

Metody Obliczeniowe w Nauce i Technice

Laboratorium I

Maciej Trątnowiecki

AGH, Semestr Letni, 2020

Sumowanie liczb pojedynczej precyzji

W ramach laboratorium zaimplementowałem trzy algorytmy obliczające sumę N liczb z tablicy, każdej o wartości v . Za N przyjąłem 10^7 , za v początkowo $v = 0.53125$.

Obliczyłem błąd wzgledny i bezwzględny jakim obarczone były przeprowadzone operacje arytmetyczne, uzyskując poniższe wyniki.

Mierzona wartość	Sumowanie proste	Sumowanie rekursywne	Sumowanie Kahana
Wynik sumowania	5030840.50	5312500.00	5312500.00
Błąd bezwzględny	281659.50	0.00	0.00
Błąd wzgledny	5.30%	0.00%	0.00%

Tabela 1: Wyniki obliczeń dla $v = 0.53125$

Następnie zmieniłem wartość v na 0.63211, uzyskując poniższe wyniki.

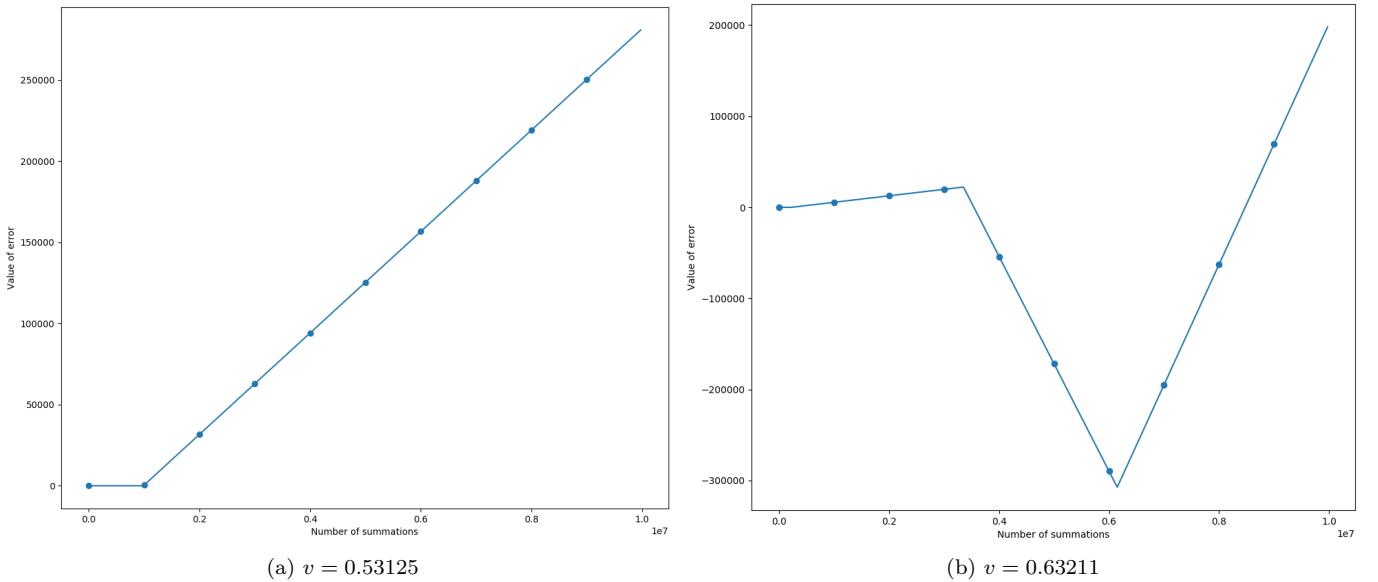
Mierzona wartość	Sumowanie proste	Sumowanie rekursywne	Sumowanie Kahana
Wynik sumowania	6119759.50	6321100.00	6321100.00
Błąd bezwzględny	201340.50	0.00	0.00
Błąd wzgledny	3.18%	0.00%	0.00%

Tabela 2: Wyniki obliczeń dla $v = 0.63211$

Błąd wzgledny w algorytmie prostym znacznie rośnie, ponieważ wielokrotnie sumuje on liczby zupełnie różnych rzędów wielkości. Błąd wzgledny jest znacznie mniejszy w algorytmie Kahana, jak i sumowania rekursywnego względem algorytmu prostego. W algorytmie rekursywnym, błąd nie występuje, ponieważ sumowane liczby są tego samego rzędu. Algorytm Kahana śledzi występujący błąd obliczeń i minimalizuje go w kolejnych sumowaniach. Jest to możliwe, ponieważ w zmiennej err przechowywany jest występujący na mniej znaczących bitach błąd dokładności. Żeby go obliczyć, algorytm najpierw oblicza różnicę stosunkowo dużych liczb reprezentujących aktualną i poprzednią sumę, dopiero następnie odejmuje od niego stosunkowo małą dodawaną aktualnie wartość.

Prześledziłem również jak wyglądał wzrost błędu w algorytmie prostego sumowania. Zebrane wyniki przedstawiłem na poniższych wykresach. Błękitnymi kropkami oznaczono każdy milion kroków.

Łatwo zauważyc, że wykres narastania błędu jest ciągłą sumą funkcji liniowych. W początkowej fazie, gdy dodawane wartości są tego samego rzędu, błąd nie rośnie w ogóle, lub rośnie wolno.



Rysunek 1: Wykres zmiany błędu bezwzględnego w stosunku do liczby sumowań.

Wykonałem także porównanie czasów wykonania dla każdego z trzech algorytmów. Do pomiarów czasu w sposób powtarzalny użyłem pakietu sio2jail (<https://github.com/sio2project/sio2jail>), którego specyfika została lepiej opisana w pracy licencjackiej (<https://hitagi.dasie.mimuw.edu.pl/files/licencjat/pracalic-logo.pdf>), za pośrednictwem skryptu opakowującego oiejq (<https://oi.edu.pl/static/attachment/20181007/oiejq.tar.gz>), używanego przez Ogólnopolską Olimpiadę Informatyczną. Otrzymałem następujące wyniki.

Algorytm	Czas wykonania
Prosty	21ms
Rekursywny	156ms
Kahana	21ms

Tabela 3: Pomiar czasów wykonania

Dla uprzednio rozważanych wartości v zarówno algorytm rekursywny, jak i algorytm Kahana zwracał dokładny wynik. Jednak dla $v = 0.666666$ w wyniku działania algorytmu rekursywnego otrzymałem wynik obarczony błędem bezwzględnym rzędu 0.50, podczas gdy algorytm Kahana zwrócił wynik dokładny.

Sumy częściowe

Obliczyłem wartość funkcji ζ i η dla wskazanych parametrów w przód i wstecz. Dla $n < 200$ nie zauważylem rozbieżności. Otrzymane wyniki zebrałem w poniższej tabeli. Występujące różnice oznaczylem kolorem pomarańczowym.

Wartość s	ζ w przód	ζ wstecz	η w przód	η wstecz
2.0	1.625133	1.625133	0.822271	0.822271
3.6667	1.109399	1.109400	0.934693	0.934693
5.0	1.036927	1.036928	0.972120	0.972120
7.2	1.007228	1.007228	0.993527	0.993527
10.0	1.000995	1.000995	0.999040	0.999040

Tabela 4: $n = 50$, float

Wartość s	ζ w przód	ζ wstecz	η w przód	η wstecz
2.0	1.634984	1.634984	0.822417	0.822417
3.6667	1.109409	1.109409	0.934693	0.934693
5.0	1.036927	1.036928	0.972120	0.972120
7.2	1.007228	1.007228	0.993527	0.993527
10.0	1.000995	1.000995	0.999040	0.999040

Tabela 5: $n = 100$, float

Wartość s	ζ w przód	ζ wstecz	η w przód	η wstecz
2.0	1.639947	1.639946	0.822455	0.822455
3.6667	1.109409	1.109410	0.934693	0.934693
5.0	1.036927	1.036928	0.972120	0.972120
7.2	1.007228	1.007228	0.993527	0.993527
10.0	1.000995	1.000995	0.999040	0.999040

Tabela 6: $n = 200$, float

Wartość s	ζ w przód	ζ wstecz	η w przód	η wstecz
2.0	1.642936	1.642936	0.822465	0.822465
3.6667	1.109409	1.109411	0.934693	0.934693
5.0	1.036927	1.036928	0.972120	0.972120
7.2	1.007228	1.007228	0.993527	0.993527
10.0	1.000995	1.000995	0.999040	0.999040

Tabela 7: $n = 500$, float

Wartość s	ζ w przód	ζ wstecz	η w przód	η wstecz
2.0	1.643935	1.643934	0.822467	0.822467
3.6667	1.109409	1.109411	0.934693	0.934693
5.0	1.036927	1.036928	0.972120	0.972120
7.2	1.007228	1.007228	0.993527	0.993527
10.0	1.000995	1.000995	0.999040	0.999040

Tabela 8: $n = 1000$, float

Następnie wykonałem te same obliczenia stosując zmienne zmiennoprzecinkowe podwójnej precyzji, otrzymując te same wyniki licząc w przód i wstecz, dla obu rozpatrywanych funkcji. W miejscach, gdzie przy zastosowaniu pojedynczej precyzji wystąpiły rozbieżności kolorem zielonym oznaczyłem dokładne wartości. W celu wytłumaczenia zaobserwowanych rozbieżności dla wyniku uzyskiwanego w przód i wstecz dla reprezentacji pojedynczej precyzji rozpatrzmy błąd operacji zmiennoprzecinkowych ϵ dla obu sposobów prowadzenia obliczeń.

$$fl(fl(x + y) + z) = fl((x + y)(1 + \epsilon) + z) = ((x + y)(1 + \epsilon) + z)(1 + \epsilon) = (x + x\epsilon + y + y\epsilon + z)(1 + \epsilon)$$

$$fl(fl(x + y) + z) = (x + y + z + \epsilon(x + y))(1 + \epsilon)$$

$$fl(x + fl(y + z)) = fl(x + (y + z)(1 + \epsilon)) = (x + (y + z)(1 + \epsilon))(1 + \epsilon) = (x + y + y\epsilon + z + z\epsilon)(1 + \epsilon)$$

$$fl(x + fl(y + z)) = (x + y + z + \epsilon(z + y))(1 + \epsilon)$$

Zauważmy, że gdy $|x + y| < |y + z|$ w drugiej równości otrzymujemy większy błąd wzajemny. Możemy stąd wnioskować, że wartość funkcji obliczanej w przód jest obarczona większym błędem wzajemnym, gdyż w pamięci reprezentujemy większą sumę. Przyjmuje ona postać $\frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \dots + \frac{1}{n^s}$, gdzie $s > 1$. Łatwo zauważać, że początkowe wyrazy ciągu są co do wartości większe od końcowych, zatem ich suma też jest większa. Nie znaczy to jednak, że przy sumowaniu wstecz nie występuje błąd - stąd nie zawsze otrzymamy wynik pokrywający się z obliczeniami dokładnymi uzyskanymi po zastosowaniu podwójnej precyzji.

Błędy zaokrągleń i odwzorowanie logistyczne

W celu prześledzenia zbieżności procesu iteracyjnego określonego zadanym odwzorowaniem logistycznym przygotowałem interaktywny wykres wartości kolejnych elementów ciągu dla wartości stałej r z przedziału $[1.0, 4.0]$ i $x_0 = 0.77$. Ze względu na ograniczenia techniczne formatu pdf, wspomniany wykres umieściłem pod poniższym adresem http://users.v-lo.krakow.pl/~maciektr/mownit/lab1_logistic_map.gif.

Za jego pomocą łatwo wywnioskować, że:

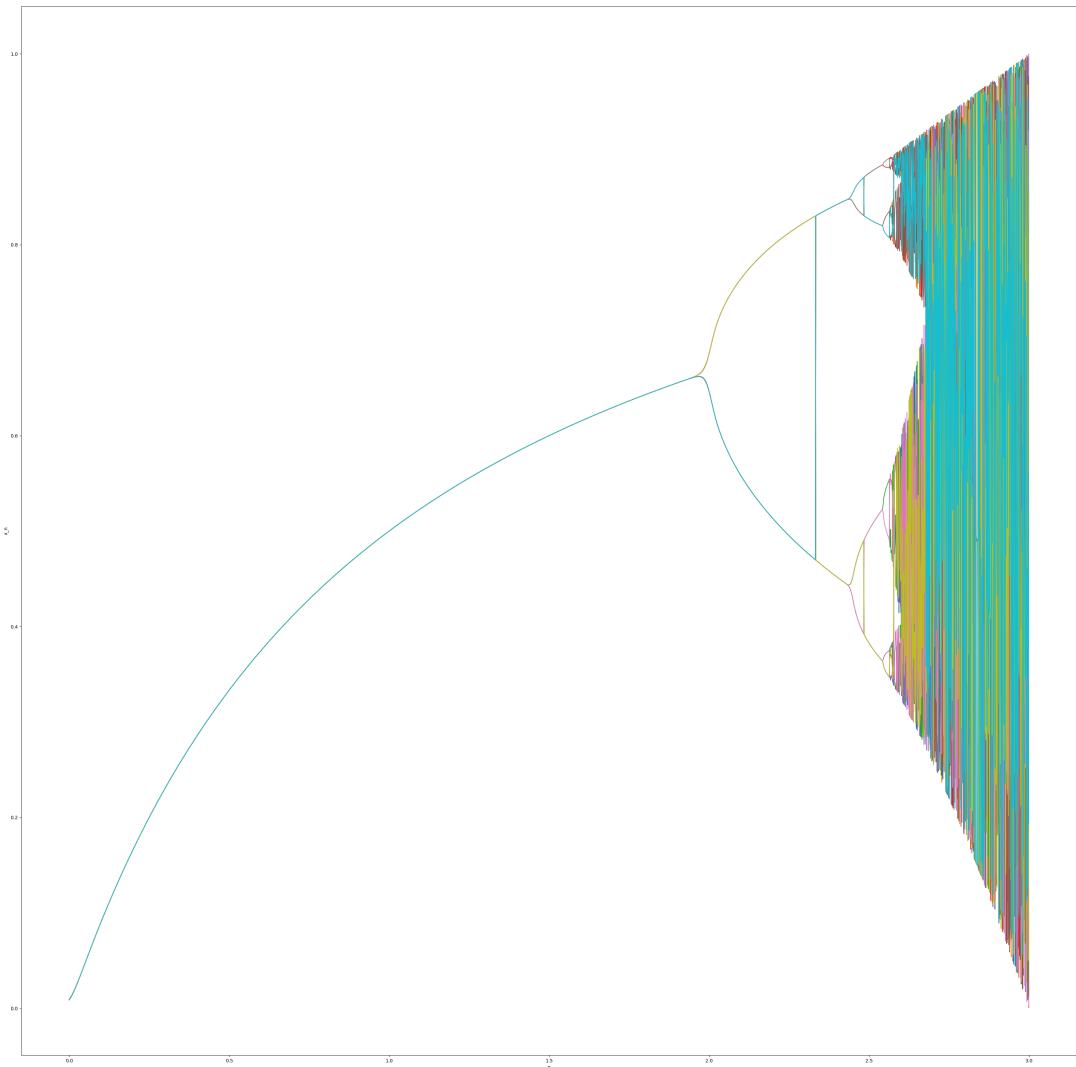
- dla $r \in [1.0, 2.0]$ - kolejne elementy ciągu szybko zbiegają do swojej granicy. Możemy prosto wyznaczyć ją sposobem wspomniany poniżej.
- dla $r \in [2.0, 3.0]$ - ciąg jest zbieżny, ale w początkowej fazie wahaje się.
- dla $r \in [3.0, 3.5]$ - odwzorowanie tworzy oscylator.
- dla $r \in [3.5, 4.0]$ - otrzymujemy chaotyczne drgania.

Załóżmy, że ciąg x_n jest zbieżny. Niech $\lim_{n \rightarrow \infty} x_n = g$. Wtedy:

$$\lim_{n \rightarrow \infty} x_{n+1} = g = \lim_{n \rightarrow \infty} rx_n(1 - x_n) = rg(1 - g) \implies r(1 - g) = 1 \implies g = \frac{r - 1}{r}$$

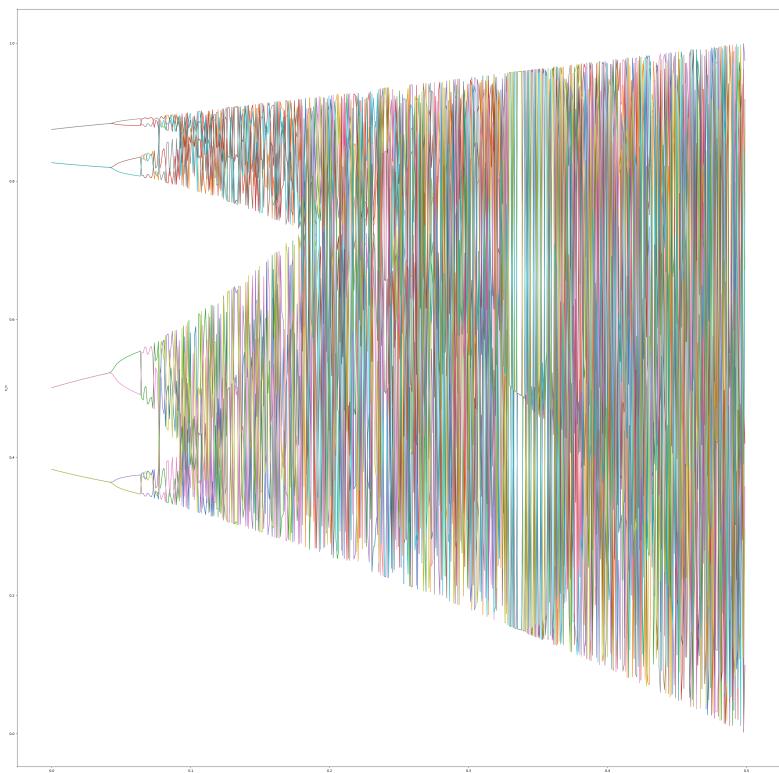
Następnie przygotowałem wykres bifurkacyjny dla zadанego odwzorowania. Oryginalny obraz, o rozdzielczości 4000×4000 pikseli umieściłem pod następującym adresem http://users.v-lo.krakow.pl/~maciektr/mownit/lab1_bifurcation.png.

Diagram bifurkacyjny pokazuje rozwidlanie się stanów stabilnych odwzorowania logistycznego. Widzimy, że oscylacje odbywają się wokół kolejno jednej, dwóch, czterech itd. wartości, aż nie przejdą w chaos. Łatwo zauważać związek otrzymanego wykresu, z wcześniej określonymi przedziałami zbieżności.

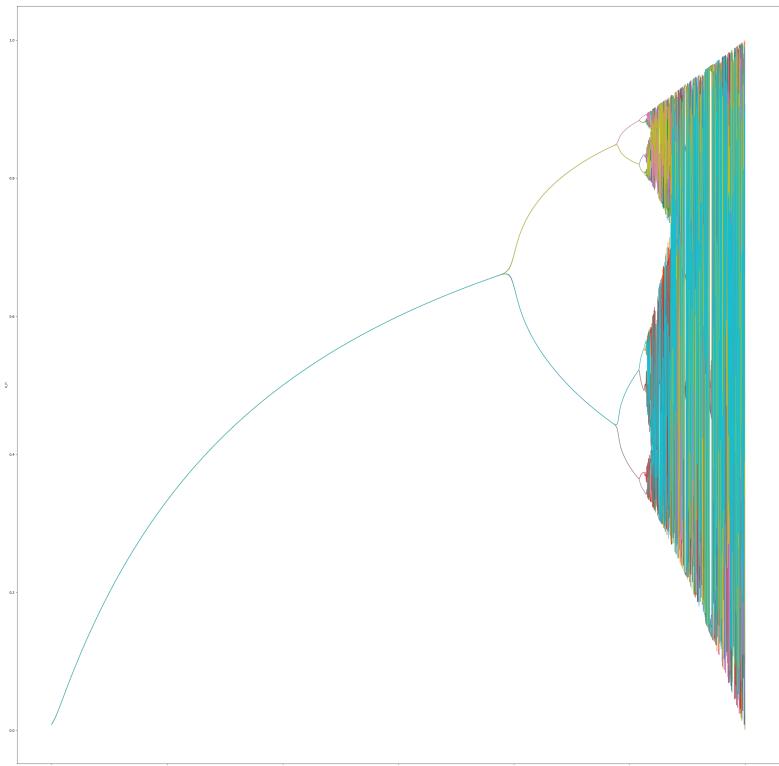


(a) $x_0 = 0.7; r \in (1.0, 4.0)$

Rysunek 2: Diagram bifurkacyjny.

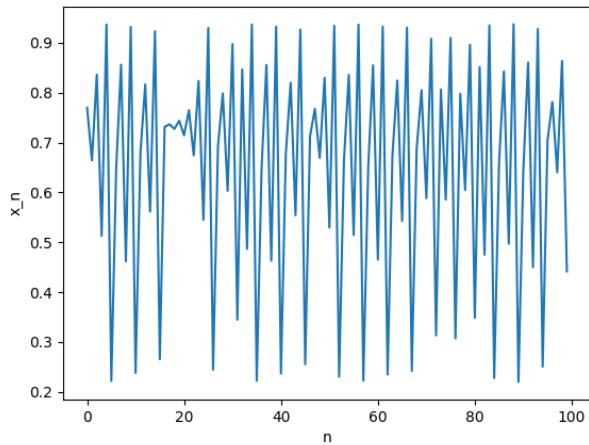


(a) $x_0 = 0.7; r \in (3.5, 4.0)$

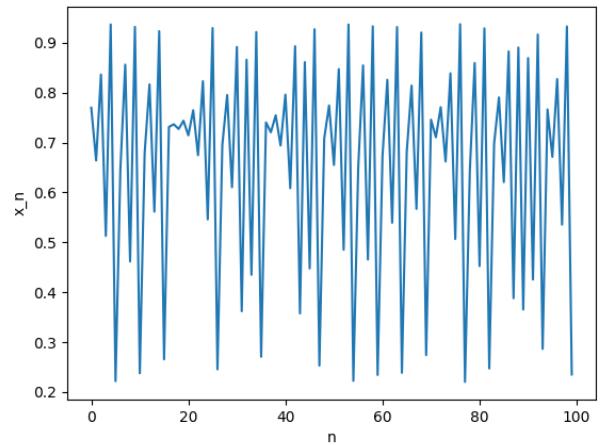


(b) $x_0 = 0.8; r \in (1.0, 4.0)$

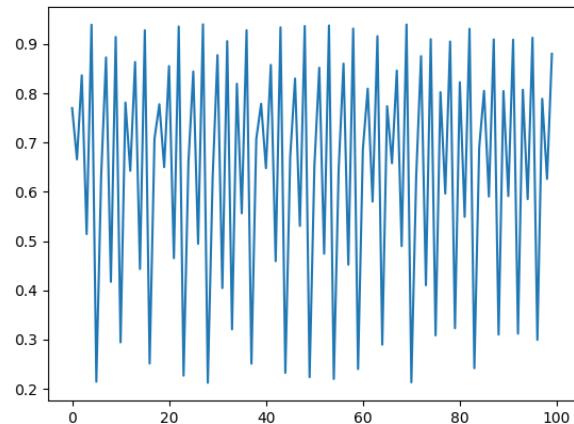
Prześledziłem również zmianę trajektorii liczonych z użyciem pojedynczej i podwójnej precyzji, dla $x_0 = 0.7$.



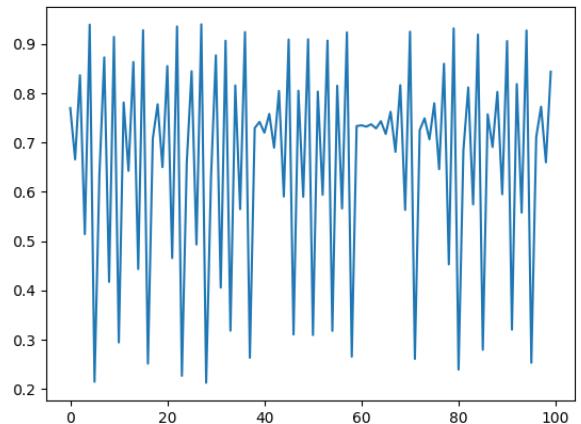
(a) $r = 3.75$, float



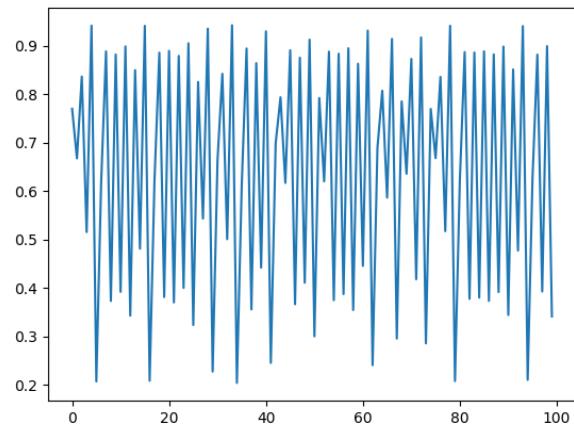
(b) $r = 3.75$, double



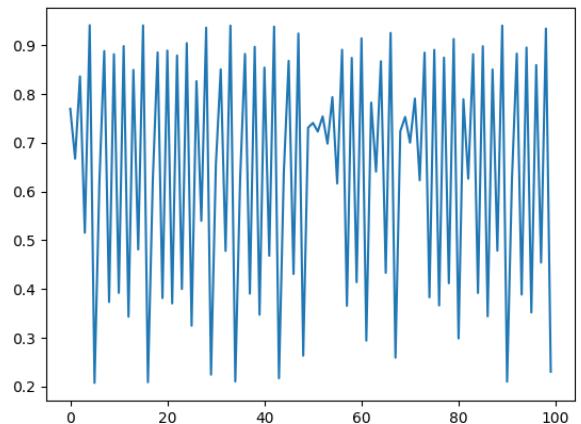
(c) $r = 3.76$, float



(d) $r = 3.76$, double

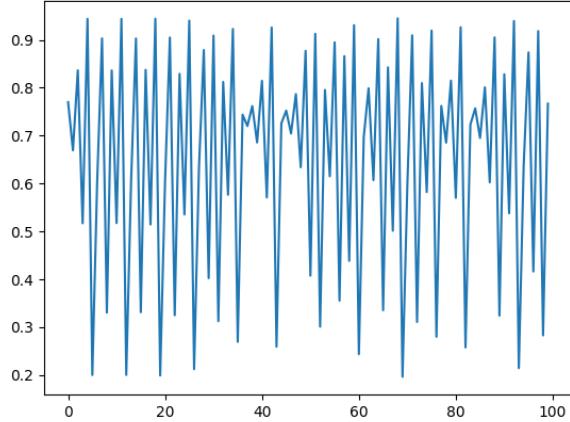


(e) $r = 3.77$, float

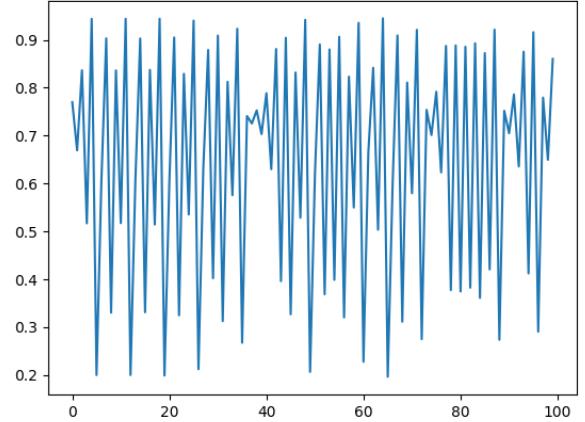


(f) $r = 3.77$, double

Rysunek 4: Trajektorie w pojedynczej i podwójnej precyzji.



(a) $r = 3.78$, float



(b) $r = 3.78$, double

Rysunek 5: Trajektorie w pojedynczej i podwójnej precyzyji.

Widzimy, że użycie podwójnej precyzyji obliczeń skutkuje otrzymaniem zmienionej trajektorii w porównaniu z pojedynczą precyją, jednak dalej ma ona charakter chaotyczny.

Następnie wyznaczyłem liczbę iteracji potrzebnych do osiągnięcia zera w pojedynczej precyzyji, dla $r = 4$. Dla $x_0 = 0.77$, otrzymałem 4135 iteracji, przyjmując dokładność $\epsilon = 4.8 * 10^{-7}$.

Dla $x_0 = 0.6$ otrzymałem 400 iteracji, z tą samą dokładnością.

Dla $v = 0.8$ otrzymałem 757 iteracji z dokładnością $\epsilon = 5.5 * 10^{-6}$.

Porównując z zerem liczby zmienoprzecinkowe pojedynczej precyzyji musimy godzić się na przyjęcie dokładności porównania. W innym wypadku nie otrzymamy poprawnego wyniku. Największa dokładność z jaką możemy dokonać porównania zależna jest od danych wejściowych.