IZP - Projekt č. 2 - Iterační výpočty

Než začnete programovat, přečtěte si pozorně celé zadání.
Poslední změna: 23. září 2010

Motivace

Cílem této úlohy je naučit se nebo si zopakovat

- implementaci různých typů iteračních algoritmů,
- tvorbu opatření pro překonání nedostatků počítače (heuristiku),
- vytváření a používání obecných podprogramů, které zjednodušují řešení a jsou použitelné v jiných programech,
- zpracování textově zapsané posloupnosti hodnot na standardním vstupu,
- správné ošetřování chybových stavů programu,
- vytváření přehledného zdrojového kódu.

Zadání úlohy

Napište program, který pomocí iteračních algoritmů zpracuje libovolně dlouhou posloupnost číselných hodnot typu double, které budou v textové podobě zapsány na standardní vstup a budou odděleny libovolně dlouhou posloupností bílých znaků. Výstupem programu bude číselná posloupnost výsledků. Používejte konstanty (makra) NAN a INFINITY pro ošetření výjimečných případů tak, aby to dávalo smysl.

Část 1. Aproximace funkcí

Implementujte pomocí základních matematických operací (+, -, *, /) aproximace těchto matematických operací se zadanou přesností:

- hyperbolický tangens $y = \tanh x$
- obecný logaritmus: $y = \log_a x, \mathrm{a} \in \mathbb{R}^+ \setminus \{1\}$

Vytvořené funkce aplikujte na každou hodnotu vstupní posloupnosti a výsledky zapište na standardní výstup. Výstupní posloupnost hodnot bude v tomto případě tedy stejně dlouhá, jako vstupní posloupnost. Počítejte v oboru reálných čísel (není potřeba pracovat s komplexními čísly). Určete maximální možné definiční obory těchto funkcí s ohledem na použitý datový typ.

Přesnost aproximace bude zadávána jako parametr příkazové řádky a bude to celé číslo udávající počet platných cifer výsledku.

Pro vlastní výpočet je zakázáno používat standardní matematické funkce, které vyčíslují hodnoty zadaných funkcí (sin, cos, tan, exp, log, pow, ... a jejich deriváty). Smyslem této části úlohy jsou iterační výpočty a jejich odvození z rekurentních vztahů. Je povoleno používat matematickou knihovnu `<math.h>` kromě výše zmíněných funkcí (můžete je použít například pro testování). Pokud si nejste jistí, jak by se měly funkce chovat ve významných bodech svých definičních oborů i mimo ně, můžete se inspirovat chováním standardních funkcí z knihovny.

Ve *zdůvodněných* případech lze tyto knihovní funkce použít pro pomocné výpočty. V žádném případě však s nimi nelze nahrazovat iterační algoritmy.

Část 2. Statistické funkce

Implementujte algoritmy, které ze vstupní posloupnosti popsané výše spočítají tyto údaje:

- vážený aritmetický průměr
- vážený kvadratický průměr

Vstupní posloupnost nyní interpretujte jako dvojice hodnot takto:

```
x1 h1 x2 h2 x3 h3 ...
```

První hodnota ve dvojici (x) je prvkem posloupnosti, z níž se počítá daný průměr. Druhá hodnota ve dvojici (h) je nezáporná váha tohoto prvku. Výsledkem programu bude v těchto případech posloupnost průběžných výsledků poloviční délky jako vstupní posloupnost (pro jeden výsledek jsou nutné dva vstupy). Skončí-li vstupní posloupnost předčasně, bude posledním výsledkem hodnota NAN.

Pokud pro výpočet bude potřeba složitější matematické funkce (odmocnina, atd.), můžete v této části úlohy použít funkce ze standardní matematické knihovny jazyka C.

Část 3. Dokumentace

Své řešení popište v dokumentaci. U výpočtů matematických funkcí se zamyslete nad jednotlivými vztahy pro výpočet uvedených funkcí a popište principy jejich použití s ohledem na co nejlepší konvergenci algoritmů. Rovněž diskutujte, jak dosáhnout u aproximačních algoritmů požadované přesnosti. U statistických funkcí rozeberte způsob, jak dosáhnout výpočtu s lineární časovou složitostí aniž by bylo nutné použít pole. V dokumentaci rovněž popište a zdůvodněte svůj návrh řešení problémů s nekorektními vstupy.

Poznámky

Program musí fungovat pro libovolně dlouhou vstupní posloupnost předem neznámé délky. Cyklus načítání hodnot ze standardního vstupu i tisk výsledku se musí dít mimo tyto výpočetní funkce. V ideálním případě půjde o jediný cyklus (nebo maximálně dva cykly) ve funkci main. Funkce realizující dané výpočty musí být bez úprav použitelné např. v programu, kde se prvky posloupnosti načítají ze souboru nebo se třeba postupně čtou z nějakého portu a s výsledky se dále pracuje. Matematické funkce by mělo jít bez problému použít ve výrazech podobně jako tomu je u matematických funkcí ve standardní knihovně.

Vše potřebné pro vyřešení všech částí této úlohy je vysvětleno v učebních textech a bude probráno na přednáškách a demonstračních cvičeních. Pokud budete projekt řešit dříve, než se látka probere na přednáškách, doporučuji přijít na konzultaci.

Rozhraní programu

Program přečte z příkazového řádku přepínač, který specifikuje jednu z operací, a u některých operací požadovanou přesnost.

Funkce	Přepínač	Popis	Poznámka
Nápověda	`proj2 -h`	Vytiskne nápovědu k použití programu.	
$y = \tanh x$	`proj2 --tanh sigdig`	sigdig - přesnost zadaná jako počet platných cifer (significant digits)	20% funkčnosti programu
$y = \log_a x, \mathrm{a} \in \mathbb{R}^+ \setminus \{1\}$	`proj2 --logax sigdig a`	sigdig - přesnost zadaná jako počet platných cifer (significant digits) a - základ logaritmu	20% funkčnosti programu
vážený aritmetický průměr	`proj2 --wam`	Tiskne na výstup průběžné hodnoty váženého aritmetického průměru již načtené posloupnosti hodnot.	20% funkčnosti programu
vážený kvadratický průměr	`proj2 --wqm`	Tiskne na výstup průběžné hodnoty váženého kvadratického průměru již načtené posloupnosti hodnot.	20% funkčnosti programu

Číselnou posloupnost načítejte ze standardního vstupu. Jednotlivé hodnoty budou odděleny "bílými znaky", tj. mezerami, tabulátory nebo konci řádků. Formátem vstupních hodnot bude reálné číslo zapsané podle zvyklostí jazyka C (můžete použít funkci scanf()). Posloupnost končí s koncem souboru (EOF) a může být potenciálně nekonečná, takže program by měl výsledky tisknout průběžně. Navrhněte a implementujte vhodné řešení problému nekorektních vstupů s ohledem na použitelnost aplikace. Řešení zdůvodněte v dokumentaci.

Aproximované funkce aplikujte zvlášť na každý prvek posloupnosti. Výsledkem aplikace těchto funkcí bude stejně dlouhá (první část úlohy) nebo poloviční (druhá část úlohy) výstupní posloupnost výsledků. Jednotlivé prvky výstupní posloupnosti vypisujte vždy na samostatný řádek standardního výstupu.

Výsledkem aplikace statistických funkcí je posloupnost průběžných výsledků (pozor, někdy není výsledek definován).

Důležité! Aby bylo možné výsledky testovat, vypisujte každý výsledek tímto způsobem: `printf("%.10e\n", vysledek);` (tedy 10 desetinných míst při zápisu s exponentem).

Příklady vstupu a výstupu

Příkazová řádka pro výpočet logaritmu o základu 2.71 s přesností na tři platné cifry:

```
$ ./proj2 --logax 3 2.71
```

Standardní vstup (psaný ručně nebo přesměrovaný ze souboru):

```
1e24
100000    1.97
```

Odpovídající výstup:

```
5.5262000000e+01
1.1512776601e+01
0.6780668847e+00
```

Poznámky:

- Toto je jenom ukázka. Výsledné hodnoty se nemusí shodovat se správným řešením.
- Pokud budete zadávat vstup ručně, je velmi pravděpodobné, že program bude průběžně vracet výsledky po každém ukončeném řádku. Takové chování je v pořádku.
- Přesnost na N cifer neznamená, že od cifry od N+1 pozice dále budou nuly, ale že na nich již nezáleží.

Maximální počet získaných bodů

Za tuto úlohu lze získat **až 10 bodů**. Součástí řešení je dokumentace, která je součástí předmětu IUS, a která je bodovaná zvlášť. Za dokumentaci lze získat **až 10 bodů**, které se počítají do IUS.

Prémie

Za tuto úlohu lze získat jeden prémiový bod, pokud implementujete nějaké další zajímavé, netriviální operace nebo program rozšíříte jiným netriviálním způsobem nebo pokud bude vaše implementace zvlášť povedená. Zde nabízím náměty pro rozšíření:

- výpočet určitého integrálu implementovaných funkcí obdélníkovou metodou (funkci lze v jazyce C předat jako argument jiné funkci)
- výpočet určitého integrálu lichoběžníkovou metodou

Co se zejména hodnotí

- Správné odvození definičního oboru.
- Správná implementace v maximálním možném rozsahu vstupních hodnot. (Nezapomeňte, že v jazyce C je legální pracovat i s nekonečnými a nečíselnými hodnotami.)
- Přesnost výpočtů musí odpovídat zadané přesnosti.
- Odvození iteračních algoritmů a jejich efektivita s ohledem na zvolenou přesnost (program nesmí nikdy skončit v nekonečném cyklu a musí vždy skončit v rozumném čase).
- Efektivita - hodnotí se i to, zda algoritmy neprovádějí zbytečné výpočty.
- Kvalita návrhu datových typů a rozdělení programu na podproblémy. Cyklus načítání hodnot ze vstupu a zápis výsledků na výstup se nesmí vyskytovat uvnitř funkcí, které provádí vlastní výpočet!
- V dokumentaci se hodnotí kvalita vyjadřování, věcná správnost, přehlednost dokumentu a schopnost jeho rozčlenění do kapitol. Pravopisné chyby budou mít na hodnocení negativní vliv!

Pomůcka

Konec souboru (EOF) se v unixovém terminálu "zapíše" kombinací kláves Ctrl+D, v DOSu (příkazovém řádku ve Windows) pomocí Ctrl+Z. Pro testování (i reálné použití) je obvyklejší mít testovací data v textovém souboru a použít operátor přesměrování. Stejně tak je rozumné si ukládat výsledek do jiného textového souboru.

```
$ ./proj2 --tanh 3 < input.txt > output.txt
```

Pro vlastní řešení můžete potřebovat matematickou knihovnu, můžou se vám z ní hodit některé funkce (fabs(), trunc(), modf(), ...). Její použití vyžaduje vložení hlavičkového souboru `<math.h>`. Dále je nutno ji k programu připojit pomocí přepínače `-lm`.

```
$ gcc -std=c99 -Wall -pedantic -g -lm proj2.c -o proj2
```

Pokud potřebujete vracet hodnotu NaN (Not a Number - není číslo) nebo nekonečno, v matematické knihovně najdete makra NAN a INFINITY. Pokud nechcete matematickou knihovnu používat stačí vrátit hodnotu `0.0/0.0` (NaN, pozor některé nePC architektury ji nepodporují) nebo `1.0/0.0` (nekonečno). Některé implementace standardní knihovny tato makra nemusí obsahovat, potom můžete použít stejný postup a definovat si je sami. Další užitečné konstanty jsou naneštěstí v nestandardním rozšíření knihovny, takže raději do svého zdrojového souboru vložte toto:

```
const double IZP_E = 2.7182818284590452354;        // e
const double IZP_PI = 3.14159265358979323846;      // pi
const double IZP_2PI = 6.28318530717958647692;     // 2*pi
const double IZP_PI_2 = 1.57079632679489661923;    // pi/2
const double IZP_PI_4 = 0.78539816339744830962;    // pi/4
```

Výchozí rekurentní vztahy je vhodné upravit tak, aby byl výpočet co nejefektivnější (některé funkce nejrychleji konvergují jen v určitém rozsahu vstupních hodnot - viz literatura).

Teorie

Pozor! Pokud někdo následující text použije v dokumentaci, bude asistent nucen jej v duchu označit nelichotivým příměrem a ohodnotit sníženým počtem bodů. ;-)

Uvedené matematické funkce lze aproximovat pomocí rekurentních vztahů, které vycházejí z Newtonovy iterační metody, Taylorova rozvoje nebo součtu jiné nekonečné řady. U všech zde uvedených vztahů jde vlastně o vyčíslování limity, jejíž hodnota představuje hledané řešení. Obecně lze jeden krok iteračního výpočtu zapsat takto:

$Y_{i+1}=F(Y_i)$, pro $i >= 0$

kde **i** je index kroku, $\mathbf{Y_i}$ je hodnota v kroku i a **F()** je funkce pro výpočet dalšího kroku na základě výsledku předchozího kroku.

Pro úspěšné rozběhnutí algoritmu je nutné zvolit hodnotu $\mathbf{Y_0}$, tedy počáteční aproximaci. Tato hodnota vyplývá z odvozeného rekurentního vzorce.

Protože limita je nekonečná, je nutné zvolit nějaké kritérium pro ukončení algoritmu. Na počítači jsme hardwarem omezeni počtem zobrazitelných číslic "reálných" čísel, takže se spokojíme s výpočtem, který se s nějakou přesností přiblíží k přesné hodnotě. Dalším důvodem pro výpočet se zadanou přesností je rychlost výpočtu - čím větší přesnost požadujeme, tím déle výpočet trvá (u některých vztahů se výpočet každého následujícího kroku zpomaluje!).

V našem případě použijeme přesnost zadanou číslem EPSILON. Algoritmus výpočtu ukončíme, když absolutní hodnota rozdílu výsledků dvou po sobě jdoucích kroků bude menší nebo rovna EPSILON:

$$|Y_{i+1} - Y_i| <= EPSILON$$

Protože v této úloze je přesnost zadána, jako počet platných cifer, je nutné tento údaj vhodně přepočítat na hodnotu EPSILON. Jak jste to udělali popíšete v dokumentaci.

Bližší informace k problematice iteračních výpočtů lze nalézt např. ve skriptech **Rábová, Honzík, Češka, Hruška: Počítače a programování** nebo v **Hans-Jochen Bartsch: Matematické vzorce** nebo na internetu. Zde ovšem můžete narazit na kvalitní, ale i na docela zavádějící informace! Nevěřte všemu, co naleznete. Zvláště pak v dokumentaci necitujte Wikipedii a jiné internetové zdroje. Ověřte si nalezené informace ve spolehlivých zdrojích.

Tip pro funkce řešené Taylorovou řadou

Zadané funkce lze definovat i pomocí jiných funkcí, které lze Taylorovou řadou vyčíslit snadněji a přesněji.

Tip pro statistické funkce

Velmi si zjednodušíte práci, když si budete průběžné výsledky ukládat do vhodné datové struktury (s konstantní prostorovou složitostí) a když se co nejdříve naučíte pracovat s podprogramy. Uvnitř podprogramů pro výpočet průběžných hodnot by se neměl vůbec vyskytovat cyklus. U obou statistických funkcí by mělo jít o řešení s lineární časovou a konstantní prostorovou složitostí. Pojmy časová a prostorová složitost budou vysvětleny na jedné z posledních přednášek (podívejte se do slajdů a do studijní opory).

Autor: David Martinek. Poslední modifikace: 23. září 2010. Pokud v tomto dokumentu narazíte na chybu, dejte mi prosím vědět.