Programowanie i metody numeryczne

Ćwiczenia 3.

Algorytmy. Arytmetyka zmiennopozycyjna.

Zadanie 1. Błąd w arytmetyce zmiennopozycyjnej.

W celu obliczenia numerycznego przybliżenia wartości funkcji

$$f(x,\, y) = x^2 - y^2$$

dla dowolnych $x,\, y \in \mathbb{R}$ możemy zastosować jeden z dwóch algorytmów:

$$\tilde{f}_1(x,\, y) = x \cdot x - y \cdot y \qquad \text{lub} \qquad \tilde{f}_2(x,\, y) = (x + y) \cdot (x - y).$$

Będziemy zakładać, że x i y są liczbami maszynowymi, czyli że są reprezentowane dokładnie.

a) Udowodnij, że oba algorytmy są numerycznie poprawne, jednak drugi z nich prowadzi do mniejszego błędu względnego.

b) Napisz trzy programy `fl16`, `fl32` oraz `fl64`. Każdy z nich powinien przyjmować trzy argumenty wywołania: liczby rzeczywiste y_0 i Δx oraz liczbę naturalną K. Po uruchomieniu każdy z programów ma wykreślić dwukrotnie, na jednym rysunku, wartości funkcji $\phi(x) \equiv f(x,\, y_0)$ dla K wartości argumentu x rozłożonych równomiernie w przedziale $[y_0 - \Delta x,\, y_0 + \Delta x]$, za pierwszym razem posługując się algorytmem $\tilde{f}_1$, zaś za drugim razem – algorytmem $\tilde{f}_2$. Wykorzystaj reprezentację zmiennopozycyjną liczb maszynowych określoną w standardzie IEEE 754: w programie `fl16` – połówkowej precyzji (`binary16`), w programie `fl32` – pojedynczej precyzji (`binary32`), w programie `fl64` – podwójnej precyzji (`binary64`).

c) Przeanalizuj wyniki Twoich programów dla różnych wartości argumentów wywołania. Czy lepsza jakość algorytmu $\tilde{f}_2$ daje o sobie znać?

Zadanie 2. Implementacja funkcji wykładniczej.

Jeden z algorytmów numerycznego obliczania przybliżonej wartości funkcji wykładniczej $f(x) = \mathrm{e}^x$ dla argumentu $x = x_0 \in \mathbb{R}$ opiera się na wzorze Maclaurina z resztą w postaci Lagrange'a:

$$\mathrm{e}^x = \sum_{n=0}^{N} \frac{x^n}{n!} + \mathrm{e}^{\xi} \frac{x^{N+1}}{(N+1)!}, \qquad \text{gdzie } \xi \text{ leży pomiędzy } 0 \text{ i } x. \tag{1}$$

Żądamy przy tym, aby błąd bezwzględny przybliżenia nie przekroczył zadanej wartości $\epsilon \in \mathbb{R}_+$, czyli aby spełniony był warunek

$$\left| f\left(x_0\right) - \tilde{f}\left(x_0\right) \right| \leq \epsilon, \tag{2}$$

gdzie $\tilde{f}\left(x_0\right)$ jest przybliżoną wartością $f\left(x_0\right)$ obliczoną przez algorytm.

a) Wyznacz w zależności od x_0 i ϵ najmniejszą wartość N we wzorze (1), dla której zachodzi warunek (2).

b) Napisz funkcję `num_exp`, która przyjmuje jako argumenty x_0 oraz ϵ i zwraca przybliżoną wartość $\tilde{f}(x_0)$ obliczoną na podstawie wzoru (1) oraz błąd bezwzględny tego przybliżenia, oszacowany na podstawie reszty we wzorze (1).

c) Wykorzystując funkcję `num_exp` napisz program `numexp`, obliczający przybliżone wartości $\tilde{f}(x)$ dla K wartości argumentu x rozłożonych równomiernie w przedziale $[a, b]$. Program powinien przyjmować jako argumenty wywołania liczby a, b, K i ϵ oraz wykreślać na osobnych rysunkach $\tilde{f}(x)$, błąd bezwzględny, błąd względny oraz koszt czasowy zastosowanego algorytmu w funkcji x.

   Przeanalizuj wyniki programu `numexp` dla różnych wartości argumentów wywołania. Czy zawsze błąd bezwzględny oszacowany przez funkcję `num_exp` jest mniejszy od ϵ, jak powinno być na mocy warunku (2)? Sprawdź w szczególności, jak zachowuje się Twój program dla $x = 2025$ i $\epsilon = 0,01$ oraz dla $x = 3$ i $\epsilon = 2,2 \cdot 10^{-15}$. Czy dzieje się wtedy coś niespodziewanego? Jeśli tak – dlaczego?

d) Załóżmy, że znamy reprezentację numeryczną liczby e (stała reprezentująca liczbę e jest dostępna w większości języków programowania i pakietów obliczeniowych). Niech $x = m + \xi$, gdzie $m \in \mathbb{Z}$ oraz $\xi \in [0, 1[$. Wówczas

$$\mathrm{e}^x = \mathrm{e}^{m+\xi} = \mathrm{e}^m \mathrm{e}^\xi = \mathrm{e}^m f(\xi), \tag{3}$$

   przy czym wartość e^m możemy obliczyć po prostu wykonując mnożenia (ponieważ m jest liczbą całkowitą), natomiast przybliżoną wartość $\tilde{f}(\xi)$ da się wyznaczyć, jak to dotychczas robiliśmy, na podstawie wzoru (1).

   Napisz funkcję `num_exp_xi`, która przyjmuje jako argumenty x_0 oraz ϵ i zwraca przybliżoną wartość $\tilde{f}(x_0)$ obliczoną na podstawie wzoru (3) oraz błąd bezwzględny tego przybliżenia. Funkcja `num_exp_xi` powinna wywołać funkcję `num_exp` i wykorzystać jej wyniki w celu obliczenia $\tilde{f}(\xi)$ oraz oszacowania błędu bezwzględnego.

e) Napisz program `numexpxi` działający analogicznie do programu `numexp`, korzystający jednak z funkcji `num_exp_xi` zamiast funkcji `num_exp`.

   Porównaj wyniki programów `numexp` i `numexpxi` dla różnych wartości argumentów wywołania. Czy zastosowanie kroku pośredniego w postaci wzoru (3) (program `numexpxi`) poprawiło jakość algorytmu obliczającego wartość $\tilde{f}(x_0)$ w stosunku do bezpośredniego jej obliczania na podstawie wzoru (1) (program `numexp`)?

f) Korzystając z funkcji `num_exp`, napisz program `numexpn100`, obliczający przybliżone wartości $\tilde{f}(-100)$ dla K wartości maksymalnego błędu bezwzględnego ϵ rozłożonych równomiernie w przedziale $[\epsilon_1, \epsilon_2]$. Program powinien przyjmować jako argumenty wywołania liczby ϵ_1, ϵ_2 i K oraz wykreślać na osobnych rysunkach $\tilde{f}(-100)$, błąd bezwzględny, błąd względny oraz koszt czasowy zastosowanego algorytmu w funkcji ϵ.

   Przeanalizuj wyniki programu `numexpn100` dla różnych wartości argumentów wywołania. Jak zmienia się $\tilde{f}(-100)$ przy coraz mniejszych wartościach ϵ – czy kolejne przybliżenia dążą do wartości e^{-100}? Jeśli nie – dlaczego?

Zadanie 3. Algorytmy sortowania.

Sortowanie to zadanie polegające na uporządkowaniu elementów pewnego zbioru, na którym określona jest relacja porządku. Sortować możemy więc na przykład liczby rzeczywiste, posługując się naturalnym porządkiem $\leq$, oraz słowa, wykorzystując porządek alfabetyczny.

Istnieje wiele algorytmów sortowania. Przyjrzymy się trzem z nich:

- sortowanie bąbelkowe (ang. *bubble sort*),
- sortowanie szybkie (ang. *quick sort*),

- sortowanie przez scalananie (ang. *merge sort*).

a) Wykorzystując sztuczną inteligencję lub materiały dostępne w internecie zapoznaj się z wymienionymi algorytmami sortowania.

b) Oblicz czasową złożoność obliczeniową każdego z tych algorytmów w najlepszym, średnim i najgorszym przypadku.

c) Napisz trzy funkcje `bubble_sort`, `quick_sort` i `merge_sort`, których zadaniem jest sortowanie liczb rzeczywistych za pomocą algorytmu określonego w nazwie funkcji. Każda z funkcji powinna przyjmować jako argument listę liczb rzeczywistych i zwracać listę złożoną z tych samych liczb uporządkowanych rosnąco.

d) Wykorzystując te funkcje napisz program `sort`, przyjmujący jako argument wywołania liczbę naturalną N. Program powinien N–krotnie, dla $n = 1, 2, \ldots, N$, tworzyć n–elementową listę losowych liczb rzeczywistych, a następnie sortować tę listę czterema sposobami: każdą z napisanych przez Ciebie funkcji oraz za pomocą funkcji bibliotecznej udostępnianej przez język programowania albo pakiet obliczeniowy, którego używasz, mierząc czas wykonywania każdej operacji sortowania. Wynikiem programu powinien być rysunek przedstawiający wykresy czasu wykonywania Twoich funkcji sortujących oraz funkcji bibliotecznej w zależności od n.

e) Przeanalizuj wyniki programu `sort` dla różnych wartości argumentu wywołania. Która z funkcji jest najszybsza? Czy zależność czasu sortowania od rozmiaru sortowanej listy odzwierciedla złożoności obliczeniowe poszczególnych algorytmów?

Opracowanie: Bartłomiej Zglinicki.