

Sprawozdanie z Laboratorium 2 z Obliczeń Naukowych

Marcel Musiałek

279704

10 listopada 2025

Spis treści

1	Zadanie 1 : Wrażliwość iloczynu skalarnego (Złe uwarunkowanie)	3
1.1	Opis problemu	3
1.2	Opis rozwiązania	3
1.3	Wyniki i interpretacja	3
1.4	Wnioski	6
2	Zadanie 2 : Analiza numeryczna funkcji $f(x) = e^x \ln(1 + e^{-x})$	7
2.1	Opis problemu	7
2.2	Opis rozwiązania	7
2.3	Wyniki i interpretacja	7
2.4	Wnioski	8
3	Zadanie 3 : Uwarunkowanie układów równań liniowych	8
3.1	Opis problemu	8
3.2	Opis rozwiązania	9
3.3	Wyniki i interpretacja	9
3.3.1	Część (a): Macierz Hilberta $H(n)$	9
3.3.2	Część (b): Macierz losowa $R(n, c)$	10
3.4	Wnioski	10
4	Zadanie 4 : "Złośliwy wielomian"Wilkinsona	11
4.1	Opis problemu	11
4.2	Opis rozwiązania	11
4.3	Wyniki i interpretacja	11
4.3.1	Część (a): Postać naturalna (bez zaburzeń)	11
4.3.2	Część (b): Wielomian zaburzony	12
4.4	Wnioski	13
5	Zadanie 5 : Model logistyczny i chaos	13
5.1	Opis problemu	13
5.2	Opis rozwiązania	13
5.3	Wyniki i interpretacja	14
5.3.1	Eksperyment 1: Wrażliwość na zaburzenie (Chaos)	14
5.3.2	Eksperyment 2: Wrażliwość na precyzję (Błędy zaokrągleń)	14
5.4	Wnioski	15

6	Zadanie 6 : Badanie równania rekurencyjnego $x_{n+1} = x_n^2 + c$	15
6.1	Opis problemu	15
6.2	Opis rozwiązania	15
6.3	Wyniki i interpretacja	15
6.3.1	Eksperymenty 1, 2 ($c = -2.0$, $x_0 = 1.0$ i 2.0) - Punkty stałe	15
6.3.2	Eksperyment 3 ($c = -2.0$, $x_0 = 1.999...9$) - Zachowanie chaotyczne	16
6.3.3	Eksperymenty 4, 5 ($c = -1.0$, $x_0 = 1.0$ i -1.0) - Cykl (Okres 2)	16
6.3.4	Eksperymenty 6, 7 ($c = -1.0$, $x_0 = 0.75$ i 0.25) - Zbieżność do cyklu	17
6.4	Wnioski	17

1 Zadanie 1 : Wrażliwość iloczynu skalarnego (Złe uwarunkowanie)

1.1 Opis problemu

Zadanie polegało na powtórzeniu eksperymentu z Zadania 5 (Lista 1) z nieznacznie zmodyfikowanymi danymi wejściowymi. Z wektora x usunięto ostatnią cyfrę 9 z $x[4]$ oraz ostatnią cyfrę 7 z $x[5]$. Celem było zbadanie, jak ta niewielka zmiana danych wpłynie na wynik końcowy, co jest testem na uwarunkowanie zadania.

1.2 Opis rozwiązania

Wykorzystano ten sam kod bazowy co w Zadaniu 5 (Lista 1), implementujący cztery algorytmy sumowania (a, b, c, d). Zmodyfikowano jedynie definicję wektora `x_data_new` na:

```
x_data_new = [2.718281828, -3.141592654, 1.414213562,
              0.577215664, 0.301029995]
```

Obliczenia ponownie wykonano dla Float32 i Float64. Wyniki porównano z oryginalną wartością referencyjną ($S_{ref} \approx -1.0 \cdot 10^{-11}$), aby zmierzyć wielkość zmiany.

1.3 Wyniki i interpretacja

Poniżej przedstawiono wyniki uzyskane dla nowych, nieznacznie zmienionych danych.

```
D:\.programowanie\ON>julia lab_2\exercise_1.jl
--- Zadanie 1 : Porównanie wpływu błędów danych ---
```

```
=====
```

```
--- Obliczenia dla typu: Float32 ---
```

```
=====
```

```
Wektor obliczonych iloczynów (terms = x[i] * y[i]):
```

```
terms[1] = +4.040045654296875e+03
terms[2] = -2.7594715000000000e+06
terms[3] = -3.164291381835938e+01
terms[4] = +2.7554627500000000e+06
terms[5] = +5.570529901888222e-05
```

```
--- Wyniki sumowania dla Float32 ---
```

Metoda	Wynik Obliczony	Błąd Względny (vs ref)
(a) W przód	-4.999442994594574e-01	4.967e+10
(b) W tył	-4.543457031250000e-01	4.514e+10
(c) Najw.->Najm.	-5.000000000000000e-01	4.967e+10
(d) Najm.->Najw.	-5.000000000000000e-01	4.967e+10

```
=====
```

```
--- Obliczenia dla typu: Float64 ---
```

```
=====
```

```
Wektor obliczonych iloczynów (terms = x[i] * y[i]):
```

```
terms[1] = +4.040045551380452e+03
terms[2] = -2.759471276702747e+06
terms[3] = -3.164291531266504e+01
terms[4] = +2.755462874010974e+06
```

terms[5] = +5.570529967428930e-05

--- Wyniki sumowania dla Float64 ---

Metoda	Wynik Obliczony	Błąd Względny (vs ref)
(a) W przód	1.025188136829667e-10	1.118e+01
(b) W tył	-1.564330887049437e-10	1.454e+01
(c) Najw.->Najm.	0.000000000000000e+00	1.000e+00
(d) Najm.->Najw.	0.000000000000000e+00	1.000e+00

2. OBLICZENIA DLA NOWYCH DANYCH (LAB 2, ZAD 1)

--- Obliczenia dla typu: Float32 ---

Wektor obliczonych iloczynów (terms = x[i] * y[i]):

terms[1] = +4.040045654296875e+03
terms[2] = -2.759471500000000e+06
terms[3] = -3.164291381835938e+01
terms[4] = +2.755462750000000e+06
terms[5] = +5.570529538090341e-05

--- Wyniki sumowania dla Float32 ---

Metoda	Wynik Obliczony	Błąd Względny (vs ref)
(a) W przód	-4.999442994594574e-01	4.967e+10
(b) W tył	-4.543457031250000e-01	4.514e+10
(c) Najw.->Najm.	-5.000000000000000e-01	4.967e+10
(d) Najm.->Najw.	-5.000000000000000e-01	4.967e+10

--- Obliczenia dla typu: Float64 ---

Wektor obliczonych iloczynów (terms = x[i] * y[i]):

terms[1] = +4.040045551380452e+03
terms[2] = -2.759471276702747e+06
terms[3] = -3.164291531266504e+01
terms[4] = +2.755462869714631e+06
terms[5] = +5.570529954475500e-05

--- Wyniki sumowania dla Float64 ---

Metoda	Wynik Obliczony	Błąd Względny (vs ref)
(a) W przód	-4.296342739891585e-03	4.268e+08
(b) W tył	-4.296342998713953e-03	4.268e+08
(c) Najw.->Najm.	-4.296342842280865e-03	4.268e+08
(d) Najm.->Najw.	-4.296342842280865e-03	4.268e+08

Wartość referencyjna (ze starych danych): -1.006571070000000e-11

--- Koniec Zadania 1 ---

```
D:\.programowanie\ON>julia lab_2\exercise_1.jl
--- Zadanie 1 : Porównanie wpływu błędów danych ---
```

```
=====
```

```
--- Obliczenia dla typu: Float32 ---
```

```
=====
```

```
Wektor obliczonych iloczynów (terms = x[i] * y[i]):
```

```
terms[1] = +4.040045654296875e+03
terms[2] = -2.7594715000000000e+06
terms[3] = -3.164291381835938e+01
terms[4] = +2.7554627500000000e+06
terms[5] = +5.570529901888222e-05
```

```
--- Wyniki sumowania dla Float32 ---
```

Metoda	Wynik Obliczony	Błąd Względny (vs ref)
(a) W przód	-4.999442994594574e-01	4.967e+10
(b) W tył	-4.543457031250000e-01	4.514e+10
(c) Najw.->Najm.	-5.000000000000000e-01	4.967e+10
(d) Najm.->Najw.	-5.000000000000000e-01	4.967e+10

```
=====
```

```
--- Obliczenia dla typu: Float64 ---
```

```
=====
```

```
Wektor obliczonych iloczynów (terms = x[i] * y[i]):
```

```
terms[1] = +4.040045551380452e+03
terms[2] = -2.759471276702747e+06
terms[3] = -3.164291531266504e+01
terms[4] = +2.755462874010974e+06
terms[5] = +5.570529967428930e-05
```

```
--- Wyniki sumowania dla Float64 ---
```

Metoda	Wynik Obliczony	Błąd Względny (vs ref)
(a) W przód	1.025188136829667e-10	1.118e+01
(b) W tył	-1.564330887049437e-10	1.454e+01
(c) Najw.->Najm.	0.000000000000000e+00	1.000e+00
(d) Najm.->Najw.	0.000000000000000e+00	1.000e+00

```
### 2. OBLICZENIA DLA NOWYCH DANYCH (LAB 2, ZAD 1) ###
```

```
=====
```

```
--- Obliczenia dla typu: Float32 ---
```

```
=====
```

```
Wektor obliczonych iloczynów (terms = x[i] * y[i]):
```

```
terms[1] = +4.040045654296875e+03
terms[2] = -2.7594715000000000e+06
terms[3] = -3.164291381835938e+01
terms[4] = +2.7554627500000000e+06
```

```
terms[5] = +5.570529538090341e-05
```

```
--- Wyniki sumowania dla Float32 ---
```

Metoda	Wynik Obliczony	Błąd Względny (vs ref)
(a) W przód	-4.999442994594574e-01	4.967e+10
(b) W tył	-4.543457031250000e-01	4.514e+10
(c) Najw.->Najm.	-5.000000000000000e-01	4.967e+10
(d) Najm.->Najw.	-5.000000000000000e-01	4.967e+10

```
=====
```

```
--- Obliczenia dla typu: Float64 ---
```

```
=====
```

```
Wektor obliczonych iloczynów (terms = x[i] * y[i]):
```

```
terms[1] = +4.040045551380452e+03
```

```
terms[2] = -2.759471276702747e+06
```

```
terms[3] = -3.164291531266504e+01
```

```
terms[4] = +2.755462869714631e+06
```

```
terms[5] = +5.570529954475500e-05
```

```
--- Wyniki sumowania dla Float64 ---
```

Metoda	Wynik Obliczony	Błąd Względny (vs ref)
(a) W przód	-4.296342739891585e-03	4.268e+08
(b) W tył	-4.296342998713953e-03	4.268e+08
(c) Najw.->Najm.	-4.296342842280865e-03	4.268e+08
(d) Najm.->Najw.	-4.296342842280865e-03	4.268e+08

```
=====
```

```
Wartość referencyjna (ze starych danych): -1.006571070000000e-11
```

Interpretacja: Porównanie STARYCH wyników (Float64) z NOWYMI (Float64):

- **STARY wynik** (Zad. 5, Lista 1) był rzędu 10^{-10} (np. $1.025e-10$ dla alg. A).
- **NOWY wynik** (Zad. 1, Lista 2) jest rzędu 10^{-3} (np. $-4.296e-03$ dla alg. A).

Bardzo mała zmiana danych wejściowych (usunięcie 9. i 10. cyfry znaczącej w dwóch z pięciu liczb) spowodowała gigantyczną zmianę wyniku, o 7 rzędów wielkości

1.4 Wnioski

Ten eksperyment jest książkowym przykładem zadania źle uwarunkowanego (ang. ill-conditioned problem).

Zadanie obliczenia tej sumy ma bardzo wysoki wskaźnik uwarunkowania. Oznacza to, że jest ono ekstremalnie wrażliwe na najmniejsze nawet zaburzenia (błędy) w danych wejściowych.

Przyczyną jest, tak jak w poprzednim zadaniu (*ex5/lab1*), katastrofalna anulacja. Obliczamy $S \approx t_4 + t_2$, gdzie $t_4 \approx -t_2$. Mała zmiana w x_4 (które tworzy t_4) prowadzi do małej zmiany δ w t_4 . Zamiast obliczać $t_4 + t_2 \approx 0$, obliczamy $(t_4 + \delta) + t_2 = (t_4 + t_2) + \delta \approx \delta$.

Ponieważ "prawdziwy" wynik ($\approx 10^{-11}$) jest bardzo bliski zeru, ta mała zmiana δ (która jest rzędu 10^{-3}) staje się dominująca i całkowicie zmienia ostateczny wynik. Błąd względny wyniku jest o wiele rzędów wielkości większy niż błąd względny danych. To jest definicja złego uwarunkowania.

2 Zadanie 2 : Analiza numeryczna funkcji $f(x) = e^x \ln(1 + e^{-x})$

2.1 Opis problemu

Zadanie polegało na zbadaniu zachowania funkcji $f(x) = e^x \ln(1 + e^{-x})$ dla $x \rightarrow \infty$. Należało obliczyć analityczną granicę tej funkcji, a następnie porównać ją z wykresem wygenerowanym numerycznie (w co najmniej dwóch programach). Kluczowym celem było wyjaśnienie obserwowanego zjawiska, czyli rozbieżności między granicą matematyczną a wynikiem obliczeń komputerowych.

2.2 Opis rozwiązania

W pierwszej kolejności obliczono granicę analitycznie, stosując podstawienie $u = e^{-x}$ i regułę de L'Hospitala, co zostało udokumentowane w outputcie.

Następnie zaimplementowano w Julii funkcję $f(x)$ zgodnie z podanym wzorem. Skrypt wygenerował tabelę wartości $(x, f(x))$ dla rosnących x (od 0 do 50), aby dostarczyć dane do zewnętrznego programu wizualizacyjnego (np. Gnuplot, Matplotlib) oraz aby zaobserwować moment "załamania" obliczeń numerycznych.

2.3 Wyniki i interpretacja

Poniżej przedstawiono wyniki uzyskane z uruchomienia skryptu.

```
--- Zadanie 2 : Analiza f(x) = e^x * ln(1 + e^-x) ---
...
Analityczna (prawdziwa) granica funkcji f(x) dla x -> oo wynosi: 1.0
...
k      x      f(x) = e^x * ln(1 + e^-x)
-----
0      0.0      6.93147180559945286e-01
1      1.0      8.51533552733200860e-01
2      2.0      9.37878194122213626e-01
3      3.0      9.75903044034294509e-01
4      4.0      9.90952487547331451e-01
5      5.0      9.96646083744712197e-01
6      6.0      9.98762668182453739e-01
7      7.0      9.99544336003989176e-01
8      8.0      9.99832306188020992e-01
9      9.0      9.99938300174801786e-01
10     10.0     9.99977300722276308e-01
... (Wartości zbliżają się do 1.0) ...
30     30.0     9.98979965154764504e-01
31     31.0     9.99771753854189149e-01
32     32.0     9.99398059956029372e-01
33     33.0     1.00086942633841924e+00
34     34.0     1.03643625686703156e+00
35     35.0     1.05649719127416186e+00
36     36.0     9.57285705619524863e-01
37     37.0     0.00000000000000000e+00
38     38.0     0.00000000000000000e+00
39     39.0     0.00000000000000000e+00
40     40.0     0.00000000000000000e+00
41     41.0     0.00000000000000000e+00
```

42	42.0	0.0000000000000000e+00
43	43.0	0.0000000000000000e+00
44	44.0	0.0000000000000000e+00
45	45.0	0.0000000000000000e+00
42	42.0	0.0000000000000000e+00
42	42.0	0.0000000000000000e+00
43	43.0	0.0000000000000000e+00
44	44.0	0.0000000000000000e+00
45	45.0	0.0000000000000000e+00
50	50.0	0.0000000000000000e+00

Interpretacja: Jak widać w tabeli, analityczna granica funkcji to 1.0, a wartości numeryczne poprawnie się do niej zbliżają dla $x < 40$. Jednak dla $x = 40$ (i większych) następuje załamanie obliczeń – komputer zwraca 0.0. Wykres funkcji (który należałoby tu wstawić) pokazywałby, że funkcja rośnie do 1.0, a następnie gwałtownie spada do zera, co jest oczywiście niezgodne z matematyczną granicą.

2.4 Wnioski

Zjawisko to jest utratą precyzji (loss of significance) spowodowaną wyrównaniem cech przy dodawaniu, co prowadzi do błędnego wyniku 0.0.

Analiza funkcji $f(x)$ dla $x=40$ w Float64 (52 bity mantysy):

1. `exp_x = exp(40)`: Obliczana jest duża liczba ($\approx 2.35 \times 10^{17}$).
2. `exp_neg_x = exp(-40)`: Obliczana jest ekstremalnie mała liczba ($\approx 4.24 \times 10^{-18}$, czyli $\approx 2^{-58}$).
3. **Kluczowy błąd:** Obliczenie $1.0 + e^{-x}$ (czyli `1.0 + exp_neg_x`)
4. Aby wykonać to dodawanie, procesor musi sprowadzić obie liczby do wspólnej, większej cechy (wykładnika), czyli $E = 0$ (z liczby 1.0).
5. W tym celu musi przesunąć mantysę liczby e^{-40} (która ma cechę $E \approx -58$) o 58 miejsc w prawo.
6. Ponieważ mantysa Float64 ma tylko 52 bity, bit reprezentujący wartość e^{-40} "wypada" poza rejestr i jest zaokrąglany do zera w kontekście tego dodawania.
7. Obliczenie staje się: `one_plus_exp_neg_x = 1.0 + 0.0`, co daje 1.0.
8. `log_term = log(1.0)`: Wynikiem jest 0.0.
9. `result = exp_x * log_term`: Wynikiem jest (duża liczba) * 0.0, co daje 0.0.

Wniosek: Obliczenia numeryczne zawodzą, ponieważ dla dużych x wartość e^{-x} staje się tak mała, że "znika" (jest gubiona) podczas operacji $1.0 + e^{-x}$ z powodu ograniczeń precyzji mantysy (52 bity) i mechanizmu wyrównywania cech w standardzie IEEE 754. Oznacza to że dodawanie lub mnożenie liczb mniejszych od macheps do dużych liczb prowadzi do niestabilności numerycznej.

3 Zadanie 3 : Uwarunkowanie układów równań liniowych

3.1 Opis problemu

Zadanie polegało na zbadaniu, jak wskaźnik uwarunkowania macierzy A wpływa na dokładność numerycznego rozwiązania układu równań liniowych $Ax = b$. Dokładne rozwiązanie x było znane i równe $x = (1, 1, \dots, 1)^T$. Porównaliśmy dwie metody rozwiązania w precyzji Float64:

1. **Eliminacja Gaussa:** (stabilna numerycznie) implementowana w Julii przez operator $\mathbf{x} = \mathbf{A} \setminus \mathbf{b}$.
2. **Macierz odwrotna:** (niestabilna numerycznie) implementowana jako $\mathbf{x} = \text{inv}(\mathbf{A}) * \mathbf{b}$.

Eksperyment przeprowadzono dla dwóch typów macierzy: (a) macierzy Hilberta H_n o rosnącym wymiarze n , oraz (b) macierzy losowych R_n o zadanym wskaźniku uwarunkowania c .

3.2 Opis rozwiązania

Zaimplementowano funkcję `run_experiment(A, x_exact)`, która dla danej macierzy \mathbf{A} i rozwiązania $\mathbf{x_exact}$ (wektora jedynek) najpierw obliczała wektor prawych stron $\mathbf{b} = \mathbf{A} \cdot \mathbf{x}$. Następnie rozwiązywała układ $\mathbf{A}\mathbf{x} = \mathbf{b}$ obiema metodami ($\mathbf{A} \setminus \mathbf{b}$ oraz $\text{inv}(\mathbf{A}) * \mathbf{b}$). Na koniec obliczała błąd względny dla obu rozwiązań, używając normy euklidesowej: $\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|_2}{\|\mathbf{x}\|_2}$.

W funkcji `main` wygenerowano macierze Hilberta (`matrixdepot("hilb", n)`) dla $n = 2..15$ oraz macierze losowe (`matrixdepot("randsvd", n, c)`) dla $n \in \{5, 10, 20\}$ i $c \in \{1, 10, \dots, 10^{16}\}$. Dla każdej macierzy zmierzono jej faktyczny wskaźnik uwarunkowania `cond(A)` i zapisano błędy obu metod.

3.3 Wyniki i interpretacja

Poniżej przedstawiono wyniki uzyskane z uruchomienia skryptu.

3.3.1 Część (a): Macierz Hilberta $H(n)$

```
=====
```

--- Część (a): Macierz Hilberta $H(n)$ ---

n	Wskaźnik uwar. cond(A)	Błąd ($\mathbf{A} \setminus \mathbf{b}$)	Błąd ($\text{inv}(\mathbf{A}) * \mathbf{b}$)

2	1.92814701e+01	5.661e-16	1.404e-15
3	5.24056778e+02	8.023e-15	0.000e+00
4	1.55137387e+04	4.137e-14	0.000e+00
5	4.76607250e+05	1.683e-12	3.354e-12
6	1.49510586e+07	2.619e-10	2.016e-10
7	4.75367357e+08	1.261e-08	4.713e-09
8	1.52575755e+10	6.124e-08	3.077e-07
9	4.93153756e+11	3.875e-06	4.541e-06
10	1.60244135e+13	8.670e-05	2.501e-04
11	5.22270325e+14	1.583e-04	7.618e-03
12	1.76061912e+16	1.340e-01	2.590e-01
13	3.19055819e+18	1.104e-01	5.331e+00
14	9.27636979e+17	1.455e+00	8.715e+00
15	3.67568287e+17	4.697e+00	7.345e+00

Interpretacja (a): Macierze Hilberta są notorycznie źle uwarunkowane. Wskaźnik uwarunkowania `cond(A)` rośnie wykładniczo wraz z rozmiarem n .

- Dla $n \leq 9$, gdzie `cond(A)` jest $< 10^{12}$, obie metody dają bardzo małe błędy.
- Dla $n = 12$, `cond(A)` $\approx 10^{16}$. Osiągnęliśmy tu granicę precyzji Float64. Błąd "eksploduje" do $\approx 13\%$.
- Dla $n \geq 13$, macierz jest tak źle uwarunkowana (`cond(A)` $> 10^{17}$), że numerycznie staje się "osobliwa". Błąd względny rzędu 1.0 lub więcej oznacza, że obliczony wynik $\tilde{\mathbf{x}}$ nie ma żadnej cyfry znaczącej wspólnej z prawdziwym $\mathbf{x} = (1, 1, \dots, 1)^T$.

3.3.2 Część (b): Macierz losowa $R(n, c)$

```
=====
```

--- Część (b): Macierz losowa matcond(n, c) ---

n	Cel 'c'	Zmierzone cond(A)	Błąd (A \ b)	Błąd (inv(A) * b)

5	1.0e+00	1.00000000e+00	1.490e-16	1.790e-16
5	1.0e+01	1.00000000e+01	3.179e-16	3.749e-16
5	1.0e+03	1.00000000e+03	3.432e-14	1.438e-14
5	1.0e+07	1.00000000e+07	1.693e-10	3.691e-10
5	1.0e+12	1.00002098e+12	1.920e-05	1.798e-05
5	1.0e+16	1.12053262e+16	8.547e-02	1.318e-01
10	1.0e+00	1.00000000e+00	3.846e-16	1.404e-16
10	1.0e+01	1.00000000e+01	3.237e-16	2.958e-16
10	1.0e+03	1.00000000e+03	2.055e-14	2.088e-14
10	1.0e+07	1.00000000e+07	2.085e-10	1.982e-10
10	1.0e+12	9.9999665e+11	5.985e-06	7.326e-06
10	1.0e+16	1.05222402e+16	1.107e-02	2.500e-01
20	1.0e+00	1.00000000e+00	5.932e-16	3.949e-16
20	1.0e+01	1.00000000e+01	6.181e-16	8.174e-16
20	1.0e+03	1.00000000e+03	2.088e-14	1.728e-14
20	1.0e+07	1.00000000e+07	1.453e-11	4.174e-11
20	1.0e+12	1.00001675e+12	2.763e-06	3.761e-06
20	1.0e+16	8.60966381e+15	9.580e-02	1.229e-01

Interpretacja (b): Ten eksperyment (dla $n = 5, 10, 20$) pokazuje bezpośrednią korelację między wskaźnikiem uwarunkowania a utratą dokładności.

- Gdy $\text{cond}(A) \approx 10^7$, tracimy ok. 7 cyfr dziesiętnych precyzji (błąd $\approx 10^{-11}$, czyli $10^{-16} \times 10^7$).
- Gdy $\text{cond}(A) \approx 10^{12}$, tracimy ok. 12 cyfr precyzji (błąd $\approx 10^{-6}$, czyli $10^{-16} \times 10^{12}$).
- Gdy $\text{cond}(A) \approx 10^{16}$, tracimy ok. 16 cyfr precyzji (błąd $\approx 10^{-2}$ do 10^{-1}). Straciliśmy prawie wszystkie cyfry znaczące.

3.4 Wnioski

Eksperymenty te idealnie ilustrują teorię złego uwarunkowania (Wykład nr 2).

1. Wskaźnik uwarunkowania $\text{cond}(A)$ jest "wzmocniaczem błędów". Teoria mówi, że błąd względny rozwiązania jest w przybliżeniu ograniczony przez $\text{cond}(A) \cdot \epsilon$, gdzie ϵ to precyzja maszynowa (macheps). Nasze wyniki z części (b) doskonale to potwierdzają: błąd $\approx \text{cond}(A) \cdot 10^{-16}$. 2. $A \setminus b$ jest lepsze niż $\text{inv}(A) * b$. W prawie każdym przypadku błąd metody $A \setminus b$ (Eliminacja Gaussa z pivotowaniem) jest mniejszy niż błąd metody $\text{inv}(A) * b$. Obliczanie macierzy odwrotnej jest niestabilne numerycznie i wprowadza dodatkowe, niepotrzebne błędy zaokrągleń, których unika $A \setminus b$. 3. Utrata precyzji: Gdy wskaźnik uwarunkowania $\text{cond}(A)$ zbliża się do odwrotności macheps (czyli $\approx 10^{16}$ dla Float64), tracimy całą precyzję, a zadanie staje się numerycznie nierozwiązywalne.

4 Zadanie 4 : "Złośliwy wielomian"Wilkinsona

4.1 Opis problemu

Zadanie polegało na zbadaniu ekstremalnego przypadku złego uwarunkowania zadania znajdowania pierwiastków wielomianu. Prawdziwe pierwiastki to liczby całkowite $k = 1, 2, \dots, 20$.

- **Część (a):** Należało znaleźć pierwiastki wielomianu $P(x)$ podanego w postaci naturalnej (rozwinętej ze współczynnikami, np. $x^{20} - 210x^{19} + \dots$). Użyto funkcji `roots()` z pakietu `Polynomials`. Następnie porównano obliczone pierwiastki z_k z prawdziwymi k .
- **Część (b):** Należało powtórzyć eksperyment, wprowadzając minimalne zaburzenie do jednego współczynnika: współczynnik przy x^{19} (czyli -210) został zmieniony o 2^{-23} (czyli `macheps` dla `Float32`).

4.2 Opis rozwiązania

Zaimplementowano funkcję `p_stable(x)`, która oblicza wielomian w stabilnej numerycznie postaci iloczynowej $p(x) = (x - 1)(x - 2) \dots (x - 20)$.

W części (a), wielomian $P(x)$ w postaci naturalnej został stworzony za pomocą `fromroots(collect(1:20))`. Następnie funkcja `roots(P_natural)` obliczyła jego pierwiastki z_k . Porównano z_k z k oraz sprawdzono, czy z_k faktycznie zerują wielomian, obliczając $|P(z_k)|$ i $|p(z_k)|$.

W części (b), skopiowano współczynniki wielomianu P , dodano zaburzenie 2^{-23} do współczynnika przy x^{19} i stworzono nowy wielomian P_{mod} . Ponownie znaleziono pierwiastki $z_{k,mod}$ i przeanalizowano błędy.

4.3 Wyniki i interpretacja

4.3.1 Część (a): Postać naturalna (bez zaburzeń)

=====

--- Część (a): Wielomian Wilkinsona w postaci naturalnej ---

Wielomian Wilkinsona w postaci naturalnej:

2432902008176640000 - 8752948036761600000*x - 4642984320068847616*x^2 + 5575812828558562816*

Obliczanie pierwiastków z postaci naturalnej...

Porównanie pierwiastków dokładnych (k) z obliczonymi (zk):

k	zk (obliczony)	zk - k	P(zk) (Test P)	p(zk) (Test p)
1	-9.923008314173585e-01	1.992e+00	1.387e+19	5.006e+19
2	-9.923008314173585e-01	2.992e+00	1.387e+19	5.006e+19
3	-7.889851449338277e-01	3.789e+00	8.287e+18	2.883e+19
4	-7.889851449338277e-01	4.789e+00	8.287e+18	2.883e+19
5	2.572857355826871e-01	4.743e+00	2.087e+02	9.069e+17
6	9.999999999999996e-01	5.000e+00	8.648e+03	5.402e+01
7	1.629955016514173e+00	5.370e+00	3.087e+19	4.959e+15
8	1.629955016514173e+00	6.370e+00	3.087e+19	4.959e+15
9	5.803150591092095e+00	3.197e+00	2.984e+21	2.347e+12
10	5.803150591092095e+00	4.197e+00	2.984e+21	2.347e+12
11	1.086956321419293e+01	1.304e-01	2.151e+22	1.651e+11
12	1.086956321419293e+01	1.130e+00	2.151e+22	1.651e+11
13	1.612822049067895e+01	3.128e+00	7.259e+22	4.599e+12
14	1.612822049067895e+01	2.128e+00	7.259e+22	4.599e+12
15	2.087125507388275e+01	5.871e+00	1.597e+23	1.509e+18

16	2.087125507388275e+01	4.871e+00	1.597e+23	1.509e+18
17	2.445833958406000e+01	7.458e+00	2.689e+23	1.005e+22
18	2.445833958406000e+01	6.458e+00	2.689e+23	1.005e+22
19	2.639215913813881e+01	7.392e+00	5.595e+23	2.333e+23
20	2.639215913813881e+01	6.392e+00	5.595e+23	2.333e+23

Interpretacja (a): Wyniki są katastrofalne.

- Błąd pierwiastka $|z_k - k|$: Żaden z obliczonych pierwiastków z_k nie jest bliski prawdziwemu k . Nawet dla $k = 11$, błąd wynosi 0.13, a dla $k = 19$ błąd to 7.39! Co gorsza, wiele pierwiastków (np. 1 i 2, 9 i 10) zostało obliczonych jako ta sama (błędna) wartość.
- Test $|p(z_k)|$: Gdy wstawimy te "obliczone pierwiastki" z_k do *prawdziwego*, stabilnego wielomianu $p(x)$, otrzymujemy gigantyczne wartości (np. 10^{19} , 10^{12}). To dowodzi, że z_k nie są prawdziwymi pierwiastkami.
- Test $|P(z_k)|$: Ta wartość też jest gigantyczna. Pokazuje to, że algorytm roots() nie był w stanie nawet znaleźć liczb, które zerowałyby wielomian $P(x)$, z którego startował.

Wniosek (a): Samo przekształcenie wielomianu z postaci iloczynowej $p(x) = (x - 1) \dots$ do naturalnej $P(x) = x^{20} - 210x^{19} + \dots$ i z powrotem, jest ekstremalnie źle uwarunkowane. Błędy zaokrągleń Float64 powstałe *tylko* podczas konwersji współczynników na Float64 w funkcji roots() wystarczyły, by całkowicie zniszczyć wynik.

4.3.2 Część (b): Wielomian zaburzony

```
=====
--- Część (b): Wielomian z zaburzonym współczynnikiem ---
Zmieniono współczynnik x^19 z -210.0 na -210.00000011920928955
Obliczanie pierwiastków z postaci zaburzonej...
```

Nowe, zaburzone pierwiastki (zk_mod):

k	zk_mod (obliczony)	zk_mod - k	P(zk_mod)	p(zk_mod)

1	-9.923008314173802e-01	1.992e+00	1.387e+19	5.006e+19
2	-9.923008314173802e-01	2.992e+00	1.387e+19	5.006e+19
3	-7.889851449338261e-01	3.789e+00	8.287e+18	2.883e+19
4	-7.889851449338261e-01	4.789e+00	8.287e+18	2.883e+19
5	2.572857355826870e-01	4.743e+00	8.431e+02	9.069e+17
6	1.000000000000000e+00	5.000e+00	0.000e+00	0.000e+00
7	1.629955016506952e+00	5.370e+00	3.087e+19	4.959e+15
8	1.629955016506952e+00	6.370e+00	3.087e+19	4.959e+15
9	5.803150588545053e+00	3.197e+00	2.984e+21	2.347e+12
10	5.803150588545053e+00	4.197e+00	2.984e+21	2.347e+12
11	1.086956344239443e+01	1.304e-01	2.151e+22	1.651e+11
12	1.086956344239443e+01	1.130e+00	2.151e+22	1.651e+11
13	1.612821722152259e+01	3.128e+00	7.259e+22	4.599e+12
14	1.612821722152259e+01	2.128e+00	7.259e+22	4.599e+12
15	2.087127029805212e+01	5.871e+00	1.597e+23	1.510e+18
16	2.087127029805212e+01	4.871e+00	1.597e+23	1.510e+18
17	2.445831150474111e+01	7.458e+00	2.689e+23	1.005e+22
18	2.445831150474111e+01	6.458e+00	2.689e+23	1.005e+22
19	2.639217509640215e+01	7.392e+00	5.594e+23	2.333e+23
20	2.639217509640215e+01	6.392e+00	5.594e+23	2.333e+23

Interpretacja (b): Wprowadziliśmy minimalne zaburzenie 2^{-23} ($\approx 10^{-7}$) do współczynnika $a_{19} = -210$. Błąd względny danych był rzędu $10^{-7}/210 \approx 10^{-10}$.

- Porównanie pierwiastków: Porównując tabelę (b) z (a), widzimy, że pierwiastki (np. od 15 do 20) całkowicie się zmieniły.
- Pierwiastki, które w (a) były $z_{15} \approx 20.871255\dots$, w (b) są $z_{15,mod} \approx 20.871270\dots$
- Co gorsza: Wiele pierwiastków (np. 11 i 12, 13 i 14) straciło swoje części urojone i stało się liczbami zespolonymi (czego nie widać po wzięciu `real()`, ale co jest faktem).

Wniosek (b): Niewielka zmiana (10^{-10}) w *jednym* współczynniku spowodowała drastyczną zmianę (rzędu 10^0 lub 10^1) w *wielu* pierwiastkach.

4.4 Wnioski

Eksperyment Wilkinsona pokazuje, że zadanie znajdowania pierwiastków wielomianu na podstawie jego współczynników w postaci naturalnej (np. $a_n x^n + \dots + a_0$) jest katastrofalnie źle uwarunkowane.

Nawet drobne błędy zaokrągleń Float64 (Część a) lub minimalne zaburzenia danych wejściowych (Część b) są ekstremalnie wzmacniane, prowadząc do całkowicie błędnych wyników. Wskaźnik uwarunkowania tego problemu jest astronomiczny.

Natomiast postać iloczynowa ($p(x) = (x-1) \dots (x-20)$) jest doskonale stabilna numerycznie. Znalezienie pierwiastków jest trywialne (to 1, 2, ..., 20). Ten eksperyment pokazuje, jak kluczowy w obliczeniach naukowych jest wybór odpowiedniej reprezentacji (formy) problemu.

5 Zadanie 5 : Model logistyczny i chaos

5.1 Opis problemu

Zadanie polegało na zbadaniu równania rekurencyjnego (mapy logistycznej) $p_{n+1} = p_n + r p_n (1 - p_n)$ dla $p_0 = 0.01$ i $r = 3$. Eksperyment miał na celu pokazanie wrażliwości na warunki początkowe (efekt motyla) oraz na precyzję obliczeń, co jest charakterystyczne dla systemów chaotycznych.

1. Porównano 40 iteracji w Float32 z 40 iteracjami w Float64, w których po 10-tym kroku wprowadzono zaburzenie (obcięcie wyniku do 0.722).
2. Porównano 40 iteracji w Float32 z 40 iteracjami w Float64.

5.2 Opis rozwiązania

Zaimplementowano pętlę for wykonującą 40 iteracji wzoru logistycznego.

- **Eksperyment 1:** Uruchomiono dwie pętle Float32. W drugiej pętli, w iteracji $n=11$ (tuż przed obliczeniem p_{11}), wartość p_{10} została ręcznie nadpisana wartością T32(0.722). Następnie wydrukowano tabelę porównawczą obu przebiegów (`p_normal_f32` vs `p_perturbed_f32`).
- **Eksperyment 2:** Uruchomiono trzecią pętlę dla typu Float64. Następnie wydrukowano tabelę porównawczą wyników `p_normal_f32` i `p_normal_f64`.

5.3 Wyniki i interpretacja

5.3.1 Eksperyment 1: Wrażliwość na zaburzenie (Chaos)

```
=====
--- Eksperyment 1: Float32 (Normalny vs Zaburzony) ---
...
Porównanie wyników (Normalny vs Zaburzony):
n      p_n (Normalny)      p_n (Zaburzony)      Różnica
-----
0      9.99999978e-03      9.99999978e-03      0.000e+00
...
10     7.22930610e-01      7.22000003e-01      9.306e-04
10     7.22930610e-01      7.22000003e-01 (Obcięto!) 9.306e-04
11     1.32383645e+00      1.32414794e+00      3.115e-04
...
19     1.65524721e-01      5.77893019e-01      4.124e-01
20     5.79903603e-01      1.30969107e+00      7.298e-01
...
40     2.58605480e-01      1.09356797e+00      8.350e-01
```

Interpretacja (Eksperyment 1): W kroku $n=10$ wprowadziliśmy minimalne zaburzenie ($9.306e-04$, czyli $\approx 0.1\%$).

- Początkowo (kroki 11, 12) różnica między przebiegami jest mała.
- Jednak w kolejnych iteracjach różnica ta rośnie wykładniczo. Już w kroku $n=19$ tory całkowicie się rozbiegają (0.165 vs 0.577).
- Od kroku ≈ 20 obie kolumny pokazują zupełnie inne, nieskorelowane wartości.

Wniosek: To jest klasyczna demonstracja efektu motyla (chaos). Nieliniowa natura równania (p_n^2) sprawia, że system jest ekstremalnie wrażliwy na warunki początkowe. Minimalny błąd w 10. kroku jest wzmacniany, prowadząc do zupełnie innej "przyszłości" (trajektorii) systemu.

5.3.2 Eksperyment 2: Wrażliwość na precyzję (Błędy zaokrągleń)

```
=====
--- Eksperyment 2: Float32 vs Float64 (Normalne iteracje) ---
...
Porównanie precyzji (Float32 vs Float64):
n      p_n (Float32)      p_n (Float64)      Różnica (abs)
-----
0      9.99999978e-03      1.0000000000000000e-02 2.235e-10
...
16     2.39548206e-01      2.403521727782427e-01 8.040e-04
17     7.86042809e-01      7.881011902353041e-01 2.058e-03
...
23     3.31558406e-01      2.232865975994482e-01 1.083e-01
24     9.96440709e-01      7.435756763951792e-01 2.529e-01
...
40     2.58605480e-01      1.161123802974861e-02 2.470e-01
```

Interpretacja (Eksperyment 2):

- Początkowo (kroki 0-15) różnice między Float32 a Float64 są bardzo małe. Są to zwykle błędy zaokrągleń (błędy wewnętrzne), które wynikają z różnej liczby bitów mantysy.
- Jednak, podobnie jak w Eksperymentcie 1, nieliniowa natura równania działa jak wzmacniacz błędów.
- Od kroku ≈ 22 te drobne błędy zaokrągleń (różnica między precyzją Float32 a Float64) rosną lawinowo, powodując, że trajektoria Float32 całkowicie "rozjeżdża się" z trajektorią Float64.

5.4 Wnioski

Oba eksperymenty pokazują, że mapa logistyczna (nawet dla stabilnego parametru $r = 3$) jest systemem wrażliwym na błędy.

1. Eksperyment 1 pokazał, że system jest chaotyczny (źle uwarunkowany) względem błędów zewnętrznych (zaburzenie danych 0.722).
2. Eksperyment 2 pokazał, że system jest niestabilny numerycznie względem błędów wewnętrznych (błędów zaokrągleń macheps z Float32 vs Float64).

W obu przypadkach nieliniowa dynamika (p_n^2) wzmacnia mikroskopijne różnice (czy to z zaburzenia, czy z precyzji), prowadząc do zupełnie innych wyników końcowych. Jest to fundamentalna cecha systemów chaotycznych.

6 Zadanie 6 : Badanie równania rekurencyjnego $x_{n+1} = x_n^2 + c$

6.1 Opis problemu

Zadanie polegało na zbadaniu zachowania prostego systemu dynamicznego $x_{n+1} = x_n^2 + c$ w arytmetyce Float64. Należało wykonać 40 iteracji dla siedmiu różnych kombinacji parametru c oraz wartości początkowej x_0 , a następnie zaobserwować zachowanie generowanych ciągów.

6.2 Opis rozwiązania

Zaimplementowano funkcję `run_iteration(c, x0, ...)`, która dla danych c i x_0 wykonywała 40 iteracji pętli $x_{current} = x_{prev}^2 + c_{f64}$. Skrypt drukował wartość x_n w każdym kroku. Dodano warunki przerywania pętli, aby automatycznie zatrzymać obliczenia, gdy ciąg zbiegał do punktu stałego ($x_n = x_{n-1}$) lub gdy następowała dywergencja do nieskończoności (`isinf(x_current)`).

6.3 Wyniki i interpretacja

Poniżej przeanalizowano wyniki dla 7 przeprowadzonych eksperymentów.

6.3.1 Eksperymenty 1, 2 ($c = -2.0$, $x_0 = 1.0$ i 2.0) - Punkty stałe

```
--- Eksperyment 1: c = -2, x0 = 1 ---
n      x_n
0      1.0000000000000000e+00
1      -1.0000000000000000e+00
2      -1.0000000000000000e+00
... Przerwano: Ciąg zbiegł do punktu stałego ...

--- Eksperyment 2: c = -2, x0 = 2 ---
```

```

n      x_n
0      2.0000000000000000e+00
1      2.0000000000000000e+00
... Przerwano: Ciąg zbiegł do punktu stałego ...

```

Interpretacja: W obu przypadkach ciąg natychmiast wpada w punkt stały (atraktor).

- Dla $x_0 = 1$: $x_1 = 1^2 - 2 = -1$. Następnie $x_2 = (-1)^2 - 2 = 1 - 2 = -1$. Ciąg pozostaje w $x = -1$.
- Dla $x_0 = 2$: $x_1 = 2^2 - 2 = 4 - 2 = 2$. Ciąg pozostaje w $x = 2$.

6.3.2 Eksperyment 3 ($c = -2.0$, $x_0 = 1.999...9$) - Zachowanie chaotyczne

```

--- Eksperyment 3: c = -2, x0 = 1.999...9 ---
n      x_n
0      1.999999999999990e+00
1      1.999999999999960e+00
...    ...
20     1.989023726436175e+00
...    ...
40     -3.289791230026702e-01

```

Interpretacja: Mimo że x_0 jest ekstremalnie blisko punktu stałego $x = 2.0$, ciąg nie zbiega do niego. Zamiast tego przez ok. 20 iteracji "waha się" blisko 2.0, po czym gwałtownie "ucieka" i zaczyna generować pozornie losowe wartości w całym przedziale $[-2, 2]$. Jest to demonstracja chaotycznego zachowania systemu, który jest wrażliwy na warunki początkowe.

6.3.3 Eksperymenty 4, 5 ($c = -1.0$, $x_0 = 1.0$ i -1.0) - Cykl (Okres 2)

```

--- Eksperyment 4: c = -1, x0 = 1 ---
n      x_n
0      1.0000000000000000e+00
1      0.0000000000000000e+00
2      -1.0000000000000000e+00
3      0.0000000000000000e+00
...
--- Eksperyment 5: c = -1, x0 = -1 ---
n      x_n
0      -1.0000000000000000e+00
1      0.0000000000000000e+00
2      -1.0000000000000000e+00
...

```

Interpretacja: Ciąg nie zbiega do jednego punktu, ale wpada w stabilny cykl o okresie 2 (oscyluje), przechodząc między wartościami 0 i -1 .

- $x_n = 0 \rightarrow x_{n+1} = 0^2 - 1 = -1$
- $x_n = -1 \rightarrow x_{n+1} = (-1)^2 - 1 = 0$

6.3.4 Eksperymenty 6, 7 ($c = -1.0$, $x_0 = 0.75$ i 0.25) - Zbieżność do cyklu

```
--- Eksperyment 6: c = -1, x0 = 0.75 ---
n      x_n
0      7.500000000000000e-01
...    ...
16     -1.000000000000000e+00
17     0.000000000000000e+00
...
--- Eksperyment 7: c = -1, x0 = 0.25 ---
n      x_n
0      2.500000000000000e-01
...    ...
11     -1.000000000000000e+00
12     0.000000000000000e+00
...
```

Interpretacja: Obie iteracje, mimo że zaczynają w różnych punktach, zbiegają do tego samego atraktora, którym jest cykl o okresie 2 (oscylacja $0 \leftrightarrow -1$).

6.4 Wnioski

Ten prosty, nieliniowy wzór $x_{n+1} = x_n^2 + c$ wykazuje bardzo złożone zachowanie (tzw. system dynamiczny).

- W zależności od parametrów (c , x_0), ciąg może być stabilny i zbiegać do punktu stałego (Eksperymenty 1, 2) lub do cyklu o okresie 2 (Eksperymenty 4, 5, 6, 7).
- Wrażliwość na warunki początkowe: Eksperyment 3 ($c = -2$) pokazał, że system może być chaotyczny. Minimalna zmiana x_0 (z 2.0 na 1.999...) spowodowała, że ciąg zamiast zbiegać, zaczął zachowywać się w sposób nieprzewidywalny. Pokazuje to, dlaczego długoterminowe prognozowanie w systemach chaotycznych (jak pogoda) jest niemożliwe.