Obliczenia naukowe
Sprawozdanie – Lista 2

Bartosz Rzepkowski

8.11.2015

1. Zadanie 1

1.1 Opis problemu

Powtórzyć eksperyment z zadania 5 z listy 1 (obliczanie iloczynu skalarnego) dla wektorów, w których wprowadzono drobne zmiany (usunięcie ostatniej 9 z x_4 oraz ostatniej 7 z x_5). Nowe wektory:

x = [2.718281828, -3.141592654, 1.414213562, 0.577215664, 0.301029995]

y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]

1.2 Rozwiązanie

W celu rozwiązania zadania zastosowano te same algorytmy co w zadaniu 5 z listy 1, tj:

A) „W przód" - sumując otrzymane iloczyny po kolei od pierwszego do ostatniego

B) „ W tył" = sumują otrzymane iloczyny od ostatniego do pierwszego

C) „Od największej do najmniejszej" - obliczyć sumy częściowe – zsumować liczby dodatnie od największej do najmniejszej oraz liczby ujemne od najmniejszej do największej, po czym dodać do siebie otrzymane sumy częściowe.

D) „Od najmniejszej do największej" - sposób odwrotny do podpunktu C)

1.3 Wyniki

Typ zmiennopozycyjny	„W przód"	„W tył"	„Od największej do najmniejszej"	„Od najmniejszej do największej"
Float 32 (oryginalne dane)	-0.4999443	-0.4543457	-0.5	-0.5
Float 64 (oryginalne dane)	1.025188136829 6672e-10	-1.56433088704 94366e-10	0.0	0.0
Float 32 (nieznacznie zmienione dane)	-0.4999443	-0.4543457	-0.5	-0.5
Float 64 (nieznaczenie zmienione dane)	-0.00429634273 9891585	-0.00429634299 8713953	-0.004296342842 280865	-0.004296342842 280865

1.4 Wnioski

Możemy zauważyć, że wyniki eksperymentu w arytmetyce Float32 są takie same dla oryginalnego wektora oraz wektora z wprowadzoną niedokładnością. Jest to spowodowane tym, że wprowadzony błąd jest mniejszy niż precyzja arytmetyki, w której wykonywane są obliczenia (dla Float32 jest to 1.1920929e-7).

W arytmetyce Float64 możemy zauważyć, jak niewielka zmiana może wpłynąć na końcowy wynik. Ponieważ w sumie

$$\sum a_i b_i$$

występują $a_i b_i$ o różnych znakach, zadanie obliczenia iloczynu skalarnego staje się źle uwarunkowane (dla $a_i b_i$ o tych samych znakach jest dobrze uwarunkowane).

2. Zadanie 2

2.1 Opis problemu

Rozważamy zadanie rozwiązywania równań liniowych:

$$Ax = b$$

gdzie A jest macierzą współczynników rozmiaru n x n, a b jest wektorem prawych stron rozmiaru n. Macierz A generujemy na dwa sposoby:

a) $A = H_n$, gdzie H_n jest macierzą Hilberta stopnia n,

b) $A = R_n$, gdzie R_n jest losowa macierzą stopnia n z zadanym wskaźnikiem uwarunkowania c.

Wektor b jest postaci b = Ax dla x = (1, …, 1)^T. Dzięki temu znamy dokładną postać A oraz b.
Zadanie polega na obliczeniu $\tilde{x}$ za pomocą dwóch algorytmów:
a) eliminacji Gaussa ($\tilde{x}$ = A \ b),
b) $\tilde{x}$ = A^{-1} b
dla H_n z rosnącym n oraz R_n dla n = 5, 10, 20 oraz c = 1, 10, 10^3, 10^7, 10^{12}, 10^{16}. Porównać $\tilde{x}$ z wartością dokładną x.

2.2 Rozwiązanie

W celu utworzenia zadanych macierzy posłużyłem się funkcjami hilb() oraz matcond(). Następnie w celu obliczenia $\tilde{x}$ obliczyłem wartości wyrażeń `A \ b` (eliminacja Gaussa) oraz `inv(A) * b`. Za pomocą funkcji cond() sprawdzałem współczynnik uwarunkowania danej macierzy. Dla danych macierzy w pętlach zwiększałem odpowiednie argumenty (n oraz c).

2.3 Wyniki

Dla macierzy Hilberta H_n otrzymaliśmy następujące wyniki:

n	Wskaźnik uwarunkowania macierzy	x = A \ b	x = A^{-1} b
1	1.0	1.0	1.0
2	19.281470067903967	1.0000000000000004	0.9999999999999993
3	524.056777586062	1.0000000000000018	0.9999999999999897
4	15513.738738929262	1.0000000000000095	0.9999999999999165
5	476607.2502421033	0.9999999999999957	0.9999999999999949
6	1.4951058641626492e7	0.9999999999996649	1.0000000000088654
7	4.7536735571017563e8	0.9999999999981822	1.0000000000677616
8	1.5257575514263742e10	0.999999999993595	1.000000000329728
9	4.931539971476631e11	1.0000000000257598	0.9999999979597309
10	1.602500311938691e13	1.0000000015318404	0.9999998696797339
20	2.906214331972671e18	1.000000001398261	0.9999993967226837

Dla macierzy R_n, n = 5

c – wskaźnik uwarunkowania macierzy	x = A \ b	x = A^{-1} b
1.0000000000000002	1.0000000000000002	1.0
10.000000000000004	1.0000000000000002	1.0
1000.0000000000007	0.9999999999999939	0.9999999999999911
1.0000000004015287e7	0.9999999999727814	0.9999999999804556
9.999343157195112e11	0.9999785847432546	0.9999906029122019
7.298040051658818e16	0.9406736793793593	0.9619027928107008

Dla macierzy R_n, n = 10

c – wskaźnik uwarunkowania macierzy	x = A \ b	x = A^{-1} b
1.0000000000000009	0.9999999999999997	1.0
9.999999999999998	1.0000000000000002	1.0000000000000004
999.9999999999466	1.0000000000000413	1.0000000000000226
9.999999991954071e6	0.9999999999458604	0.9999999999298717
1.0000142389224015e12	0.9999942191264157	0.9999911060424005
6.976019891862081e15	1.1435140149220968	1.0995438009169303

Dla macierzy R_n, n = 20

c – wskaźnik uwarunkowania macierzy	x = A \ b	x = A^{-1} b
1.000000000000001	0.999999999999999	1.0000000000000007
9.999999999999998	1.0	1.0000000000000004
1000.0000000000258	0.9999999999999829	0.999999999999986
1.0000000000809174e7	0.9999999999664703	0.9999999999685064
1.0000079336862542e12	1.0000195207243292	1.0000240472759387
8.511314017431578e15	1.090101677338083	1.1018981505894256

2.4 Wnioski

Wraz ze wzrostem wskaźnika uwarunkowania macierzy otrzymywane wyniki zaczynają coraz bardziej odbiegać od oczekiwanych. Ponieważ precyzja arytmetyki Float64 wynosi 2.220446049250313e-16 (jest rzędu 10^{-16}) możemy zauważyć, że otrzymywane wyniki zaczynają się różnić o wartości rzędu c * macheps np.:

dla R_n, n = 20 i c = 1 błąd jest rzędu 10^{-16} → x = 1.0000000000000007,

dla R_n, n = 20 i c = 10^{16} błąd jest rzędu 10^{-1} → x = 1.1018981505894256

Macierz Hilberta jest źle uwarunkowana, dlatego dla coraz większych n wskaźnik uwarunkowania rośnie bardzo szybko, a co za tym idzie otrzymywane wyniki stają się niedokładne.

3. Zadanie 3

3.1 Opis problemu

a) Obliczyć 20 zer wielomianu Wilkinsona P w postaci naturalnej:

$$P(x) = 1\,x^{20} - 210.0\,x^{19} + 20615.0\,x^{18} - 1256850.0\,x^{17} + 53327946.0\,x^{16} - 1672280820.0\,x^{15} + 40171771630.0\,x^{14} - 756111184500.0\,x^{13} + 11310276995381.0\,x^{12} - 135585182899530.0\,x^{11} + 1307535010540395.0\,x^{10} - 10142299865511450.0\,x^{9} + 63030812099294896.0\,x^{8} - 311333643161390640.0\,x^{7} + 1206647803780373360.0\,x^{6} - 3599979517947607200.0\,x^{5} + 8037811822645051776.0\,x^{4} - 12870931245150988800.0\,x^{3} + 13803759753640704000.0\,x^{2} - 8752948036761600000.0\,x + 2432902008176640000.0$$

Wielomian Wilkinsona p jest postaci:

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

Następnie sprawdzić obliczone pierwiastki z_k, $1 \leqslant k \leqslant 20$ obliczając $|P(z_k)|$, $|p(z_k)|$ i $|z_k - k|$.

b) Powtórzyć eksperyment zamieniając współczynnik -210 na $-210-2^{-23}$.

3.2 Rozwiązanie

W celu uzyskania postaci normalnej wielomianu Wilkinsona użyłem konstruktora Poly() podając jako argument współczynniki wielomianu. Wielomian p wygenerowałem używając metody poly(), generując wielomian na podstawie jego pierwiastków (od 1 do 20). Aby policzyć wartości wielomianów w danych punktach użyłem metody polyval(„wielomian", „wartość").

3.3 Wyniki

a)

z_k	$\|P(z_k)\|$	$\|p(z_k)\|$	$\|z_k - k\|$
0.9999999999997752	26624.0	27648.0	2.248201624865942e-13
2.0000000000835962	528896.0	545280.0	8.359624104059549e-11
2.99999999238026	5.30688e6	5.380608e6	7.619739950115445e-9
4.000000258252059	3.1365632e7	3.1627776e7	2.582520588489956e-7
4.999995526518891	1.3515008e8	1.3566208e8	4.47348110910184e-6
6.0000468280376715	4.93355008e8	4.94682112e8	4.682803767153132e-5
6.999672907671254	1.440342528e9	1.443151872e9	0.0003270923287459482
8.001635039841798	3.817224192e9	3.821419008e9	0.001635039841797692
8.993981354169168	1.0342221312e10	1.0349671936e10	0.006018645830831559
10.017588262682127	2.2865122816e10	2.2875401216e10	0.017588262682126654
10.962314851409891	4.9934647296e10	4.9948185088e10	0.03768514859010885
12.070465818821347	1.24841463808e11	1.24862945792e11	0.07046581882134717
12.90686527219602	1.96617889792e11	1.96644164608e11	0.0931347278039798
14.101867195266468	5.21279421952e11	5.21319753728e11	0.10186719526646826
14.913372910784355	8.37568397312e11	8.37616226816e11	0.08662708921564466
16.04980905799017	1.50003613184e12	1.500103663104e12	0.04980905799017066
16.976860105569482	2.446652635648e12	2.446732916224e12	0.02313989443051767
18.00659818178863	4.620619615232e12	4.620727186944e12	0.006598181788628921
18.99883919844017	8.107033942016e12	8.107160339456e12	0.0011608015598305599
20.00008723809686	1.64081617664e13	1.6408325607424e13	8.723809686017603e-5

b)

z_k	$\lvert P(z_k)\rvert$	$\lvert p(z_k)\rvert$	$\lvert z_k - k\rvert$
1.00000000000025 + 0.0im	29696.0 + 0.0im	29696.0 + 0.0im	2.5002222514558525e-13 + 0.0im
1.9999999999663314 + 0.0im	212480.0 + 0.0im	196096.0 + 0.0im	3.36686234447825e-11 + 0.0im
3.0000000007917986 + 0.0im	759808.0 + 0.0im	667648.0 + 0.0im	7.917986266647858e-10 + 0.0im
4.000000022782163 + 0.0im	2.066432e6 + 0.0im	2.328576e6 + 0.0im	2.2782162822920782e-8 + 0.0im
4.999998639496471 + 0.0im	3.8202368e7 + 0.0im	3.8970368e7 + 0.0im	1.3605035293906553e-6 + 0.0im
6.0000315067615535 + 0.0im	2.50441216e8 + 0.0im	3.23432448e8 + 0.0im	3.1506761553501406e-5 + 0.0im
6.999431874279081 + 0.0im	1.200517632e9 + 0.0im	2.511001088e9 + 0.0im	0.0005681257209193546 + 0.0im
8.009247744577655 + 0.0im	4.4818816e9 + 0.0im	2.2222267392e10 + 0.0im	0.00924774457765487 + 0.0im
8.909918231150565 + 0.0im	1.3927051776e10 + 0.0im	1.47126630912e11 + 0.0im	0.0900817688494353 + 0.0im
10.094662661572864 + 0.6538101709831287im	5.400541713629769e10 + 0.0im	1.5315415509070642e12 + 0.0im	0.6606275495141316 + 0.0im
10.094662661572864 - 0.6538101709831287im	5.400541713629769e10 + 0.0im	1.5315415509070642e12 + 0.0im	1.1167378546603133 + 0.0im
11.79544654376153 + 1.6553515163077377im	3.461261571662791e11 + 0.0im	3.324711352544336e13 + 0.0im	1.6679420730353407 + 0.0im
11.79544654376153 - 1.6553515163077377im	3.461261571662791e11 + 0.0im	3.324711352544336e13 + 0.0im	2.0472268246284706 + 0.0im
13.993304384308534 + 2.5195795843777025im	2.888985534816439e12 + 0.0im	9.571230893400511e14 + 0.0im	2.5195884809393783 + 0.0im
13.993304384308534 - 2.5195795843777025im	2.888985534816439e12 + 0.0im	9.571230893400511e14 + 0.0im	2.7132484856100696 + 0.0im
16.731138189999534 + 2.8128555694464437im	2.690520979252151e13 + 0.0im	2.744959485258138e16 + 0.0im	2.9063240541002426 + 0.0im
16.731138189999534 - 2.8128555694464437im	2.690520979252151e13 + 0.0im	2.744959485258138e16 + 0.0im	2.8256756939611107 + 0.0im
19.502623913431854 + 1.94042123806624891im	1.9339159319274578e14 + 0.0im	4.254936489234442e17 + 0.0im	2.4542031713686248 + 0.0im
19.502623913431854 - 1.94042123806624891im	1.9339159319274578e14 + 0.0im	4.254936489234442e17 + 0.0im	2.0044613689198667 + 0.0im
20.847020713254746 + 0.0im	4.53235943554048e14 + 0.0im	1.374962172803668e18 + 0.0im	0.8470207132547465 + 0.0im

3.4 Wnioski

Float64 ma od 15 do 17 cyfr znaczących. Ponieważ współczynniki w wielomianie Wilkinsona przekraczają tę liczbę powoduje to zaistnienie niedokładności, która wpływa na otrzymywane wyniki.

Wprowadzenie niewielkiej zmiany ($2^{-23} \approx 1.2 * 10^{-7}$) do obliczania pierwiastków wielomianu Wilkinsona spowodowało duże zmiany w otrzymywanych wynikach (otrzymaliśmy także pierwiastki zespolone). Możemy z tego wnioskować, że zadanie obliczenia pierwiastków tego wielomianu jest źle uwarunkowane.

4. Zadanie 4

4.1 Opis problemu

Przeprowadzić eksperyment dla danego równania rekurencyjnego:

$$P_{n+1} := p_n + rp_n(1 - p_n) \text{ dla } n = 0, 1, \ldots$$

1) Dla $p_0 = 0.01$ oraz $r = 3$ wykonać 40 iteracji, a następnie wykonać 10 iteracji, obciąć wynik do 3 miejsc po przecinku i kontynuować obliczenia.
2) Dla $p_0 = 0.01$ i $r = 3$ wykonać 40 iteracji w arytmetyce Float32 i Float64.

4.2 Rozwiązanie

W celu rozwiązania zadanie stworzyłem funkcję recurse(n), którą następnie używałem w pętlach obliczając odpowiednie wartości równania.

```
A[n] // ← tablica, w której przechowywane są wyniki
recurse(n)
        if n = 0 return A[0]
        else if A[n] != 0 return A[n]
        else
                A[n] = recurse(n-1)
                return A[n]
        end if
end
```

Pseudokod funkcji recurse(n)

4.3 Wyniki

A)

Numer iteracji	Wyniki bez obcięcia w 10 iteracji	Wyniki z obcięciem w 10 iteracji
1	0.0397	0.0397
2	0.15407173000000002	0.15407173000000002
3	0.5450726260444213	0.5450726260444213
4	1.2889780011888006	1.2889780011888006
5	0.17151914210917552	0.17151914210917552
6	0.5978201201070994	0.5978201201070994
7	1.3191137924137974	1.3191137924137974
8	0.056271577646256565	0.056271577646256565
9	0.21558683923263022	0.21558683923263022
10	**0.722914301179573**	**0.722**
11	1.3238419441684408	1.324148
12	0.03769529725473175	0.03648822228799964
13	0.14651838271355924	0.14195871805478313
14	0.521670621435246	0.5073780393238604
15	1.2702617739350768	1.2572147329310672
20	0.5965293124946907	1.3097310226799155
25	1.315588346001072	1.0891739070296693
30	0.37414648963928676	1.3331050322407778
35	0.9253821285571046	0.2204442995206507
40	0.011611238029748606	0.7305550338104317

B)

Numer iteracji	Float32	Float64
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
15	1.2704837	1.2702617739350768
20	0.5799036	0.5965293124946907
25	1.0070806	1.315588346001072
30	0.7529209	0.37414648963928676
35	1.021099	0.9253821285571046
40	0.25860548	0.011611238029748606

4.4 Wnioski

Model przedstawiony w zadaniu nosi nazwę *modelu logistycznego.* Eksperyment pokazuje, jak drobne błędy (spowodowane np. niedokładnym przechowywaniem danych, błędnymi pomiarami) mogą się kumulować, doprowadzając do zupełnie błędnych wyników. Jest to zjawisko chaosu w matematycznych układach sprzężeń zwrotnych. Z powodu małych niedokładności po pewnym czasie możliwość przewidywania wyniku załamuje się (w tabeli z przykładu A widać brak korelacji między wynikami z późniejszych iteracjach eksperymentu). Zjawisko to będzie występować nawet dla w pełni poprawnego modelu matematycznego z powodu *czułej zależności od warunków początkowych*.

Możemy także zauważyć, że te same obliczenia wykonywane w różnych arytmetykach (z różną dokładnością) również mogą prowadzić do różnych wyników. Wyniki otrzymane w arytmetykach Float32 oraz Float64 różnią się z powodu dokładności, z jaką przechowywane są liczby. Przez to, że niemożliwe jest uniknięcie niedokładnego przechowywania danych zjawisko chaosu będzie występowało nawet na komputerach z ogromna mocą obliczeniową. Błędy wynikają również ze sposobu, w jaki wykonywane są obliczenia. Dla równoważnych równań

$$p + rp(1 - p) \text{ oraz } (1 + r)p - rp^2$$

otrzymamy różne wyniki.

5. Zadanie 5

5.1 Opis problemu

Dla danego równania rekurencyjnego

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

gdzie c jest pewną stałą przeprowadzić 40 iteracji obliczeń dla następujących wartości zmiennych:

1) $c = -2$, $x_0 = 1$
2) $c = -2$, $x_0 = 2$
3) $c = -2$, $x_0 = 1.99999999999999$
4) $c = -1$, $x_0 = 1$
5) $c = -1$, $x_0 = -1$
6) $c = -1$, $x_0 = 0.75$
7) $c = -1$, $x_0 = 0.25$

5.2 Rozwiązanie

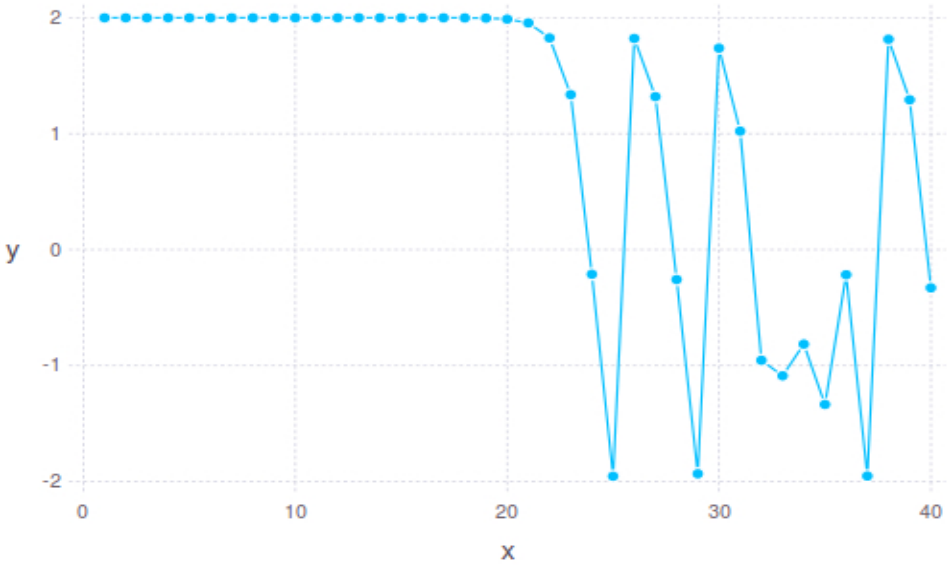
W celu rozwiązania zadania stworzyłem funkcję recurse(n, c, x_0), którą następnie używałem w pętlach podstawiając odpowiednie wartości c, n, oraz x_0.

```
recurse(n, c, x0)
        if n = 0 return x0
        else
                x = recurse(n-1, c, x0)
                return x^2 + c
        end if
end
```

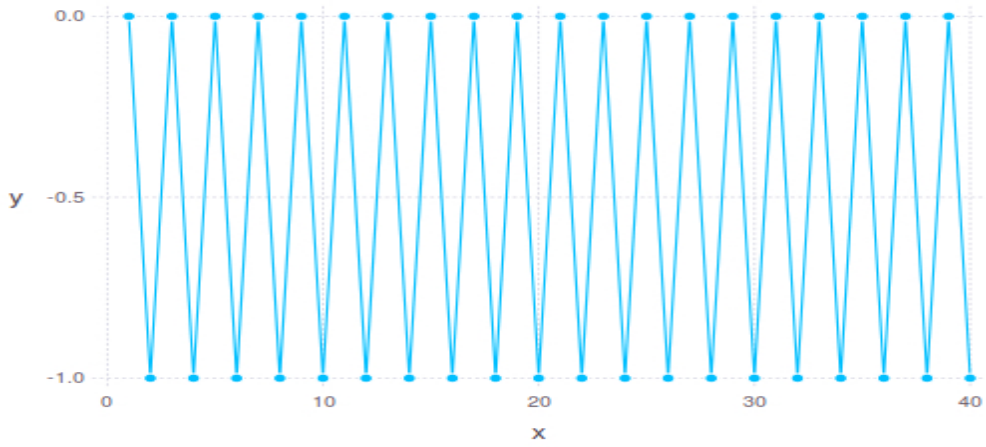
Pseudokod funkcji recurse(n, c, x_0).

5.3 Wyniki

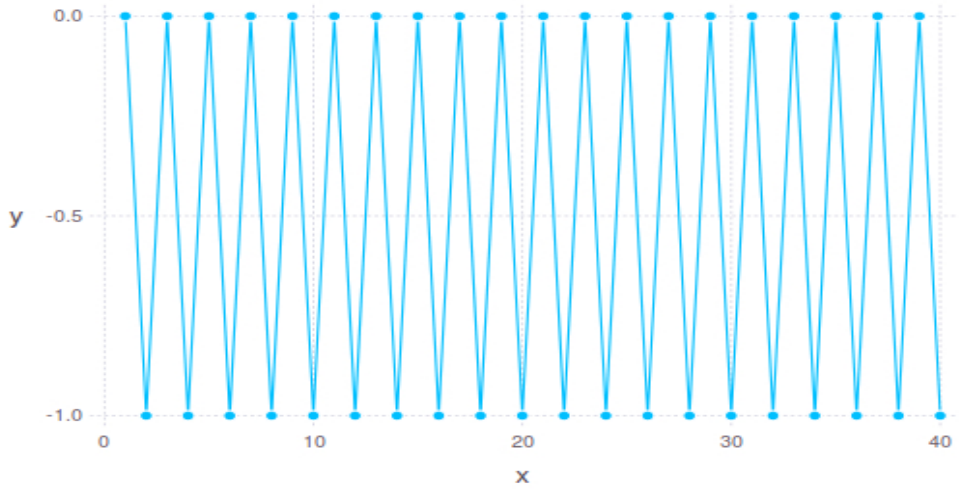
1) x_n = const = -1
2) x_n = const = 2
3)



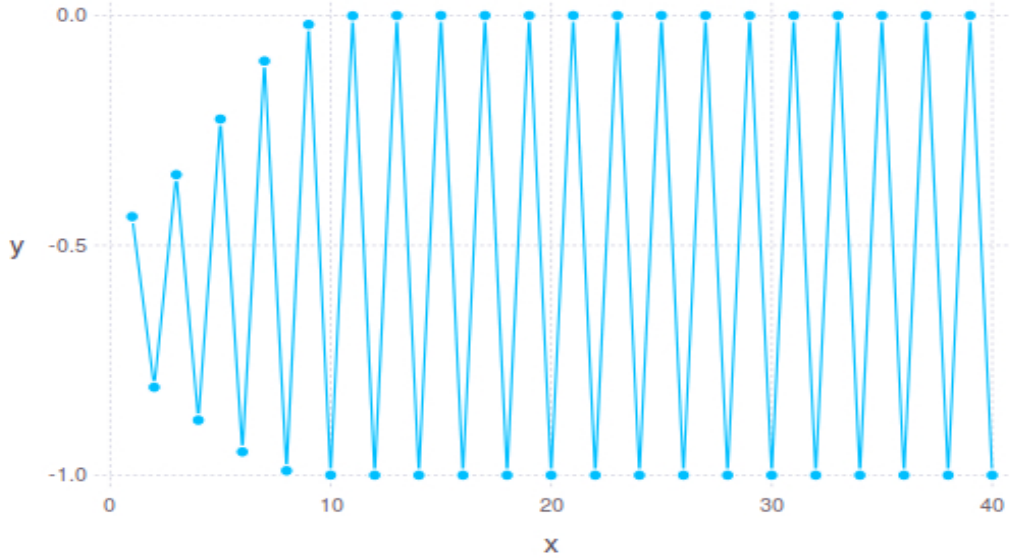
4)



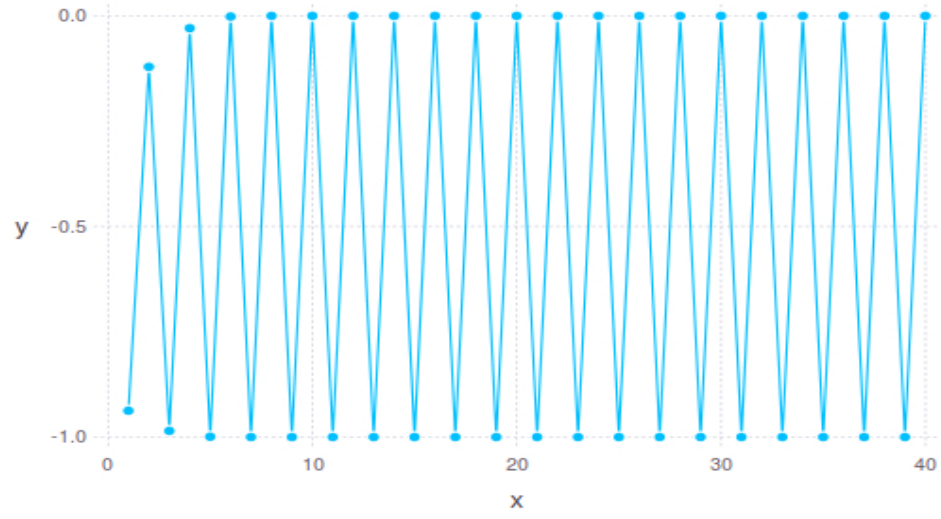
5)


0.0
-0.5
-1.0
y
0
10
20
30
40
x

6)


0.0
-0.5
-1.0
y
0
10
20
30
40
x

7)


0.0
-0.5
-1.0
y
0
10
20
30
40
x

5.4 Wnioski

Rozpatrywane równanie $x_n^2 + c$ jest równoważne z równaniem z poprzedniego zadania. Podpunkty 1 oraz 2 pokazują wyjątkowe sytuacje, w których funkcja zachowuje się stabilnie. W podpunkcie 3 możemy zauważyć, jak niewielka zmiana po pewnym czasie znowu sprawia, że otrzymywane wyniki są nieuporządkowane. W podpunktach 4 oraz 5 możemy zauważyć, że przebieg funkcji zależy od procesu podnoszenia do kwadratu. Natomiast w podpunktach 6 i 7 widzimy, jak funkcja po kilku iteracjach się stabilizuje. Aby przewidzieć w inny sposób niż analizowanie wyników, czy iteracja doprowadza do stanu stabilnego możemy posłużyć się iteracją graficzną.

Otrzymane wyniki obrazują nam, jak trudne jest przeprowadzenie analizy chaosu dla danego problemu.