MOwNiT – arytmetyka komputerowa

Przygotował:
Szymon Budziak

Problem:
Niech ciąg x_k będzie zdefiniowany:

$$x_1 = 4, \quad x_{k+1} = 2^{2(k+1)+1} \cdot \frac{\sqrt{1 + x_k^2 / 2^{2(k+1)}} - 1}{x_k}$$

Zaproponować inną postać tego związku i obliczyć x_{30} dwoma sposobami.
Skomentować i spróbować objaśnić otrzymane wyniki.

Inna postać związku:
Podany wzór przekształcamy mnożąc ułamek przez "sztuczną jedynkę" jaką jest licznik tego ułamka tylko ze zmienionym znakiem przy 1 (+ 1).

$$x_{k+1} = 2^{2(k+1)+1} \cdot \frac{\sqrt{1 + x_k^2 / 2^{2(k+1)}} - 1}{x_k} =$$

$$= 2 \cdot 2^{2(k+1)} \cdot \frac{\sqrt{1 + x_k^2 / 2^{2(k+1)}} - 1}{x_k} \cdot \frac{\sqrt{1 + x_k^2 / 2^{2(k+1)}} + 1}{\sqrt{1 + x_k^2 / 2^{2(k+1)}} + 1} =$$

$$= 2 \cdot 2^{2(k+1)} \cdot \frac{1 + x_k^2 / 2^{2(k+1)} - 1}{x_k \cdot (\sqrt{1 + x_k^2 / 2^{2(k+1)}} + 1)} = 2 \cdot 2^{2(k+1)} \cdot \frac{x_k^2}{x_k \cdot 2^{2(k+1)} \cdot (\sqrt{1 + x_k^2 / 2^{2(k+1)}} + 1)} =$$

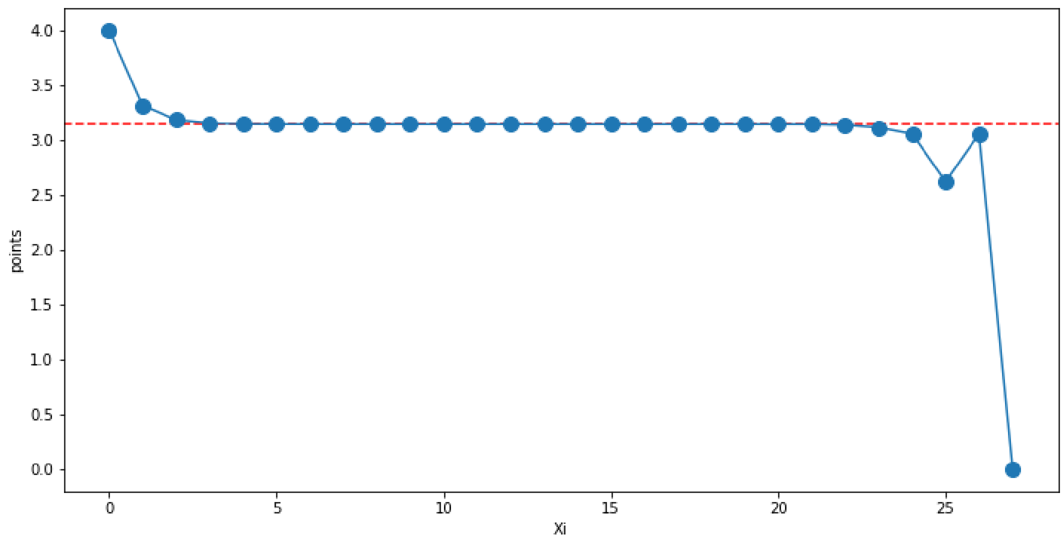
$$= \frac{2 \cdot x_k}{\sqrt{1 + x_k^2 / 2^{2(k+1)}} + 1}$$

W zadaniu zostały wykorzystane 3 typy danych z biblioteki numpy:

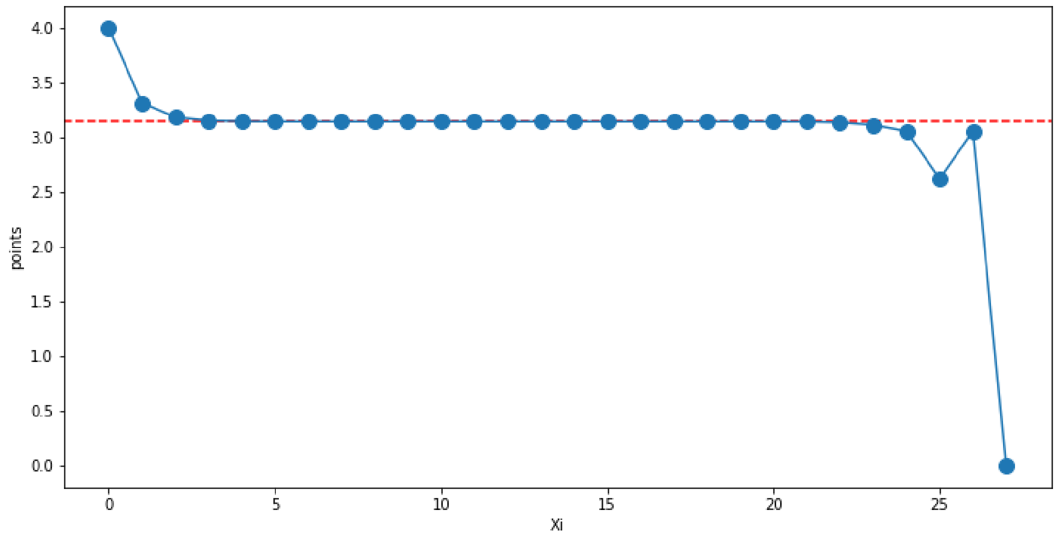
1) float32 - jest to typ liczby zmiennoprzecinkowej o pojedynczej precyzji, taki sam jak typ float z C, posiada on 32 bity precyzji, gdzie 8 przeznaczonych jest na cechę i 23 na mantysę,
2) double - w numpy jest to typ liczby zmiennoprzecinkowej o podwójnej precyzji, taki sam jak typ double z C, posiada on 64 bity precyzji z czego 11 przeznaczonych jest na cechę i 52 na mantysę,
3) long double - jest to typ liczby zmiennoprzecinkowej o rozszerzonej precyzji, taki sam jak typ long double z C, posiada on 128 bitów precyzji,

Dokładny opis typ zmiennych możemy otrzymać wywołując funkcję **np.finfo(zmienna)**, gdzie `zmienna` jest naszym typem danych. Funkcja ta użyta jest w jupyter notebooku.

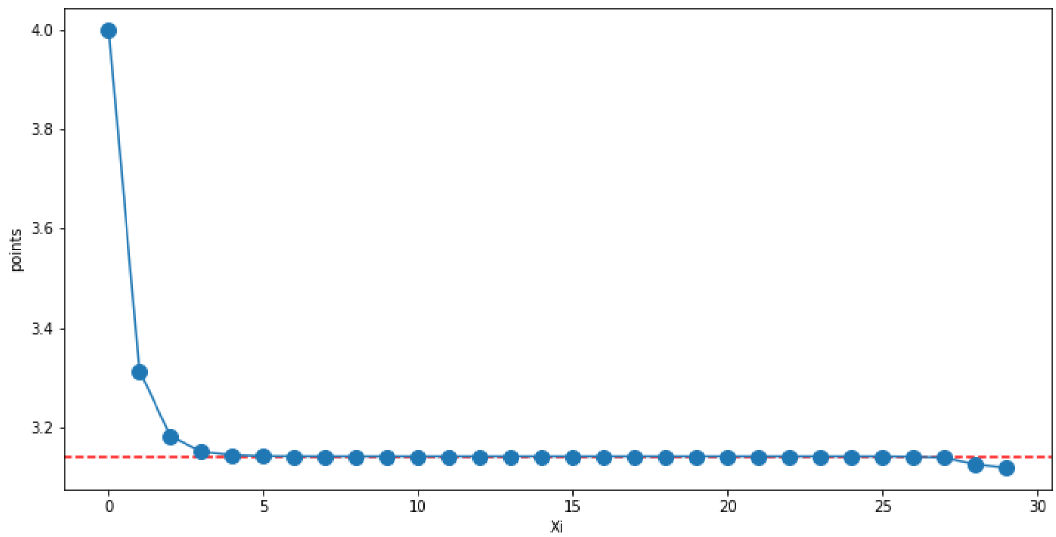
Obliczenia oraz wykresy dla kolejnych x przy pomocy podanego wzoru:



Wykres 1: Wykres dla 30 kolejnych wartości x obliczonych przy pomocy podanego wzoru dla typu float



Wykres 2: Wykres dla 30 kolejnych wartości x obliczonych przy pomocy podanego wzoru dla typu double



Wykres 3: Wykres dla 30 kolejnych wartości x obliczonych przy pomocy podanego wzoru dla typu long double

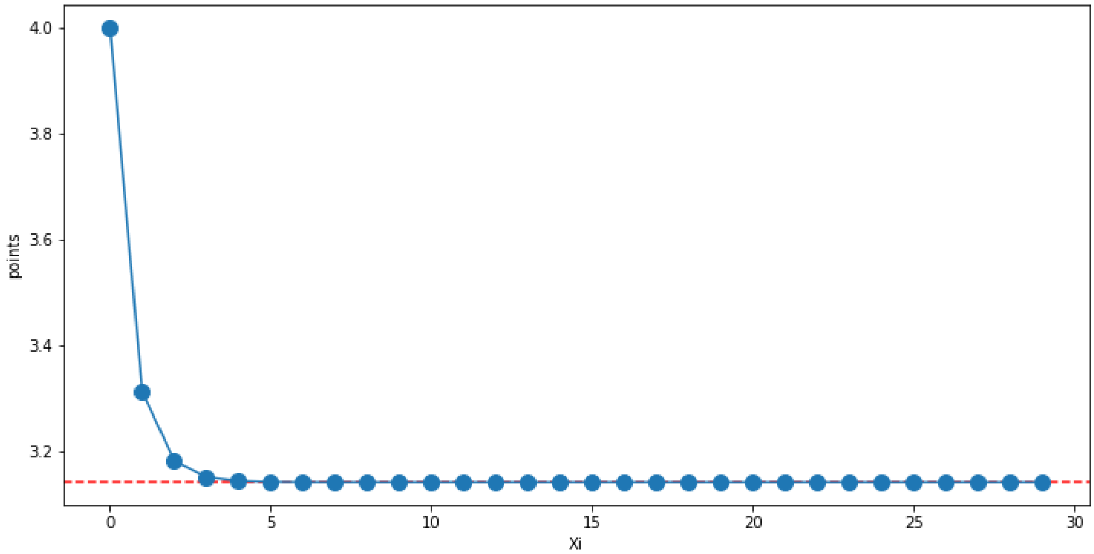
x_n	float	double	long double
x_1	4.0	4.0	4.0
x_2	3.3137085	3.313708498984761	3.3137084989847603901
x_3	3.1825979	3.1825978780745294	3.182597878074528113
x_4	3.1517248	3.151724907429258	3.1517249074292560925
x_5	3.1441183	3.1441183852458665	3.144118385245904277
…	…	…	…
x_{23}	3.1339834	3.1339832938853593	3.141591364112935589
x_{24}	3.1110568	3.1110567880253206	3.1415823152036629893
x_{25}	3.0536246	3.0536247478882985	3.141533079640214992
x_{26}	2.6198373	2.6198372951792175	3.1415046013554204743
x_{27}	3.0536249	3.0536247478882985	3.140911363218358626
x_{28}	0.0	0.0	3.1390172659624233085
x_{29}	nan	nan	3.12597834564362792
x_{30}	nan	nan	3.119023525414891949

Tabela 1: Obliczenia dla kolejnych wartości x przy pomocy podanego wzoru

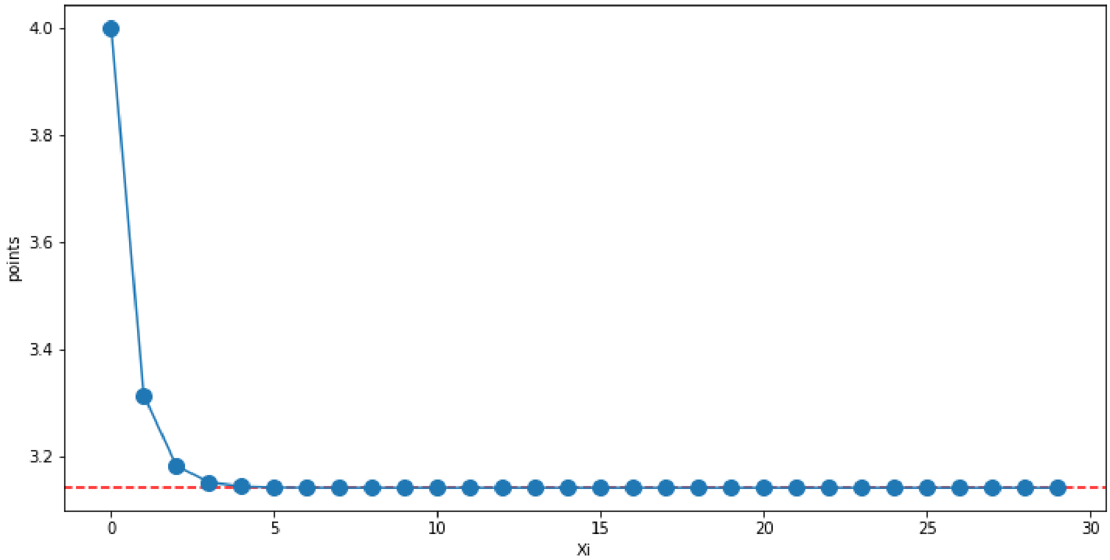
Ciąg który mamy zadany w zdaniu zbiega do wartości π.

Możemy zauważyć, że dla różnych typów danych mamy mniej więcej zbliżone wartości, co można zauważyć w tabeli oraz na wykresach. Jednak w tabeli widać różnice w ostatnich liczbach, które wynikają z różnych precyzji. Typy float i double nie zwracają już poprawnych wyników dla liczby π od 23 miejsca w obydwu przypadkach. Jeśli chodzi o typ long double tutaj od 28 miejsca możemy zauważyć błąd w dokładności drugiej cyfry po przecinku. Wynika z tego, że wszystkie przetestowane typy są za mało precyzyjne aby określić 28, 29 i 30 wyraz ciągu. W przypadku typu float i double 28 wyraz jest już równy 0.0, co powoduje błędy w obliczaniu kolejnych wyrazów, ponieważ to 0.0 zostaje użyte w mianowniku.

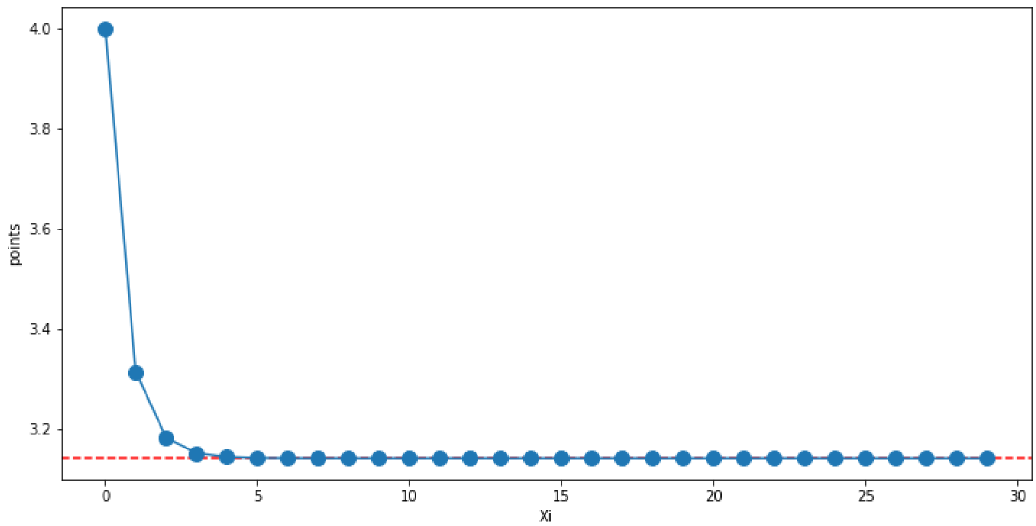
Obliczenia dla kolejnych x przy pomocy przekształconego wzoru:



Wykres 4: Wykres dla 30 kolejnych wartości x obliczonych przy pomocy przekształconego wzoru dla typu float



Wykres 5: Wykres dla 30 kolejnych wartości x obliczonych przy pomocy przekształconego wzoru dla typu double



Wykres 6: Wykres dla 30 kolejnych wartości x obliczonych przy pomocy przekształconego wzoru dla typu long double

x_n	float	double	long double
x_1	4.0	4.0	4.0
x_2	3.3137085	3.3137084989847607	3.3137084989847603905
x_3	3.1825979	3.1825978780745285	3.1825978780745281106
x_4	3.1517248	3.151724907429256	3.1517249074292560986
x_5	3.1441183	3.1441183852459047	3.1441183852459042628
…	…	…	…
x_{23}	3.1415927	3.1415926535898313	3.1415926535898299572
x_{24}	3.1415927	3.1415926535898033	3.141592653589802418
x_{25}	3.1415927	3.1415926535897962	3.1415926535897955331
x_{26}	3.1415927	3.141592653589795	3.141592653589793812
x_{27}	3.1415927	3.141592653589795	3.1415926535897933818
x_{28}	3.1415927	3.141592653589795	3.1415926535897932743
x_{29}	3.1415927	3.141592653589795	3.1415926535897932474
x_{30}	3.1415927	3.141592653589795	3.1415926535897932407

Tabela 2: Obliczenia dla kolejnych wartości x przy pomocy przekształconego wzoru

Możemy zauważyć, że dla różnych typów danych mamy mniej więcej zbliżone wartości, a nawet w niektórych przypadkach takie same. Dzięki zlikwidowaniu wyrazu x w mianowniku, poprzez przekształcenie wzoru program może zakończyć obliczenia i uzyskać wszystkie wyniki. Wyniki też są różne w zależności od użytego typu danych i ich precyzji. Jednak dzięki przekształceniu wzoru wyniki do 30 wyrazu są poprawne i bliskie prawdziwej wartości liczby π. Dodatkowo wyraz x_{30} dla typu double oraz long double jest równy wartości π z dokładnością do 11 miejsc po przecinku.

Literatura:

[1] Wykłady nr 1 oraz nr 2 dr Rycerz z przedmiotu MOwNiT
[2] Oficjalna dokumentacja Numpy na temat typów danych