Obliczenia naukowe
Sprawozdanie lista 2

Wojciech Wróblewski 250349

October 2020

Zad1

Opis problemu oraz terminologia

Powtórzyć zadanie 5 z listy 1, ale usunąć ostatnią dziewiątkę z czwartej współrzędnej wektora x i ostatnią siódemkę z piątej współrzędnej wektora x . Ocenić wpływ niewielkich mian.
We wspomnianym zadaniu mieliśmy obliczyć iloczyn skalarny 2 wektorów x oraz y, korzystając z 4 różnych metod.

Rozwiązanie

Otrzymanie wyników umożliwią nam funkcje napisaje w języku Julia z wcześniejszej listy.

Wyniki

Porównajmy wyniki uzyskane po redukcji cyf we współrzednych wektorów z wynikami z poprzedniej listy.
Wyniki stare

podpunkt	Float32	Float64
a)	-0.4999443	1.0251881368296672e-10
b)	-0.4543457	-1.5643308870494366e-10
c)	-0.5	0.0
d)	-0.5	0.0

Wyniki po modyfikacji.

podpunkt	Float32	Float64
a)	-0.4999443	-0.004296342739891585
b)	-0.4543457	-0.004296342998713953
c)	-0.5	-0.004296342842280865
d)	-0.5	-0.004296342842280865

Wnioski

Obserwujemy, że w typie Float32 precyzja jest zbyt mała aby uzyskać inne wyniki przez co wszystkie dane wynikowe są równe danym z poprzedniego ćwiczenia. Arytmetyka Float64 zwraca poprawne wyniki, które są identyczne dla każdej metody. Wnioskujemy, że algorytmy zaimplementowane na poprzedniej liście są poprawne, ale samo zadanie jest źle uwarunkowane.

Zad2

Opis problemu

Policzyć granicę funkcji $f(x) = e^x \ln(1 + e^{-x})$ oraz porównać ją z wykresami funkcji wykonanymi w dwóch programach do wizualizacji.

Rozwiązanie

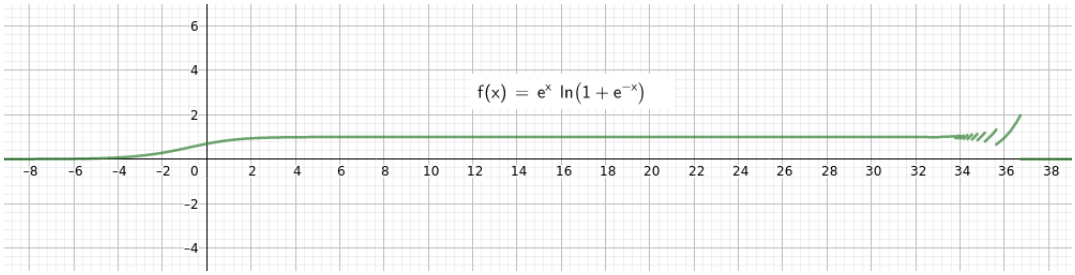
Policzmy granicę z funkcji $f(x)$ sprowadzając do postaci umożliwiającej użycie reguły L'Hospitala.

$\lim_{x\to\infty} f(x) = \lim_{x\to\infty} e^x \ln(1+e^{-x}) = \lim_{x\to\infty} \frac{\ln(1+e^{-x})}{\frac{1}{e^x}} \overset{\text{H}}{=} \lim_{x\to\infty} \left(\frac{-\frac{e^{-x}}{1+e^{-x}}}{-e^{-x}} \right) =$

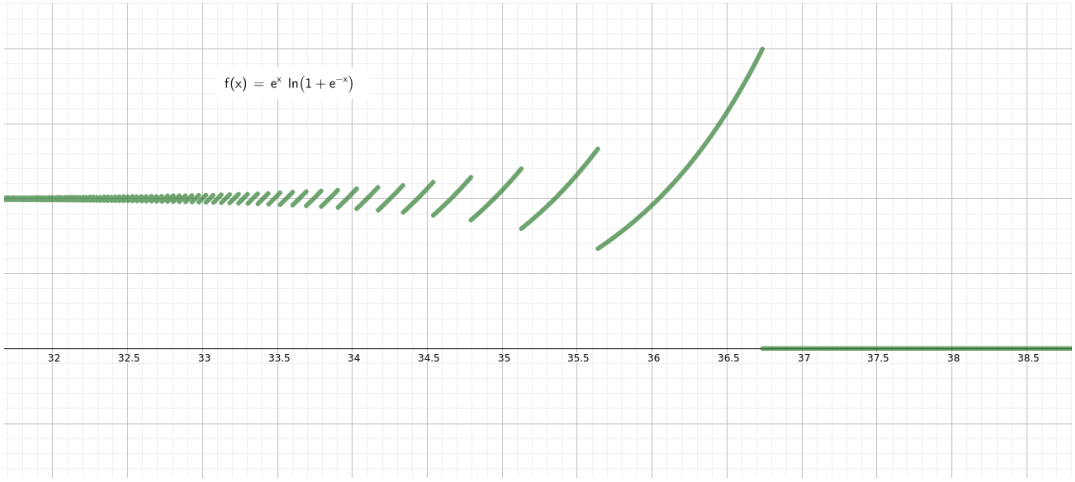
$\lim_{x\to\infty}(\frac{1}{1+e^{-x}}) = 1$

Wyniki

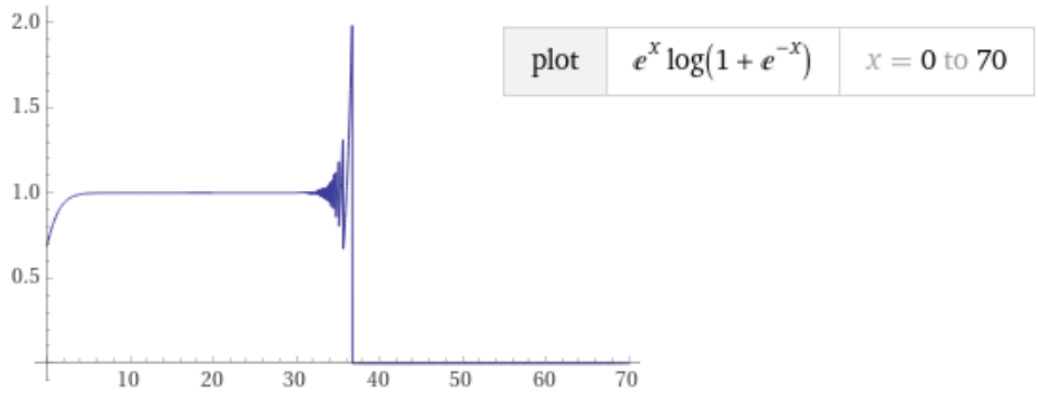
Edytory jakie będziemy uwzględniać to Geogebra i WolframAlpha.



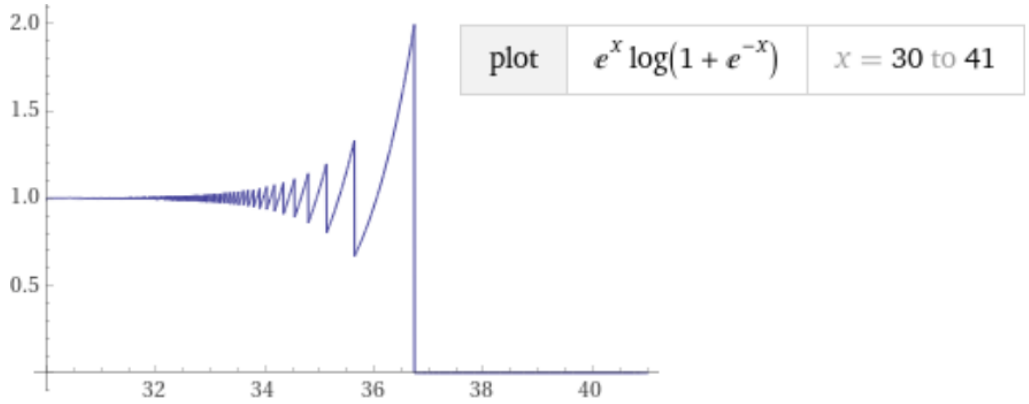
Rysunek 1: Wykres z programu geogebra



Rysunek 2: Wykres z programu geogebra z przybliżeniem



Rysunek 3: Wykres z programu wolfram aplha



Rysunek 4: Wykres z programu wolfram alpha

Wnioski

Porównując faktyczną granicę zdefiniowanej funkcji z jej graficznymi reprezentacjami zauważamy rozbieżności. Dla pierwszych dodatnich argumentów wykresy dobrze pokazują dążenie funkcji do jedynki, jednak dla przedziału x-ów większych od 30 obserwujemy oscylowanie funkcji, gdzie górna granica wartości funkcji dochodzi nawet do 2. Następnie dla x-ów bliskich 37 oraz większych obswerujemy, że wartości funkcji na wykresie spadły do 0. Rezultat ten jest efektem niedokładności wynikającej z dodawania 1 do wyrażenia e^{-x}, które dla coraz większych argumentów staje się bardzo małe względem jedynki. Dodając liczby różniące się rzędem, obserwujemy pochłonięcie wartości wyrażenia e^{-x} przez jedynkę, co w rezultacie generuje błąd w graficznej granicy funkcji.

Zad3

Opis problemu

Rozważyć rozwiązania układów równań liniowych $Ax = b$ dla macierzy współczynników $A \in R^{nxn}$ i wektora prawych stron $b \in R^n$.

Sposoby generowanie macierzy.

- Macierz $A = H_n$, gdzie H_n jest macierzą hilberta wygenerowaną poprzez funkcję hilb().
- Macierz $A = R_n$, gdzie R_n jest losową macierzą stopnia n z predefiniowanym wskaźnikiem uwarunkownia, otrzymaną z funkcji matcond().

Wektor b zadany jest jako $b = Ax$, gdzie A jest wygenerowaną macierzą oraz $x = (1, ..., 1)^T$. W celu rozwiązania układu równań korzystamy z metody eliminacji Gaussa (x=A b) oraz $x = A^{-1}b$. Eksperymenty wykonujemy dla macierzy hilberta H_n z rosnącym stopniem n oraz dla macierzy losowej R_n, dla której $n \in \{5, 10, 20\}$ z rosnącym wskaźniekiem uwarunkowania $c \in \{1, 10, 10^3, 10^7, 10^{12}, 10^{16}\}$ dla każdego z rzędów.

Rozwiązanie

Dane wygenerowałem korzystając z funkcji hilb(),matcond() podanych na stronie wykładowcy oraz implementacji funkcji gaussian_err oraz inv_err obliczające błędy względne rozwiązań. Algorytmy obliczjące błędy znajdują się poniżej.

```
#obliczanie bledu wzglednego dla metody gaussa
function gaussian_err(A,b,n,x)
    return norm(A \ b - x)/norm(x)
end
#obliczanie bledu wzgl dnego
function inv_err(A,b,n,x)
    return norm(inv(A) * b - x)/norm(x)
end
```

Wyniki

Tabela 1: Wyniki dla macierzy losowej.

n	rank(A)	cond(A)	gaussian_err	inversion_err
5	5	1.0000000000000004	1.719950113979703e-16	2.2752801345137457e-16
5	5	9.999999999999993	2.6272671962866383e-16	1.3136335981433191e-16
5	5	999.9999999999762	1.0473823066688541e-14	9.866379057398558e-15
5	5	9.999999999904515e6	2.459622302859417e-10	2.1856957296280407e-10
5	5	9.999817501999833e11	2.8256765085310778e-5	3.4459197359119305e-5
5	4	8.277994041228786e15	2.6272671962866383e-16	0.06555055301063448
10	10	1.0000000000000013	3.1791949213824894e-16	3.2177320244274193e-16
10	10	9.99999999999998	4.440892098500626e-16	4.183640918457414e-16
10	10	999.9999999999696	1.1678997426721634e-14	1.3160522106737663e-14
10	10	9.99999999844451e6	8.368738932750286e-11	1.0392146110160465e-10
10	10	1.0000170368044983e12	1.64683108177941e-5	1.7099283423416847e-5
10	9	1.7485186136661578e16	0.254945238260756	0.272296727509715
20	20	1.0000000000000016	6.834863329361955e-16	4.896322696446008e-16
20	20	9.999999999999998	6.816805534718973e-16	5.23691153334427e-16
20	20	999.999999999931	1.4355574384816614e-14	1.9404399130943875e-14
20	20	9.999999999858376e6	2.2401088763138323e-10	2.3254041292121015e-10
20	20	1.0000637820255469e12	3.920301108353138e-5	4.3334634579853395e-5
20	19	1.0305970419867404e16	0.10507788193821126	0.09722168937415984

Tabela 2: Wyniki dla macierzy hilberta.

n	rank(A)	cond(A)	gaussian_err	inversion_err
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	15513.738738928929	4.637277712035294e-13	7.542470546988852e-13
5	5	476607.25024224253	1.7697056701418277e-13	7.45602798259539e-12
6	6	1.495105864125091e7	3.496491467713994e-10	3.533151828962887e-10
7	7	4.7536735637688667e8	1.3175049864850338e-8	6.190844397992631e-9
8	8	1.5257575516147259e10	2.487433466002445e-7	3.775275483015941e-7
9	9	4.9315408927806335e11	9.643625435772316e-6	1.1659486044133412e-5
10	10	1.6024859712306152e13	0.00022035288727930986	0.0003357158826776558
11	10	5.2210348947688544e14	0.006022512934347414	0.01113776822564549
12	11	1.7255427417341868e16	0.19509235225028912	0.16218620232347905
13	11	7.126491965424366e17	7.894191771622431	5.511855154155295
14	11	6.101307732044041e17	0.8270688593203056	3.3522039875276723
15	12	4.223311222761075e17	3.10349386243609	4.354299435453685
16	12	3.535827507735838e17	9.083139658689422	54.189834405860445
17	12	3.1182808742153696e17	4.24328971542452	5.786281231941037
18	12	1.5639169583348145e18	4.7860299021083	5.7599951815224495
19	13	1.3274441976880407e18	6.114994252530053	12.309212980457932
20	13	2.2777635596453635e18	19.122235961045973	17.030822563878868
21	13	1.5088647979164173e18	5.528693844520417	4.797191888763164
22	13	2.148587035517758e18	14.91838193889066	19.452979830106727
23	13	8.53990580100839e18	7.050470984846638	6.265996982174681
24	13	1.1703742699502748e19	13.918474300172141	17.20261485961593
25	13	1.5100611248172846e18	28.59107844940893	31.685081256911236

Wnioski

W eksperymencie zauważamy że dla dwóch zdefiniowanych konstrukcji macierzy, błędy względne zależą od wskaźnika uwarunkowania. W macierzy losowej obswerujemy wzrost błędu wraz ze wzrostem wskaźnika uwarunkowania, jednak w porównaniu z macierzą hilberta bład ten jest znacząco mniejszy. Wnioskujemy, że macierz hilberta jest źle uwarunkowana dlatego, że małe zmieny n bardzo mocno zwiększają wskaźnik uwarunkowania, a w związku z tym również ostateczny błąd względny.

Zad4

W zadaniu mamy zadany wielomian Wilkinsona w dwóch postaciach.

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

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

W eksperymencie mamy sprawdzić obliczone pierwiastki z_k $k \in < 1, .., 20 >$, obliczając $|P(z_k)|$, $|p(z_k)|$ i $|z_k - k|$.

Rozwiązanie

W rozwiązaniu pierwiastki wielomianu zostały wygenerowane za pomocą funkcji roots(), a same wielomiany P(x) oraz p(x), zotały wygenerowane wbudowanymi funkcjami poly(),Poly(). Funkcja poly() tworzy wielomian z pierwiastków, a Poly() generuje wielomian ze współczynników.
W implementacji funkcje get_p() oraz get_P() zwracają odpowiednio p(x) i P(x).
Ostatecznie przechodzimy po pierwiastkach wygenerowanych przez wbudowaną funkcją roots() i wyświetlamy interesujące nas dane. Kod reprezentujący algorytm głównej funkcji znajduje się poniżej.

```
function get_data(coefficients)
    p =get_p()
    P =get_P(coefficients)
    c_roots = roots(P)
    println("k & z_k & |P(zk)| & p(zk)| &  |zk-k| ")
    for k in 1:20
        z = c_roots[k]
        println(
            "$k & $z & $(abs(polyval(P, z)))
            & $(abs(polyval(p, z))) & $(abs(z - k))"
        )
  end
end
```

Wyniki

k	z_k	$\lvert P(x_k)\rvert$	$\lvert p(z_k)\rvert$	$\lvert z_k$-k$\rvert$
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

Zmieńmy współczynnik -210 na $-210 - 2^{23}$ i zobaczmy jak wpłynie na wyniki.

k	$\lvert P(x_k)\rvert$	$\lvert p(z_k)\rvert$
1	0.9999999999998357 + 0.0im	1.6431300764452317e-13
2	2.0000000000550373 + 0.0im	5.503730804434781e-11
3	2.99999999660342 + 0.0im	3.3965799062229962e-9
4	4.000000089724362 + 0.0im	8.972436216225788e-8
5	4.99999857388791 + 0.0im	1.4261120897529622e-6
6	6.000020476673031 + 0.0im	2.0476673030955794e-5
7	6.99960207042242 + 0.0im	0.00039792957757978087
8	8.007772029099446 + 0.0im	0.007772029099445632
9	8.915816367932559 + 0.0im	0.0841836320674414
10	10.095455630535774 - 0.6449328236240688im	0.6519586830380407
11	10.095455630535774 + 0.6449328236240688im	1.1109180272716561
12	11.793890586174369 - 1.6524771364075785im	1.665281290598479
13	11.793890586174369 + 1.6524771364075785im	2.0458202766784277
14	13.992406684487216 - 2.5188244257108443im	2.518835871190904
15	13.992406684487216 + 2.5188244257108443im	2.7128805312847097
16	16.73074487979267 - 2.812624896721978im	2.9060018735375106
17	16.73074487979267 + 2.812624896721978im	2.825483521349608
18	19.5024423688181 - 1.940331978642903im	2.4540214463129764
19	19.5024423688181 + 1.940331978642903im	2.0043294443099486
20	20.84691021519479 + 0.0im	0.8469102151947894

k	z_k	$\lvert z_k$-k$\rvert$
1	20992.0	22016.0
2	349184.0	365568.0
3	2.221568e6	2.295296e6
4	1.046784e7	1.0729984e7
5	3.9463936e7	4.3303936e7
6	1.29148416e8	2.06120448e8
7	3.88123136e8	1.757670912e9
8	1.072547328e9	1.8525486592e10
9	3.065575424e9	1.37174317056e11
10	7.143113638035824e9	1.4912633816754019e12
11	7.143113638035824e9	1.4912633816754019e12
12	3.357756113171857e10	3.2960214141301664e13
13	3.357756113171857e10	3.2960214141301664e13
14	1.0612064533081976e11	9.545941595183662e14
15	1.0612064533081976e11	9.545941595183662e14
16	3.3151034759817633e11	2.7420894016764064e16
17	3.3151034759817633e11	2.7420894016764064e16
18	9.539424609817828e12	4.2525024879934694e17
19	9.539424609817828e12	4.2525024879934694e17
20	1.114453504512e13	1.3743733197249713e18

Wnioski

Analizując błędy podczas wyznaczania pierwiastków wielomianu Wilkinsona musimy pamiętać że Float64 ma od 15 do 17 cyfr znaczących w reprezentacji, a zadane współczynniki są dłuższe co oznacza, że tracimy dokładność. Dlatego otrzymujemy zaistniałe błędy. Modyfikując jeden współczynik o bardzo małą wartość widzimy silne zaburzenie wyników, gdyż pojawiają się pierwiastki zespolone. Zachowanie to znaczy, że obliczanie pierwiastów wielomianu Wilkinsona jest zadaniem źle uwarunnkowanym.

Zad5

Opis problemu

Rozważyć równanie rekurencyjne $p_{n+1} := p_n + rp_n(1 - p_n)$ dla $n = 0, 1, ...$ oraz przeprowadzić eksperymenty.

- Dla danych $p_0 = 0.01$ i $r = 3$ wykonać 40 iteracji wyrażenia (1), a następnie wykonać ponownie 40 iteracji wyrażenia (1) z niewielką modyfikacją tj. wykonać 10 iteracji, zatrzymać, zastosować obcięcie wyniku odrzucając

cyfry po trzecim miejscu po przecinku i kontynuować dalej obliczenia (do 40-stej iteracji) tak, jak gdyby był to ostatni wynik na wyjściu.

- Dla danych $p_0 = 0.01$ i $r = 3$ wykonać 40 iteracji wyrażenia rekurencyjnego w arytmetyce Float32 i Float64.

Rozwiązanie

Poniżej zaprezentowane są funkcje, które zwracają wyniki dla obu podpunktów. Funkcja formula() zwraca wynik wyrażenia po zadanej liczbie iteracji(limit - w pierwszym podpunkcie mamy limit równy 40 iteracji).

```
function formula(_type, p, r, limit)
    p =_type(p)
    r =_type(r)
    for i in 1:limit
        p = p + _type(r * p * (_type(1.0) - p))
    end
    return p
end
```

Funkcja formula_with_cut() zwraca wynik wyrażenia po zadanej liczbie iteracji z obcięciem . Obcięcie następuje po wykonaniu zadanej liczbie iteracji (cut_position) i następnie ponawiamy iterowanie aż do uzyskania zdefiniowanej liczby iteracji.

```
function formula_with_cut(_type, p, r, limit , cut_position)
    p = formula(_type,p,r,cut_position)
    p = trunc(p, digits=3)
    return formula(_type, p, r, limit - cut_position)
end
```

Wyniki

typ	wynik
Float64	0.011611238029748606
Float32	0.25860548
Float32 (z obcięciem)	1.093568

Wnioski

Eksperyment pokazuje, że zaburzenie wartości zmiennej od której zależą kolejne iteracje powoduje drastyczne zaburzenie ostatecznego rezultatu poprzez gromadzenie błędów. Błędy gromadzone w kolejnych iteracjach od zaburzenia

kumulują się i prowadzą do dużych rozbieżności względem wartości oczekiwanych. Zauważamy również, że obliczenia prowadzone w różnych arytmetykach prowadzą do innych wyników. Liczy przechowywane są jakąś dokładnością przez co niemożliwe jest uzyskać zawsze perfekcyjnie precyzyjny wynik, na co wpływa sposób przechowywyania liczb.

Zad6

Opis problemu

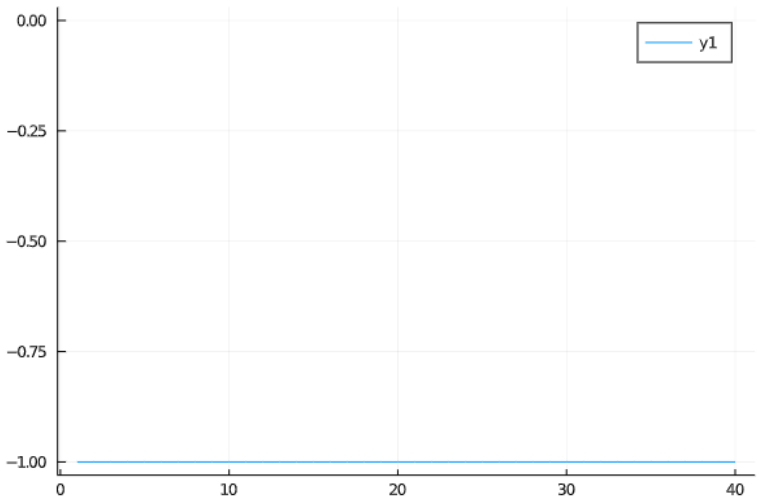
Rozważamy równanie rekurencyjne. $x_{n+1} := x_n^2 + c$
dla n = 0, 1, . . . ,
gdzie c jest pewną daną stałą.
Chcemy przeanalizować zachowanie ciągów dla 40 iteracji równania dla danych:
1. c = -2 i $x_0 = 1$
2. c = -2 i $x_0 = 2$
3. c = -2 i $x_0 = 1.99999999999999$
4. c = -1 i $x_0 = 1$
5. c = -1 i $x_0 =$ -1
6. c = -1 i $x_0 = 0.75$
7. c = -1 i $x_0 = 0.25$

Rozwiązanie

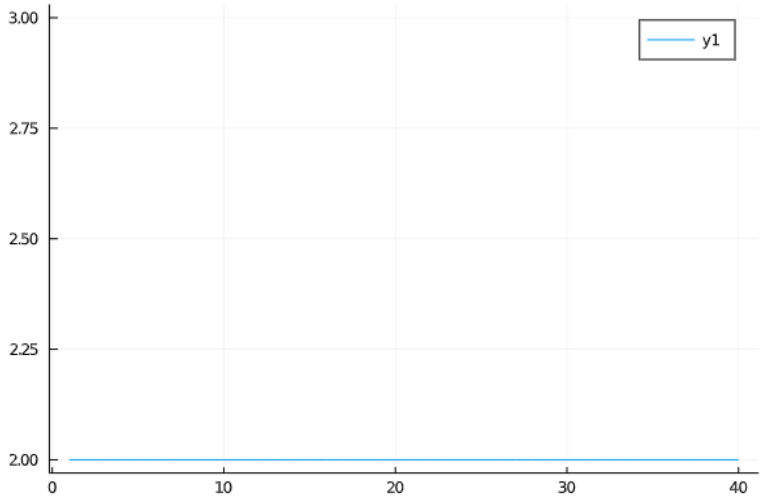
Parametry zostały zapisane w tablicy. W pętli uruchamiamy funkcję recursion_formula(), która zwraca wynik zadanego równania rekurencyjnego dla ilości iteracji jaką zadeklarujemy. Dlatego dla każdej pary współczynników uruchamiamy funkcję recursion_formula(), która wygeneruje nam wyniki równania dla każdej z 40 iteracji i zapisze w tablicy. Nastepnie na podstawie tablicy wykonywany jest wykres przy pomocy biblioteki Plots(). Algorytm głównej funkcji znajduje się poniżej.

```
function get_data(parameters,iterations)
    for i in 1:length(parameters)
        array=zeros(40)
        for k in 1:iterations
            array[k]=
            recursion_formula(k,parameters[i][1],parameters[i][2])
        end
        println(array)
    end
end
```

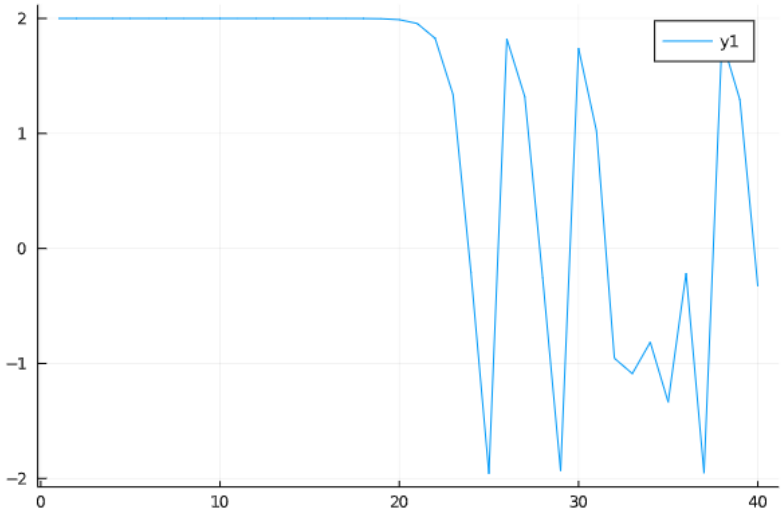
Wyniki



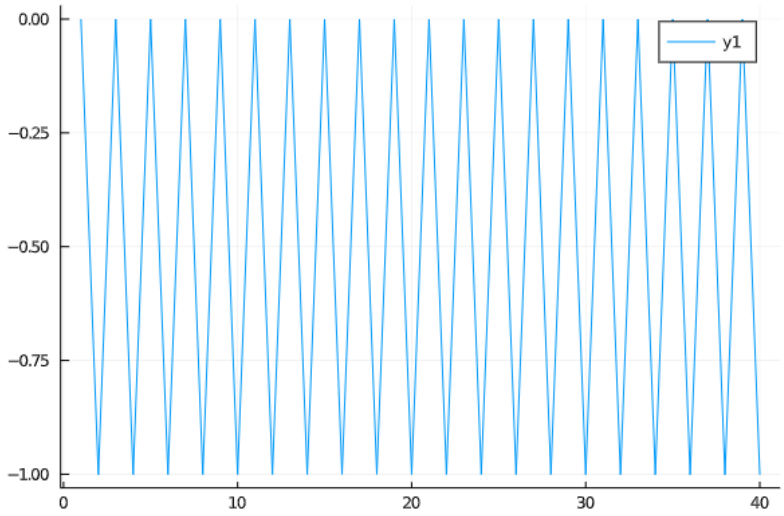
Rysunek 5: wykres dla danych 1



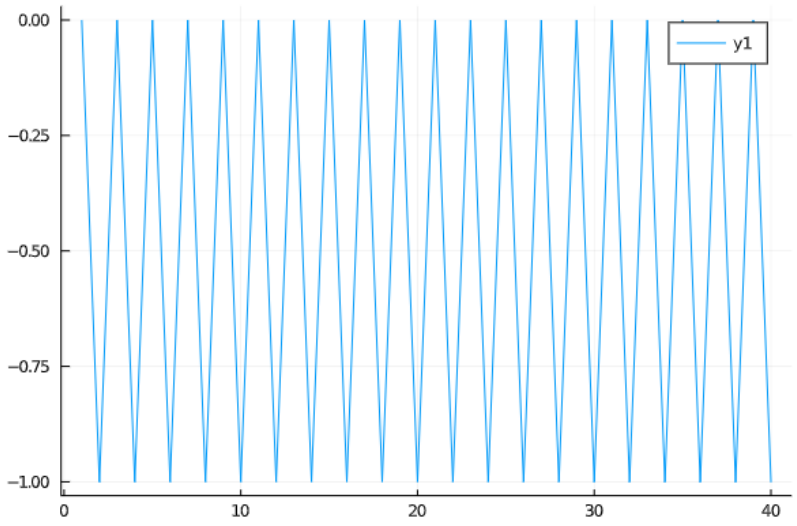
Rysunek 6: wykres dla danych 2



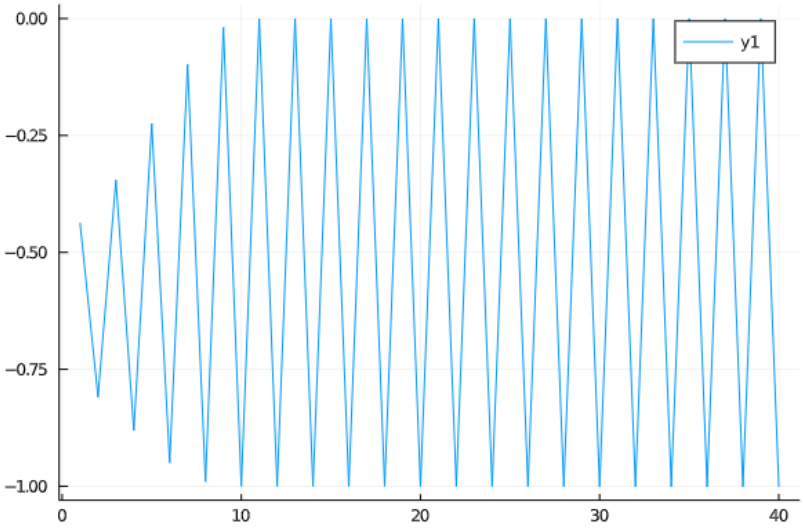
Rysunek 7: wykres dla danych 3



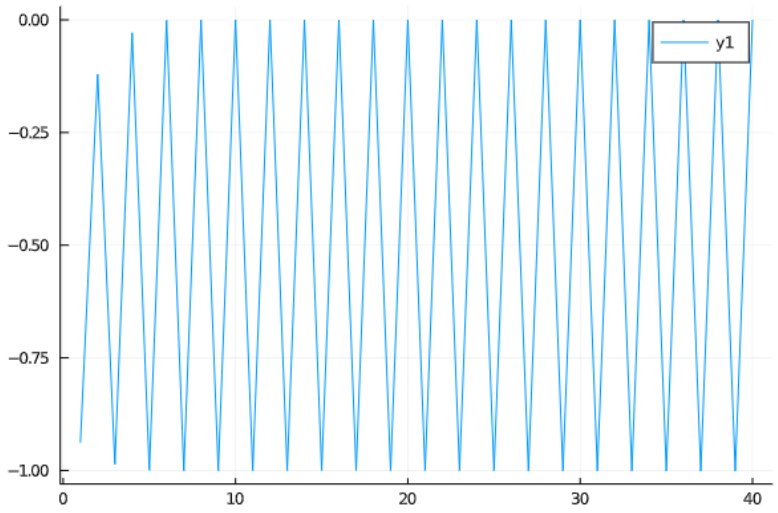
Rysunek 8: wykres dla danych 4



Rysunek 9: wykres dla danych 5



Rysunek 10: wykres dla danych 6



Rysunek 11: wykres dla danych 7

Wnioski

Obserwujemy, że dla dwóch pierwszych przypadków, gdy (c = - 2 i $x_0 = 1$) oraz (c = - 2 i $x_0 = 2$), funkcja przyjmuje stałe wartości. W 3 przypadku otrzymujemy bardzo niestabilne wyniki co jest spowodowane podnoszeniem do kwadratu liczby 1.99999999999999. Niedokładna reprezentacja wspomnianej liczby potęguje błędy z każdą iteracją. Dla zestawu danych 4 oraz 5 otryzmujemy stabilne wyniki, których wartości to na przemian liczby 0 i -1. Dla dwóch ostatnich przypadków początkowe wartości stabilizują się od pewnego miejsca, by zwracać na przemian 0 oraz -1. Zadanie pokazuje, że niektóre zestawy danych nie gwarantują stabilnych wyników oraz błedy, które wystąpują w reprezentacji potrafią się nawarstwiać przez co finalny wynik może różnić się od wartości oczekiwanej.