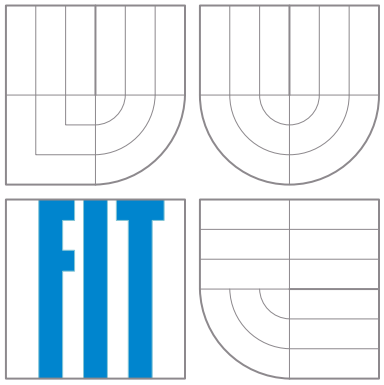


Dokumentace k projektu pro předmět IZP

Iterační výpočty

projekt č. 2

25. listopadu 2010

Autor: Vlastimil Slinták, xslint01@stud.feec.vutbr.cz
Fakulta elektrotechniky a komunikačních technologií
Vysoké Učení Technické v Brně

Obsah

1 Úvod

Tento dokument popisuje implementaci matematických funkcí `tanh` – hyperbolický tangens, `log` – obecný logaritmus a dvou statistický funkcí – aritmetický a kvadratický vážený průměr.

Implementace prvních dvou funkcí nevyužívá standardní knihovnu `math.h`, ani složitější datové typy. Statistické funkce již využívají matematickou knihovnu i jeden vlastní datový typ. V dalších sekcích tohoto dokumentu se podrobně podíváme na implementace všech čtyř funkcí.

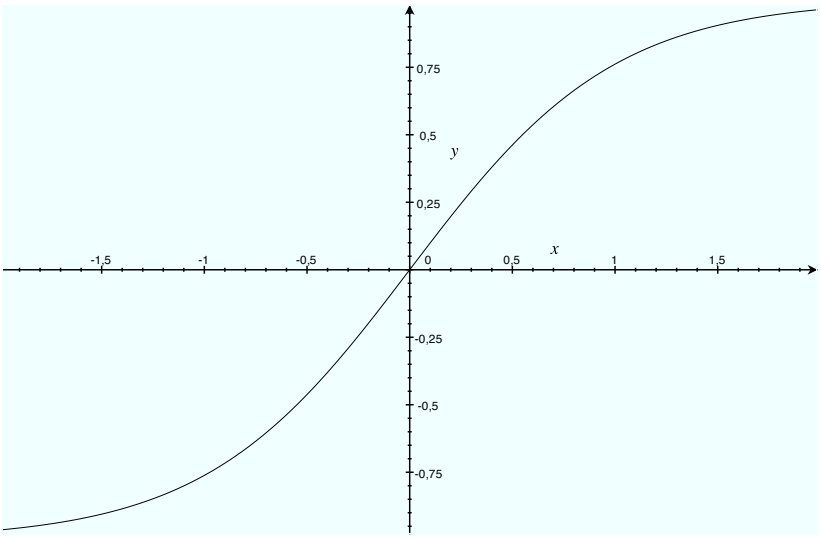
2 Analýza problému a princip jeho řešení

Pro úspěšnou implementaci zadání je nutná znalost chování zde uváděných matematických funkcí. V této části se proto podíváme na jednotlivé funkce a popíšeme si jejich chování.

2.1 Hyperbolický tangens

Graf funkce `tanh` je na obrázku číslo 1. Je zřejmé, že argumentem funkce mohou být všechna reálná čísla. Grafem je lichá funkce, která se pro rostoucí argument `x` asymptoticky blíží 1, resp. -1. O této funkci tedy platí následující tvrzení:

$$f(-x) = -f(x);\ \lim_{x\to\infty} \tanh x = 1;\ \lim_{x\to-\infty} \tanh x = -1; x \in \mathbb{R};$$



Obrázek 1: Graf funkce `tanh(x)`

Výpočet lze realizovat Taylorovou řadou, která ale pro `tanh` obsahuje Bernoulliho čísla. Vhodnější implementace se jeví přes `sinh` a `cosh`:

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{\sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!}}{\sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}}$$

Pro tento výpočet je tedy potřeba implementovat funkce `sinh` a `cosh`, případně provést výpočet obou funkcí během jednoho vyklu. Já jsem nakonec zvolil jiný způsob výpočtu – přes exponenciální funkci:

$$\tanh x = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Exponenciální funkce lze spočítat pomocí Taylorovi řady následovně:

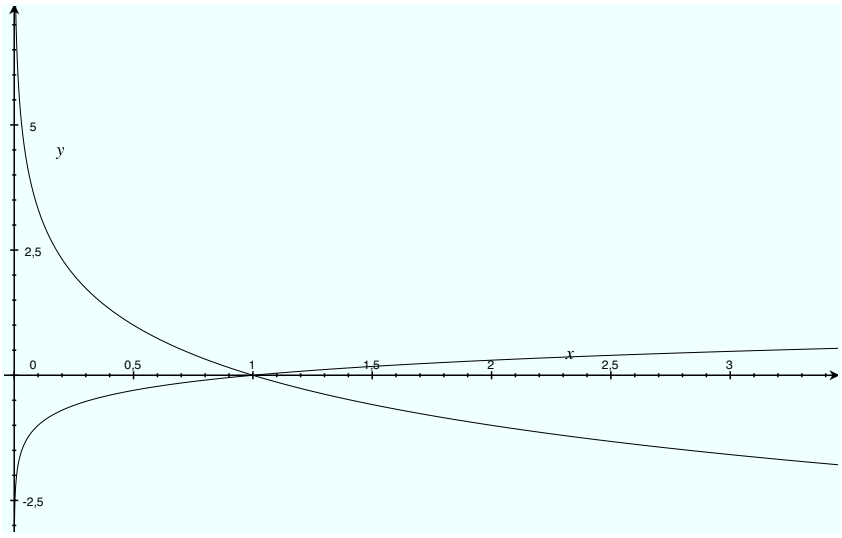
$$e^x = \sum_{x=0}^{\infty} \frac{x^n}{n!}$$

2.2 Obecný logaritmus

Obecný logaritmus je definován pouze pro kladná reálná čísla kromě nuly a to jak pro argument, tak pro základ. Pro $\log_a(x)$ platí:

$$a, x \in \mathbb{R}^+; f(x) \in \mathbb{R}$$

Graf této funkce je na obrázku 2. Klesající křivka je logaritmus o základu 0.5, zatímco rostoucí je dekadický logaritmus, tedy o základu 10.



Obrázek 2: Grafy funkcí $\log_{10}(x)$ a $\log_{0.5}(x)$.

Dále platí následující výroky:

$$\log_a x = \frac{\log_b x}{\log_b a}$$

$$\log_a x = -\log_{\frac{1}{a}} x, \quad a \in (0, 1)$$

$$\log_a x = -\log_a \frac{1}{x}, \quad x \in (0, 1)$$

Využití těchto dvou rovnic bude vysvětleno v kapitole 4.3 věnující se implementaci obecného logaritmu.

2.3 Statistické funkce

V této kapitole jsou uvedeny vzorce pro vážený aritmetický a kvadratocký průměr. Popis implementace je pak v kapitole 4.4.

V obou níže uvedených vzorcích platí — x_i je i-tý prvek průměru a w_i je jeho váha.

Vážený aritmetický průměr

$$\bar{x} = \frac{\sum_{i=1}^{k} x_i w_i}{\sum_{i=1}^{k} w_i}$$

Vážený kvadratický průměr

$$\bar{x} = \sqrt{\frac{\sum_{i=1}^{k} x_i^2 w_i}{\sum_{i=1}^{k} w_i}}$$

3 Návrh řešení problému

3.1 Analýza vstupních dat

Na vstupu od uživatele očekáváme pouze desetinná čísla ve tvaru například 3.4, $3e-4$, případně speciální konstantu inf, která představuje nekonečno. Jakýkoliv jiný vstup bude programem vyhodnocen jako chybný a uživateli to bude oznámeno chybovým hlášením.

3.2 Specifikace testů

Test 1: Chybný vstup ⟶ Detekce chyby.

```
./proj2 --tanh 5 <<< "bla"
./proj2 --tanh 5 <<< "nan"
./proj2 --tanh 5 <<< "0,4"

./proj2 --logax 5 10 <<< "bla"
./proj2 --logax 5 10 <<< "nan"
./proj2 --logax 5 10 <<< "0,1"

./proj2 --wam <<< "bla"
./proj2 --wam <<< "4.2"
./proj2 --wqm <<< "4.2 bla"
./proj2 --wqm <<< "4,2 nan"
```

Test 3: Vstup mimo povolený rozsah hodnot ⟶ Detekce chyby.

```
./proj2 --logax 10 10 <<< "-3.0"
./proj2 --logax 10 -1 <<< "3.0"
./proj2 --logax 10 0 <<< "3.0"
./proj2 --logax 10 2 <<< "0"
```

Test 4: Správný vstup ⟶ Výsledek

vstup	očekávaný výstup
`./proj2 --tanh 5 <<< "0.5"`	4.6211715726e-01
`./proj2 --tanh 5 <<< "-100"`	-1.0000000000e+00
`./proj2 --logax 5 2 <<< "16"`	4.0000000000e+00
`./proj2 --wam <<< \`	1.5000000000e+00
"1.5 1.0 2.5 2.0 3.5 3.0"	2.1666666667e+00
	2.8333333333e+00

4 Popis řešení

Vlastní implementace jednotlivých funkcí zohledňuje jejich chování, které bylo popsáno v kapitole 2.

4.1 Ovládání programu

Program je ovládán z terminálu a jeho výstup je čistě textový. Veškerá uživatelská data jsou načítána ze standardního vstupu a jsou zapisována na standardní výstup. Chybová hlášení se tisknou na standardní chybový výstup. V případě výskytu chyby program končí a vrací `1`, jinak `0`. Program tak může uživatel spouštět interaktivně i neinteraktivně (například ve skriptech).

Argumenty programu jsou:

Argument	Popis
−h	Vypíše nápovědu a skončí.
−−tanh sigdig N	Vypočítá tanh s přesností na `sigdig` desetinných míst.
−−logax sigdig a	Vypočítá logaritmus o základu 'a' s přesností na `sigdig`.
−−wam	Průběžně počítá vážený aritmetický průmer.
−−wqm	Průběžně počítá vážený kvadratocký průměr.

4.2 Implementace `tanh`

Implementace využívá poznatků z kapitoly 2. Využívám faktu, že je funkce lichá, a proto není potřeba nijak speciálně řešit záporné argumenty funkce.

Jak již bylo popsáno výše, moje implementace nevyužívá funkcí `sinh` ani `cosh`, ale počítá hyperbolický tangens přes exponenciální funkci. Tato funkce je rychle rostoucí, již pro malé argumenty nabývá velkých hodnot. Obecně by se dala řešit pomocí Taylorovy řady, která ale pro velká číslo konverguje příliš pomalu. Nejrychlejší konvergence je pro čísla menší jak jedna. Proto je samotný výpočet exponeniální funkce rozdělen do dvou částí. Na část celočíselnou a zlomkovou.

Argument exponenciální funkce je rozdělen na dvě části. Mějme například číslo 3.5, které rozdělíme na 3 a 0.5. Funkce `approx_exp()`, která bere dva argumenty – číslo `x` a uživatelem zadanou přesnost výpočtu ε – postupně volá dvě funkce `approx_by_int_exp()` a `approx_by_dbl_exp()`. Prvně jmenovaná funkce počítá celočíselnou část argumentu a druhá desetinnou. Oba výsledky se pak vynásobí a vrátí uživateli. Výpočet využívá rovnosti:

$$e^{a+b} = e^a \cdot e^b$$

Funkce `approx_by_dbl_exp` implementuje Taylorovu řadu. Jelikož ji voláme pouze pro čísla menší než jedna a větší než nula, podává uspokojivé výsledky při malém počtu iterací. Druhá funkce, `approx_by_int_exp` pracuje s celými čísly. Pro příklad čísla 3 funkce provede dvě iterace. Číslo 3 se dá ve dvojkové soustavě zapsat jako 0011 – tedy součet čísel 1 a 2 a tedy jako:

$$e^3 = e^1 \cdot e^2$$

Funkce v každé iteraci násobí exponent mocnin dvojky k průběžnému výsledku. Počet nutných iterací je roven nejvyššímu bitu argumentu exponenciální funkce. Pro číslo 8 jsou to 4 iterace, pro číslo 32 jen 6, atd…

Výpočet `tanh` závisí na rychlosti a přesnoti implementace exponenciální funkce. Jelikož je funkce lichá, není potřeba upravovat algoritmus pro záporné argumenty, stačí jenom výsledku změnit znaménko. Také se při výpočtu využívá faktu, že funkce se velmi rychle přiblíží číslu jedna. Stačí tedy na začátku zjistit maximální hodnotu argumentu `tanh`, pro který již nestačí přesnost datového typu `double`. To vypočítáme přes funkci `artanh`:

$$x_{max} = artanh\ \varepsilon_{dbl} = \frac{1}{2} \ln \left(\frac{1 + \varepsilon_{dbl}}{1 - \varepsilon_{dbl}} \right)$$

kde x_{max} je hledaná maximální hodnota, kterou jsme ještě schopni vyčíslit pomocí `double` a ε_{dbl} je konstanta `DBL_EPSILON` z knihovny `float.h`.

Tato maximální hodnota se vypočítá jednou, při spuštění programu a ukládá se do `IZP_TANH_MAX`. Pro argumenty $x > IZP\_TANH\_MAX$ se proto vrací výsledek 1 bez dalšího výpočtu.

4.3 Implementace `logax`

Funkce obecného logaritmu se volá pomocí `approx_log_a()`. Zde se rozhodne, jestli je argument nebo základ logaritmu menší jak jedna. Pokud ano, vypočítá se převrácená hodnota čísla a výsledek se pak vrací s opačným znaménkem. Tento poznatek, popsaný v kapitole 2.2, jsem využil při návrhu algoritmu pro výpočet logaritmu – očekávám vstupní hodnoty pouze větší jak 1.

Podobně jako u hyperbolického tangens i zde je samotný výpočet rozdělen do dvou částí – na celočíselný a desetinný výsledek. Výsledky jsou pak sečteny a vráceny uživateli. Toto je implementováno ve funkcích `integer_log_a1_x1()` a `fractional_log_a1_x1()`.

Princip výpočtu si ukážeme na funkci `integer_log_a1_x1()`. V následujícím textu předpokládám toto:

$$y = \log_a x; \quad a^y = x$$

Funkce bere čísla `a` a `x` jako své argumenty. `y` je výsledek, který je předáván jako návratová hodnota. Základní myšlenka algoritmu je projít všechny bity čísla `y` a zjistit jestli jsou nastavené, nebo ne. Toto probíhá v jednom cyklu, kdy argument `x` dělím číslem a^n (`n` je n-tý bit v `y`). Pokud je výsledek dělení větší jak jedna, je daný bit čísla `y` nastaven. Můžeme tak napsat:

$$x_n = \frac{x_{n-1}}{a^n}$$

Jestli je `n`-tý bit výsledku `y` nastaven, je v `n`-té iteraci x_n větší jak jedna. Číslo `y` inkrementujeme o jedničku, posuneme jeho bity doleva a pokračujeme další iterací. Jakmile takto určíme všechny bity čísla `y`, máme hledaný vysledek uložen v `y` a zbytek v `x`. Tento zbytek je dále předán druhé funkci – `fractional_log_a1_x1()` – která je principiálně stejná jako první, ale počítá logaritmus čísla `x` menší jak základ `a`.

4.4 Implementace `wam` a `wqm`

Statistické funkce jsou implementovány ve funkcích `calc_wam` – vážený aritmetický průměr – a `calc_wqm` – vážený kvadratický průměr. Obě funkce berou jako první parametr ukazatel na strukturu `s_mean` do které se ukládají průběžné výsledky, druhým a třetím parametrem jsou čísla typu `double`, které představují hodnotu, resp. váhu `n`-tého prvku. Návratová hodnota obou funkcí je průběžný výsledek.

Reference

[1] KNUTH E. Donald: *The Art Of Computer Programming, Volume 2*. Addison-Wesley, 1997, ISBN 0-20-189684-2.

A Metriky kódu

Počet souborů: 1 soubor

Počet řádků zdrojového textu: 632 řádků

Velikost statických dat: 15361B

Velikost spustitelného souboru: 14544B (systém Darwin 10.5.0, 64 bitová architektura, gcc 4.2.1, při překladu bez ladicích informací)