13.03.2022 r.

MOWNiT Laboratorium nr 1 -

Arytmetyka komputerowa

opracował: Kacper Kłusek

Dane techniczne

Program obliczający kolejne wyrazy ciągu dla odpowiednich typów danych został napisany w języku C++ i skompilowany za pomocą gcc 9.4.0.
Uruchomienie programu odbywało się pod 64 bitowym systemem operacyjnym Ubuntu 20.04.3 LTS na komputerze z procesorem AMD® Ryzen 7 architektury x86 o bazowym taktowaniu procesora 2.9GHz (4.2GHz boost) oraz 16GB pamięci RAM.

Pomiar Czasu

Do mierzenia czasu użyto steady_clock z biblioteki chrono. Z uwagi na bardzo krótki czas obliczeń, nie została użyta żadna optymalizacja kompilatora aby nie zmniejszać dokładności pomiaru. Same wyniki pomiaru dotyczą wykonania k (=70) obliczeń kolejnego wyrazu ciągu i zostały podane w nanosekundach.

Dokładne wyniki

Do wykonania dokładnych obliczeń zmiennoprzecinkowych w celu wyznaczenia wzorcowych wartości ciągu użyto biblioteki decimal języka Python. Wyniki są zgodne z wynikami obliczonymi za pomocą kalkulatora naukowego WolframAlpha.

Sposób obliczeń

Obliczenia wykonywane były za pomocą podanych wzorów, wyliczając wyraz x_{k+1} za pomocą wyrazu x_k , przy zadanych warunkach początkowych $x_0 = 0.1$.
Policzono 70 pierwszych wyrazów ciągu, dla większej ilości dane na wykresach przestawały być czytelne, a błąd bezwzględny oscylował w tych samych granicach dla wszystkich typów danych.

Wzory:

wzór nr 1: $$x_{k+1} = x_k + 3x_k(1 - x_k)$$

wzór nr 2: $$x_{k+1} = 4x_k - 3x_k x_k$$

Użyte typy danych:

float:

Manstysa: 23 bity
Wykładnik: 8 bitów
Dokładność: 7-8 liczb znaczących

double:

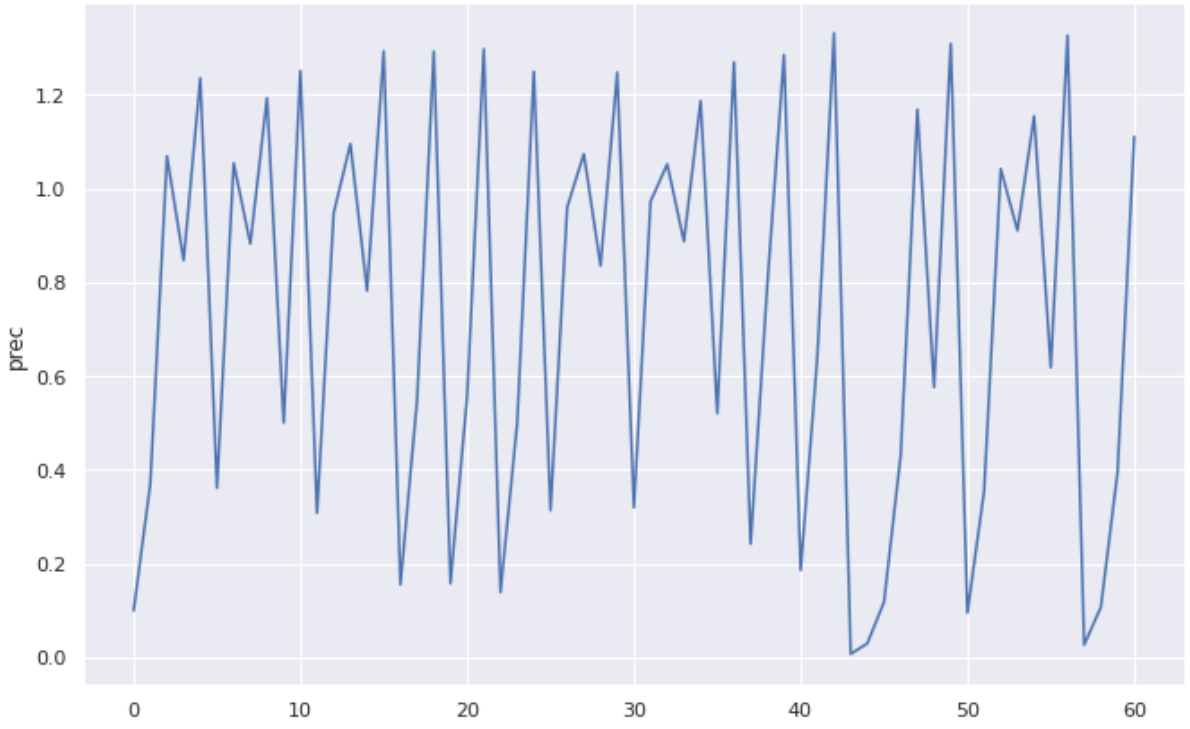
Mastysa: 53 bity
Wykładnik: 11 bitów
Dokładność: 15-16 liczb znaczących

long double:

Mantysa: 63 bitów
Wykładnik: 15 bitów
Dokładność: ok. 19 liczb znaczących (eksperymentalnie 15-16 tak jak double)

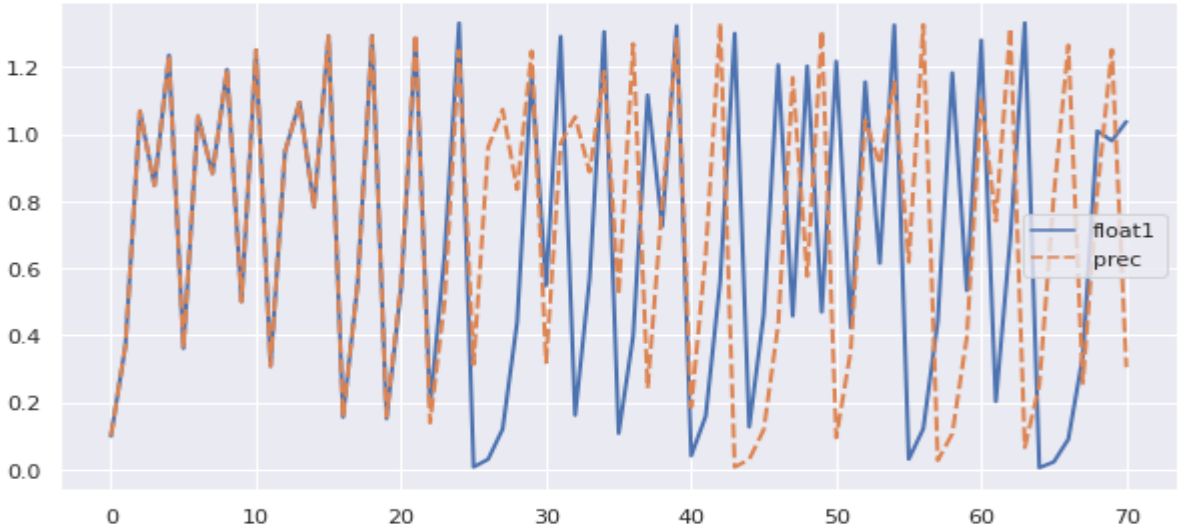
Wyniki eksperymentu

dokładne wartości ciągu:

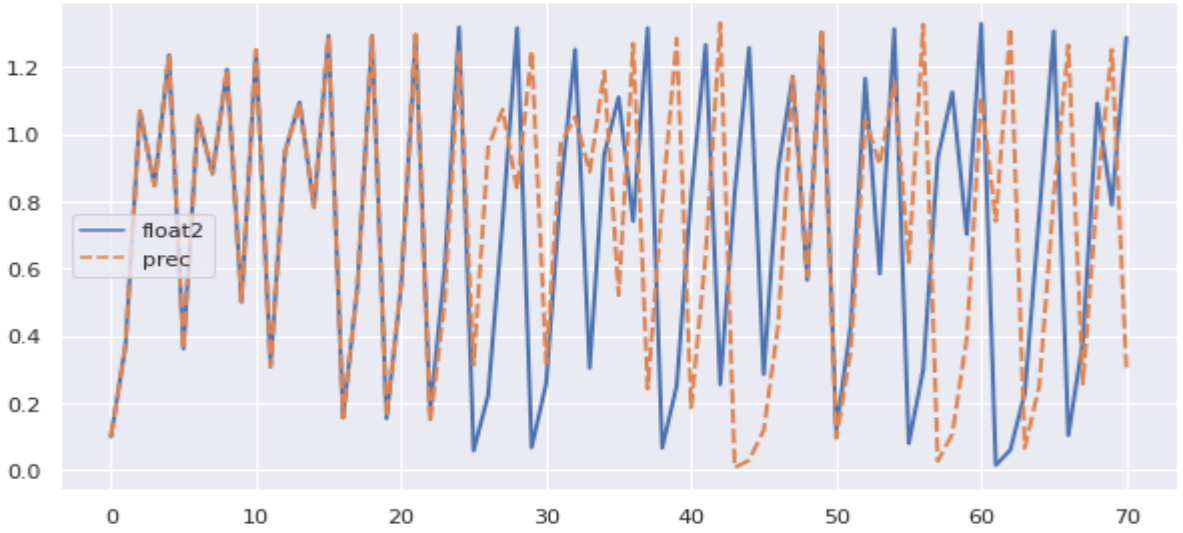


wartości dla typu **float** i dokładne dane (**prec**)

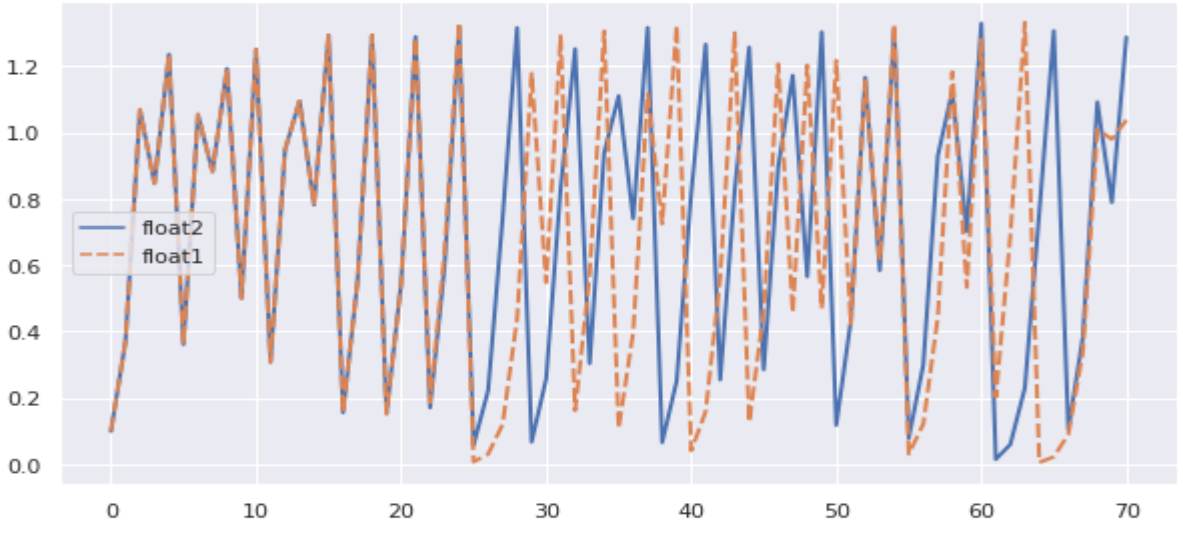
a) wzór nr 1



b) wzór nr 2

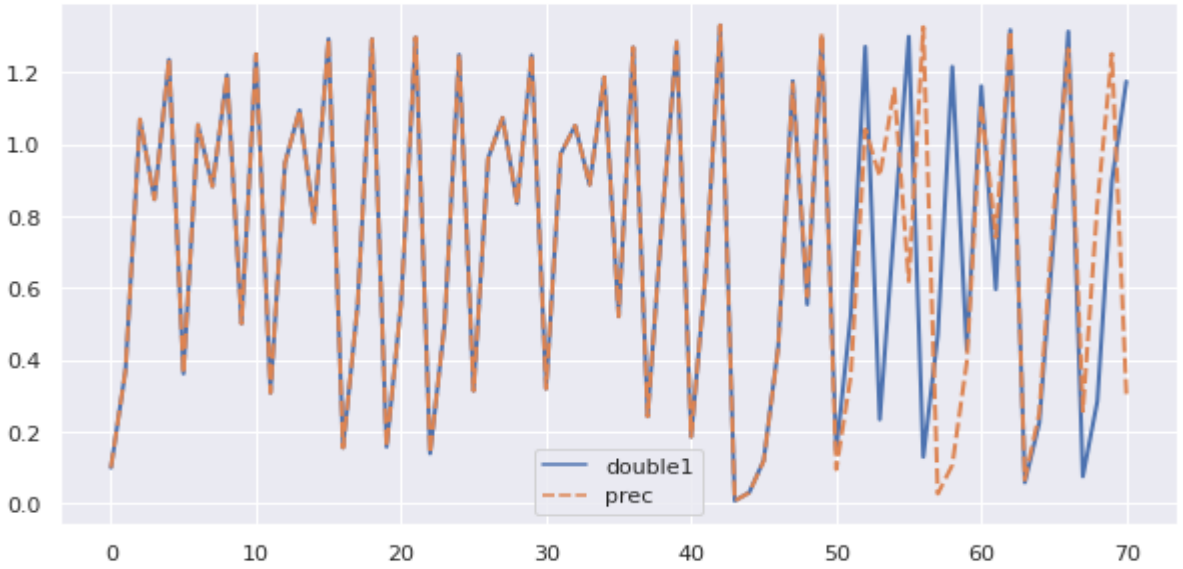


wartości dla typu **float** dla obu wzorów

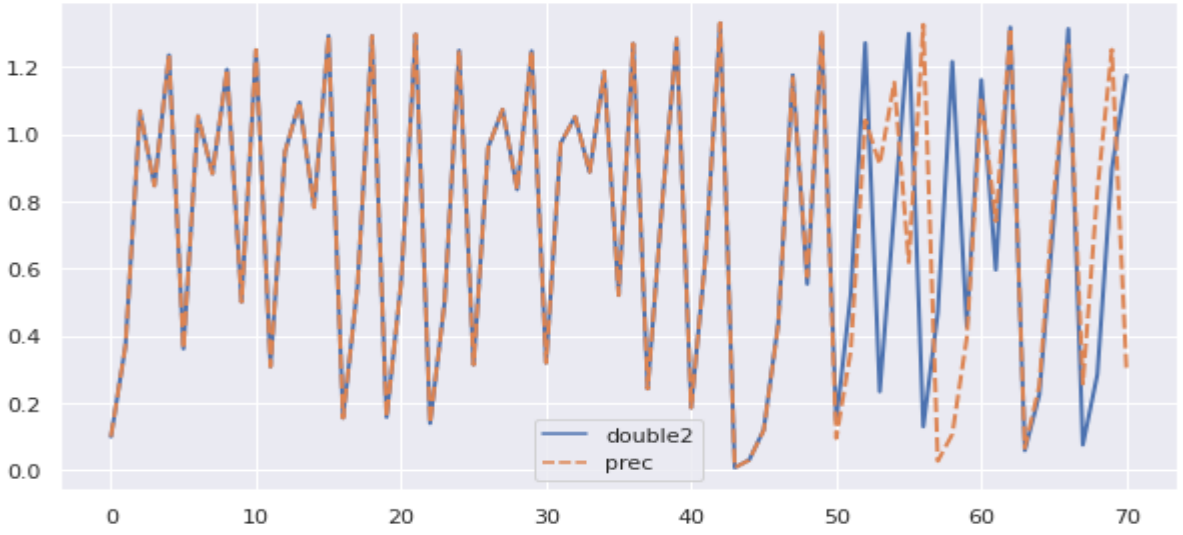


wartości dla typu **double** i dokładne wyniki (**prec**)

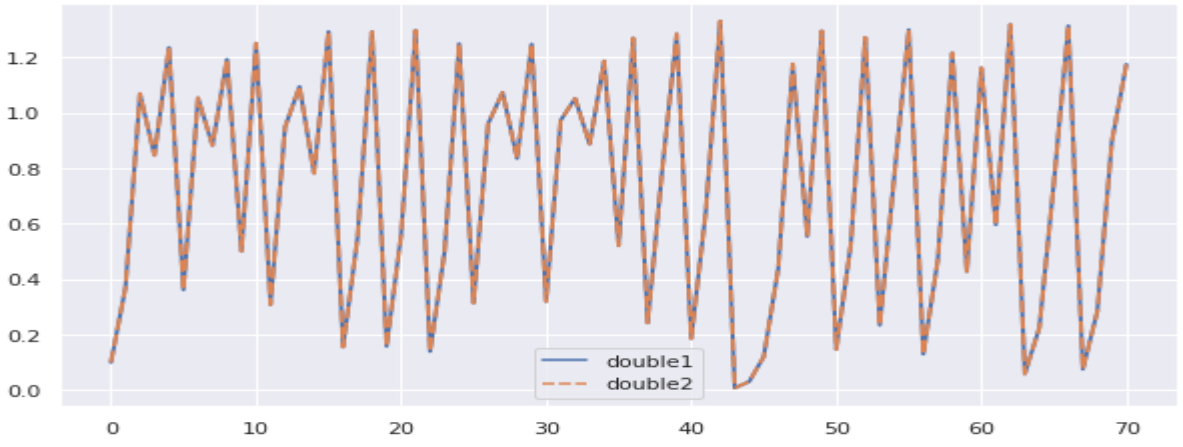
a) wzór nr 1



b) wzór nr 2

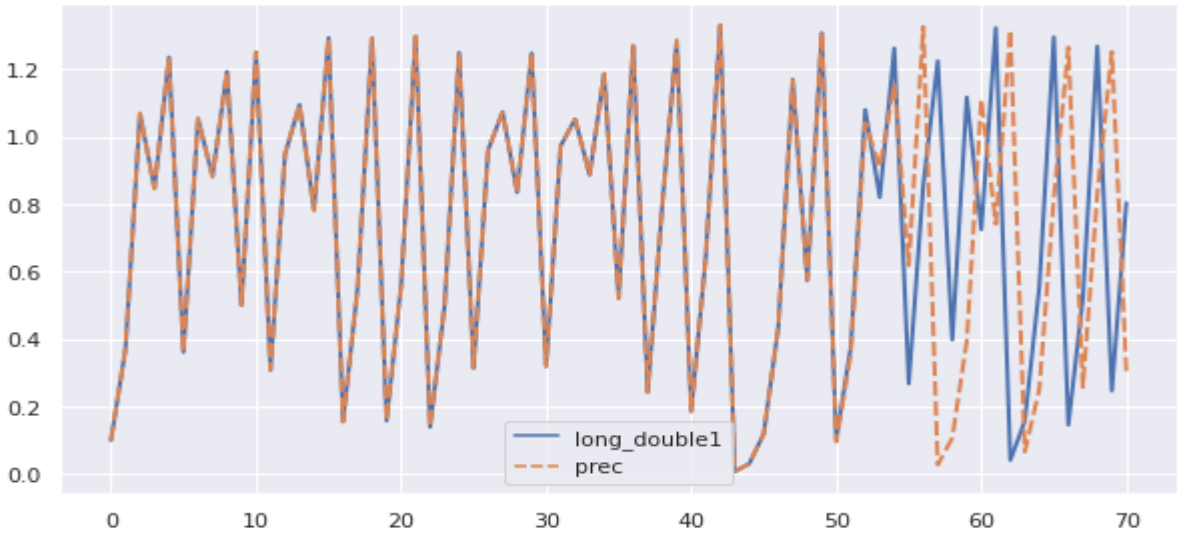


wartości dla typu **long** dla obu wzorów

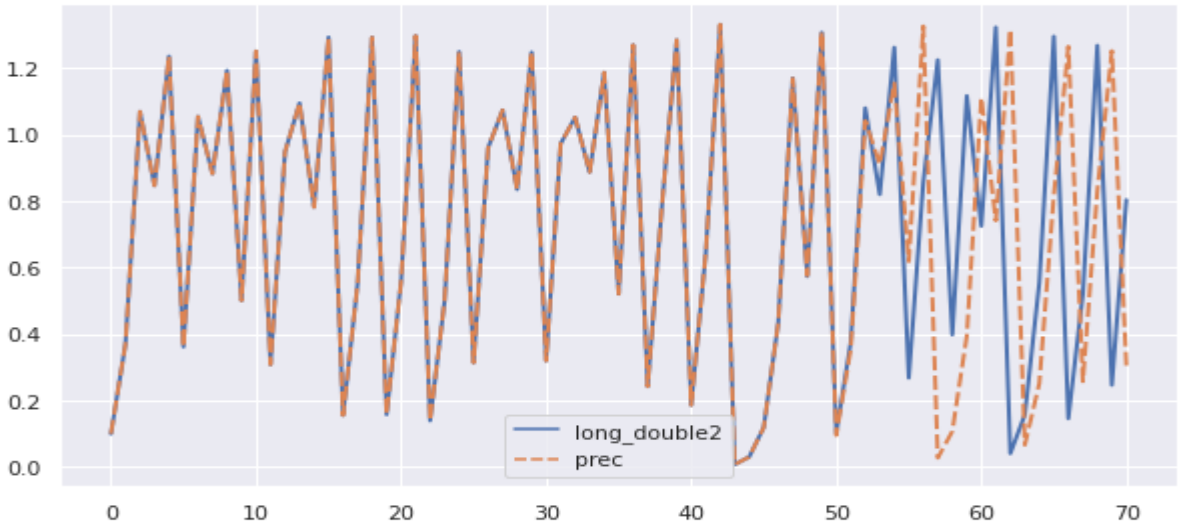


wartości dla typu **long double** i dokładne wartości (**prec**)

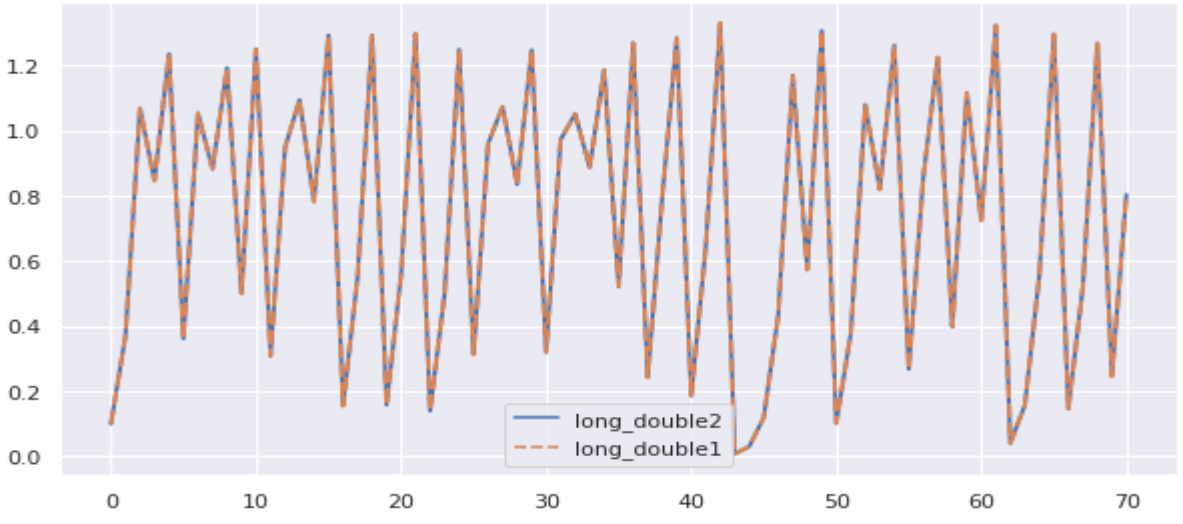
a) wzór nr 1



b) wzór nr 2

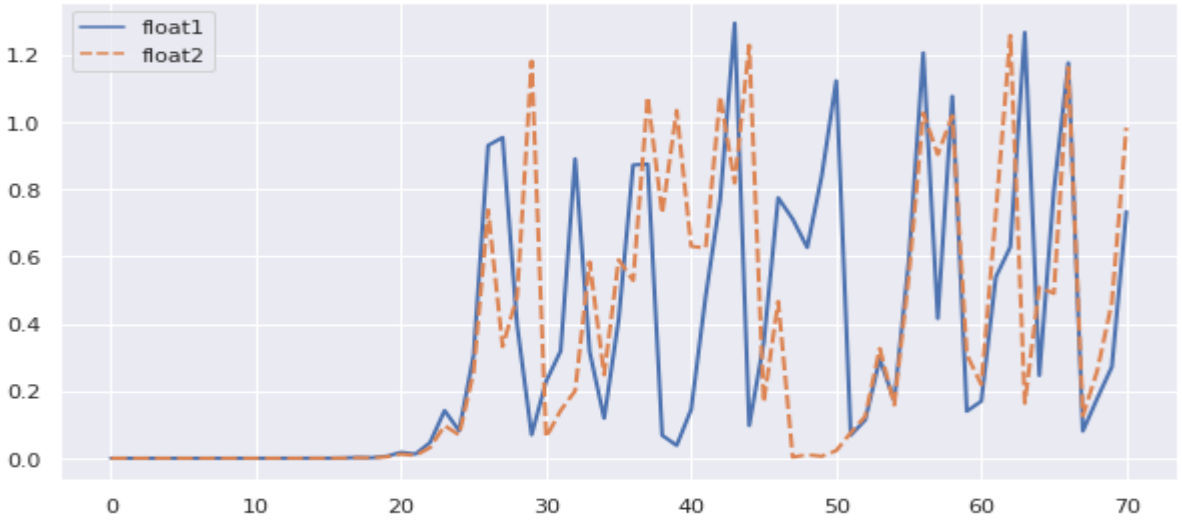


wartości **long double** dla obu wzorów

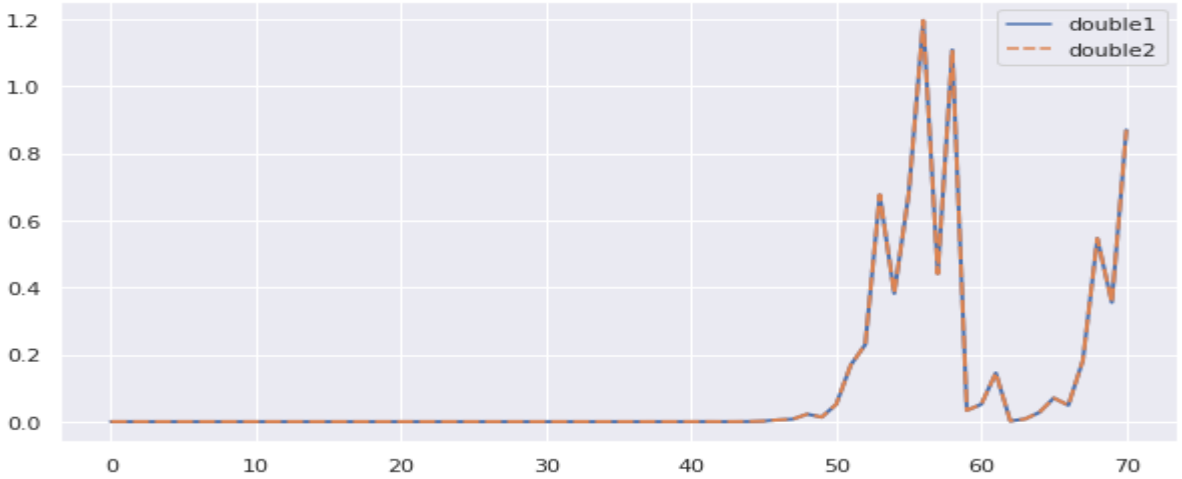


Błąd bezwzględny

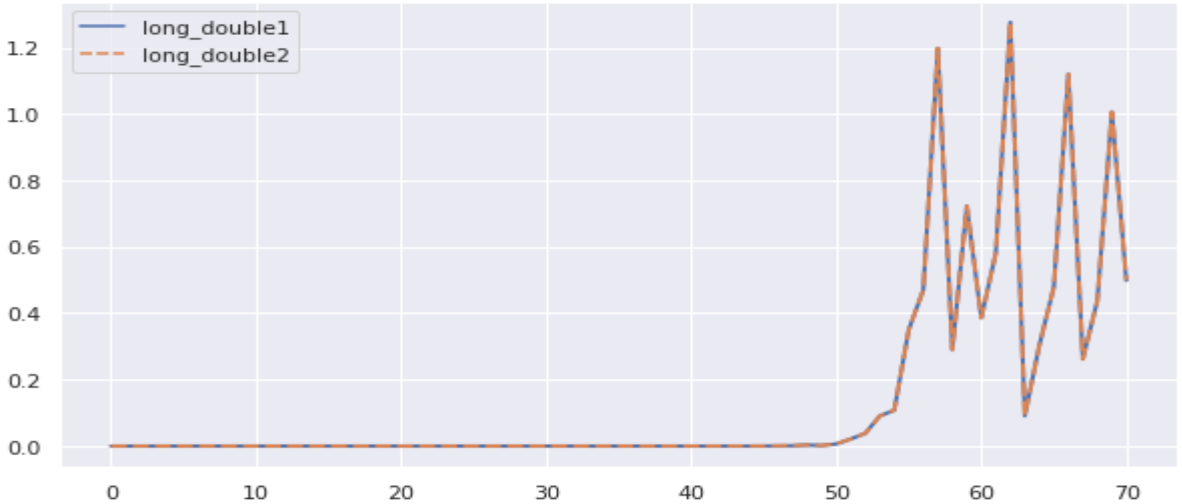
Typ float: (wzór nr 1 i 2)



Typ double: (wzór nr 1 i 2)



Typ long double: (wzór nr 1 i 2)



Pomiar czasu

Wyniki prędkości obliczeń dla typu float: (nanosekundy)

wzór 1 - 300
wzór 2 - 341

Wyniki prędkości obliczeń dla typu double: (nanosekundy)

wzór 1 - 311
wzór 2 - 310

Wyniki prędkości obliczeń dla typu double: (nanosekundy)

wzór 1 - 641
wzór 2 - 681

Wnioski:

Wybór typu danych bezdyskusyjnie ma ogromny wpływ na precyzję obliczeń na liczbach zmiennoprzecinkowych.
Z wyników obliczeń wynika, iż obliczenia na typie **float (4 bajty)** zajmującym 2 razy mniej pamięci niż **double (8 bajtów)** przestają być dokładne 2 razy szybciej, niż wspomniany double.
Co ciekawe **long double (16 bajtów)** nie kontynuuje trendu, a obliczenia z jego wykorzystaniem przestają być dokładne znacznie szybciej niż dwukrotność dla double - różnica ok. 2 wyrazów ciągu.

Błąd wynika z reprezentacji liczb w komputerze. W momencie pojawienia się liczby której rozwinięcie dziesiętne wykracza poza zakres dokładności dla interesującego nas typu danych, mamy do czynienia z błędem zaokrąglenia. Każde kolejne działanie zawierające tę liczbę będzie propagowało ten błąd. W naszym przypadku mnożymy liczbę przez stałą oraz przez **samą siebie** - co sprawia iż błąd jest propagowany jeszcze szybciej.