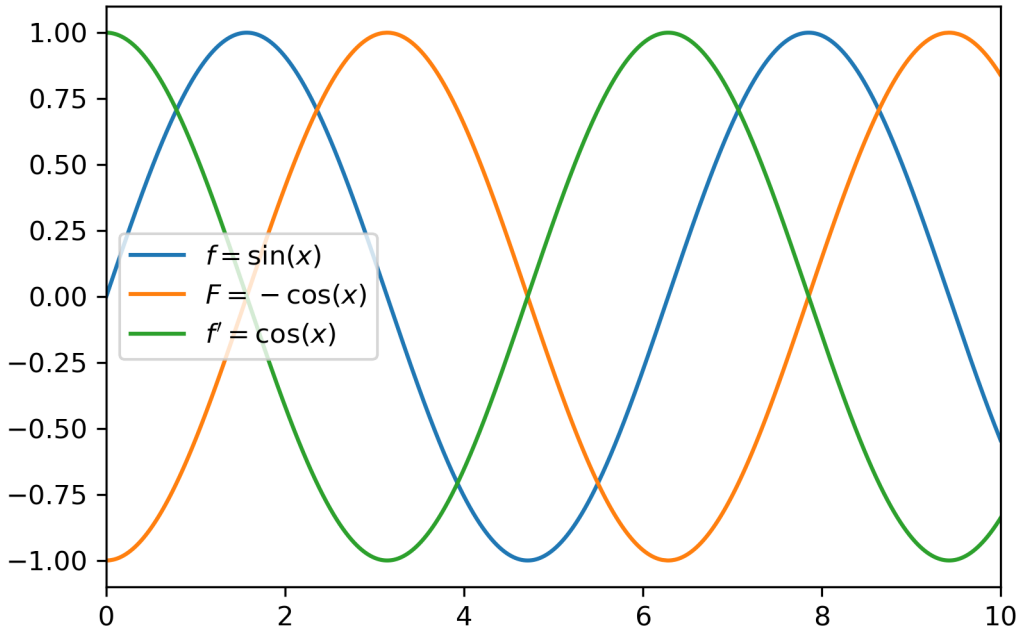
Jegyzőkönyv

6. Függvény Osztály

Korszerű Számítógépes Módszerek



Jegyzőkönyvet készítette: Werderits Richárd Tamás
AQB9DN

1. Feladat leírása

Az osztály valamilyen általunk választott reprezentációban tárolja egy folytonos függvény valamilyen diszkretizációját, a cél, hogy lehessen kiértékelni bárhol (interpoláció), határozott es határozatlan integrált számolni, deriválni, gyököt, minimumot, maximumot keresni.

2. Implementáció

A függvény osztályt a FUN_class.hpp header fájlban hoztam létre. Itt található az összes member function, valamint egyéb segédfüggvények. A main.cpp-ben található kód csupán az osztály funkcionalitásainak demonstrálására szolgál.

2.1. Diszkretizált függvény tárolása

Törekedtem arra, hogy a lehető legkevesebb adattal kellő képpen eltárolhassam a diszkretizált függvényt. Úgy döntöttem végül, hogy egy tetszőleges elemszámú std::vector típusú tárolóban fogom eltárolni a függvény y értékeit, amik egyenletes h mintavételezéssel vett x értékekhez tarzotnak. Az x értékeket azonban nem tárolom el mindet, csupán az elsőt (x_0) és az utolsót (x_1). Ez azonban azzal jár, hogy az osztály nem alkalmas egyenletlen mintavételezéssel vett y értékek tárolására. Előnye viszont az, hogy nem kell a h mintavételezési köz értékét minden lépésben újra és újra kiszámolni deriválás és integrálás esetén.

2.2. Interpoláció

A függvény tetszőleges helyen vett kiértékeléséhez lineáris interpolációt alkalmaztam. Erre a value(x) és interpolate(x) member function-ök alkalmazhatóak.

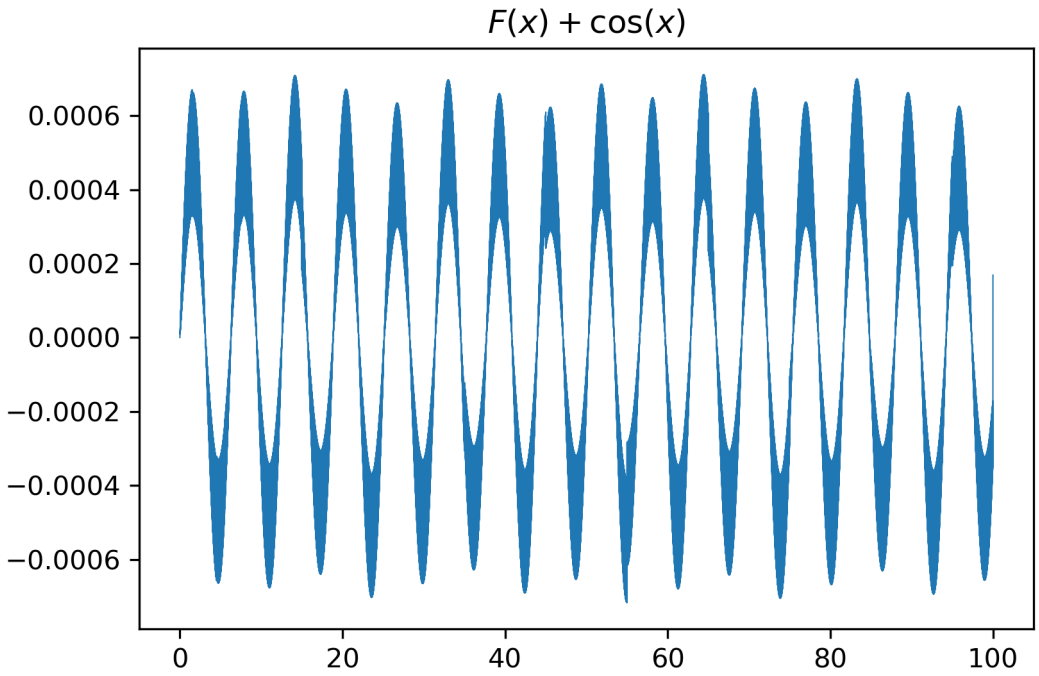
$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0}$$

2.3. Határozott és határozatlan integrál

Elsőként a határozatlan integrálhoz készítettem eljárást (integral(C)), ez a Simpson 1/3 szabály alapján működik. Ez az eljárás elkészíti a függvény határozatlan integrálját egy általunk meghatározott C konstanssal eltolva.

A határozott integrált kiszámító eljárás (integrate(a, b)) is ezt az eljárást használja: elkészíti a határozatlan integrált, majd kiértékeli (interpoláció segítségével) a és b között és kiszámolja a különbségüket.

Létrehoztam az integral(C) eljárás segítségével a $\sin(x)$ függvény primitív függvényét, és összehasonlítottam az analitikus eredménnyel (-cos(x)). A két adatsor különbségét ábrázoltam a Python3 Matplotlib könyvtárának segítségével.



1. ábra. Numerikus és analitikus integrál eltérése

Ugyan az integrálás nem hibamentes, láthatóan a hibák nem felhalmozódnak, hanem oszcillálnak avalódi eredmény körül.

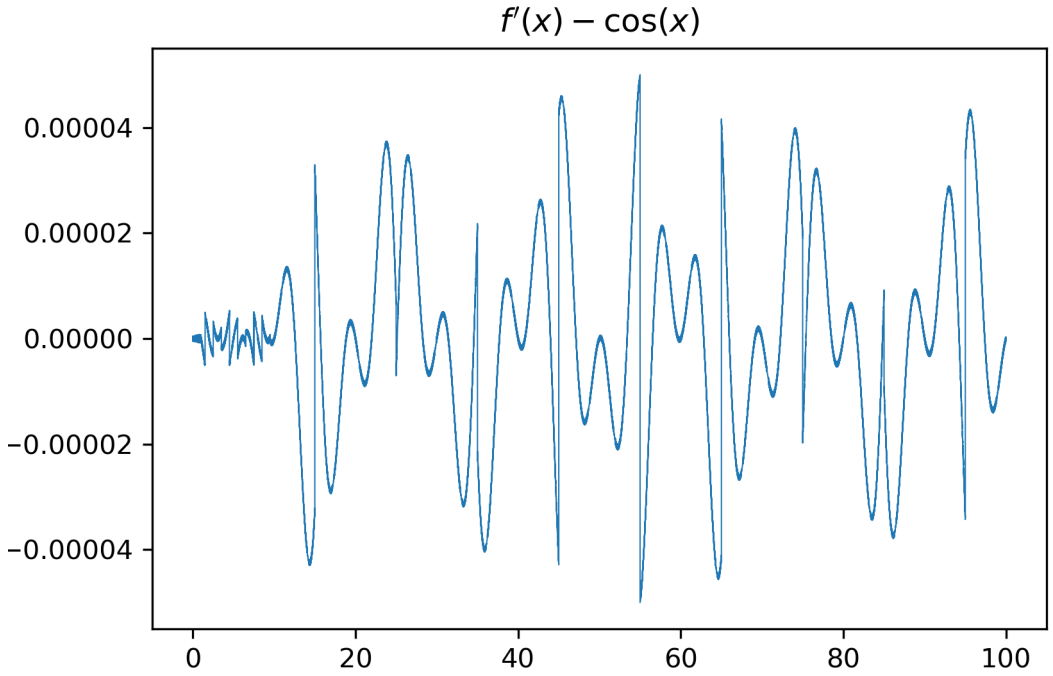
2.4. Deriválás

A derivált függvényt elkészítő eljárás (diff()) a következő közelítést alkalmazza:

$$f'(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x+2h)}{12h}$$

Mivel az adott x helyen a deriváltat az előző kettő és következő kettő y értékből számolja ki, az adathalmaz első két és utolsó két elemére nem tudunk ezzel az eljárásssal deriváltat számolni, az eljárás egy minimális adatvesztéssel jár.

Az integrálhoz hasonlóan itt is összehasonlítottam az eljárás által lértrehozott függvényt az analitikus eredménnyel ($\cos(x)$).



2. ábra. Numerikus és analitikus derivált eltérése

Amint láthatjuk, a hibák itt sem halmozódnak, hanem oszcillálnak. Láthatóan $x = 10$ alatt lényegesen kisebb a hiba, mint felette, $x = 1$ alatt pedig mégkisebb. Úgy gondolom, ez a floating point tulajdonságai miatt történhet.

2.5. Gyökök keresése

Gyökök keresésére két eljárást írtam.

Az első eljárás (where(y)) a diszkrét y értékekhez tartozó x értékekkel tér vissza. Minden diszkrét y értékhez rendel egy tartományt, ami pont nincs átfedésben a szomszédos y értékhez rendelt tartománnyal. Ha a keresett y érték egy ilyen tartományon belül van, az eljárás feljegyzi a tartomány diszkrét y értékéhez tartozó x értéket.

A második eljárás (where_ri(y)) két szomszédos diszkrét y értéket vizsgál. Ha a keresett y érték a szomszédos értékek között van, akkor az eljárás kiszámítja, milyen x értékhez tartozik a keresett y érték a lineáris interpoláció inverz műveletével.

$$x = x_0 + (y - y_0)\frac{x_1 - x_0}{y_1 - y_0}$$

Mindkét eljárás problémákba tud futni, ha a keresett y érték lokális szélsőértéke a függvénynek, ugyanis a diszkretizáció miatt maga a szélsőérték nem garantált, hogy része az adatsornak.

2.6. Szélsőértékek keresése

Külön eljárásokat írtam lokális és globális szélsőértékek keresésére.

A lokális szélsőrtékeket kereső eljárások (local_mins(), local_maxs()) elkészíték a függvény deriváltját, majd megkeresem ennek a zérushelyeit. Ezután leellenőrzöm az előző és a következő elemmel való összehasonlítás segítségével, hogy melyik értékek minimumok vagy maximumok.

A globális szélsőértékeket kereső eljárások (mins(d), maxs(d)) ugyanígy járnak el, azonban csak azokat a szélsőértékeket tartják meg, amik az y adatsor minimumától vagy maximumától nem térnek el egy bizonyos küszöbértéknél jobban. Ezt a küszöbértéket a következő képlet adja meg: $|y_min - y_max|/d$, ahol d alapértelmezett értéke 100.

Mivel mindkét fajta eljárás deriváltra alapszik, ezért nem találják meg a minimum- és maximum helyeket, ha azok az y tömb első két vagy utolsó két elemén találhatóak.

2.7. Kiíró eljárások

A matematikai műveleteket végző eljárások mellett írtam még két eljárást, amik kiírják az objektumokban tárolt diszkretizált függvényeket CSV formátumban. Egyik eljárás a terminálba írja ki az első n adatpontot. Ha $n = 0$, akkor mindent kiír. A másik eljárás egy általunk megadott fájlba írja ki az adatsort.

3. Diszkusszió

Mindent egybevetve úgy vélem, idő és energia befektetésével lehetne még mit hozzáadni a FUN osztály funkcionalitásaihoz (például spline interpoláció), illetve lehetne még szépíteni a már implementált member functionökön (például olyan deriválás, ami nem jár adatveszteséggel). Esetlegesen lehetne gondolkozni más féle tárolási módokon is (például x_0 és h tárolása, vagy egy olyan std::vector tárolása, ami az x, y számpárokat tárolja.

Ennek ellenére elégedett vagyok az implementáció végső állapotával, figyelembe véve, hogy nem hagytam rá túl sok időt magamnak, és hogy ez az első ilyen jellegű C++ projektem.