Zadanie numeryczne 1

Jakub Heczko

1 Opis uzytego algorytmu wraz z optymalizacja

Algorytm jakiego uzylem jest algorytmem lekko skroconym od pierwotnej jego wersji, albowiem uzylem uproszczonego arytmetycznie wyrazenia, ktore wyglada nastepujaco:

$$\frac{e^{3n}-e^{3n}cos(nx^4)^2+cos(nx^4)}{e^{4n}}$$

Uzycie takiego wyrazenia sprawia, ze moge z praktycznie taka sama dokladnoscia wyliczac wartosci, ale o wiele szybciej, poniewaz licze wartosc cos() tylko raz, a nastepnie podstawiam pod odpowiednie miejsca w moim rownaniu, jesli chodzi o blad jaki jest miedzy dwiema wyrazeniami w sumach, to jest on taki sam, jesli porownamy go z prawdziwa wartoscia, wyliczona przez program wolframalpha:

-Dla starego wyrazenia:

1.4007813613268894074

-Dla nowego wyrazenia:

1.4007813613268893018

-Dla wolframalpha:

1.4007813613268893571

Ze wzgledu na dlugosc wyniku w wolframie skrocnej dlugosc do takiej samej cyfry, jakiej dostalem wynik w moich wyrazeniach. Widzimy ze nasz blad dla jednego i drugiego wyrazenia jest bardzo zblizony, a zaoszczedza nam to obliczen. W tym "nowym wyrazeniu" niestety dalej bedziemy mieli do czynienia z bledem pochodzacym z odejmowania bliskich liczb od siebie, ale przynajmniej nie bedziemy musieli, wyliczac tej samej wartosci po kilka razy w kodzie

2 Uzasadnienie wyboru n

Z wyborem n jest troche dwuznacznie, dlatego ze wybierajac n = 50 gwarantujemy maksymalnie dokladny wynik dla danego wyrazania przy uzyciu float64, ale na potrzeby zadania aby zminimalizowac ilosc obliczen, ale rowniez otrzymac blad mniejszy od 10^{-10} to najlepiej uzyc n = 25. Taki blad mowi nam,

ze musimy miec co najmniej 8 cyfr znaczacych po przecinku, aby nasz blad nie przekroczyl podanej wartosci.
Moj wybor wiec uzasadnie kilkoma rzeczami, po pierwsze danymi, ktore dostawalem; wygenerowalem okolo 20 roznych zestawow i dla kazdego, owy blad wynosil nawet mniej bo 10^{-12}. Podam przyklad dla x = 15(oczywiscie jest to wynik sumy): -Dla programu:

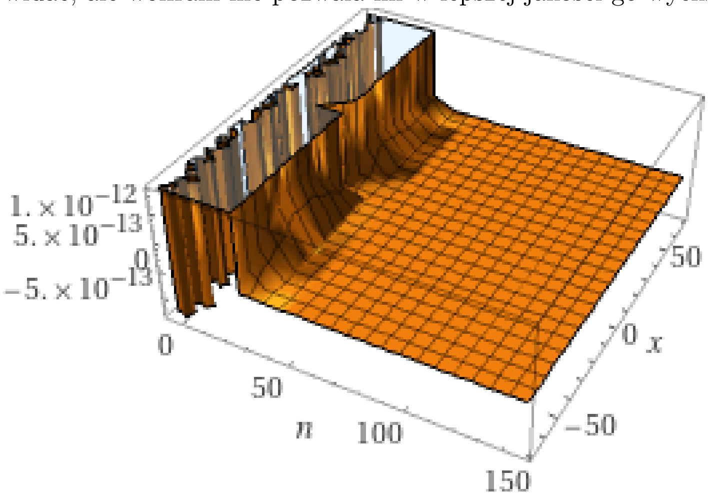
1.4251827394830521315

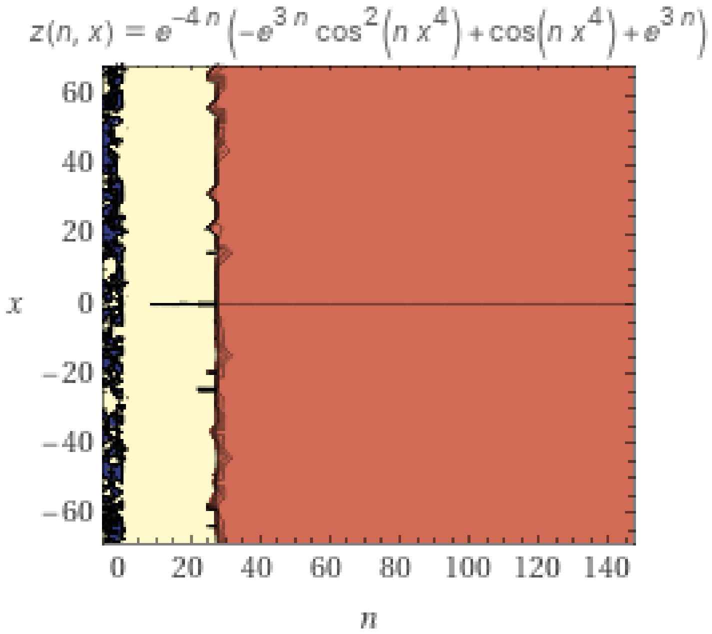
-Dla wolframalpha:

1.4251827394889998731

-Wyliczylem blad i wynosi w przyblizeniu:

$9 * 10^{-12}$
Jak widac, nie jest tak zle, kiedy bedziemy porownywac wartosci z programu, z wartosciami z wolframa. Ale mam jeszcze drugi powod do wybierania takich n jak podalem powyzej, a jest to wykres funckji jakiej sume liczymy. Bardzo z gory przepraszam za jakos zdjecia mam nadzieje, ze bedzie na nim wszystko widac, ale wolfram nie pozwala mi w lepszej jakosci go wyeksportowac:





Pierwszy wykres, ukazuje sam wykres, a drugi jedynie mape poziomu skoku wykresu. Jak widac na wykresie wybranie dowolnego x nie zmieni nam, jak bedzie wygladac skladnik sumy dla n = 50, bo bedzie on bardzo zblizony do zera, ale jak mozna rowniez zauwazyc, ze nasze n dla okolo wartosci n = 20-23, wartosc tego elementu, ktory bedziemy dodawac do sumy bedzie bliska 0, co sprawi ze nasza suma sie nie zmieni, albo w bardzo malo znaczacy sposob. Lecz na wykresie pierwszym wydac, ze lekkie skoki sa nawet po n ≈ 25, lecz dopiera na n = 50 wszystko sie stabilizuje, co tlumaczy, dlaczego, nasza suma jest najbardziej dokladna dla n = 50. Finalnie w dalszych obliczeniach stosuje n = 25, w celu zaoszczedzenia na ilosci wykonywanych obliczen, gdyz wyniki sa wystarczajaco dobre

3 Polecenia do zadania

Teraz mozemy zajac sie sprawami zwiazanymi z samymi zadaniami, zacznijmy od wyliczenia wartosci f(1)

Wartosc dla f(1)

$f(1) = 1.4007813613$
$\epsilon = 0.0000000000002433608869978343$
lub
$\epsilon = 2.433608869978343e - 13$

3.1 Ilosc obliczen

```
(1)import exp from math
(2)import numpy as np
(3)def func(x):
(4)  value1 = np.longdouble(0.0)
(5)  for n in range(0,25):
(6)    cos_val = np.cos(n*(x**4))
(7)    value1 += np.longdouble( (exp(3*n) - exp(3*n)*cos_val**2 +
    cos_val) / exp(4*n))
```

To teraz linika po linice przelicze:

(1) 50 (parse)=50

(2) 50 (parse)=50

(3) 0.5(=)=0.5

(4) 50 (parse) + 0.5(=) + 1*25(warunek) + 1*25(+=) = 100.5

(5) 0.5(=) + 30(cos) + 1(*) + 30(**) to wszystko razy 25(petla) = 1537.5

(6) 1(+=) + 3*30(exp, mamy 3 takie wywołania) + 30(cos**2) + 1*6(-/+/*/dzielenie) to wszystko razy 25 =3175

(7) 50(return)=50

(8) Nie licze print bo podaje ostateczny wynik ale +1(bo wywoluje funkcje)=1

Lacznie mamy operacji = 4964,5