

3.2. Rozwiązanie

Do wygenerowania macierzy Hilberta n-stopnia, użyto funkcji *hilb(n)*,
co było podane w treści polecenia zadania, natomiast do wygenerowania losowej macierzy stopnia n i wskaźnika uwarunkowania użyto funkcji *matcond(n, c)*,
co również było podane. Błędy względne zostały wyliczone przy pomocy normy wektora: $\partial = \frac{\|\tilde{x}-x\|}{\|x\|}$.

3.3. Wyniki

Stopień	Rząd	Wskaźnik uwarunkowania	Błąd względny	
			Eliminacja Gaussa	Odwrotność macierzy A
1	1	1.0	0.0	0.0
2	2	19.28147006790397	5.661048867003676e-16	1.4043333874306803e-15
3	3	524.0567775860644	8.022593772267726e-15	0.0
4	4	15513.73873892924	4.137409622430382e-14	0.0
5	5	476607.25024259434	1.6828426299227195e-12	3.3544360584359632e-12
6	6	1.4951058642254665e7	2.618913302311624e-10	2.0163759404347654e-10
7	7	4.75367356583129e8	1.2606867224171548e-8	4.713280397232037e-9
8	8	1.5257575538060041e10	6.124089555723088e-8	3.07748390309622e-7
9	9	4.931537564468762e11	3.8751634185032475e-6	4.541268303176643e-6
10	10	1.6024416992541715e13	8.67039023709691e-5	0.0002501493411824886
11	11	5.222677939280335e14	0.00015827808158590435	0.007618304284315809
12	11	1.7514731907091464e16	0.13396208372085344	0.258994120804705
13	11	3.344143497338461e18	0.11039701117868264	5.331275639426837
14	12	6.200786263161444e17	1.4554087127659643	8.71499275104814
15	12	3.674392953467974e17	4.696668350857427	7.344641453111494
16	12	7.865467778431645e17	54.15518954564602	29.84884207073541
17	12	1.263684342666052e18	13.707236683836307	10.516942378369349
18	12	2.2446309929189128e18	10.257619124632317	24.762070989128866
19	13	6.471953976541591e18	102.15983486270827	109.94550732878284
20	13	1.3553657908688225e18	108.31777346206205	114.34403152557572

Tabela1: Wyniki z Macierzy Hilberta, otrzymane z zad3.jl

Stopień	Rząd	Wskaźnik uwarunkowania	Błąd względny	
			Eliminacja Gaussa	Odwrotność macierzy A
5	5	1.0	1.719950113979703e-16	1.4895204919483638e-16
5	5	10.0	3.2934537262255424e-16	9.930136612989092e-17
5	5	1.0e3	1.5102624605787574e-14	1.6511521094205152e-14
5	5	1.0e7	1.9988766985489306e-10	1.5010355224632225e-10
5	5	1.0e12	2.221971369993138e-5	2.3109636277558834e-5
5	4	1.0e16	0.348665346160947	0.32264773747850767
10	10	1.0	3.3306690738754696e-16	2.742048596011341e-16
10	10	10.0	2.3551386880256624e-16	2.3551386880256624e-16
10	10	1.0e3	2.1485727154431577e-14	1.9002659797115724e-14
10	10	1.0e7	2.774674323484301e-10	2.1681865365464676e-10
10	10	1.0e12	1.9706842115917344e-5	1.9279942140900783e-5
10	9	1.0e16	0.09040595149920845	0.048412291827592706
20	20	1.0	3.040470972244059e-16	2.730787568572e-16
20	20	10.0	3.2934537262255424e-16	4.0792198665315547e-16
20	20	1.0e3	6.159887137044856e-15	1.7919426092099404e-14
20	20	1.0e7	4.245028176337727e-11	5.3549083199213835e-11
20	20	1.0e12	7.458600485844564e-6	4.40397258339646e-6
20	19	1.0e16	0.0079748940288458	0.02470529422006546

Tabela2: Wyniki z losowej macierzy, otrzymane z zad3.jl

3.4. Wnioski

Macierz Hilberta jest przykładem macierzy źle, a nawet bardzo źle uwarunkowanej. Wskaźnik jej uwarunkowania już dla niewielkich stopni jest wysoki, np. dla $n = 10, cond(n) \approx 1{,}6 \times 10^{13}$, a błąd względny jest już rzędu 10^{-5}. Jej elementy wylicza się ze wzoru $h_{i_j} = \frac{1}{i+j-1}$, co już mieliśmy okazję się przekonać w poprzedniej liście, nie jest w większości możliwa do dokładnego zapisu w reprezentacji binarnej, dlatego już dla niewielkich układów równań dokładny wynik nie jest możliwy do uzyskania.

Błąd macierzy losowej jest zależny od stopnia uwarunkowania, czym wskaźnik jest większy, tym błąd jest również większy. Zauważyłem również, że dla wskaźnika uwarunkowania wynoszącego 10^{16}, gdy stopień wynosi $n, rank(n) = n - 1$. W pozostałych przypadkach rząd jest równy stopniowi macierzy.

4. Zadanie 4 - "*złośliwy wielomian*, Wilkinson"

4.1. Opis problemu

(a) Należy użyć funkcji roots (z pakietu *Polynomials*) do obliczenia 20 zer wielomianu *P* w postaci naturalnej, podanego w treści zadania oraz sprawdzić obliczone pierwiastki zk, $1 \le k \le 20$ obliczając $|P(zk)|$, $|p(zk)|$ i $|zk - k|$.

(b) Powtórzyć eksperyment Wilkinsona (zmienić współczynnik -210 na $-210 - 2^{23}$)

4.2. Rozwiązanie

Do rozwiązania tego zadania użyłem następujących funkcji z pakietu *Polynomials* języka Julia:

- *Poly([a,b,c,...])* - tworzy wielomian: $ax^0 + bx^1 + cx^2 + ..$
- *poly([a,b,c,...])* - tworzy wielomian: $(x - a)(x - b)(x - c)..$
- *polyval(P,x)* - zwraca wartość $P(x)$

Należy pamiętać, że intuicyjnie myślimy o a, jako współczynniku przy największej potędze x, dlatego używałem funkcji *reverse()*, która odwraca kolejność składników w tablicy.

4.3. Wyniki

Miejsce zerowe z	$roots(z)$	$\|P(z)\|$	$\|p(z)\|$	$\|z - k\|$
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

Tabela3: Wyniki wielomianu przed modyfikacją z zad4.jl

Miejsce zerowe z	$\lvert P(z)\rvert$	$\lvert p(z)\rvert$
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

Tabela4: Wynik wielomianu po modyfikacji z zad4.jl

4.4. Wnioski

Program nie jest w stanie zwrócić dokładnych miejsc zerowych wielomianu w większości przypadków, a nawet wyniki $P(z)$ i $p(z)$ różnią się od siebie. Spowodowane jest to niewystarczającą precyzją, gdyż podane współczynniki w zapisie dziesiętnym mają do 20 cyfr znaczących, a Float64 jest dokładny do 15-17 cyfr znaczących.

Po dokonaniu niewielkiej zmiany w punkcie b) zadania, program zwracał miejsca zerowe w postaci liczb zespolonych. Jest to niewątpliwie duża zmiana wyniku przy niewielkim zaburzeniu współczynnika, zatem zadanie znalezienia miejsc zerowych w wielomianie Wilkinsona jest źle uwarunkowane.

5. Zadanie 5

5.1. Opis problemu

W zadaniu piątym rozważamy równanie rekurencyjne:

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

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 $p_0 = 0.01$ 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 40stej iteracji) tak, jak gdyby był to ostatni wynik na wyjściu. Porównać otrzymane wyniki. Obliczenia wykonać w arytmetyce Float32 (w języku Julia).

2) Dla danych $p_0 = 0.01$ i $r = 3$ wykonać 40 iteracji wyrażenia w arytmetyce Float32 i Float64 (w języku Julia). Porównać otrzymane wyniki.

5.2. Rozwiązanie

Utworzyłem funkcje 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.

```
# Funkcja, licząca wyrażenie z zadania w arytmetyce Float32
function zad532(pn,r,i)
        pnext=(Float32)(pn+((Float32)(r*pn)*((Float32)(1.0-pn))))

# Instrukcja pomocnicza uwzględniająca to, że te liczby składają się z cyfry mniej
        if (i<10)
                println(" $i:  $pn")
                zad532(pnext,r,i+1)
        end
        if (i<41 && i>9)
                println("$i:  $pn")
                zad532(pnext,r,i+1)
        end
end
```

Listing5: Fragment kodu z zad5.jl, prezentujący funkcje, liczącą w arytmetyce Float32

Dla arytmetyki Float64 funkcja wygląda analogicznie.
W części głównej kodu następuje wywołanie tych funkcji z podanymi w zadaniu parametrami.

5.3. Wyniki

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

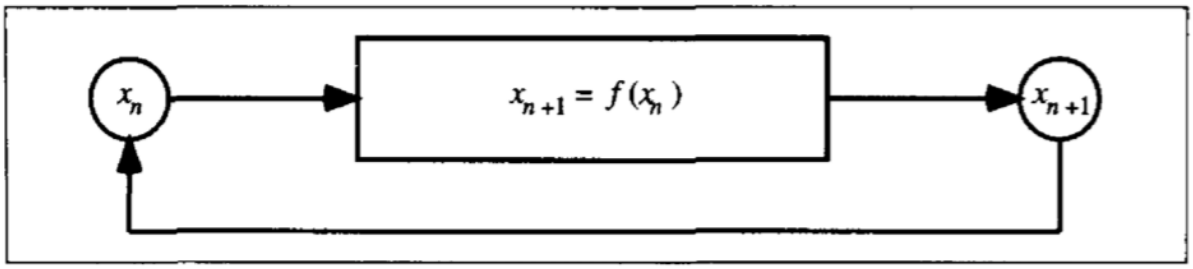
Tabela5: Wyniki z zad5.jl dla podpunktu a)

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

Tabela6: Wyniki z zad5.jl dla podpunktu b)

5.4. Wnioski

Funkcja rozważana w tym zadaniu sprowadza się do $x_{n+1} = f(x_n)$,
a dokładnie $p_{n+1} = -{p_n}^2 + (r+1)p_n$, czyli mamy do czynienia ze sprzężeniem zwrotnym o zależności prostej.



Rysunek3: Zasada działania urządzenia sprzężenia zwrotnego z zależnością prostą.[1]

Wyniki te zatem zależne są tylko od wartości otrzymanej w poprzednim kroku.

Analizując wyniki dochodzimy do wniosku, że wykonywane mnożenia w szybki sposób stają się niedokładne, co powoduje, że różnica między Float32 a Float64 w czterdziestym kroku wynosi aż 25-krotność, a obcięcie wyniku w kroku dziesiątym spowodowało czterokrotną różnicę w kroku czterdziestym.

6. Zadanie 6

6.1. Opis problemu

W ostatnim zadaniu mamy podane równanie rekurencyjne:
$x_{n+1} := {x_n}^2 + c \; dla \; n = 0, 1, ..$, gdzie c jest pewną stałą

Musimy 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.

[1] Rysunek 1.16 z książki H.O. Preitgen, H. Jürgens, D. Saupe "Granice chaosu. Fraktale, część 1, Wydawnictwo Naukowe PWN, Warszawa 1997

6.2. Rozwiązanie

Utworzyłem funkcje przyjmującą za argumenty stałą rekurencji, wartość początkową rekurencji oraz iterator, który pozwala wykonywać kolejne iteracje. Funkcję liczę w arytmetyce Float64, w sposób, jaki nakazuje zadanie.

```
# Funkcja, wyliczająca rekurencyjnie problem z zadania w arytmetyce Float64
function wylicz(c, x, i)
        if (i==0)
                println("c = $c, x0 = $x\n")
        end
        xn=(Float64)((Float64)((Float64)(x)*((Float64)(x))) + (Float64)(c))

        if (i<41)
        println("$i  $x")
        wylicz(xn, c, i+1)
 end
end
```

Listing6: Fragment kodu z zad6.jl, prezentujący funkcje, która spełnia warunki zadania

W części głównej kodu następuje wywołanie tych funkcji z podanymi w zadaniu parametrami.

```
    # Funkcja organizacyjna
    function zad6()
        wylicz(-2, 1.0, 0)
        println()
        wylicz(-2, 2.0, 0)
        println()
        wylicz(-2, 1.99999999999999, 0)
        println()
        wylicz(-1, 1.0, 0)
        println()
        wylicz(-1, -1.0, 0)
        println()
        wylicz(-1, 0.75, 0)
        println()
        wylicz(-1, 0.25, 0)
    end
```

Listing7: Fragment kodu z zad6.jl, prezentujący w jaki sposób wywołujemy funkcje

6.3. Wyniki

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

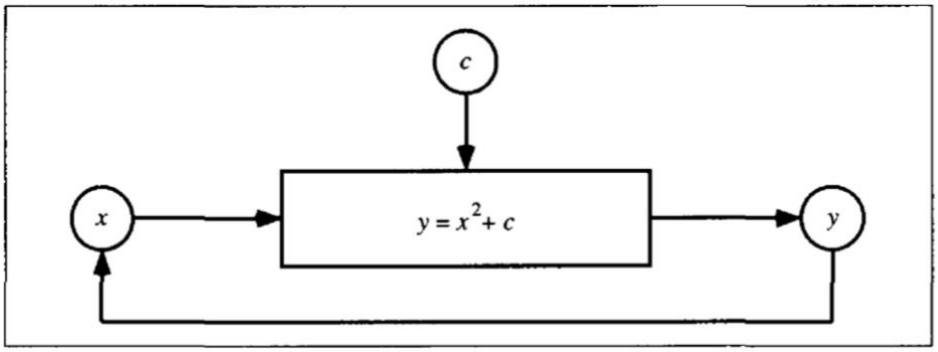
Tabela7: Wyniki z zad6.jl dla pierwszych czterech wywołań funkcji

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	-5.7415793692783 27e-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

Tabela7: Wyniki z zad6.jl dla ostatnich trzech wywołań funkcji

6.4. Wnioski

W tym zadaniu rozpatrywaliśmy iterowanie funkcji kwadratowej.



Iterowanie funkcji kwadratowej

Rysunek4: Iterowanie funkcji kwadratowej jako urządzenie sprzężenia zwrotnego. Procesor został zaprogramowany tak, aby przy danych *x* i c obliczał x^2+ c.[2]

Dla obecnych wyników można stwierdzić, że ciągi 1,2,4,5 są stabilne, a 3,6,7 kumulują błędy w kolejnych iteracjach, powodujące poważną utratę dokładności obliczeń, ponieważ podnosimy do kwadratu, liczba miejsc znaczących po przecinku, potrzebnych do przedstawienia kolejnych wyników, podwaja się w każdym kroku. Uniemożliwia to otrzymanie dokładnych wyników dla więcej niż kilku iteracji, gdyż komputery reprezentują liczby tylko z pewną określoną, a na pewno skończoną, liczbą miejsc po przecinku. Zatem konsekwentnie dokonują się przybliżenia, a błąd się nawarstwia.

[2] Rysunek 1.22 z książki H.O. Preitgen, H. Jürgens, D. Saupe "Granice chaosu. Fraktale, część 1, Wydawnictwo Naukowe PWN, Warszawa 1997