Wydział Fizyki Technicznej i Matematyki Stosowanej
Instytut Fizyki i Informatyki Stosowanej

Gdańsk, dnia 09.11.2021

Sprawozdanie nr 1

Algorytmy i Struktury Danych
Laboratorium Komputerowe

Algorytm: Błąd numeryczny w obliczeniach iteracyjnych

student: Aleksander Kruczkowski, s185390
semestr: Fizyka Techniczna, semestr III
stopień studiów: I
specjalność: informatyka stosowana
rok akademicki: 2021/2022
prowadzący: dr hab. inż. arch. Jan Kozicki, profesor uczelni

I. Wstęp teoretyczny

Opracowywano zachowanie funkcji (1.1) dla liczb zmiennoprzecinkowych o różnych dokładnościach. Kolejne wartości x_n otrzymywano według poniższego wzoru:

$$x_n = f\left(x_{n-1}, x_{n-2}\right) = 108 - \frac{815 - \frac{1500}{x_{n-2}}}{x_{n-1}} \tag{1.1}$$

Dla wartości początkowych:

$$x_0 = 4$$
$$x_1 = 4.25$$

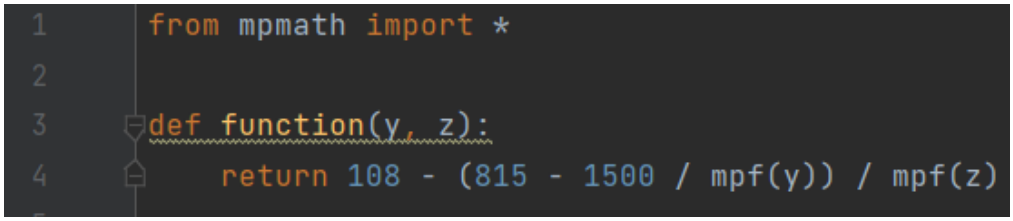
funkcja (1.1) zbiega do wartości 5.

Przedmiotem badań jest po ilu krokach iteracyjnych funkcja odchyla się od tej wartości, co zależy od dokładności zmiennych wykorzystanych do zapisu kolejnych wartości x_n.

Występujące błędy wynikają z faktu, że reprezentacja otrzymywanej liczby rzeczywistej w postaci zmiennoprzecinkowej jest jedynie pewnym przybliżeniem.

II. Opis implementacji

1. Fragment przedstawiający zapis funkcji (1.1) w Python:



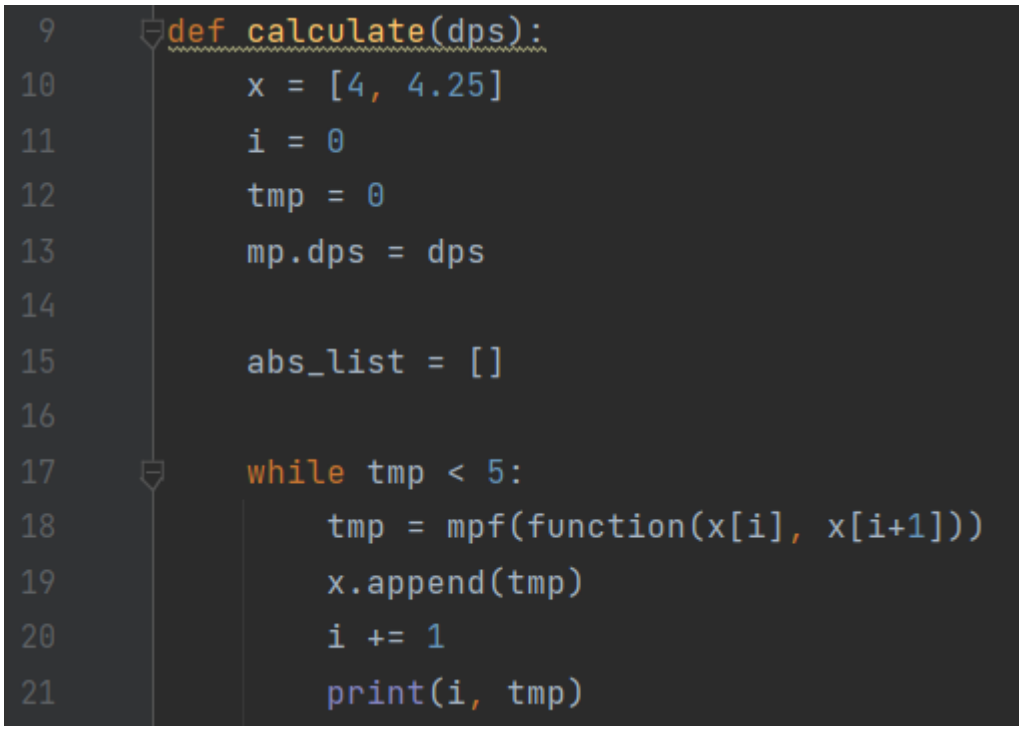
```
from mpmath import *

def function(y, z):
    return 108 - (815 - 1500 / mpf(y)) / mpf(z)
```

Wykorzystano bibliotekę **mpmath** w celu określenia do ilu miejsc dziesiętnych mają mieć otrzymywane wartości.

mpf – odpowiednik zmiennej typu float w bibliotece mpmath, wykorzystanie jej pozwala na użycie funkcji zawartych w mpmath do edycji precyzji tej zmiennej

2. W poniższym fragmencie przedstawiono deklaracje użytych zmiennych i pętlę wyznaczającą kolejne wartości x_n do do czasu gdy znajdzie się ona poza pożądanym zakresem (funkcja ma dążyć do 5). Wartości x są dodawane do tablicy w celu wykorzystania ich w kolejnych iteracjach.



```
def calculate(dps):
    x = [4, 4.25]
    i = 0
    tmp = 0
    mp.dps = dps

    abs_list = []

    while tmp < 5:
        tmp = mpf(function(x[i], x[i+1]))
        x.append(tmp)
        i += 1
        print(i, tmp)
```

mp.dps – pozwala na ustawienie ile miejsc dziesiętnych ma zawierać wartość w zmiennej typu mpf

3. Żądana ilość miejsc dziesiętnych jest przekazywana jako argument przy wywołaniu funkcji obliczajacej x_n – „calculate()”.

```
24          minimum = abs(tmp - 5)
25          abs_list.append(minimum)
26
27
28      iteracje.append(abs_list.index(min(abs_list))+1)
29
30  calculate(15)
31  calculate(16)
32  calculate(17)
```

Dla wygody odczytu wyniku, czyli do której iteracji wartość x zbiegała do 5, dodano tablicę „iteracje”. Znajdują się w niej indeksy najmniejszych wartości bezwzględnych z różnicy wyrazów x oraz liczby 5 – czyli liczba iteracji, w której x najbardziej zbliżył się do oczekiwanej wartości.

W podobny sposób zaimplementowano w/w funkcję w językach programowania:

- **C++**
- **Julia**

(przy czym w C++ pominięto implementację tablicy wartości bezwzględnych).

III. Wyniki i obserwacje

Obiektem testów było po ilu krokach otrzymywane x przestaną zbiegać do oczekiwanej wartości = 5. Wykorzystano w tym celu różne typy zmiennych.

Wyniki otrzymane w **Python**:

Typ zmiennej	**Liczba iteracji**
mpf – 15 miejsc dziesiętnych	10
mpf – 16 miejsc dziesiętnych	11
mpf – 17 miejsc dziesiętnych	12
mpf – 20 miejsc dziesiętnych	13
mpf – 50 miejsc dziesiętnych	33
mpf – 250 miejsc dziesiętnych	164
mpf – 1000 miejsc dziesiętnych	657

Wyniki otrzymane w **C++:**

Typ zmiennej	**Liczba iteracji**
float	5
double	10
long double	13
__float128	22

Wyniki otrzymane w **Julia**:

Typ zmiennej	**Liczba iteracji**
float16	2
float32	5
Double	10
BigFloat(128)	25
BigFloat(256)	51

Wartości wyrazów w kolejnych iteracjach zbliżają się do 5, następnie przez pewną ilość operacji (zależną od zastosowanej precyzji) odchodzą od niej w stosunkowo niewielkim stopniu zmniejszając się, aż nagle znacznie się zmieniają (najczęściej pojawia się liczba ujemna zbliżona do -7, a następnie liczba ok. 170 i 102. Potem wartości zbiegają do 100).

Jeśli kontynuowano by działanie algorytmu po odbiegnięciu od 5, można zauważyć, że kolejne x zbliżają się do 5, w pewnym momencie przekraczają ją i otrzymuje się kompletnie niepoprawne wartości, co widać poniżej:

```
13 4.9721527263222729
14 4.4704953665574915
15 -6.8239207793099945
16 178.26264366091512
17 102.19499813680639
18 100.10738818628326
19 100.00536351590866
20 100.00026815740838
21 100.00001340771434
22 100.00000067038202
23 100.00000003351899
24 100.00000000167595
25 100.0000000000838
26 100.00000000000419
27 100.00000000000021
28 100.00000000000001
29 100.0
30 100.0
```

Po otrzymaniu wyniku ok. 170 (zależne od precyzji) kolejne x dążą do 100. Ponadto widać, że x przestaje dążyć do 5 jeszcze przed przekroczeniem tej wartości i powoli spada, aż w pewnym momencie kompletnie odchyla się od poprzedniej tendencji.

IV. Wnioski

Liczby zmiennoprzecinkowe są jedynie przybliżoną reprezentacją otrzymywanego wyniku, więc to jaki on będzie silnie zależy od precyzji przybliżenia. W związku z tym, nie może istnieć taka precyzja, dla której wynik funkcji (1.1) zawsze zbiegałby do 5 przy powyższych danych. Wymagałoby to nieskończonej precyzji. Można jedynie zwiększać dokładność obliczeń przez stosowanie pojemniejszych typów danych pozwalających na lepsze przybliżanie.

Ciężko przewidzieć zachowanie algorytmu w momencie, gdy nie zastosuje się odpowiedniej precyzji, ponieważ przy przekroczeniu skutecznego rozmiaru zmiennych można otrzymać

V. Literatura

1. Skrypt do sprawozdania 1. nr A, autor: dr hab. inż. arch. Jan Kozicki, profesor uczelni

Załączniki

1. Plik **aisd.zip** zawierający implementacje algorytmu w C++, Python, Julia