Arytmetyka Komputerowa – Zestaw 1

Autor: Adrian Żerebiec

Zadanie 6.

Wyznaczyć kolejne elementy ciągu $x_{k+1} = x_k + 3x_k (1 - x_k)$, $x_0 = 0.1$, i porównać otrzymane wartości dla różnej precyzji zmiennych (float, double, long double). Powtórzyć doświadczenie dla przekształconej postaci wzoru: $x_{k+1} = 4x_k - 3x_k x_k$. Spróbować wyjaśnić otrzymane wyniki.

Opis doświadczenia

Do obliczeń wykorzystałem język programowanie C++ ze względu na łatwość typowania. Dla podanej wartości k, która oznacza ilość elementów ciągu, iteracyjnie wyliczamy kolejne elementy ciągu. Wykonujemy to za równo dla podstawowej formy równania, jak i przekształconej postaci wzoru. Wyniki zapisywane są początkowo w tablicy, a następnie przenoszone do pliku dat.csv.

Do pliku check.csv natomiast przenoszone są różnice pomiędzy poszczególnymi wariantami. Później dane wykorzystujemy do stworzenia przykładowych wykresów, porównujących odpowiednie elementy ciągu. Wykresy zostały stworzone za pomocą biblioteki Matplot.lib oraz biblioteki Pandas z języka Python.

Sprzęt

Doświadczenie zostało wykonane na systemie Windows 10 (64-bity), o procesorze AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz oraz 8 GB RAM. Wykorzystany kompilator to MinGW GNU GCC Compiler, wcześniej wykorzystywany był kompilator G++.

Przykładowa pętla wykonująca obliczenia:

```
void fdouble(int k, double x,double tab[]) {
    tab[0] = x;
    for (int i = 1; i < k; i++) {
        x = x+3*x*(1-x);
        tab[i] = x;
    }
}
```

Funkcja ta, modyfikuje tablicę i zapisuje w niej uzyskane wyniki dla k wykonań kodu. Jako parametry przyjmuje: liczbę wykonań k wyrażoną intem, wyraz początkowy w naszym przypadku 0.1 oraz tablicę. Analogicznie wyglądają funkcje dla double, long double oraz 3 pozostałe obliczające przekształconą wersję wzoru.

Przykładowe uzyskane dane

Dla k = 20:

```
Wersja podstawowa dla float:
0.1, 0.37, 1.0693, 0.846993, 1.23578, 0.36166, 1.05425, 0.882679, 1.19335,
0.50115, 1.25115, 0.308486, 0.948452, 1.09512, 0.782607, 1.29301, 0.156427,
0.5523, 1.29409, 0.152338

Wersja podstawowa dla double:
0.1, 0.37, 1.0693, 0.846993, 1.23578, 0.36166, 1.05425, 0.882681, 1.19335,
0.501158, 1.25115, 0.308456, 0.948389, 1.09523, 0.782331, 1.2932, 0.155706,
0.550092, 1.29256, 0.15809

Wersja podstawowa dla long double:
0.1, 0.37, 1.0693, 0.846993, 1.23578, 0.36166, 1.05425, 0.882681, 1.19335,
0.501158, 1.25115, 0.308456, 0.948389, 1.09523, 0.782331, 1.2932, 0.155706,
0.550092, 1.29256, 0.15809

Wersja przekształcona dla float:
0.1, 0.37, 1.0693, 0.846993, 1.23578, 0.36166, 1.05425, 0.88268, 1.19335,
0.501153, 1.25115, 0.308476, 0.948433, 1.09516, 0.782521, 1.29307,
0.156202, 0.551612, 1.29362, 0.15412

Wersja przekształcona dla double:
0.1, 0.37, 1.0693, 0.846993, 1.23578, 0.36166, 1.05425, 0.882681, 1.19335,
0.501158, 1.25115, 0.308456, 0.948389, 1.09523, 0.782331, 1.2932, 0.155706,
0.550092, 1.29256, 0.15809

Wersja przekształcona dla long double:
0.1, 0.37, 1.0693, 0.846993, 1.23578, 0.36166, 1.05425, 0.882681, 1.19335,
0.501158, 1.25115, 0.308456, 0.948389, 1.09523, 0.782331, 1.2932, 0.155706,
0.550092, 1.29256, 0.15809
```

Analiza wyników

Najpierw porównajmy 3 wersje podstawowe. Jak widzimy wyniki są na pierwszy rzut oka podobne. Możemy zauważyć, że wyniki dla double oraz long double są identyczne. Wynika to ze specyfikacji języka c++, gdyż jak możemy przeczytać long double jest analogiczny z double a różnicę miedzy nimi pojawią się dopiero na odległych miejscach po przecinku.

W związku z tym, póki co w dalszej analizie będę pisał wyłącznie o double, gdyż nie ma różnic w wynikach między nim a long double dla ustalonej precyzji. Różnicę natomiast możemy zauważyć między double a float. Różnią się one nie tylko zakresem, ale także rozmiarem. Poniższa tabela prezentuje dokładniejsze parametry float oraz double.

Podstawa porównania	Float	Double
Precyzja	Pojedyncza/ mniej dokładna	Podwójna/ bardziej dokładna
Bity	32	64
Bajty	4	8
Przybliżony zakres	Od 1,4e-045 do 3,4e + 038/ 1 bit na jeden znak	4,9e-324 do 1,8e + 308/ 1 bit na jeden znak
Reprezentacja bitów	8 bitów na wykładnik 23 bity na mantysę	11 bitów na wykładnik 52 bity na mantysę

Po dokładniejszym spojrzeniu na wyniki możemy zobaczyć, iż już 7 element ciągu nieznacznie różni się dla flota oraz double. Nieznaczne różnicę widać też na późniejszych elementach np. na 12 elemencie ciągu (zaznaczony kolorem morskim).

Do dokładniejszej analizy postanowiłem jednak zwiększyć precyzję funkcją setprecision z biblioteki, ustawiając ją na 20. W ten sposób możemy zaobserwować różnice na dalszych miejsca po przecinku oraz zobaczyć po raz pierwszy różnice między long double a double.

Wyniki dla k = 10 z precyzją 20 miejsc po przecinku:

```
Wersja podstawowa float:
0.10000000149011611938, 0.37000000476837158203, 1.0692999362945556641,
0.84699267148971557617, 1.2357809543609619141, 0.36166012287139892578,
1.0542464256286621094, 0.88267910480499267578, 1.1933492422103881836,
0.50114977359771728516

Wersja podstawowa double:
0.10000000000000000555, 0.37000000000000010658, 1.0693000000000001393,
0.84699252999999963265, 1.2357810823725976501, 0.36165967884042371505,
1.0542455454648194113, 0.88268117146203384227, 1.1933465344873701586,
0.50115828383043214966

Wersja podstawowa long double:
0.1, 0.37, 1.0693000000000000001, 0.84699252999999999987, 1.2357810823725973001,
0.36165967884042485986, 1.0542455454648216801, 0.88268117146202853195,
1.1933465344873770905, 0.50115828383041032299

Wersja przekształcona float:
0.10000000149011611938, 0.37000000476837158203, 1.0692999362945556641,
0.84699273109436035156, 1.2357809543609619141, 0.361660003662109375,
1.0542461872100830078, 0.88267970085144042969, 1.1933484077453613281,
0.50115251541137695312

Wersja przekształcona double:
0.10000000000000000555, 0.36999999999999999556, 1.0693000000000001393,
0.84699252999999963265, 1.2357810823725978722, 0.36165967884042249381,
1.0542455454648174129, 0.88268117146203817214, 1.1933465344873646075,
0.50115828383045002425

Wersja przekształcona long double:
0.1, 0.37, 1.0693000000000000001, 0.84699252999999999976, 1.2357810823725973004,
0.36165967884042485591, 1.0542455454648216787, 0.88268117146202853351,
1.1933465344873770864, 0.50115828383041033595
```

Z ciekawości również dodałem funkcję obliczająca różnice między poszczególnymi wariantami.

```
Wersja podstawowa float vs wersja podstawowa double:
1.4901161138336505019e-09, 4.768371475449839636e-09, -6.3705444475203876209e-08,
1.4148971594352133252e-07, -1.280116357360583379e-07, 4.4403097521072965037e-07,
8.8016384269806735574e-07, -2.066657041166486408 2e-06, 2.7077230180250211333e-06, -
8.5102327148645073862e-06

Wersja podstawowa float vs wersja przekształcona float:
0, 0, 0, -5.9604644775390625e-08, 0, 1.1920928955078125e-07, 2.384185791015625e-07,
-5.9604644775390625e-07, 8.3446502685546875e-07, -2.74181365966796875e-06


Wersja przekształcona float vs wersja podstawowa double:
1.4901161138336505019e-09, 4.768371475449839636e-09, -6.3705444475203876209e-08,
2.0109436071891195752e-07, -1.280116357360583379e-07, 3.2482168565994840037e-07,
6.4174526359650485574e-07, -1.4706105934125801582e-06, 1.8732579911695523833e-06, -
5.7684190551965386362e-06

Wersja  podstawowa double vs wersja podstawowa long double:
0, 8.7711955754077308711e-17, 1.0560129160008813187e-16, -2.8601253310167606969e-
16, 2.6216008530699497214e-16, -8.4491875301795360542e-16, -1.7199783264310042341e-
15, 4.0340452332754406228e-15, -5.2776793352249384839e-15, 1.6599026840535824334e-
14
```

Analizując dane, możemy zobaczyć że float dokładny jest do 8 miejsca po przecinku, double do 17 miejsca po przecinku, a long double dopiero na 21 miejscu po przecinku traci dokładność. Zwykle typ float ma 32 bity, double 64, a long double 80, 96 lub 128 bitów.

Wpływa to na precyzje obliczeń, i o ile przy kilku miejscach po przecinku różnice zazwyczaj nie widać o tyle dla coraz większej ich liczby, można zauważyć rozbieżności w wynikach.

Zmiany wyników

Ze względu na trudności z kompilacją za pomocą komend (wcześniej program uruchamiałem bezpośrednio w programie Clion, a nie z terminala), zmieniłem kompilator na G++. Przez to wyniki uległy nieznacznej zmianie. Poniżej znajdują się nowe wyniki.

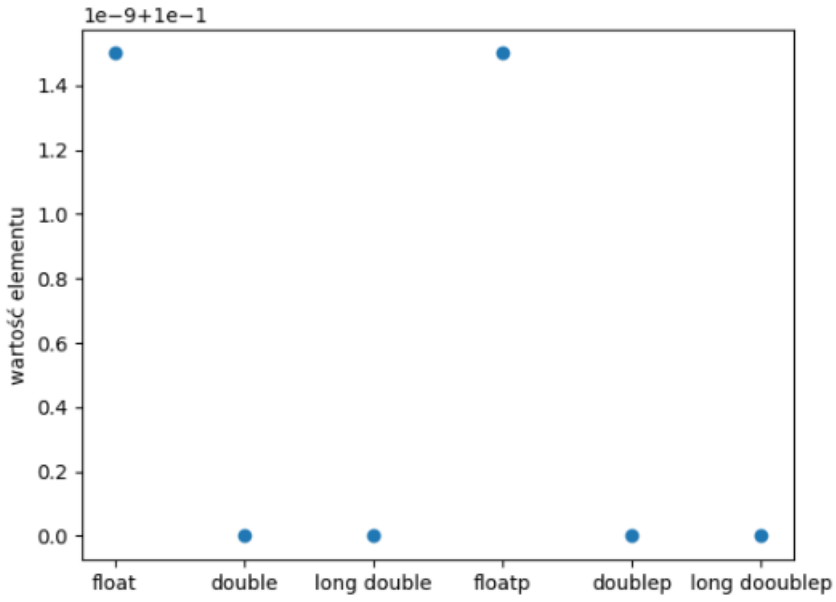
Wyniki dla k =10 z precyzją 20:

```
Wersja podstawowa float:
0.10000001490116119384, 0.37000000476837158203, 1.06930005550384521484,
0.84699237346649169922, 1.23578119277954101562, 0.36165928840637207031,
1.05424487590789794922, 0.88268274068832397461, 1.19334447383880615234,
0.50116479396820068359

Wersja podstawowa double:
0.10000000000000000555, 0.36999999999999999556, 1.06929999999999999172,
0.84699253000000018776, 1.23578108237259720600, 0.36165967884042515834,
1.05424555454648222979, 0.88268117146202706991, 1.19334653448737904040,
0.50115828383040417204
Wersja podstawowa long double:

0.1, 0.37, 1.06930000000000000001, 0.84699252999999999987, 1.23578108237259730010,
0.36165967884042485986, 1.05424555454648216801, 0.88268117146202853195,
1.19334653448737709050, 0.50115828383041032299

Wersja przekształcona float:
0.10000001490116119384, 0.37000000476837158203, 1.06930005550384521484,
0.84699237346649169922, 1.23578119277954101562, 0.36165928840637207031,
1.05424487590789794922, 0.88268274068832397461, 1.19334447383880615234,
0.50116479396820068359

Wersja przekształcona double:
0.10000000000000000555, 0.36999999999999999556, 1.06929999999999999172,
0.84699253000000018776, 1.23578108237259720600, 0.36165967884042515834,
1.05424555454648222979, 0.88268117146202706991, 1.19334653448737904040,
0.50115828383040417204


Wersja przekształcona long double:
0.1, 0.37, 1.06930000000000000001, 0.84699252999999999976, 1.23578108237259730040,
0.36165967884042485591, 1.05424555454648216787, 0.88268117146202853510,
1.19334653448737708640, 0.50115828383041032595
```

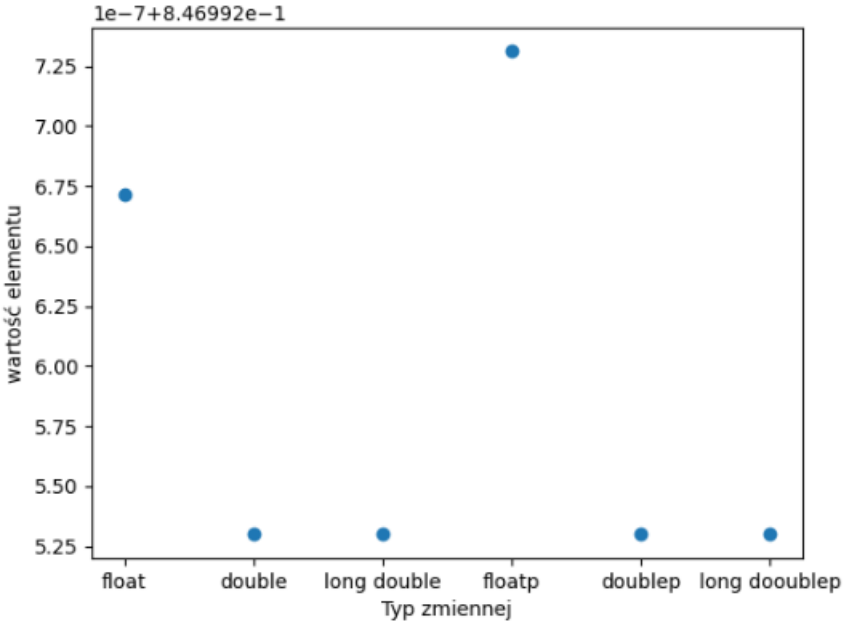
```
Wersja podstawowa float vs wersja podstawowa double:
1.4901161138336505019e-009, 4.7683715864721420985e-009,
5.5503845297621978716e-008, -1.5653350848854330479e-007,
1.1040694380959337195e-007, -3.9043405308802903164e-007, -
6.6955692434866875828e-007, 1.5692262969047021 67e-006, -
2.0606485728880130637e-006, 6.5101377965115503343e-006

Wersja podstawowa float vs wersja przekształcona float:
0, 0, 0, 0, 0, 0, 0, 0, 0, 0

Wersja przekształcona float vs wersja podstawowa double:
1.4901161138336505019e-009, 4.7683715864721420985e-009,
5.5503845297621978716e-008, -1.5653350848854330479e-007,
1.1040694380959337195e-007, -3.9043405308802903164e-007, -
6.6955692434866875828e-007, 1.5692262969047021 67e-006, -
2.0606485728880130637e-006, 6.5101377965115503343e-006

Wersja  podstawowa double vs wersja podstawowa long double:
0, -2.3310346708438345331e-017, -1.1644331332494317621e-016,
2.6909897921090220052e-016, -1.8192912454306764403e-016,
5.9837117899474989713e-016, 1.1666015375944027 71e-015, -
2.7383152169380142738e-015, 3.6041048617763138395e-015, -
1.1378593380018120484e-014
```

Wnioski pozostają takie same jak poprzednio. Jedyne co ciekawe, po zmianie kompilatora podstawowa wersja floata jest równa tej przekształconej., czego wcześniej nie uzyskaliśmy. Jak widać zmiana kompilatora również wpływa na wyniki doświadczenia.

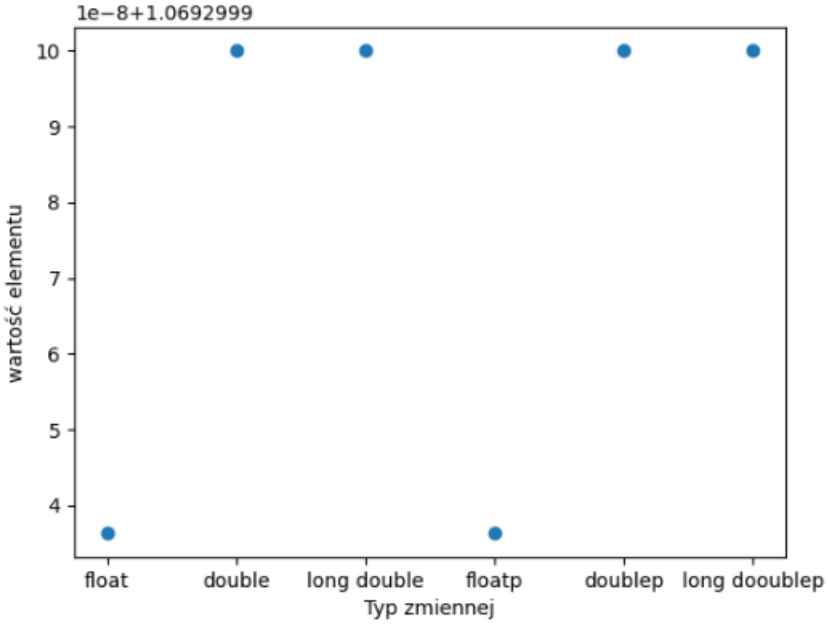
Wykresy



Wykres 1: wartość elementu dla x_0



Wykres 2: Wartość elementu dla x_3



Wykres 3: porównanie wyników dla x_2 z precyzją ustawioną na 20

Jak widać long double jest bardzo zbliżony do double, różnica między nimi jest marginalna dla precyzji 20.

Wnioski

Double/Long double zapewnia zdecydowanie większą precyzję obliczeń niż float, jednak odbywa się to kosztem czasu. Wynika to z ilości bitów przeznaczonych dla tych typów, co powoduje że double wykorzystuje do spamiętywania większą ilość miejsc po przecinku.

Dla największej dokładności powinniśmy wykorzystywać typ long double. W związku z rozwojem technologii i zwiększeniem ilości pamięci, wielkości nie będą nas raczej ograniczać i dzięki temu powinniśmy wykorzystywać long double, aby wyniki naszych doświadczeń były najdokładniejsze.

Po zmianach, okazuję się iż jeśli chcemy uzyskać jak najdokładniejsze wyniki powinniśmy nie tylko wykorzystywać long double ale także powtarzać doświadczenie dla wielu różnych kompilatorów, gdyż jak pokazał przykład na górze, inny kompilator potrafi uzyskać niekiedy całkiem inne wyniki.