Obliczenia naukowe, sprawozdanie z listy 2

Data Paweł 250105

November 8, 2020

1 Zadanie

Zadanie polega na policzeniu iloczynu skalarnego dwoch wektorów na 4 różne sposoby. Raz dla domyslnych danych, drugi raz dla zmienionych danych (2 liczby zmieniły się na 10 miejscu po przecinku)

W tabeli znajdują się wyniki domyślnych danych:

sposób	Float32	Float64
1	-0.2499442995	1.0251881368e-10
2	-0.2043457031	-1.5643308870e-10
3	-0.2500000000	0.0000000000e+00
4	-0.2500000000	0.0000000000e+00

W tej tabeli są wyniki dla zmienionych danych:

sposób	Float32	Float64
1	-0.4999442995	-4.2963427399e-03
2	-0.4543457031	-4.2963429987e-03
3	-0.5000000000	-4.2963428423e-03
4	-0.5000000000	-4.2963428423e-03

W typie Float32 wyniki się nie zmieniły. Jednak widać sporą różnicę przy liczeniu w typie Float64.

Okazuje się, że wartości x4

0.577215664

0.5772156649

We Float32 są zapisane tak samo (mają tą samą najbliższą wartość maszynową), a wartości x5

$$0.301029995$$

$$0.3010299957$$

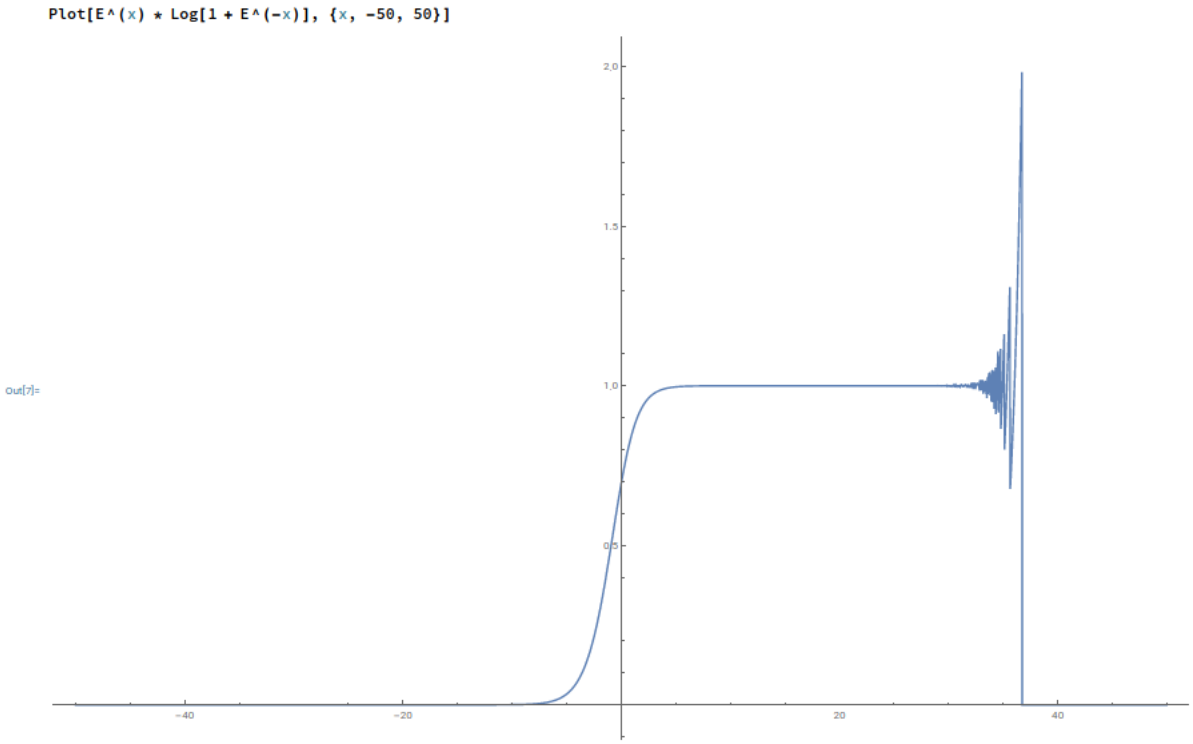
Różnią się o ostatni znak mantysy. Zatem wynik się nie zmienił. We Float64 wynik się zmienił, ponieważ mamy wyraźnie większą mantysę. Algorytmy są wrażliwe na niewielkie miany danych, zatem zadanie jest źle uwarunkowane.

2 Zadanie

Narysowałem wykres funkcji

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

w dwóch programach: Mathematica oraz Desmos.



W przedziale $x \in [30, 40]$ wykresy się różnią i nie są do końca prawdziwe. By zweryfikować wykresy, policzyłem granicę funkcji w nieskończoności:

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

Zatem oba programy nie są odporne na błędy obliczeń. Dla dużych wartości x wyrażenie $1 + e^{-x}$ wynosi 1. Skoro dwa popularne programy nie poradziły sobie z tym problemem można stwierdzić, że jest on trudny bądź niemożliwy do uniknięcia.

3 Zadanie

Zadanie polega na policzeniu błędów w rozwiązywaniu układów liniowych.

Macierz generujemy na 2 sposoby:

- macierz hilberta
- macierz losowa o konkretnym współczynniku stopniu uwarunkowania

Rozwiązanie również liczymy na 2 sposoby

- eliminacją Gaussa
- mnożenie przez macierz odwrotną

W poniższych tabelach są pokazane błędy względne
Macierz Hilberta:

Rozmiar	cond(A)	rank(A)	Gauss	Mnożenie odwrotne
1	1.0	1	0.0	0.0
2	19.28147006790397	2	5.661048867003676e-16	1.4043333874306803e-15
3	524.0567775860644	3	8.022593772267726e-15	0.0
4	15513.73873892924	4	4.137409622430382e-14	0.0
5	476607.25024259434	5	1.6828426299227195e-12	3.3544360584359632e-12
6	1.4951058642254665e7	6	2.618913302311624e-10	2.0163759404347654e-10
7	4.75367356583129e8	7	1.2606867224171548e-8	4.713280397232037e-9
8	1.5257575538060041e10	8	6.124089555723088e-8	3.07748390309622e-7
9	4.931537564468762e11	9	3.8751634185032475e-6	4.541268303176643e-6
10	1.6024416992541715e13	10	8.67039023709691e-5	0.0002501493411824886
11	5.222677939280335e14	10	0.00015827808158590435	0.007618304284315809
12	1.7514731907091464e16	11	0.13396208372085344	0.258994120804705
13	3.344143497338461e18	11	0.11039701117868264	5.331275639426837
14	6.200786263161444e17	11	1.4554087127659643	8.71499275104814
15	3.674392953467974e17	12	4.696668350857427	7.344641453111494
16	7.865467778431645e17	12	54.15518954564602	29.84884207073541
17	1.263684342666052e18	12	13.707236683836307	10.516942378369349
18	2.2446309929189128e18	12	9.134134521198485	7.575475905055309
19	6.471953976541591e18	13	9.720589712655698	12.233761393757726
20	1.3553657908688225e18	13	7.549915039472976	22.062697257870493

Macierz losowa:

Rozmiar	cond(A)	c	błąd - Gauss	błąd - Mnożenie odwrotne
5	1.0000000000000004	1	1.3136335981433191e-16	1.1102230246251565e-16
5	9.999999999999991	10	4.657646926676369e-16	3.475547814546182e-16
5	999.9999999999277	10^3	3.379209585970305e-14	3.0975182594076356e-14
5	9.999999996073918e6	10^7	2.589576407235501e-10	3.0690399758666906e-10
5	9.999490255717208e11	10^{12}	1.5743573043284717e-5	1.2536374454616533e-5
5	5.688093730857876e15	10^{16}	0.23392472579874266	0.2404423007708918
10	1.0000000000000007	1	3.5108334685767007e-16	2.895107444979072e-16
10	10.000000000000002	10	2.0471501066083614e-16	1.9229626863835638e-16
10	1000.0000000000326	10^3	1.3055769609472668e-14	1.2639780648013057e-14
10	1.0000000009922506e7	10^7	3.8645090555330353e-10	3.9685371678287364e-10
10	1.0000149423202512e12	10^{12}	1.9269741388083112e-5	2.101764917042606e-5
10	1.3005256883026896e16	10^{16}	0.25460088962530575	0.24903990448006016
20	1.000000000000001	1	4.489201592495935e-16	4.0792198665315547e-16
20	9.999999999999995	10	7.368569535002363e-16	5.093734210850115e-16
20	1000.000000000027	10^3	1.850518991434869e-14	1.8016973444998713e-14
20	9.99999999431808e6	10^7	1.0179029517031063e-10	5.059930268268656e-11
20	9.999851591910955e11	10^{12}	7.5911149340471105e-6	1.700644922268582e-5
20	7.645175564580154e15	10^{16}	0.04802778565364382	0.031985962649897295

Dla macierzy hilberta błąd eliminacji Gaussa jeest mniejszy niż bląd liczenia przy pomocy macierzy odwrotnej.

Zadanie dla macierzy Hilberta jest źle uwarunkowane, ponieważ wyniki bardzo różnią się od prawdziwych. Widać to również po wskaźniku uwarunkowania.

4 Zadanie

Zadanie polega na policzeniu miejsc zerowych wielomianu w postaci kanonicznej i sprawdzenie ich wartości na tym wielomianie zapisanym na dwa sposoby: w postaci kanonicznej (oznaczonej jako P) oraz iloczynowej (małe p).

Obliczenia robiłem na tzw. wielomianie Wilkinsona (ma 20 miejsc zerowych, od 1 do 20).

Wyniki:

i	x_i	$\lvert P(x_i)\rvert$	$\lvert p(x_i)\rvert$	$\lvert x_i - i\rvert$
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

Powtórzyłem eksperyment, zmieniając jeden współczynik o małą wartość (z -210 na $-210 - 2^{-23}$)

Wyniki:

i	x_i	$\lvert P(x_i)\rvert$	$\lvert p(x_i)\rvert$	$\lvert x_i - i\rvert$
1	0.9999999999998357 + 0.0im	20992.0	22016.0	1.6431300764452317e-13
2	2.0000000000550373 + 0.0im	349184.0	365568.0	5.503730804434781e-11
3	2.99999999660342 + 0.0im	2.221568e6	2.295296e6	3.3965799062229962e-9
4	4.000000089724362 + 0.0im	1.046784e7	1.0729984e7	8.972436216225788e-8
5	4.99999857388791 + 0.0im	3.9463936e7	4.3303936e7	1.4261120897529622e-6
6	6.000020476673031 + 0.0im	1.29148416e8	2.06120448e8	2.0476673030955794e-5
7	6.99960207042242 + 0.0im	3.88123136e8	1.757670912e9	0.00039792957757978087
8	8.007772029099446 + 0.0im	1.072547328e9	1.8525486592e10	0.007772029099445632
9	8.915816367932559 + 0.0im	3.065575424e9	1.37174317056e11	0.0841836320674414
10	10.095455630535774 - 0.6449328236240688im	7.143113638035824e9	1.4912633816754019e12	0.6519586830380407
11	10.095455630535774 + 0.6449328236240688im	7.143113638035824e9	1.4912633816754019e12	1.1109180272716561
12	11.793890586174369 - 1.6524771364075785im	3.357756113171857e10	3.2960214141301664e13	1.665281290598479
13	11.793890586174369 + 1.6524771364075785im	3.357756113171857e10	3.2960214141301664e13	2.0458202766784277
14	13.992406684487216 - 2.5188244257108443im	1.0612064533081976e11	9.545941595183662e14	2.518835871190904
15	13.992406684487216 + 2.5188244257108443im	1.0612064533081976e11	9.545941595183662e14	2.7128805312847097
16	16.73074487979267 - 2.812624896721978im	3.3151034759817633e11	2.7420894016764064e16	2.9060018735375106
17	16.73074487979267 + 2.812624896721978im	3.3151034759817633e11	2.7420894016764064e16	2.825483521349608
18	19.5024423688181 - 1.940331978642903im	9.539424609817828e12	4.2525024879934694e17	2.4540214463129764
19	19.5024423688181 + 1.940331978642903im	9.539424609817828e12	4.2525024879934694e17	2.0043294443099486
20	20.84691021519479 + 0.0im	1.114453504512e13	1.3743733197249713e18	0.8469102151947894

Jak widać w obu tabelach, wyniki są dalekie od zera, są nawet rzędu 10^{13}. Wartosci samych błędów miejsc zerowych nie są duże, jednak w czasie liczenia wartości funkcji w punkcie wykonje się wiele obliczeń, przez co błąd rośnie.

Błęde wyniki są spowodowane ograniczonym miejscem na przechowywanie liczb w komputerze, dla Float64 mamy 15-17 miejsc po przecinku w zapisie dziesiętnym.

W drugim eksperymencie widzimy, że niewielka zmiana o 2^{-23} powoduje, że miejscami zerowymi są liczby zespolone. Zatem eksperyment jest źle uwarunkowany.

5 Zadanie

Zadanie polega na policzeniu 40 pierwszych elementów ciagu z pewnymi założeniami.

Ciąg:

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

Dla danych $p_0 = 0.01$ i $r = 3$ mamy policzyć pierwsze 40 elementów 3 sposobami:

- używając Float32
- używajac Float32 i przy 10 iteracji obciąć wynik do 3 miejsc po przecinku
- używajac Float64

W pierwszej tabeli są Wyniki dla obu Float32, w nastepnej dla Float32 i Float64.

i	Float32	Float32 z obięciem
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

i	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
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

Każda metoda zwróciła inny wynik. Zatem widzimy, że małe zaniedbanie wyniku pośredniego może znacząco zmienić wynik końcowy. Warto zwrócić uwagę na znaczną różnice między wynikiem w Float32 a Float64.

6 Zadanie

Zadanie polega na policzeniu 40 pierwszych wartości ciągu

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

dla Float64 i dla:

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 $x_0 = 0.25$

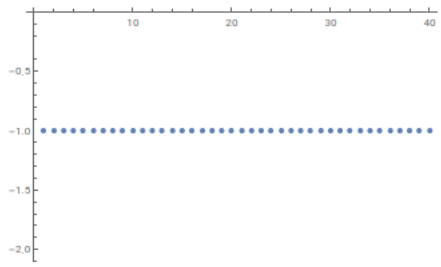
Poniżej są graficzne wyniki (wykonane w programie Mathematica)

Ciągi 1 oraz 2 są stałe. Wartość początkowa ciągu 3 jest niemal taka sama, jak ciągu 2, jednak od n $>$ 20 jego wartości wyglądają na losowe.

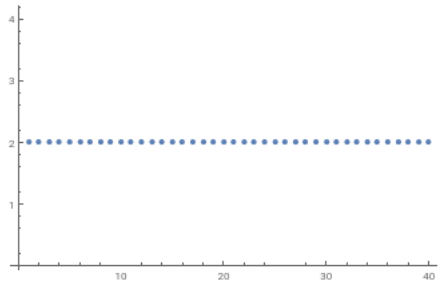
Ciągi 4 oraz 5 są poza zerową wartością są takie same. Ciągi 6 oraz 7 na początku mają inne wartości, lecz później zbiegają do tych samych wartości, co ciągi 4 i 5.

Tu również można stwierdzić, że mała zmiana początkowych wartosci może znacznie zmienić wyniki końcowe (ciąg 3). Jest to tak zwana numeryczna niestabilność. Jednak nie zawsze tak się dzieje, co widać na przykładzie ciągów 6 i 7.

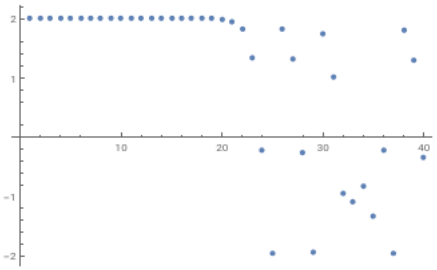
1


10
20
30
40
−0.5
−1.0
−1.5
−2.0

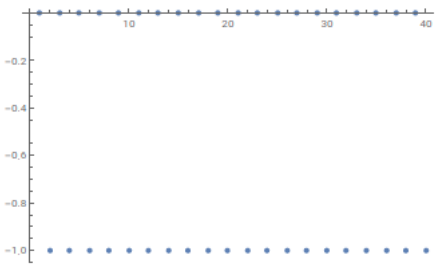
2


4
3
2
1
10
20
30
40

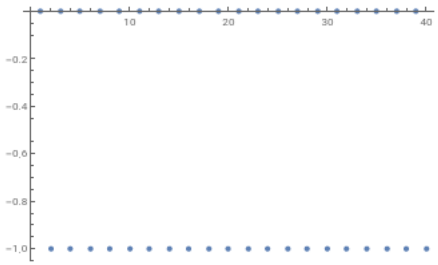
3


2
1
10
20
30
40
−1
−2

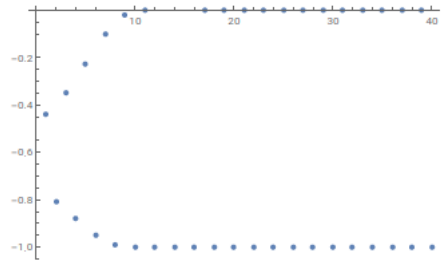
4


10
20
30
40
−0.2
−0.4
−0.6
−0.8
−1.0

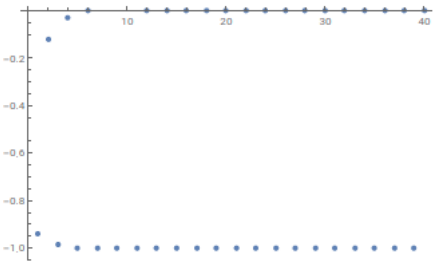
5


10
20
30
40
−0.2
−0.4
−0.6
−0.8
−1.0

6


10
20
30
40
−0.2
−0.4
−0.6
−0.8
−1.0

7


10
20
30
40
−0.2
−0.4
−0.6
−0.8
−1.0