Metody Obliczeniowe w Nauce i Technice
Laboratorium 1
Arytmetyka komputerowa

Patryk Wojtyczek

## Zadanie 1

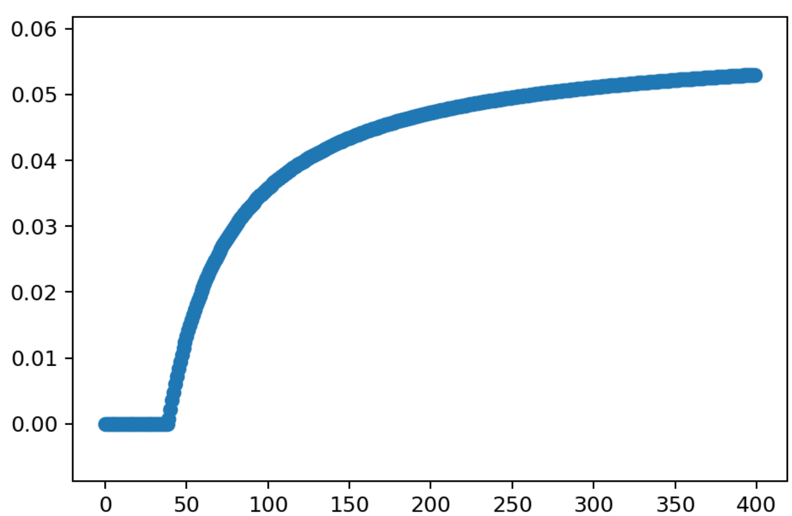
1. Napisz program, który oblicza sume N liczb pojedynczej precyzji przechowywanych w tablicy o N = 107 elementach. Tablica wypełniona jest ta sama wartościa v z przedziału [0.1, 0.9] np. v = 0.53125.

   **Jupyter Notebook**

2. Wyznacz bezwzgledny i wzgledny bład obliczeń. Dlaczego bład wzgledny jest tak duży?

   Bład bezwzgledny wyniósł 281659.5, wzgledny $\approx 5\%$. Duża wartość błedu bierze sie z różnicy wielkości pomiedzy liczba v a akumulatorem, który po wielu iteracjach jest znacznie wiekszy od dodwanej do niego wartości v. Dodajac liczby, które znancznie sie od siebie różnia gubimy dokładność przez normalizacje mantysy.

3. W jaki sposób rośnie bład wzgledny w trakcie sumowania? Przedstaw wykres (raportuj wartość błedu co 25000 kroków) i dokonaj jego interpretacji.




Z wykresu widać, że do momentu gdy różnica pomiedzy akumulatorem a v była nieznaczna bład był zerowy. Po około $50 \cdot 25000$ iteracji różnica ta zaczeła prowadzić do akumulowania sie coraz znaczniejszego błedu (tak jak opisano w punkcie wyżej). Wykres dalej wypłaszcza sie, co nie znaczy, że bład nie rośnie tylko jest coraz mniej istotny wobec akumulatora (wykres wartości bezwzglednej z błedy byłby funkcja stała nastepnie prosta).

4. Zaimplementuj rekurencyjny algorytm sumowania, działajacy jak na rysunku poniżej.

   **Jupyter Notebook**

5. Wyznacz bezwzgledny i wzgledny bład obliczeń. Dlaczego bład wzgledny znacznie zmalał?

   Zarówno bład wzgledny jak i bezwzgledny wynosi 0. Algorytm upewnia sie, że nie dodajemy do siebie liczb znacznie sie od siebie różniacych (dodajemy zawsze dwie liczby o tej samej wartości) przez co nie tracimy dokładności.

6. Porównaj czas działania obu algorytmów dla tych samych danych wejściowych. Otrzymane czasy działania

| | |
|---|---|
| Proste sumowanie | 2.384s |
| Rekurencyjne sumowanie | 7.865s |

   Proste sumowanie to jedno przejście po tablicy, sumowanie rekurencyjne to logn przejść po coraz mniejszych kawałkach tablicy.

7. Przedstaw przykładowe dane wejściowe, dla których algorytm sumowania rekurencyjnego zwraca niezerowy bład.

   Edge case dla rekurencyjnego algorytmu to sumowanie liczb uporzadkowanych dlatego, że wtedy nie udaje nam sie uciec od sumowania liczb dalekich od siebie - poczatkowe wartości beda znacznie mniejsze od tych dalszych w miare postepowania algorytmu. Niezerowy bład pojawia sie już przy 10000 kolejnych liczb naturalnych.

# Zadanie 2

1. Wyznacz bezwzgledny i wzgledny bład obliczeń dla tych samych danych wejściowych jak w przypadku testów z Zadania 1.

   Bład wzgledny i bezwzgledny wyniósł 0.

2. Wyjaśnij dlaczego w algorytm Kahana ma znacznie lepsze własności numeryczne? Do czego służy zmienna err?

   Algorytm Kahana stara sie niejako odzyskiwać bity utracone przy dodawaniu liczb znacznie różniacych sie od siebie. Algebraicznie zmienna ‘err‘ powinna zawsze być zerem, operujac na liczbach zmiennoprzecinkowych nie zawsze tak jest (miedzy innymi gdy liczby sum i tab[i] znacznie sie od siebie różnia). Zmienna ‘err‘ odzyskuje utracone bity przy dodawaniu (jeśli takowe były) - ‘(temp - sum) - y;‘ i w nastepnej iteracji dodaje utracona wartość do zmiennej która dodajemy do akumulatora.

3. Porównaj czasy działania algorytmu Kahana oraz algorytmu sumowania rekurencyjnego dla tych samych danych wejściowych.

| | |
|---|---|
| Rekurencyjne sumowanie | 6.984s |
| Kahan | 6.755s |

   Algorytmy maja podobny czas działania - wykonuja podobna ilość operacji, choć na pierwszy rzut oka algorytm Kahana wydawał sie szybszy.

# Zadanie 3

Różnice sa niewielkie - wystepuja na szóstym miejscu po przecinku. Podwójna precyzja daje wiecej miejsc po przecinku, dokładniejsze jest sumowanie od tyłu - co widać, porównujac wartości sumowania z pojedyńcza i podwójna precyzja.

| | N | S | Dzeta forward | Dzeta forward - double | Dzeta backwards | Dzeta backwards - double |
|---|---|---|---|---|---|---|
| x | 50.0 | 2.0000 | 1.625133 | 1.625133 | 1.625133 | 1.625133 |
| x | 100.0 | 2.0000 | 1.634984 | 1.634984 | 1.634984 | 1.634984 |
| x | 200.0 | 2.0000 | 1.639947 | 1.639947 | 1.639946 | 1.639947 |
| x | 500.0 | 2.0000 | 1.642936 | 1.642936 | 1.642936 | 1.642936 |
| x | 1000.0 | 2.0000 | 1.643935 | 1.643935 | 1.643934 | 1.643935 |
| x | 50.0 | 3.6667 | 1.109399 | 1.109400 | 1.109400 | 1.109400 |
| x | 100.0 | 3.6667 | 1.109409 | 1.109409 | 1.109409 | 1.109409 |
| x | 200.0 | 3.6667 | 1.109409 | 1.109410 | 1.109410 | 1.109410 |
| x | 500.0 | 3.6667 | 1.109409 | 1.109411 | 1.109411 | 1.109411 |
| x | 1000.0 | 3.6667 | 1.109409 | 1.109411 | 1.109411 | 1.109411 |
| x | 50.0 | 5.0000 | 1.036927 | 1.036928 | 1.036928 | 1.036928 |
| x | 100.0 | 5.0000 | 1.036927 | 1.036928 | 1.036928 | 1.036928 |
| x | 200.0 | 5.0000 | 1.036927 | 1.036928 | 1.036928 | 1.036928 |
| x | 500.0 | 5.0000 | 1.036927 | 1.036928 | 1.036928 | 1.036928 |
| x | 1000.0 | 5.0000 | 1.036927 | 1.036928 | 1.036928 | 1.036928 |
| x | 50.0 | 7.2000 | 1.007228 | 1.007228 | 1.007228 | 1.007228 |
| x | 100.0 | 7.2000 | 1.007228 | 1.007228 | 1.007228 | 1.007228 |
| x | 200.0 | 7.2000 | 1.007228 | 1.007228 | 1.007228 | 1.007228 |
| x | 500.0 | 7.2000 | 1.007228 | 1.007228 | 1.007228 | 1.007228 |
| x | 1000.0 | 7.2000 | 1.007228 | 1.007228 | 1.007228 | 1.007228 |
| x | 50.0 | 10.0000 | 1.000995 | 1.000995 | 1.000995 | 1.000995 |
| x | 100.0 | 10.0000 | 1.000995 | 1.000995 | 1.000995 | 1.000995 |
| x | 200.0 | 10.0000 | 1.000995 | 1.000995 | 1.000995 | 1.000995 |

Figure 1: Obliczenia dla dzeta

| | N | S | Dzeta forward | Dzeta forward - double | Dzeta backwards | Dzeta backwards - double |
|---|---|---|---|---|---|---|
| x | 50.0 | 2.0000 | 1.625133 | 1.625133 | 1.625133 | 1.625133 |
| x | 100.0 | 2.0000 | 1.634984 | 1.634984 | 1.634984 | 1.634984 |
| x | 200.0 | 2.0000 | 1.639947 | 1.639947 | 1.639946 | 1.639947 |
| x | 500.0 | 2.0000 | 1.642936 | 1.642936 | 1.642936 | 1.642936 |
| x | 1000.0 | 2.0000 | 1.643935 | 1.643935 | 1.643934 | 1.643935 |
| x | 50.0 | 3.6667 | 1.109399 | 1.109400 | 1.109400 | 1.109400 |
| x | 100.0 | 3.6667 | 1.109409 | 1.109409 | 1.109409 | 1.109409 |
| x | 200.0 | 3.6667 | 1.109409 | 1.109410 | 1.109410 | 1.109410 |
| x | 500.0 | 3.6667 | 1.109409 | 1.109411 | 1.109411 | 1.109411 |
| x | 1000.0 | 3.6667 | 1.109409 | 1.109411 | 1.109411 | 1.109411 |
| x | 50.0 | 5.0000 | 1.036927 | 1.036928 | 1.036928 | 1.036928 |
| x | 100.0 | 5.0000 | 1.036927 | 1.036928 | 1.036928 | 1.036928 |
| x | 200.0 | 5.0000 | 1.036927 | 1.036928 | 1.036928 | 1.036928 |
| x | 500.0 | 5.0000 | 1.036927 | 1.036928 | 1.036928 | 1.036928 |
| x | 1000.0 | 5.0000 | 1.036927 | 1.036928 | 1.036928 | 1.036928 |
| x | 50.0 | 7.2000 | 1.007228 | 1.007228 | 1.007228 | 1.007228 |
| x | 100.0 | 7.2000 | 1.007228 | 1.007228 | 1.007228 | 1.007228 |
| x | 200.0 | 7.2000 | 1.007228 | 1.007228 | 1.007228 | 1.007228 |
| x | 500.0 | 7.2000 | 1.007228 | 1.007228 | 1.007228 | 1.007228 |
| x | 1000.0 | 7.2000 | 1.007228 | 1.007228 | 1.007228 | 1.007228 |
| x | 50.0 | 10.0000 | 1.000995 | 1.000995 | 1.000995 | 1.000995 |
| x | 100.0 | 10.0000 | 1.000995 | 1.000995 | 1.000995 | 1.000995 |
| x | 200.0 | 10.0000 | 1.000995 | 1.000995 | 1.000995 | 1.000995 |

Figure 2: Obliczenia dla eta

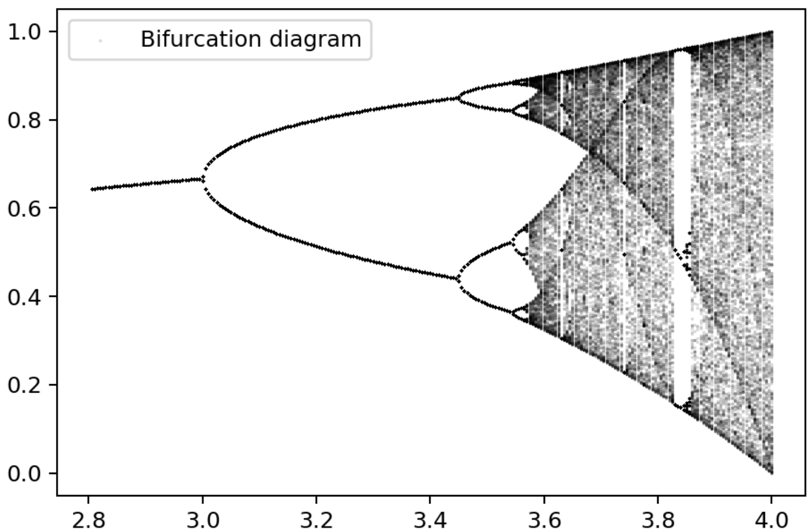
## Zadanie 4

Rozważ odwzorowanie logistyczne dane nastepujacym wzorem rekurencyjnym

$$x_{n+1} = rx_n \cdot (1 - x_n)$$

Przy czym $0 \leq x_n \leq 1$ i $r > 0$. Zbadaj zbieżność procesu iteracyjnego określonego tym równaniem w zależności od wartości parametru r oraz $x_0$.

- Dla różnych wartości r ($1 \leq r \leq 4$) oraz kilku wybranych wartości $x_0$ przedstaw na wykresie wartości $x_n$ uzyskane po wielu iteracjach odwzorowania logistycznego (diagram bifurkacyjny). Dokonaj interpretacji otrzymanych wyników.



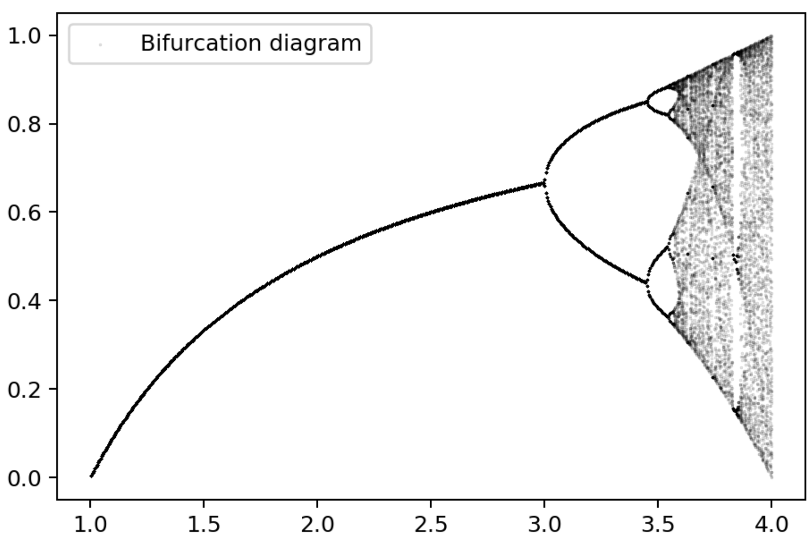




Figure 3: Przykładowe digagramy bifurkacji z dla różnych x (jeden $10^{-5}$ drugi 0.7). Różnia sie jedynie tym, że w jednym r zaczyna sie od 2.8, a w drugim od 1.0 co pozwala nam bliżej przyjrzeć sie chaosowi

Wykonanie wielu takich wykresów zaczynajacych sie z różnych x pokazuje, że zbieżność równania logistycznego nie zależy od $x_0$ ale wyłacznie od r.

- Dla tych samych wartości x0 oraz r ($3.75 \leq r \leq 3.8$) porównaj trajektorie obliczone z użyciem pojedynczej i podwójnej precyzji. Wyjaśnij otrzymane wyniki.

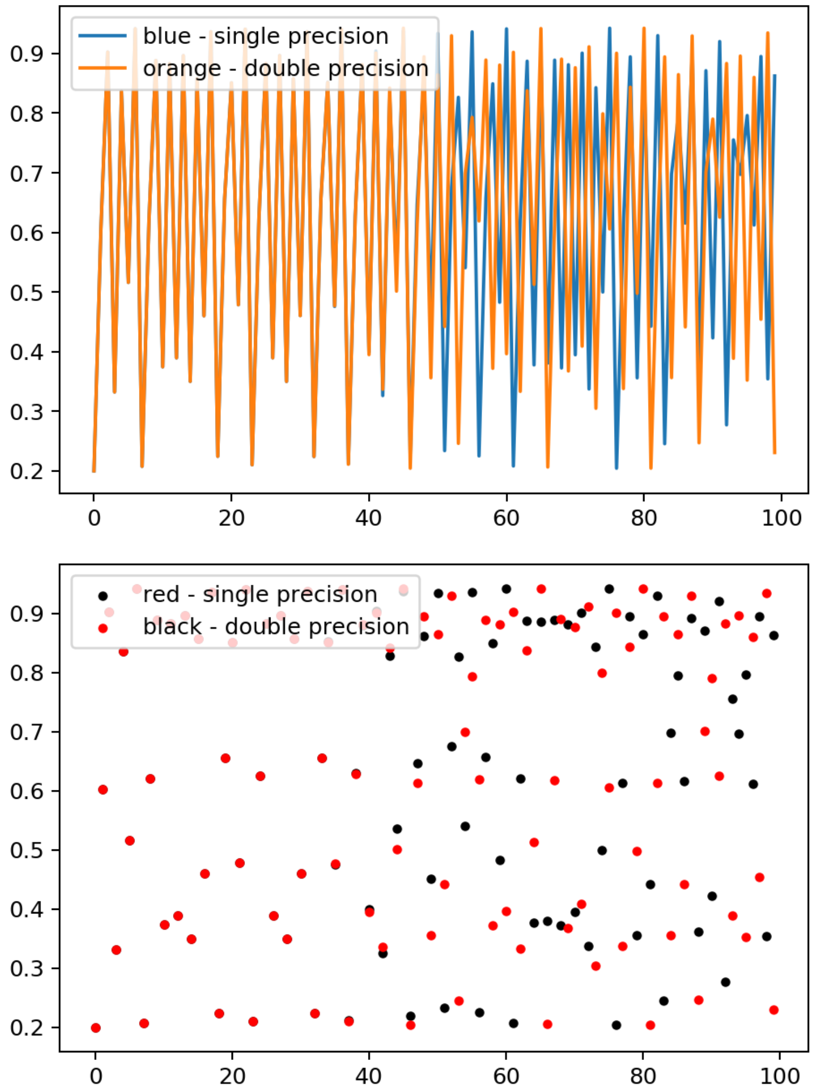



Figure 4: Wartości kolejnych $x_n$ obliczane przy różnej reprezentacji

Mały bład z poczatku nie zauważalny (do 40-tej iteracji) prowadzi do znacznego rozbiegniecia sie wartości odwzorowania logistycznego. Gdy wartości zaczeły sie nieco różnić w nastepnych iteracjach rozbiegały sie coraz bardziej.

- Dla r = 4 i różnych wartości x0 wyznacz (pojedyncza precyzja) liczbe iteracji potrzebnych do osiagniecia zera. Przedstaw interpretacje rezultatów.

| | Starting point | Iteration required to hit 0.0 |
|---|---|---|
| 0 | 0.0 | 0 |
| 1 | 0.1 | 1082 |
| 2 | 0.2 | 157 |
| 3 | 0.3 | 608 |
| 4 | 0.4 | 445 |
| 5 | 0.5 | 2 |
| 6 | 0.6 | 1227 |
| 7 | 0.7 | 608 |
| 8 | 0.8 | 669 |
| 9 | 0.9 | 763 |

Figure 5: Tabelka z iteracjami potrzebnymi do osiagniecia zera

Liczba iteracji do zbliżenia sie do zera jest bardzo różna dla różnych punktów startowych: od kilku do ponad tysiaca. Z poprzedniego diagramu bifurkacji widać, że dla r = 4, $x_n$ przechodza gesto przez liczby z zakresu 0.0 - 1.0. Pewne punkty startowe moga "opóźniać" zbieżność ale nie moga jej zatrzymać.