Sprawozdanie Obliczenia Naukowe
lista 2

Łukasz Bratos

Listopad 2019

1 Zadanie 1

1.1 O zadaniu

W tym zadaniu mamy wykonać polecenie z zadania 5 z poprzedniej listy, ale tym razem ze zmodyfikowanymi danymi wejściowymi, tj. usunąć ostatnią 9 z x_4 i ostatnią 7 z x_5. Mamy zaobserwować jaki wpływ mają niewielkie zmiany danych na wyniki.

1.2 Rozwiązanie

Korzystamy z kodu, który napisaliśmy na potrzeby ostatniej listy. Modyfikujemy tylko nasze dane wejściowe zgodnie z poleceniem.

1.3 Wyniki

Algorytm	Float64	Zmodyfikowany Float64
W przód	1.0251881368296672e-10	-0.004296342739891585
W tył	-1.5643308870494366e-10	-0.004296342998713953
Od największego do najmniejszego	0.0	-0.004296342842280865
Od najmniejszego do największego	0.0	-0.004296342842280865

Tabela 1: Wyniki dla zadania 5 w arytmetyce Float64

Algorytm	Float32	Zmodyfikowany Float32
W przód	-0.3472038161853561	-0.3472038161889941
W tył	-0.3472038162872195	-0.3472038162872195
Od największego do najmniejszego	-0.5	-0.5
Od najmniejszego do największego	-0.5	-0.5

Tabela 2: Wyniki dla zadania 5 w arytmetyce Float32

Widzimy, że otrzymane wyniki w przypadku *Float64* różnią się znacząco od poprzednich oraz dla każdego algorytmu zwracają podobną wartość. W przypadku *Float32* niewiele się zmienia (usunięte cyfry i tak nie zawierały się w tej precyzji).

1.4 Wnioski

Po otrzymanych wynikach możemy przypuszczać, że zadanie to jest źle uwarunkowane. Bardzo mała zmiana danych wejściowych daje ogromną różnicę w wyniku oraz sprawia, że algorytmy zachowują się podobnie.

2 Zadanie 2

2.1 O zadaniu

W tym zadaniu mamy narysować wykres funkcji $f(x)$ w co najmniej dwóch dowolnych programach do wizualizacji, a następnie policzyć granicę funkcji $\lim_{x\to\infty} f(x)$ i porównać wykres funkcji z policzoną granicą.

$$f(x) = e^x \ln(1 + e^{-x})$$

2.2 Rozwiązanie

Pierwszy wykres rysuję za pomocą biblioteki `matplotlib` w Pythonie. Drugi wykres tworzę z pomocą programu Gnuplot korzystając z następujących komend:

```
set xrange [-10.0:45.0]
plot exp(x) * log(1 + exp(-x))
```

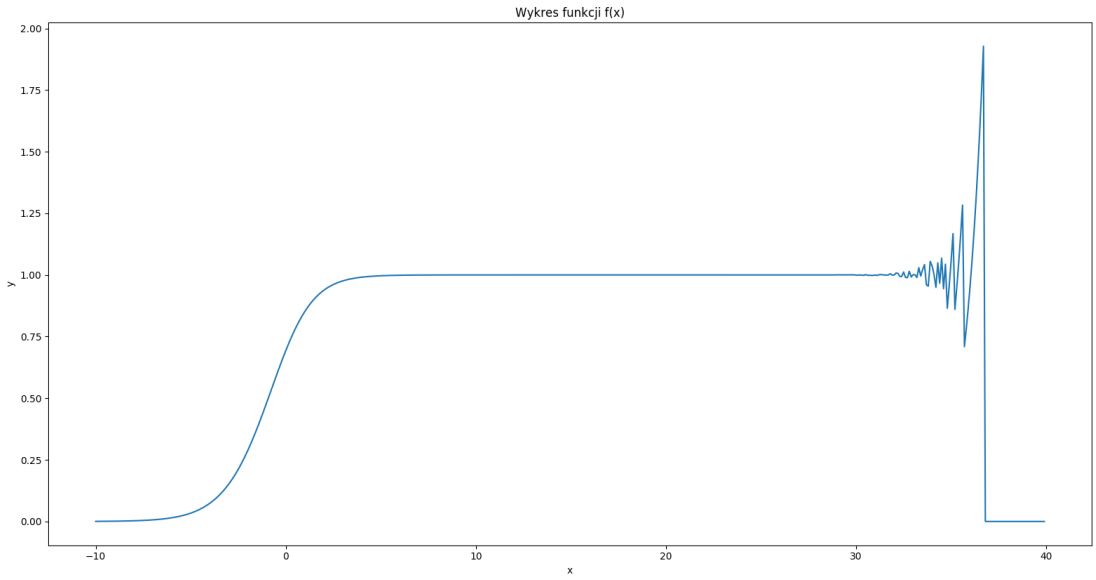
Granicę liczę korzystając z programu WolframAlpha.

2.3 Wyniki

Otrzymana granica to:

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

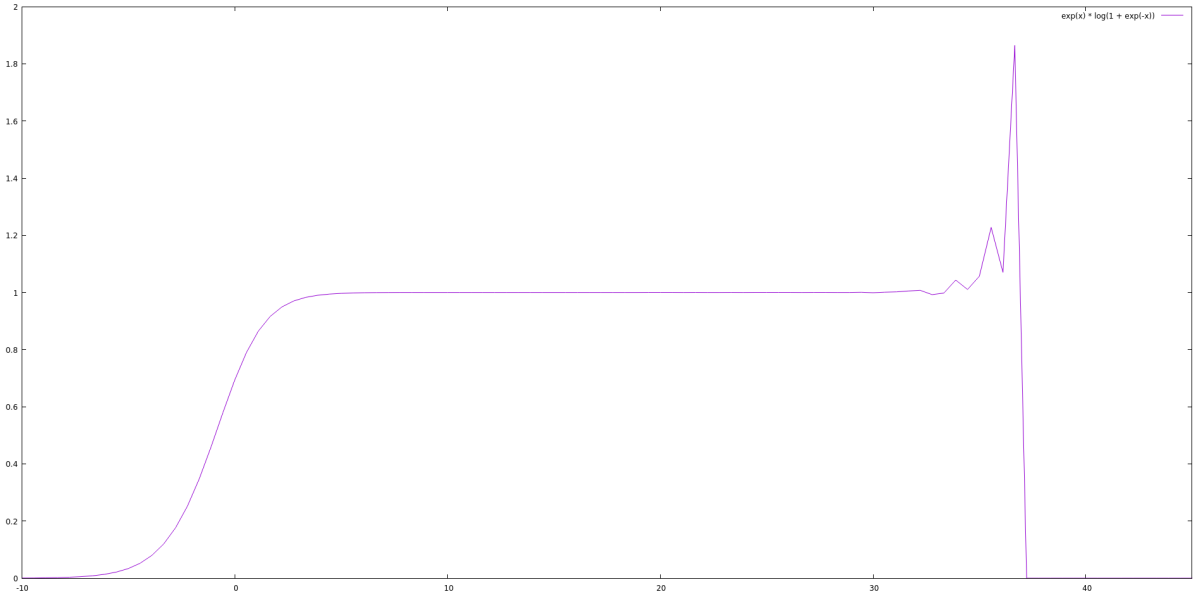
Widzimy, że na wykresach po przekroczeniu wartości 32 zaczyna dochodzić do anomalii. Pomimo, że granica funkcji wynosi 1, to wartości funkcji coraz bardziej od niej dobiegają, aby ostatecznie osiągnąć wartość 0.



Rysunek 1: Wykres funkcji $f(x) = e^x \ln(1 + e^{-x})$ wykonany w PyPlot

2.4 Wnioski

Dla wartości x powyżej 36 wartość e^{-x} zaczyna być mniejsza od epsilona maszynowego, więc wartość logarytmu zaczyna być równa 0, a co za tym idzie cała funkcja osiąga wartość 0.



Rysunek 2: Wykres funkcji $f(x) = e^x \ln(1 + e^{-x})$ wykonany w GNUPlot

3 Zadanie 3

3.1 O zadaniu

W tym zadaniu mamy rozwiązać układy równań liniowych postaci $Ax = b$ (gdzie x jest wektorem jednostkowym, A jest macierzą współczynników, a b wektorem prawych stron) dla macierzy Hilberta oraz macierzy losowych o ustalonym wskaźniku uwarunkowania. Mamy skorzystać z metody eliminacji Gaussa oraz metody inwersji.

3.2 Rozwiązanie

Korzystając z pakietu `LinearAlgebra` wyliczamy b metodą eliminacji Gaussa oraz metodą inwersji ($x = A^{-1} \cdot b$). Liczymy błędy względne w porównaniu z wektorem jednostkowym.

3.3 Wyniki

Wyniki dla macierzy Hilberta umieszczam w tabeli 3, a dla macierzy losowych w tabeli 4.

3.4 Wnioski

Macierz Hilberta jest przykładem macierzy źle uwarunkowanej. Niezależnie od algorytmu blad względny jest znaczny. Dla stosunkowo niewielkich rozmiarów wskaźnik uwarunkowania jest bardzo duży. Błąd względny w przypadku macierzy losowych jest znacznie mniejszy.

4 Zadanie 4

4.1 O zadaniu

W tym zadaniu mamy zbadać pierwiastki wielomianu Wilkinsona w postaci naturalnej obliczone przy pomocy pakietu `Polynomials`. Porównamy wyliczone pierwiastki z rzeczywistymi pierwiastkami oraz wyliczymy wartość wielomianu (dla postaci normalnej i iloczynowej) w z_k, gdzie z_k, to k-ty wyliczony pierwiastek wielomianu. W drugiej części zadania modyfikujemy współczynnik przy x_{19} zmniejszając go o 2^{-23}. Wielomian Wilkinsona wygląda następująco:

$$p(x) = \prod_{i=0}^{20} (x - i)$$

n	rank(H)	cond(H)	Błąd metody Gaussa	Błąd metody inwersji
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	10	5.222677939280335e14	0.00015827808158590435	0.007618304284315809
12	11	1.7514731907091464e16	0.13396208372085344	0.258994120804705
13	11	3.344143497338461e18	0.11039701117868264	5.331275639426837
14	11	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	9.134134521198485	7.575475905055309
19	13	6.471953976541591e18	9.720589712655698	12.233761393757726
20	13	1.3553657908688225e18	7.549915039472976	22.062697257870493

Tabela 3: Wyniki dla macierzy Hilberta z zadania 3

4.2 Rozwiązanie

Wielomian w postaci normalnej tworzymy korzystając z funkcji `Poly()`, a przy postaci iloczynowej z funkcji `poly()`. Pierwiastki wyliczamy przy pomocy funkcji `roots()`. Do obliczenia wartości wielomianu w punkcie wykorzystujemy funkcję `polyval()`.

4.3 Wyniki

Widzimy, że wyliczone pierwiastki różnią się od tych poprawnych, stąd też wartość wielomianu w punktach z_k jest różna od 0. Modyfikując współczynnik o zaledwie 2^{-23} otrzymujemy już pierwiastki zespolone.

4.4 Wnioski

Zadanie to jest źle uwarunkowane ze względu na zaburzenia współczynników. Pokazaliśmy, że niewielka zmiana daje duży błąd. Przez to, że pierwiastki są kolejnymi liczbami rzeczywistymi, niektóre współczynniki naszego wielomianu posiadają więcej cyfr znaczących niż *Float64* jest nam w stanie zaoferować.

5 Zadanie 5

5.1 O zadaniu

Mamy dane następujące równanie rekurencyjne:

$$p_{n+1} = p_n + rp_n(1 - p_n)$$

gdzie $n = 0, 1, ..., r$ jest pewną stałą, $rp_n(1 - p_n)$ jest czynnikiem wzrostu populacji, a p_0 jest wielkością populacji stanowiącą procent maksymalnej wielkości populacji dla danego stanu środowiska.

Mamy przeprowadzić następujące eksperymenty i porównać otrzymane wyniki wewnątrz każdego z nich.

1. Dla danych $p_0 = 0.01$ i $r = 3$ wykonać 40 iteracji naszego wyrażenia, a następnie wykonać ponownie 40 iteracji naszego wyrażenia z niewielką modyfikacją tj. wykonać 10 iteracji, zatrzymać,

n	rank(R)	cond(R)	Błąd metody Gaussa	Błąd metody inwersji
5	5	1.0	1.4043333874306804e-16	0.0
5	5	10.0	1.9860273225978183e-16	9.930136612989092e-17
5	5	10.0^3	1.3726775207334618e-14	1.9394042374125575e-14
5	5	10.0^7	4.468561475845091e-16	1.7371110714613055e-11
5	5	10.0^{12}	2.7641805370534272e-5	3.1687255336448955e-5
5	4	10.0^{16}	0.09390766327130208	0.12558457060881323
10	10	1.0	3.0606736594252445e-16	2.3551386880256624e-16
10	10	10.0	3.439900227959406e-16	3.0606736594252445e-16
10	10	10.0^3	4.399700525621334e-14	4.171224775237441e-14
10	10	10.0^7	3.2365059857557353e-10	2.880029537346403e-10
10	10	10.0^{12}	9.063798727310973e-6	2.6614178058710672e-5
10	9	10.0^{16}	0.07411320652669048	0.10504835196232257
20	20	1.0	3.7567558654335876e-16	3.640109616122045e-16
20	20	10.0	5.517707908259862e-16	4.896322696446008e-16
20	20	10.0^3	2.446322963599226e-14	2.528521665712367e-14
20	20	10.0^7	1.8900240649868923e-10	1.5613669335925758e-10
20	20	10.0^{12}	3.309470002356853e-5	3.623821230698555e-5
20	19	10.0^{16}	0.050545658363591955	0.11703114569421252

Tabela 4: Wyniki dla macierzy losowych z zadania 3

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. Obliczenia wykonać w arytmetyce *Float32*.

2. Dla danych $p_0 = 0.01$ i $r = 3$ wykonać 40 iteracji naszego wyrażenia w arytmetyce *Float32* i *Float64*.

5.2 Rozwiązanie

Implementujemy dosłownie podane równanie w języku **Julia** i korzystając z pętli `for` wykonujemy kolejne iteracje.

Drugą część podpunktu 1. wyliczamy wykonując 30 iteracji dla $p_0 = 0.722$ i $r = 3$.

5.3 Wyniki

Widzimy, że funkcja, gdzie w pewnym momencie wykonujemy zaokrąglenie, zwraca nam ponad czterokrotnie większy wynik niż funkcja bez żadnych modyfikacji. Korzystając z arytmetyki *Float64* otrzymujemy wynik który jest ponad 20-krotnie mniejszy od tego otrzymanego z obliczeń we *Float32*.

5.4 Wnioski

Dla złożonych obliczeń dokładna reprezentacja liczb ma duże znaczenie, gdyż już we wczesnych iteracjach małe różnice mogą mieć duży wpływ na ostateczny wynik.

6 Zadanie 6

6.1 O zadaniu

Rozważamy równanie rekurencyjne:

$$x_{n+1} = x_n^2 + c$$

dla $n = 0, 1, ...$, gdzie c jest pewną daną stałą.

Mamy wykonać testy dla następujących danych:

1. $c = -2, x_0 = 1$
2. $c = -2, x_0 = 2$

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

Tabela 5: Wyniki dla zadania 4

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. $c = -1, x_0 = 0.25$

w arytmetyce *Float64*, dla 40 iteracji naszego wyrażenia. Spróbujemy zaobserwować zachowanie generowanych ciągów.

6.2 Rozwiązanie

Prosta implementacja podanego równania w języku `Julia` zwraca nam kolejne wartości ciągów, które następnie przedstawiam na wykresie korzystając z paczki `Plots` i backendu `PyPlot`.

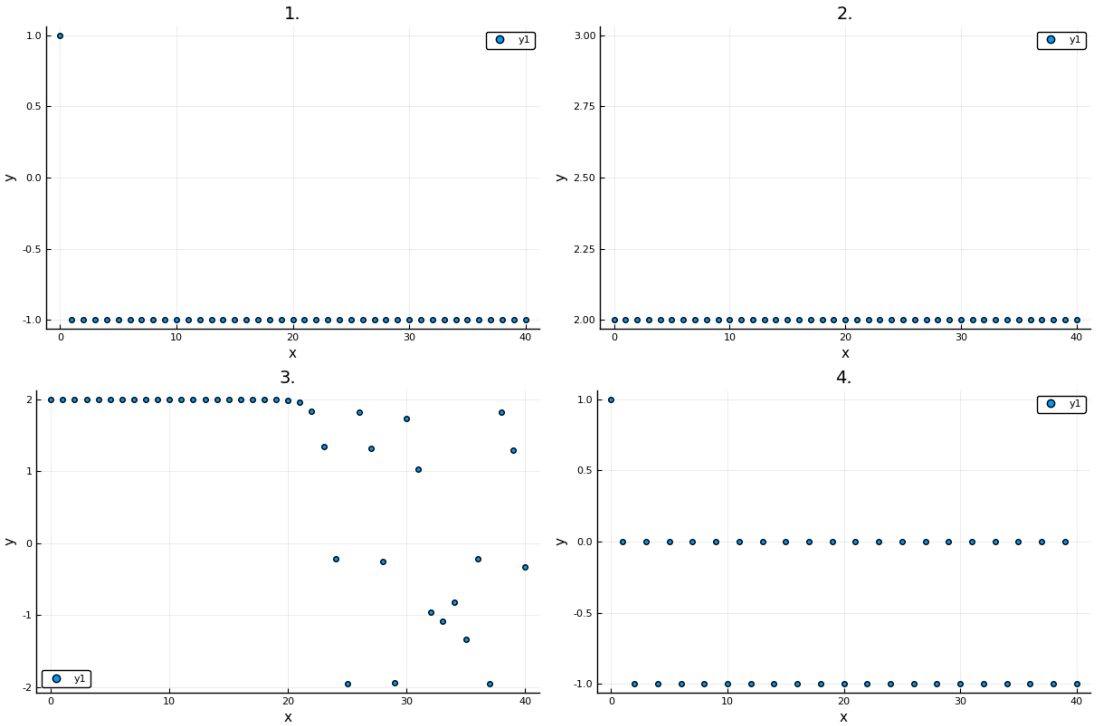
k	z_k	$\lvert P(z_k)\rvert$	$\lvert z_k - k\rvert$
1	0.9999999999998357 + 0.0im	20992.0	1.6431300764452317e-13
2	2.0000000000550373 + 0.0im	349184.0	5.503730804434781e-11
3	2.99999999660342 + 0.0im	2.221568e6	3.3965799062229962e-9
4	4.000000089724362 + 0.0im	1.046784e7	8.972436216225788e-8
5	4.99999857388791 + 0.0im	3.9463936e7	1.4261120897529622e-6
6	6.000020476673031 + 0.0im	1.29148416e8	2.0476673030955794e-5
7	6.99960207042242 + 0.0im	3.88123136e8	0.00039792957757978087
8	8.007772029099446 + 0.0im	1.072547328e9	0.007772029099445632
9	8.915816367932559 + 0.0im	3.065575424e9	0.0841836320674414
10	10.095455630535774 - 0.6449328236240688im	7.143113638035824e9	0.6519586830380406
11	10.095455630535774 + 0.6449328236240688im	7.143113638035824e9	1.1109180272716561
12	11.793890586174369 - 1.6524771364075785im	3.357756113171857e10	1.665281290598479
13	11.793890586174369 + 1.6524771364075785im	3.357756113171857e10	2.045820276678428
14	13.992406684487216 - 2.5188244257108443im	1.0612064533081976e11	2.5188358711909045
15	13.992406684487216 + 2.5188244257108443im	1.0612064533081976e11	2.7128805312847097
16	16.73074487979267 - 2.812624896721978im	3.315103475981763e11	2.9060018735375106
17	16.73074487979267 + 2.812624896721978im	3.315103475981763e11	2.825483521349608
18	19.5024423688181 - 1.940331978642903im	9.539424609817828e12	2.454021446312976
19	19.5024423688181 + 1.940331978642903im	9.539424609817828e12	2.004329444309949
20	20.84691021519479 + 0.0im	1.1144535045l2e13	0.8469102151947894

Tabela 6: Wyniki dla zmienionego wielomianu z zadania 4

	Wynik
Funkcja obliczana normalnie	0.25860548
Funkcja obliczana z zaokrągleniem	1.093568

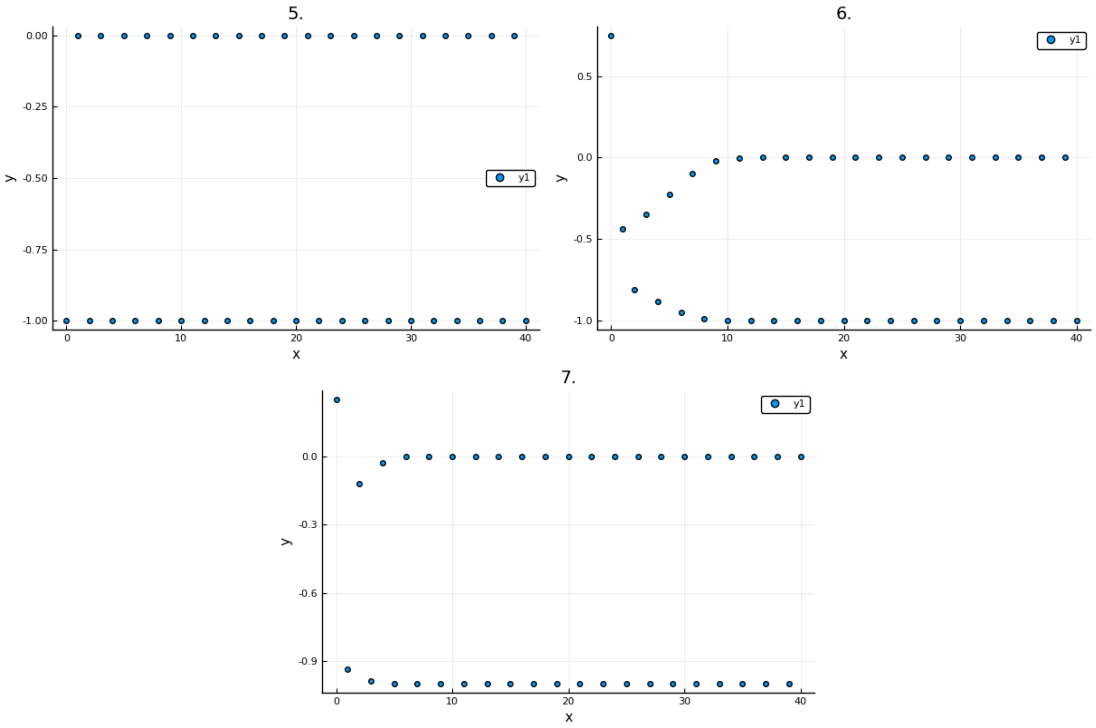
Tabela 7: Wyniki dla podpunktu 1. zadania 6

6.3 Wyniki



	Wynik
Float32	0.25860548
Float64	0.011611238029748606

Tabela 8: Wyniki dla podpunktu 2. zadania 6



Dla wykresów 1, 2, 4 oraz 5 otrzymujemy wartości całkowite zgodne z naszymi oczekiwaniami. W przykładzie 3 po przekroczeniu pewnego momentu wartości się rozbiegają. W przykładach 6 i 7 po pewnym czasie otrzymujemy po dwa podciągi zbieżne do 0 i -1.

6.4 Wnioski

Po przykładzie 3 widzimy, że mała zmiana wartości, która będzie kumulowana, może doprowadzić do otrzymania rozbieżnych wyników. W przykładach 6 i 7 przez działanie na małych liczbach doświadczamy niestabilności numerycznej.