MOWNIT

Laboratorium 6 – układy równań liniowych – metody bezpośrednie

Jakub Karbowski

25 maja 2022

Wykonane eksperymenty

Dla każdego eksperymentu wykonywano następujące kroki:

1. Wylosowanie wektora $\mathbf{x}$
2. Obliczenie $\mathbf{b} = \mathbf{A}\mathbf{x}$
3. Rozwiązanie układu $\mathbf{A}\mathbf{x}' = \mathbf{b}$
4. Obliczenie błędu jako normy L1 $||\mathbf{x} - \mathbf{x}'||_1$

Porównano błąd dla obliczeń na typach:

- Float32 (float z C)
- Float64 (double z C)

Do stworzenia wykresów/tabel wykonywano 100 prób dla każdego n, a błąd uśredniano.

Uwarunkowanie układu policzono za pomocą funkcji LinearAlgebra.cond, stosując normę L2.

Zadanie 1

Metodą eliminacji Gaussa rozwiązać układ równań $\mathbf{Ax} = \mathbf{b}$, gdzie

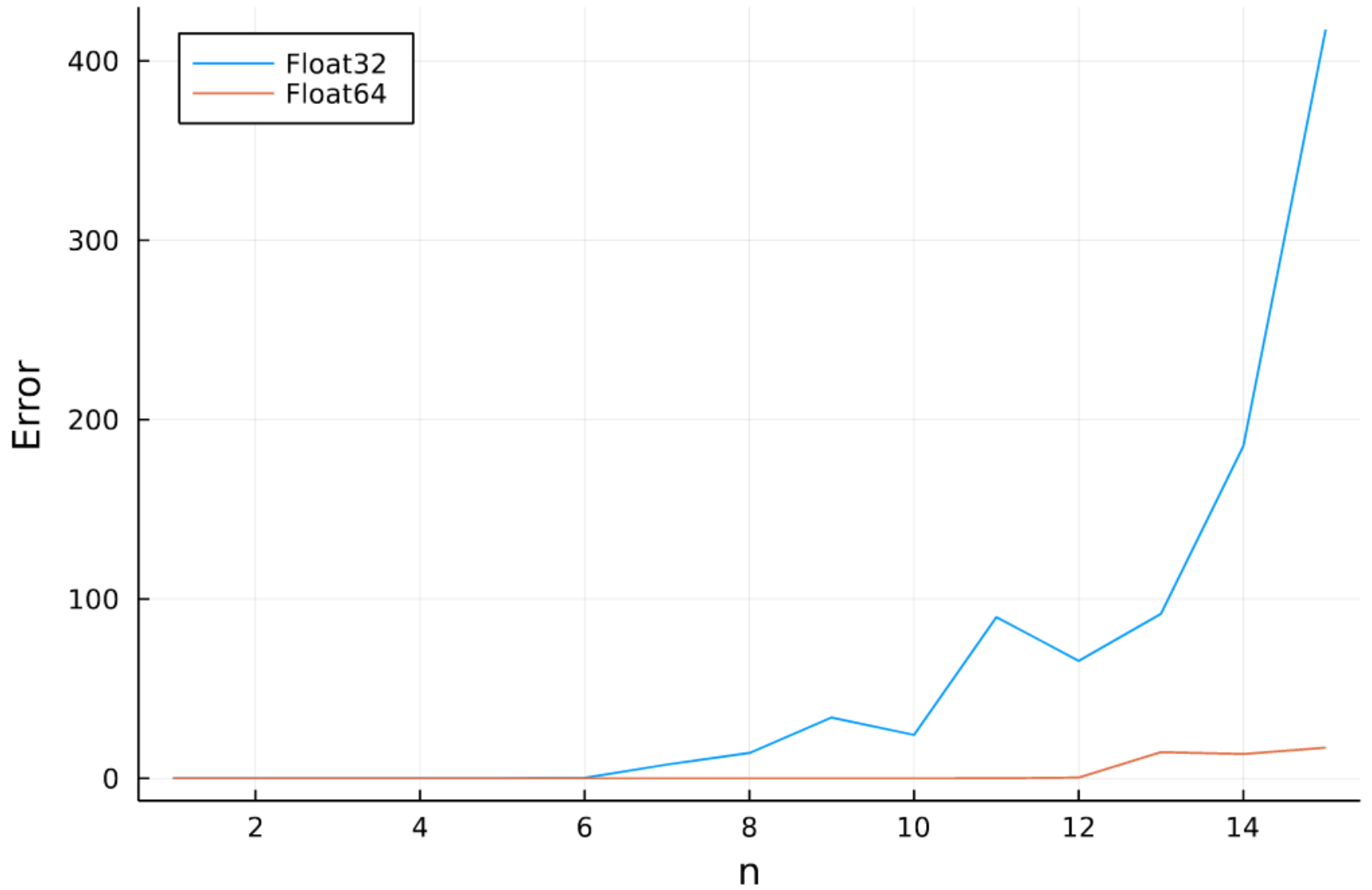
$$\begin{cases} a_{1j} = 1 \\ a_{ij} = \frac{1}{i+j-1} \quad i \neq 1 \end{cases} \qquad i, j = 1, \ldots, n$$

Zadanie 1

Tabela 1: Uwarunkowanie vs n

n	Uwarunkowanie
3	482.922
4	17 032.9
5	$5.918\,77 \times 10^{5}$
6	$2.037\,85 \times 10^{7}$
7	$6.980\,46 \times 10^{8}$
8	$2.383\,97 \times 10^{10}$
9	$8.126\,92 \times 10^{11}$
10	$2.767\,39 \times 10^{13}$
11	$9.438\,48 \times 10^{14}$
12	$3.361\,17 \times 10^{16}$
13	$5.832\,88 \times 10^{17}$
14	$7.771\,24 \times 10^{17}$
15	$6.548\,26 \times 10^{17}$

Zadanie 1



Rysunek 1: Błąd vs n – Gauss

Zadanie 1

Tabela 2: Błąd vs n – Gauss

n	Float32	Float64
3	6.4683×10^{-6}	$1.763\,92 \times 10^{-14}$
4	$0.000\,123\,52$	$2.841\,79 \times 10^{-13}$
5	$0.005\,350\,75$	$1.209\,71 \times 10^{-11}$
6	$0.225\,031$	$5.030\,95 \times 10^{-10}$
7	$7.698\,62$	$1.833\,14 \times 10^{-8}$
8	14.0992	4.5496×10^{-7}
9	33.8966	$1.803\,59 \times 10^{-5}$
10	24.196	$0.000\,485\,513$
11	89.8449	$0.017\,812\,9$
12	65.546	$0.435\,379$
13	91.8066	14.6198
14	185.234	13.55
15	417.487	17.1036

Zadanie 1

1. Układ jest bardzo źle uwarunkowany.
2. Następuje szybka eksplozja błędu przez złe uwarunkowanie układu.
3. Float64 cechuje się większą dokładnością.

Zadanie 2

Metodą eliminacji Gaussa rozwiązać układ równań $\mathbf{Ax} = \mathbf{b}$, gdzie

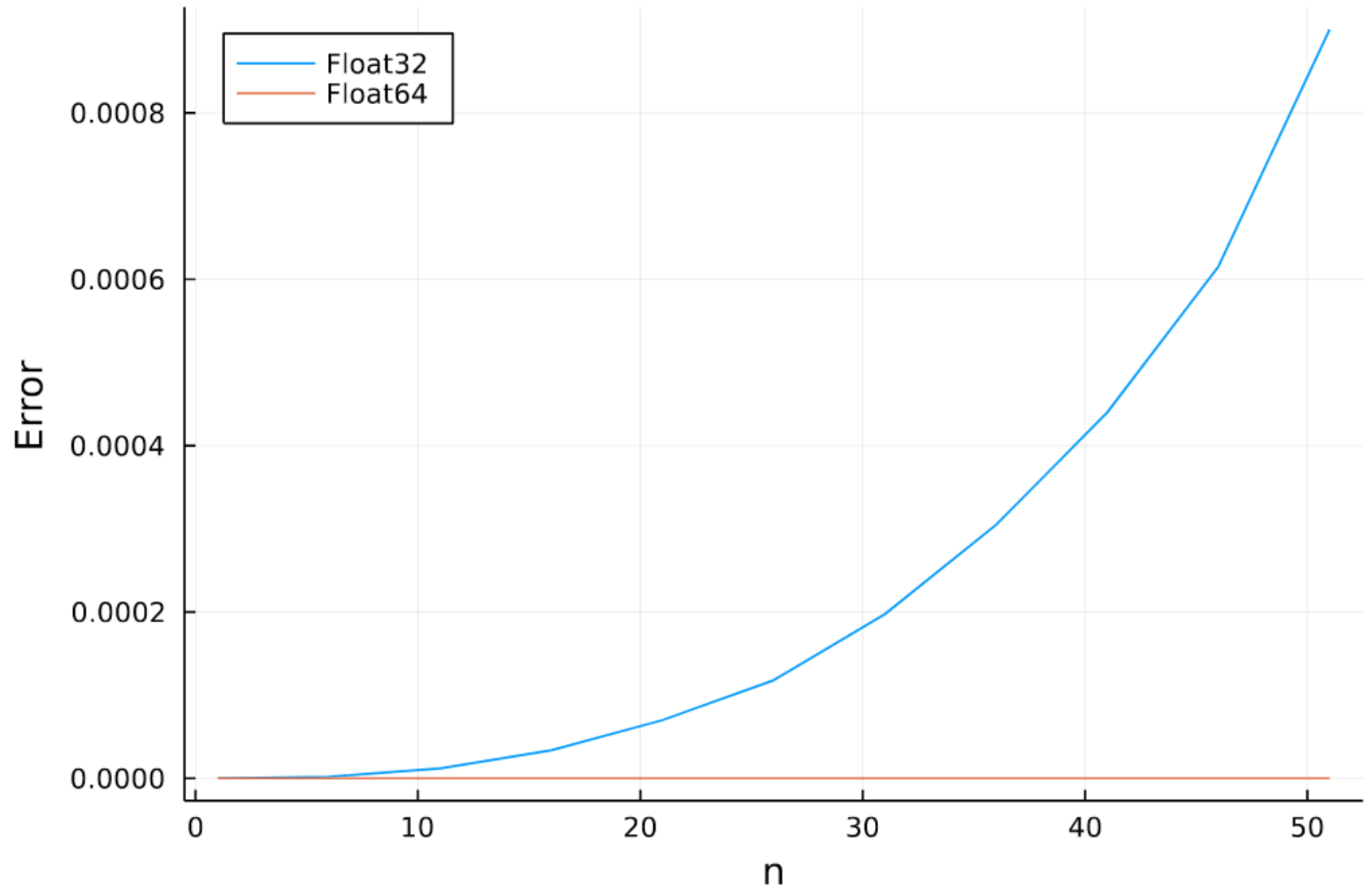
$$\begin{cases} a_{ij} = \frac{2i}{j} & j \geq i \\ a_{ij} = a_{ji} & j < i \end{cases} \qquad i, j = 1, \ldots, n$$

Zadanie 2

Tabela 3: Uwarunkowanie vs n

n	Uwarunkowanie
1	1.0
6	29.0404
11	105.757
16	233.255
21	412.389
26	643.642
31	927.333
36	1263.69
41	1652.89
46	2095.07
51	2590.34

Zadanie 2



Rysunek 2: Błąd vs n – Gauss

Zadanie 2

Tabela 4: Błąd vs n – Gauss

n	Float32	Float64
1	0.0	0.0
6	$2.015\,23 \times 10^{-6}$	$2.907\,67 \times 10^{-15}$
11	$1.200\,26 \times 10^{-5}$	$1.971\,31 \times 10^{-14}$
16	$3.367\,25 \times 10^{-5}$	$5.193\,18 \times 10^{-14}$
21	$6.995\,38 \times 10^{-5}$	$1.173\,64 \times 10^{-13}$
26	0.000 117 978	2.2271×10^{-13}
31	0.000 197 399	$3.813\,16 \times 10^{-13}$
36	0.000 304 77	5.6141×10^{-13}
41	0.000 439 96	$8.170\,44 \times 10^{-13}$
46	0.000 615 436	1.1481×10^{-12}
51	0.000 900 291	$1.579\,25 \times 10^{-12}$

Zadanie 2

1. Układ jest lepiej uwarunkowany niż poprzedni.
2. Błąd przyjmuje rozsądne wartości.
3. Błąd rośnie kwadratowo.
4. Float64 cechuje się większą dokładnością.

Zadanie 3

Metodą Thomasa dla macierzy trójdiagonalnych rozwiązać układ równań $\mathbf{Ax} = \mathbf{b}$, gdzie

$$\begin{cases} a_{i,i} = 5 & \text{główna przekątna} \\ a_{i,i+1} = \frac{1}{i+5} & \text{górna przekątna} \\ a_{i,i-1} = \frac{5}{i+5+1} & \text{dolna przekątna} \\ a_{i,j} = 0 & \text{w przeciwnym wypadku} \end{cases} \qquad i, j = 1, \ldots, n$$

Zadanie 2

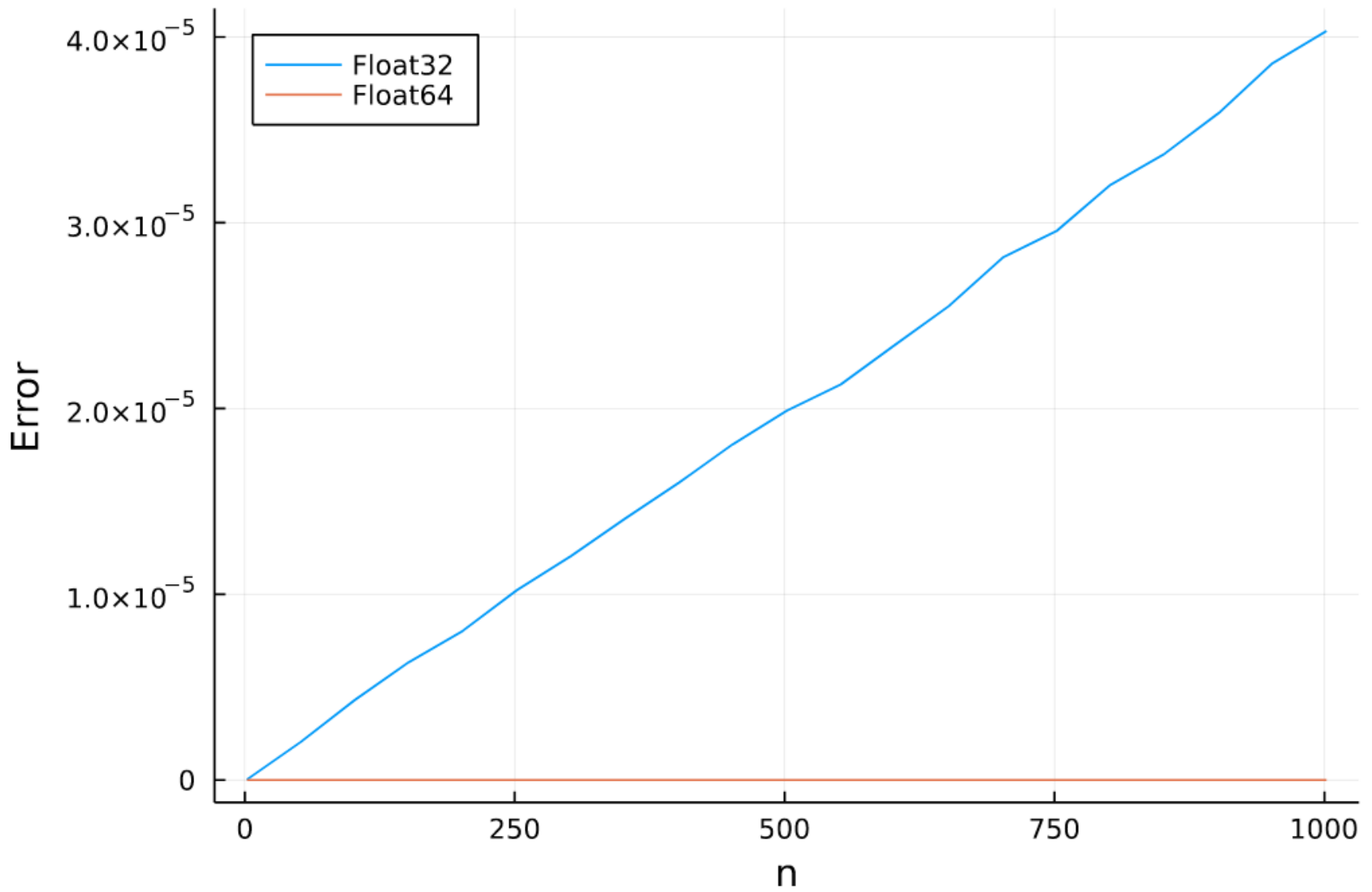
Tabela 5: Uwarunkowanie vs n

n	Uwarunkowanie
2	1.171 75
52	1.270 69
102	1.270 69
152	1.270 69
202	1.270 69
252	1.270 69
302	1.270 69
352	1.270 69
402	1.270 69
452	1.270 69
502	1.270 69
552	1.270 69
602	1.270 69

Zadanie 3

Do implementacji algorytmu użyto typu Tridiagonal z biblioteki LinearAlgebra. Typ ten przechowuje macierz trójdiagonalną jako 3 tablice reprezentujące przekątne. Pozwala na wygodne indeksowanie pomijając faktyczną implementację.

Zadanie 3



Rysunek 3: Błąd vs n – Thomas

Zadanie 3

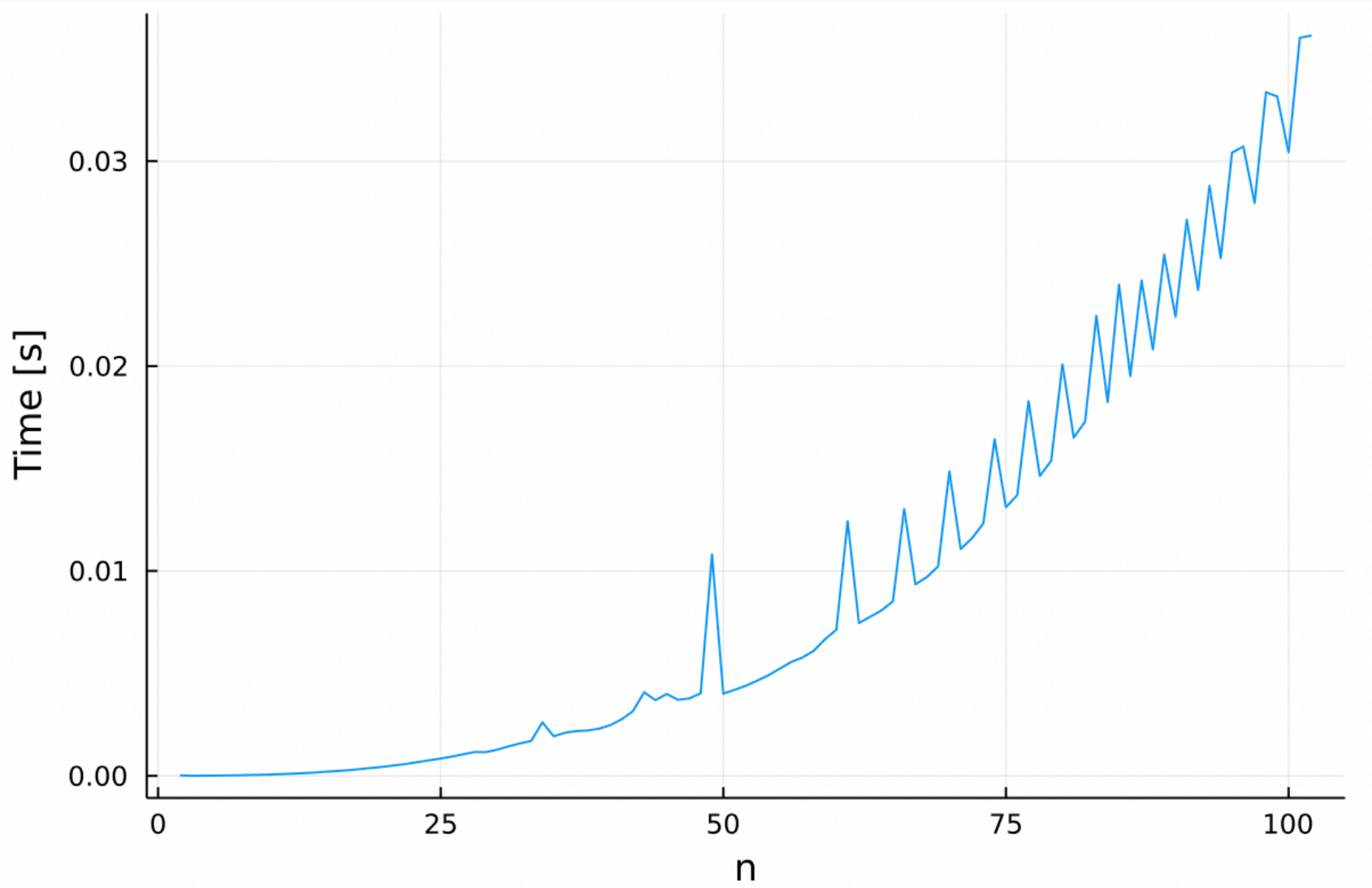
Tabela 6: Błąd vs n – Thomas

n	Float32	Float64
152	6.3498×10^{-6}	$1.002\,94 \times 10^{-14}$
202	$8.036\,26 \times 10^{-6}$	$1.393\,91 \times 10^{-14}$
252	1.0218×10^{-5}	$1.754\,42 \times 10^{-14}$
302	$1.204\,85 \times 10^{-5}$	$2.169\,68 \times 10^{-14}$
352	$1.406\,86 \times 10^{-5}$	$2.521\,25 \times 10^{-14}$
402	$1.599\,98 \times 10^{-5}$	$2.883\,85 \times 10^{-14}$
452	$1.807\,58 \times 10^{-5}$	$3.314\,33 \times 10^{-14}$
502	$1.988\,35 \times 10^{-5}$	$3.746\,44 \times 10^{-14}$
552	$2.130\,04 \times 10^{-5}$	$4.072\,38 \times 10^{-14}$
602	$2.341\,56 \times 10^{-5}$	$4.410\,02 \times 10^{-14}$
652	$2.551\,01 \times 10^{-5}$	$4.787\,54 \times 10^{-14}$
702	$2.812\,21 \times 10^{-5}$	$5.234\,99 \times 10^{-14}$

Zadanie 3

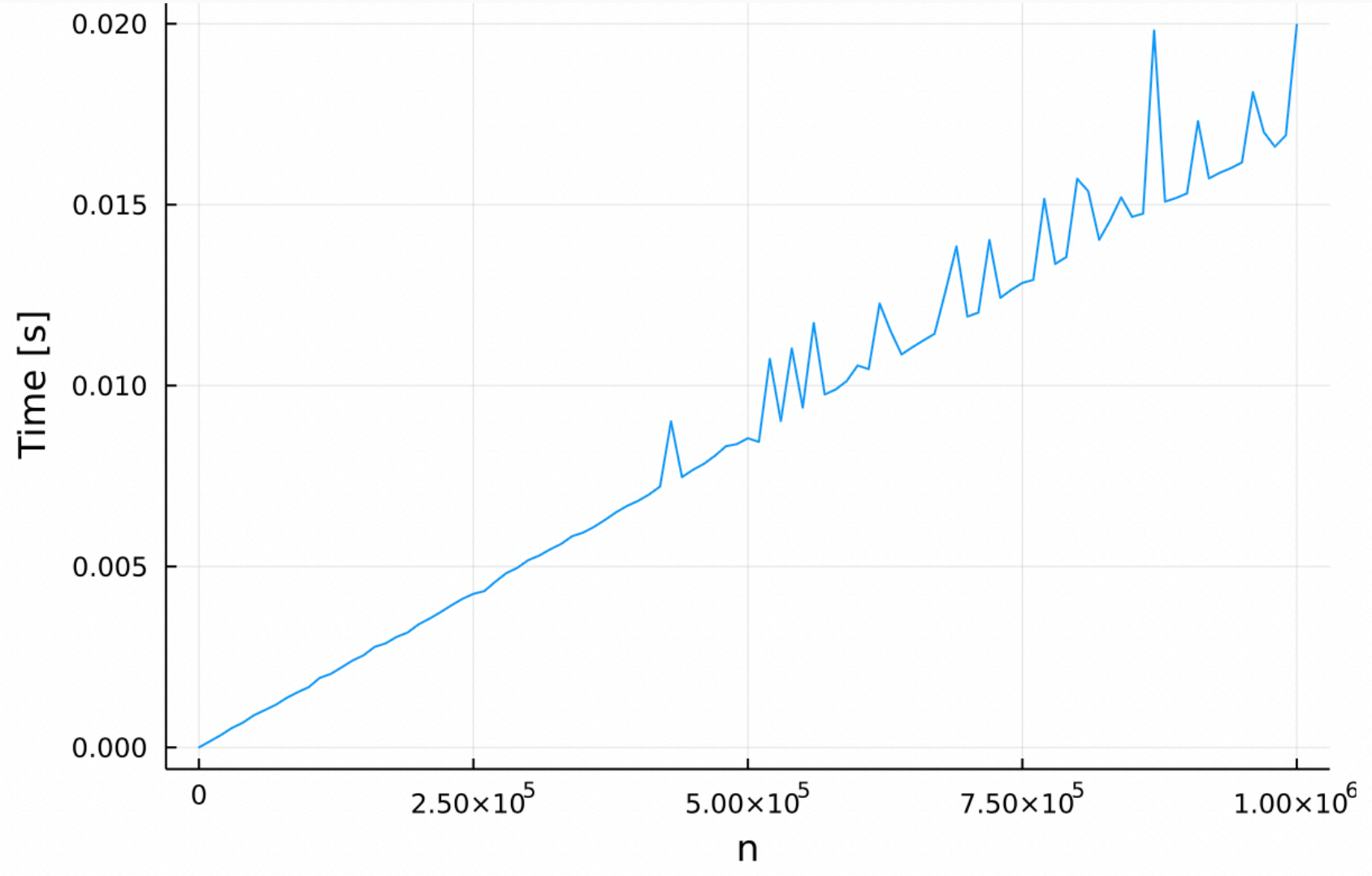
1. Układ jest bardzo dobrze uwarunkowany.
2. Błąd przyjmuje bardzo małe wartości.
3. Błąd rośnie liniowo.
4. Float64 cechuje się większą dokładnością.

Benchmark



Rysunek 4: Benchmark – Gauss

Benchmark



Rysunek 5: Benchmark – Thomas

Benchmark

1. Algorytm Thomasa jest nieporównywalnie szybszy od Gaussa – dla największych układów jakie policzy metoda Gaussa, metoda Thomasa potrzebuje czasu poniżej rozdzielczości pomiaru.
2. Algorytm Gaussa ma złożoność $O(n^3)$.
3. Algorytm Gaussa potrzebuje $O(n^2)$ pamięci.
4. Algorytm Thomasa ma złożoność $O(n)$.
5. Algorytm Thomasa potrzebuje $O(n)$ pamięci.