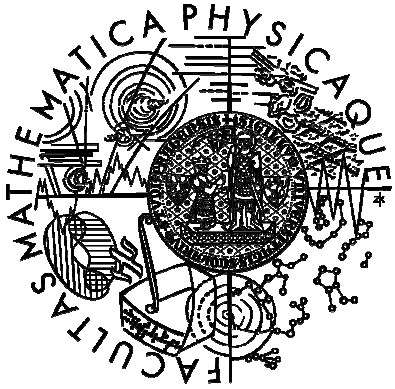
Univerzita Karlova v Praze
Matematicko-Fyzikální fakulta

Bakalárska práca



Stanislav Skotnický

Symbolická derivácia funkcie jednej reálnej premennej

Katedra

Vedúci bakalárskej práce: RNDr. Vladislav Kuboň, Ph.D.

Študíjny program: Informatika, teoretická informatika

2007

Poďakovanie.

Ďakujem svojmu vedúcemu RNDr. Vladislavovi Kuboňovi, Ph.D. za cenné rady pri písaní tejto práce a správne smerovanie pri vytváraní súvisiaceho ročníkového projektu.

Prehlasujem, že som svoju bakalársku prácu napísal sám a výhradne s použitím citovaných prameňov. Zároveň súhlasím so zapožičaním práce.

V Prahe, dňa Stanislav Skotnický

Obsah

Meno práce: Symbolická derivácia funkcie jednej reálnej premennej
Autor: Stanislav Skotnický
Katedra: Katedra
Vedúci bakalárskej práce: RNDr. Vladislav Kuboň, Ph.D.
E-mail vedúceho: Vladislav.Kubon@mff.cuni.cz

Abstrakt: Cieľom bakalárskej práce bude vytvoriť program na symbolické derivovanie funkcie jednej reálnej premennej s možnosťou využívania parametrov. Okrem derivácií bude program schopný zjednodušovať výrazy a funkcie. Program bude tiež schopný funkciu vykresľovať, a to pomocou upravenej Epsilonovej metódy.
Program bude schopný kontrolovať syntaktickú správnosť výrazov, bude v základnej variante podporovať použitie binárnych operácií +,-,*,/,^, unárnych operácií a goniometrických funkcií, je vhodné uvažovať i o rozšírení o absolútnu hodnotu, exponenciálu a logaritmus. Program by mal rozpoznať i niektoré základné konštanty ako e apod.
Samozrejmou súčasťou zadania je i požiadavka na komfortné a príjemné užívateľské rozhranie.
Kľúčové slová: Aritmetický výraz, Derivácia funkcie, Zjednodušovanie výrazov, Graf funkcie

Title: Symbolic derivations of functions with a single real variable
Author: Stanislav Skotnický
Department: Department
Supervisor: RNDr. Vladislav Kuboň, Ph.D.
Supervisor' e-mail address: Vladislav.Kubon@mff.cuni.cz

Abstract: The Main goal of this work is to create a program for the Symbolic derivations of functions with a single real variable. Program will be able to simplify arithmetic expressions and functions. It will be also able to draw functions with the improved Epsilon method. Program will check if the input is correct and will be able to use binary operators +,-,*, /, ^ unary operators and goniometric functions. Possibly it will also include exponential and logarithm functions and some basic constants like e or pi. Program will be comfortable and easy to use.

Key words: Arithmetic expressions, Derivation of function, Graphs of function, Functions with a single real variable

1 Úvod

Táto bakalárska práca slúži ako užívateľská príručka k programu Derivácia. Program je primárne určený na kreslenie grafov funkcií jednej reálnej premennej. To však nieje jediné, čo program dokáže. Okrem kreslenia funkcií program dokáže i funkcie derivovať a do istej miery, v prípade, že je to možné, aj zjednodušiť. Program podporuje všetky základné funkcie, konštanty ako Eulerovo a Rudolfovo číslo a umožňuje použitie parametrov. Nedielnou súčasťou programu je i kontrola syntaktickej správnosti zadanej funkcie a možnosť nastavenia zobrazenej časti grafu.

Program, rovnako ako táto práca, je určený hlavne pre laikov a študentov stredných a čiastočne i základných škôl. Nie pre študentov, či dokonca vyučujúcich Matematicko-fyzikálnej fakulty alebo podobne zameranej vysokej školy. Z tohoto dôvodu sú v práci veľmi podrobne vysvetlené jednotlivé termíny, riešené problémy i postupy ich riešenia. Cieľom programu a tejto práce je oboznámenie užívateľov s jednotlivými matematickými funkciami a ozrejmiť im vzťah medzi priebehom funkcie a priebehom jej derivácie. Z tohoto dôvodu sú v práci rovnako podrobne vysvetlené jednotlivé termíny, riešené problémy i postup ich riešenia.

Napriek tomu, že program a následne táto práca sú určené pre študentov stredných škôl, predsalen predpokladá znalosť niektorých základných vedomostí. Pre základné používanie programu Derivácia stačí, aby užívateľ mal predstavu, čo je to funkcia, jej správny zápis, graf funkcie poprípade parametre funkcie. Pre plné pochopenie tejto práce však autor predpokladá aspoň základnú znalosť pojmov súvisiacich s programovaním a algoritmami. Bez ďalšieho vysvetlenia budú uvedené pojmy ako program, algoritmus, premenná časová zložitosť algoritmov, rekurzia a ďalšie. Medzi ďalšími výrazmi, ktoré sú v texte použité bez predchádzajúcej definície, sú výrazy ako operátor, operand, či binárne a unárne operácie. Čitateľ by mal rovnako poznať, čo je to priorita operátorov a navyše i poznať priority jednotlivých operátorov a matematických funkcií. Tie sú dôležité hlavne na pochopenie algoritmu na prevod medzi infixovým a postfixovým zápisom výrazu.

Práca je rozdelená na dve základné časti. Časť teoretickú a praktickú. V prvej, teda teoretickej časti sa zoznámime so základnými pojmami súvisiacimi s aritmetickými výrazmi a ukážeme si rôzne spôsoby ich zápisu spolu s ich výhodami a nevýhodami. Zároveň si popíšeme najpoužívanejšie dátové štruktúry, používané na reprezentáciu dát v počítači, rovnako ako niektoré často používané algoritmy, ako napríklad prevod medzi infixovým a postfixovým zápisom. V časti venovanej derivácii sa zameriame na vysvetlenie samotného pojmu, čo pre funkciu znamená, ak v bode má deriváciu, a ukážeme si niekoľko možností praktického využitia derivácie. V časti venovanej zjednodušovaniu výrazov sa zameriame na porovnanie výsledkov, ktoré nám poskytne počítač s tými, ktoré sme schopní dosiahnuť samostatne.

V jednej z posledných kapitol si prestavíme program Derivácia, ukážeme si spôsob ovládania, možnosti nastavení a vysvetlíme si funkciu jednotlivých tlačítok a zaškrtávacích boxov v programe. V ďalšej kapitole sa zameriame na algoritmy použité na riešenie problémov v tomto programe, ktoré načrtneme v prvej časti. Pri každom algoritme uvedieme jeho špeciálne funkcie používa a vysvetlíme v čom je lepší než príslušný obecný algoritmus. Nebudú chýbať ani praktické ukážky programu, ktoré demonštrujú možnosti jeho použitia.

2 Reprezentácia aritmetického výrazu

Pojem Aritmetický výraz môžeme voľne definovať ako syntakticky správne zapísanú postupnosť funkcií, čísel a zátvoriek, ktorú môžeme vyhodnotiť a ako výsledok získať (v našom prípade) reálne číslo. Aritmetické výrazy však môžu obsahovať nielen samotné čísla ale aj rôzne konštanty (ako príklad uvediem konštanty *e* a π) a premenné. Povolenie premenných v aritmetickom výraze má za následok, že pri pokuse o jeho vyhodnotenie nemusíme vždy dospieť ku konkrétnemu číslu. V takomto prípade sa úloha riešiteľa mení a jeho snahou je výraz čo najviac zjednodušiť.

Ešte pred tým, ako sa budeme venovať reprezentácií aritmetického výrazu na počítači, budeme sa venovať spôsobu zápisu obecne. Jednotlivé typy zápisu sa medzi sebou líšia poradím operandov a operátorov. Podľa toho v akom poradí ich zapisujeme delíme zápis na človekom najčastejšie používaný infixový, postfixový a prefixový zápis výrazu. V prefixovom tvare zápisu ako prvý zapisujeme operátor a až potom operandy. Naopak v postfixovom zápise najskôr zapíšeme operandy a až potom nasleduje operátor. Ani v jednom z týchto dvoch zápisov nieje žiadne rozdiel medzi zápisom unárnych a binárnych operácií. Iná situácia je v infixovom zápise, kde je u binárnych operácií operátor zapísaný medzi operandami a u unárnych operácií je poradie operátorov a operandov rovnaké ako u prefixového zápisu. Na prvý pohľad je zrejme, že prefixový a postfixový zápis sa od seba veľmi nelíšia a jeden možno previesť na druhý jeho úplným obrátením. Je však nutné ošetriť jednu maličkosť. Po obrátení postfixového výrazu dostaneme prefixový tvar s obráteným poradím operandov. Vyriešenie tohoto problému by však nemalo robiť väzné problémy. Je ale potrebné naň pamätať. V ďalšom texte sa preto budeme zameriavať už iba na v praxi používanejší postfixový tvar.

Každý spôsob zápisu má svoje výhody aj nevýhody. Začneme výhodami a nevýhodami „ručne“ používaného infixu. Medzi významné určite patrí množstvo „grafických vylepšení“ ktoré robia zápis prehľadnejší. Najčastejšie požívane sú zlomky a exponenty mocnín, ktoré píšeme v podobe horného indexu. K prehľadnosti textu určite prispieva i špeciálny zápis odmocniny, kedy miesto sqrt(x) píšeme √x. V prípade postfixu / prefixu pravdepodobne tou najvýznamnejšou výhodou je, že v tomto druhu zápisu nie sú potrebné zátvorky ako v infixovom zápise. Naopak za nevýhodu musíme považovať nutnosť oddeliť od seba unárne

a binárne operátory plus a mínus. Ako mínus môžeme takisto označiť, že minimálne na prvý pohľad tento zápis vyzerá veľmi neprehľadne a jeho čítanie spočiatku môže spočiatku činiť mierne problémy.

Na počítači môžeme aritmetický výraz reprezentovať dvoma základnými štruktúrami. Tou prvou je jednorozmerné pole, poprípade spojový zoznam. Tou druhou je binárny strom, kde využívame faktu, že všetky operácie sú buď binárne alebo dokonca unárne. Vrcholy binárneho stromu budeme rozdeľovať na dva typy. Prvým sú listy. Ich základná vlastnosť je že nemajú žiadnych potomkov. Z tohto dôvodu sa v listoch stromu nachádzajú operandy, teda čísla, premenné, konštanty a parametre. Naopak vo vnútorných vrcholoch stromu sa nachádzajú operátory výrazu. Výhodou reprezentácie výrazu pomocou stromu oproti infixovému zápisu je, že rovnako ako v postfixovom a prefixom zápise výrazu nie sú potrebné zátvorky. Oproti postfixovému zápisu však nepotrebuje ani žiadne špeciálne znaky pre unárne operácie plus a mínus. Na rozdiel od spomínaných dvoch je jeho zápis výrazne prehľadnejší a na prvý pohľad je zrejmé, čo k čomu patrí. Za určitú nevýhodu možno považovať, že jeho zápis zaberá pomerne veľa miesta, a to najmä pri komplikovanejších výrazoch. Nevýhoda určite je aj vyššia časová náročnosť vyhodnotenia výrazu na počítači, ktorá je u stromu $N * \log_2 N$, kde N reprezentuje dĺžku vstupného výrazu.

Pravdepodobne najväčšou výhodou stromu je jeho rekurzívna štruktúra. Tá nám umožňuje výraz na počítači veľmi efektívne vyčísliť. Aritmetický výraz je pomocou stromu reprezentovaný následovne. Výpočet začína u koreňa. Ak koreň obsahuje operand, tak algoritmus končí a výsledok je práve táto hodnota. V opačnom prípade sa rovnaký postup aplikuje na oboch podstromoch (každý podstrom je samostatný binárny strom), ktoré sa samostatne vyhodnotia a koreňu vrátia iba svoju výslednú hodnotu. Po získaní výsledku z oboch podstromov (v prípade, že sa jedná o unárnu operáciu iba jedného) je postup rovnaký ako v prípade postfixového tvaru zápisu. Na oba výsledky aplikujeme príslušný operátor a výsledok vrátime ako výsledok celého stromu. V prípade, že sme sa nachádzali v uzle, iba vrátime hodnotu. V ďalšom texte sa kvôli podobnosti binárnych stromov a postfixového tvaru budeme venovať už iba postfixovému zápisu a jeho porovnaniu so zápisom infixovým, efektivitou algoritmov a nakoniec aj prevodmi medzi nimi. Najskôr však malá ukážka reprezentácie výrazu pomocou spomínaných spôsobov zápisu.

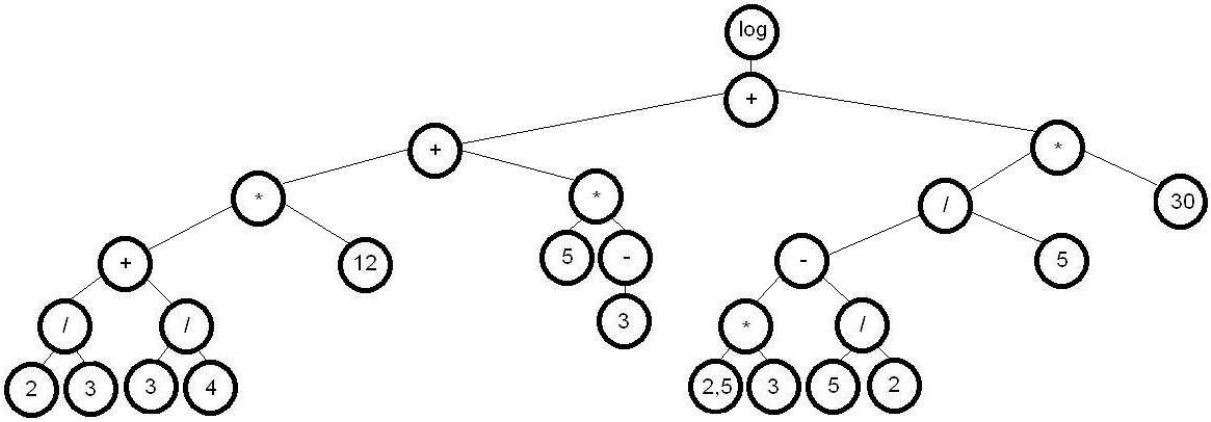
Príklad č. 1: na nasledujúcom výraze si ukážeme rozdiely medzi infixovým a postfixovým zápisom a následne vysvetlíme možný postup ich riešenia na počítači.

infixový zápis: $log_2 ((2/3 + 3/4) * 12 + 5 * (-3) + (2{,}5 * 3 - 5 / 2) / 5 * 30)$

prefixový zápis: $log_2 + * 30 / 5 - / 2 5 * 3 2{,}5 + * @ 3 5 * 12 + / 4 3 / 2 3$

postfixový zápis: $2\ 3 / 3\ 4 / + 12 * 5\ 3 @ * + 2{,}5\ 3 * 5\ 2 / - 5 / 30 * + log_2$

binárny strom:



Výsledkom tohoto výrazu je číslo 5. V postfixovom zápise pozorného čitateľa iste zaujme znak @. Jedná sa o špeciálny znak reprezentujúci unárne mínus. Keby sme ho od binárneho nerozlíšili pravdepodobne by sa nám stalo, že výraz 5 3 – by bol vyhodnotený ako 5 – 3, čo nechceme.

Spomenuli sme výhody postfixového zápisu, poprípade zápisu pomocou binárneho stromu oproti zápisu infixovému. V prechádzajúcom odstavci sme ale nenašli jednu veľmi dôležitú výhodu. Tou je rýchlosť vyhodnocovania na počítači, ktorá je výrazne vyššia ako pri vyhodnocovaní výrazu používajúceho infixový zápis. Zároveň sa jedná o hlavný dôvod, prečo sa pri spracovávaní na počítači používa skôr postfixový tvar zápisu. Pomer medzi dĺžkou operadnov a operácii a počtom operácií ktoré musí počítač vykonať na jeho vyhodnotenie je kvadratický, oproti lineárnemu u zápisu postfixového.

Postup pri vyhodnotení infixového tvaru je nasledujúci. Program najskôr nájde operátor s najnižšou prioritou (v tomto prípade je to operátor +) a následne výraz rozdelí na tri časti. Časť naľavo od operátora, samotný operátor a časť napravo od neho. Obe časti, teda prvú a tretiu samostatne vyhodnotí a na výsledok aplikuje predtým nájdený operátor. Toto platí u binárnej operácie. V prípade že sa jedná o operáciu unárnu je postup veľmi podobný. Je

dôležité, aby sme si uvedomili, že pri tomto postupe sa môže stať a pravdepodobne aj stane, že niektoré časti výrazu budeme prehľadávať niekoľkokrát. Všimnime si, že po prvom rozdelení nášho výrazu je ľavá časť takmer rovnako dlhá ako pôvodný výraz, a preto sa pri ďalšom najbližšom hľadaní operátora s najnižšou prioritou opäť prejde takmer celý pôvodný výraz. Tým sa celková doba vyhodnocovania zbytočne predĺži.

Oproti tomu postfixový zápis je napísaný presne v takom poradí, ako sa má vyhodnocovať. Program teda ako prvé načíta čísla 2 a 3. Tie si zapamätá, presnejšie uloží na vrch zásobníku. Keď následne narazí na operátor delenia vyzdvihne zo zásobníku 2 naposledy uložené čísla, vyhodnotí výraz a výsledné číslo vráti späť na vrch zásobníku. Daný výraz je preto nutné prejsť iba jedenkrát. Zároveň je daný postup menej náročný na pamäť. Niektorých čitateľov by mohlo napadnúť, že vyhodnotenie výrazu nemôže byť pre dnešné počítače problém. To je minimálne čiastočne pravda. Situácia sa zmení, ak sa snažíme nájsť napríklad korene rovnice, alebo nakresliť graf funkcie. V týchto prípadoch je potrebné vypočítať veľké množstvo hodnôt, môže byť časový rozdiel už nezanedbateľný. Pri komplikovanejších funkciách môže byť pomer aj niekoľko sekúnd oproti niekoľkým hodinám. Ostáva nám vyriešiť ešte jeden podproblém vyhodnocovania výrazu.

Tým je prevod medzi infixom a postfixom. Ten je dôležitý, pretože väčšina užívateľov pravdepodobne nebude ochotná zadávať výraz v postfixovom tvare a ako sme sa presvedčili v predchádzajúcom odstavci, vyhodnocovanie výrazu v infixovom tvare je časovo výrazne náročnejšie. V prípade zjednodušenia výrazov budeme potrebovať navyše ešte jeden algoritmus, a to prevod z postfixového zápisu do jeho infixovej formy. Dôvoď je rovnaký ako v predchádzajúcom prípade. Tento prevod by sme navyše chceli čo najrýchlejší, ideálne na jeden prechod vstupu. V prípade, že by bol prevod príliš pomalý, mohlo by sa stať, že čas ktorý sme ušetrili na vyhodnotení výrazu v postfixe, stratíme jeho prevodom. Tento problém sa čiastočne eliminuje v prípade, keď sa snažíme o vyhodnotenie funkcie, a teda o výpočet veľkého množstva výrazov. Výraz na postfixový tvar prevedieme iba jeden krát na začiatku programu a na samotné vyhodnocovanie výrazu budeme používať postfixový tvar výrazu.

Stručne si popíšme postup takéhoto prevodu. Na začiatku celého programu musíme rozlíšiť unárne mínus od binárneho. Jednou z možností ako toho dosiahnuť je prejsť celý výraz a ak narazíme na mínus a tesne pred ním bola ľavá zátvorka, zmeníme tento znak na nejaký špeciálny označujúci unárne mínus. Vhodný kandidát môže byť napríklad @. Rovnako

sa snažíme o odstránenie unárneho plus. V tomto prípade máme prácu zjednodušenú o to, že samotné unárne plus je zbytočné, a preto ho s kľudným svedomím môžeme zmazať. Pri prevode budeme používať jeden pomocný zásobník. Výsledný postfixový výraz budeme mať rovnako ako vstupný infixový uložený v poli. Postupne budeme spracovávať vstupný výraz. Vždy načítame jeden operand alebo operátor. V závislosti na tom, čo v danom kroku načítame, postupujeme ďalej. Tými možnosťami sú:

1) Na vstupe je operand. Jedná sa o najjednoduchší prípad, kedy jediná naša akcia je pridanie tohto operandu na koniec vznikajúceho zoznamu

2) Na vstupe je operátor. Najskôr zo zásobníka vyberieme všetky operátory s rovnakou alebo vyššou prioritou a pridáme ich na koniec tvoriaceho sa postfixu. Následne tento operátor uložíme na vrchol zásobníka operátorov. Na tomto mieste je dobré si pripomenúť, že unárne operátory (napríklad goniometrické funkcie) majú vyššiu prioritu ako binárne (+,-,*,/,^)

3) Na vstupe je ľavá zátvorka. Vložíme ju na vrchol zásobníku.

4) Na vstupe je pravá zátvorka. Zo zásobníka postupne odoberáme všetky operátory a následne ich pridávame na koniec postfixu. To opakujeme až kým nenarazíme buď na ľavú zátvorku alebo na dno zásobníku. V prvom prípade odstránime zátvorku zo zásobníku, v druhom ukončíme algoritmus, pretože sme práve detekovali chybne uzátvorkovaný vstupný výraz.

Po spracovaní celého výrazu vyprázdnime zvyšok zásobníku a pridáme ho na koniec výsledného postfixu. V prípade, že sa v ňom nachádzajú ešte nejaké zátvorky, opäť môžeme vstupný výraz prehlásiť za nesprávne uzátvorkovaný a teda chybný. Kontrola zátvoriek nieje jediný druh kontroly vstupu. Spomedzi ďalších spomeniem testovanie, či posledný člen výrazu je operand (infixový výraz sa nemôže končiť operátorom), alebo či v zápise čísla nepoužívame bodku namiesto čiarky, poprípade či sme v čísle omylom nezadali dve čiarky.

Nakoniec tejto časti venovanej reprezentácii aritmetického výrazu na počítači si ešte v stručnosti ukážeme prevod z postfixového zápisu na zápis infixový. Ten je potrebný v prípade, že sme sa výraz pokúsili zjednodušiť a výsledný výraz vypísať na obrazovku. Práca

je jednoduchšia aj z toho pohľadu, že tento krát sa nemusíme starať s syntaktickú správnosť vstupného výrazu. Stručný postup je následovný. Pri prechádzame postfixový zápis a postupne ho spracovávame. Tento krát si dokonca vystačíme s jedným zásobníkom. V závislosti na tom, čo v danom kroku načítame, postupujeme ďalej. Tými možnosťami sú:

1) Na vstupe je operand. V tom prípade iba pridáme prvok na vrchol zásobníka

2) Na vstupe je unárne mínus. Vyberieme vrchol zásobníka spolu s jeho prioritou. Ak ňou je + alebo, výraz obalíme do zátvoriek. Následne na začiatok výrazu pridáme operátor – a upravíme prioritu.

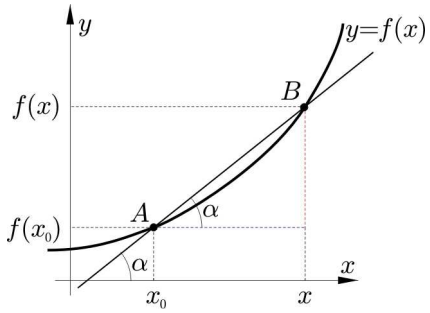
3) Na vstupe je nejaký iný unárny operátor. Pretože tieto operátory majú najvyššiu prioritu, nemôže sa stať, že by výraz v zásobníku mal prioritu vyššiu. Postup je preto vždy rovnaký. Vyberieme vrchol zásobníka, výraz obalíme do zátvoriek a daný operátor pridáme na začiatok výrazu. Ani tento krát nezabudneme nastaviť prioritu.

4) Na vstupe je binárny operátor. Tento krát vyberiem zo zásobníka hneď dva prvky. U oboch zistíme ich prioritu. Ak je pri u niektorého z nich priorita nižšia ako u pridávaného operátora, výraz zabalíme do zátvoriek. Na vrchol zásobníka vložíme už iba jeden výraz, ktorý je spojením spomínaných dvoch s operátorom uprostred. Rovnako ako v predchádzajúcich dvoch prípadoch nezabudneme nastaviť prioritu.

Na konci algoritmu budeme mať v zásobníku iba jeden prvok, ktorý bude reprezentovať nami žiadaný infixový zápis výrazu. V prípade, že sa snažíme o prevedenie výrazu, u ktorého si nie sme istí syntaktickou správnosťou vstupného postinfixového tvaru, musíme pamätať aj na kontrolu údajov. Asi najdôležitejšie je kontrolovať či pri snahe o vybratie zásobníka nie je zásobník prázdny. Tato snaha by okamžite spôsobila nepredvídateľné chovanie programu a pravdepodobne aj jeho pád. Rozumná bude i kontrola, či na konci programu je zásobník skutočne iba jeden prvok. Ak by mal prvkov viac, nutne by to znamenalo, že vstupný výraz bol syntakticky nesprávny.

3 Derivácia funkcie jednej reálnej premennej

V tejto časti sa budeme venovať derivovaniu funkcií jednej reálnej premennej. Presnejšie povedané sa jedná o derivovanie symbolické. V kapitole sa zameriame napríklad na to, aký je význam derivácií, či aký je rozdiel medzi analytickým a symbolickým derivovaním. Skôr ako pristúpime k vlastnej definícií pojmu derivácia, pozrime sa na obrázok 1. Na krivke y = f(x) je pevne zvolený bod A[x0, f(x0)]. „Kúsok“ od neho si zvoľme bod B[x, f(x)]. Sečnicu AB vyjadríme v smernicovom tvare ako priamka $y = k_s * x + q$ má smernicu $k_s = \operatorname{tg} \alpha = \frac{f(x) - f(x_0)}{x - x_0}$. Limitným prechodom (tj. bod B zvolíme nekonečne blízko bodu A) sa sečnica zmení na dotyčnicu, pre ktorú je smernica $k_t = \lim \frac{f(x) - f(x_0)}{x - x_0}$.



Definícia: Označme f reálnu funkciu a x_0 nejaký ľubovoľný ale pevný bod na reálnej ose. Ak existuje vlastná limita

$$f'(x) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

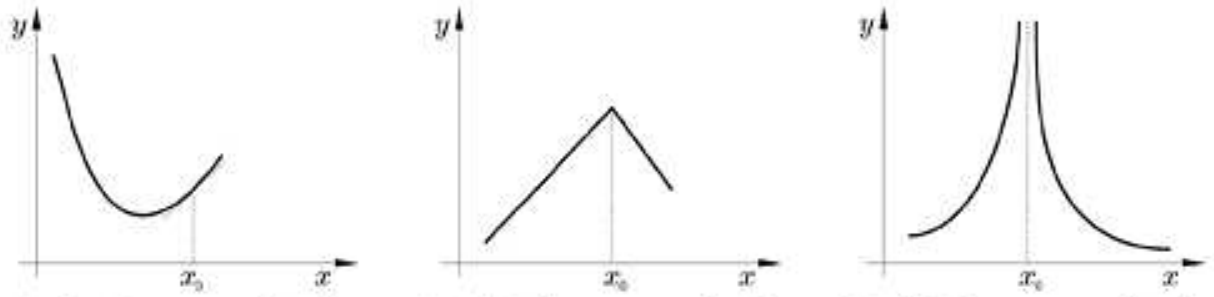
nazveme túto limitu (vlastnou) deriváciou funkcie v bode x_0 a označujeme ju $f'(x_0)$. V ostatných prípadoch, tj. limita sa rovná $\pm\infty$, alebo neexistuje, hovoríme, že funkcia f deriváciu v bode x_0 nemá.

V prípade, že funkcia má v bode x_0 deriváciu, hovoríme, že funkcia f je v bode x_0 diferencovateľná. Funkcia je diferencovateľná na intervalu I, ak je diferencovateľná v každom bode tohoto intervalu. Ak je daná funkcia diferencovateľná na nejakom intervalu, môžeme na tomto intervalu definovať funkciu, ktorá je v každom bode tohoto intervalu rovná príslušnej derivácii. Taká funkcia sa potom označuje jednoducho ako derivácia funkcie f. Deriváciou diferencovateľnej funkcie je teda opäť funkcia, ktorá navyše môže byť tiež diferencovateľná.

Deriváciu derivácie funkcie nazývame druhá derivácia, deriváciu druhej derivácie treťou deriváciou atď. Tieto derivácie vyšších radov sa obvykle označujú f″(x), f‴(x), pre ešte vyššie rady potom skôr f(3)(x), f(4)(x) atď.

Ešte jedna poznámka na okraj. Historické definície vyjadrovali deriváciu ako pomer, v akom rast nejakej premennej y odpovedá zmene inej premennej x, na ktorej má ona premenná nejakú funkčnú závislosť. Behom vývoja matematiky sa intuitívna predstava nekonečne malých (infintezimálnych) hodnoť ukázala ako nedostatočne presná a bola nahradená „ε-δ“ formalizmom limít. Najjednoduchšia predstava o derivácií je, že „derivácia je mierou zmeny funkcie v danom bode, resp. bodoch“.

Pojem derivácie je veľmi úzko spojený s ďalším dôležitým pojmom. Týmto pojmom je spojitosť. Len pre pripomenutie uvedieme, že funkcia f je bode x spojitá práve vtedy, ak sa jej funkčná hodnota v bode x_0 rovná jej limite. Platí totiž veta, ktorá tvrdí, že ak má funkcia v bode x_0 vlastnú deriváciu tak je v tomto bode spojitá. Opačné tvrdenie však neplatí. Na nasledujúcom obrázku si ozrejmíme vzťah medzi spojitosťou a deriváciou.



Na obrázku je krásne vidieť, že keď funkcia na grafe č. 2 nemá v bode vlastnú deriváciu nieje v tomto bode tzv. hladká ako na grafe č.1 , teda obsahuje ostrý prechod. Na grafe č. 3 pre zmenu vidíme prípade keď funkcia nemá ani vlastnú deriváciu a nieje ani spojitá.

Dostávame sa k meritu problému a to k výpočtu derivácie. Principiálne základnou technikou je výpočet priamo z definície, tzn. dosadením príslušnej funkcie do definujúcej limity a výpočtom tejto limity. Tento spôsob je však obvykle (až na veľmi jednoduché funkcie) dosť komplikovaný a v praxi sa nepoužíva. Namiesto toho sa derivácie funkcií počítajú zo

známych derivácií niekoľkých základných funkcií a jednoduchých algebraických pravidiel pre ich skladanie a ďalšie úpravy.

Elementárne funkcie

Derivácia niektorých elementárnych funkcií

Funkcie	Derivácia
Polynomy	
$f(x) = c$ (*c* je konštanta)	$f'(x) = 0$
$f(x) = x^c$ (*c* je konštanta)	$f'(x) = cx^{c-1}$
Mocniny, logaritmy	
$f(x) = c^x$ (*c* je konštanta, $c > 0$)	$f'(x) = c^x \ln c$
$f(x) = e^x$ (*e* je Eulerové číslo)	$f'(x) = e^x$
$f(x) = \log_a x$ (*a* je konštanta, $a > 0$, $a \neq 1$)	$f'(x) = \frac{1}{x \cdot \ln a}$
$f(x) = \ln x$	$f'(x) = \frac{1}{x}$
Goniometrické funkcie	
$f(x) = \sin x$	$f'(x) = \cos x$
$f(x) = \cos x$	$f'(x) = -\sin x$
$f(x) = \operatorname{tg} x$	$f'(x) = \frac{1}{\cos^2 x}$
$f(x) = \operatorname{cotg} x$	$f'(x) = -\frac{1}{\sin^2 x}$

Algebraická pravidla

Zo známych derivácií elementárnych funkcií sa derivácie zložitejších funkcií zostavujú tak, že sa zložitejšie funkcie rozložia na jednoduchšie pomocou jednoduchých algebraických pravidiel, ktoré pre výpočet derivácií platia:

- **Linearita derivácie:** $(af + bg)' = af' + bg'$ pre ľubovolné funkcie *f*, *g* a konštanty *a*, *b*.
  - Špeciálne platí $(af)' = a \cdot f'$ a také $(f + g)' = f' + g'$.
- **Derivácia súčinu:** $(fg)' = f'g + fg'$ pre všetky funkcie *f*, *g*.
- **Derivácia podielu:** $\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$ pre všetky funkcie *f*, *g*, kde $g \neq 0$.

- **Derivácia zloženej funkcie:** Ak $f(x) = h(g(x))$, potom $f'(x) = h'(g(x)) \cdot g'(x)$.

V tejto časti si ešte ukážeme niekoľko príkladov využitia derivácie funkcie.

Lokálne extrémy

Ak má daná diferencovateľná funkcia nejaký lokálny extrém (lokálne maximum či minimum), je zrejmé, že jej dotyčnica v tomto bode musí byť vodorovná, tzn. derivácia tejto funkcie musí byť v tomto bode nulová. (Ak funkcia v nejakých bodoch dotyčnicu, resp. deriváciu nemá, derivácia o takýchto bodoch nedokáže nič prezradiť). Ak sa v tomto bode dá spočítať i druhá derivácia, prezradí jej znamienko, o aký extrém sa jedná:

- V bodoch, kde je prvá derivácia nula a druhá derivácia je kladná, sa nachádza *lokálne minimum*.
- V bodoch, kde je prvá derivácia nula a druhá derivácia je záporná, sa nachádza *lokálne maximum*.
- V bodoch, kde je jak prvá, tak druhá derivácia nulová, sa nachádza tzv. *stacionárny bod*, ktorý môže a nemusí byť extrémom.
- (V bodoch, kde funkcia nemá prvú či druhú deriváciu, je musíme použiť iné kritéria.)

Alternatívou k rozlíšeniu pomocou druhej derivácie je znamienko prvej derivácie: v bode, kde má funkcia lokálny extrém, mení prvá derivácia znamienko: ak je nejaký bod lokálnym minimom, potom v jeho ľavom okolí je prvá derivácia záporná a v pravom okolí kladná, naopak v ľavom okolí lokálneho maxima je prvá derivácia kladná a v pravom záporná.

Tieto kritéria sa často používajú v optimalizačných úlohách. Ak máme napr. nájsť obdĺžnik, ktorý pri zadanom obvodu má maximálnu plochu, musíme nájsť maximum funkcie $f(x) = x \cdot (o/2 - x)$. Jej deriváciou je funkcia $f'(x) = o/2 - 2x$, ktorá je nulová pre $x = o/4$. Druhá derivácia funkcie f je $f''(x) = -2$, tzn. je všade záporná. V bode $x = o/4$ má teda funkcia f maximum. Znamená to teda, že všetkých obdĺžnikov o zadanom obvode má najväčší obsah ten, ktorý má všetky štyri strany rovnako dlhé, tzn. štvorec.

Analýza priebehu funkcie

Predchádzajúci odstavec popisuje spôsob, ako pre danú funkciu nájsť jej lokálne extrémy. To môže okrem optimalizačných úloh slúžiť rovnako k získaniu prehľadu o chovaní funkcie, teda

jej tzv. priebehu. Využitie je napríklad pri ručnom náčrtu jej grafu. Okrem analýzy extrémov môžeme využiť deriváciu k nasledujúcim pozorovaniam:

- V bodoch, kde je prvá derivácia kladná, je funkcia rastúca
- V bodoch, kde je prvá derivácia záporná, je funkcia klesajúca.
- V bodoch, kde je druhá derivácia kladná, je funkcia konvexná.
- V bodoch, kde je druhá derivácia záporná, je funkcia konkávna.
- V bodoch, kde je druhá derivácia nulová, sa môžu vyskytovať inflexné body.

Fyzika

Jednoznačne najdôležitejšou oblasťou použitia derivácie vo fyzike sú derivácie podľa časovej premennej, vyjadrujúce rýchlosť zmeny nejakej premennej v čase. Najbežnejšie ďalej sú časové derivácie polohy, ktoré sa vyskytujú v klasickej kinematike. Napríklad okamžitá rýchlosť je derivácia súradnice polohy telesa podľa času.

Zrýchlenie je derivácia rýchlosti podľa času, tzn. druhá derivácia polohy podľa času.

Ryv je derivácia zrýchlenia podľa času, tzn. tretia derivácia polohy podľa času.

Okrem týchto základných pojmov sa derivácia objavuje v mnohých teóriách fyzikálnych polí, Maxwellových rovniciach atd.

Diferenciálne rovnice

Mnoho vedeckých problémov vieme sformulovať v podobe rovníc, v ktorých sa vedľa seba vyskytuje nejaká funkcia i jej derivácia. Takejto rovnici hovoríme diferenciálna rovnica. Diferenciálne rovnice sa objavujú snáď vo všetkých vedeckých oboroch. Okrem matematiky a fyziky i napr. v chémií, sociológii, ekológii atd.

4 Zjednodušovanie aritmetického výrazu

Na začiatku tejto kapitoly plynulo nadviažeme na predchádzajúcu kapitolu venovanú derivovaniu funkcií jednej premennej. V nej sme sa dozvedeli, že derivácie nepočítame podľa definície. Jednoduché funkcie derivujeme podľa tabuľky derivácie základných funkcií a u komplikovanejších funkcií okrem spomínanej tabuľky používame jednoduché algebraické pravidlá na základe ktorých vieme zderivovať napríklad súčin funkcií, deriváciu zloženej funkcie a pri splnení určitých vlastností pôvodnej funkcie aj deriváciu funkcie inverznej. Zcela logicky sa preto budeme sa zamerať derivovaním symbolickým, o ktorom sme sa dozvedeli rovnako v minulej kapitole.

Ak budeme postupovať presne podľa pravidiel uvedených v predchádzajúcej kapitole, zistíme, že výsledná funkcia je zbytočne komplikovaná. Ako príklad uvediem deriváciu funkcie f(x) = 2*x. Ak budeme derivovať podľa pravidiel , výsledná derivácia bude: f'(x)= 0*x + 2*1. Našim cieľom je ale výsledok f'(x)= 2. Tento problém sa týka hlavne pri derivovaní na počítači. Preto sa v ďalšom texte zameriame na derivovanie na počítači.

Aby sme nášho cieľa čo najjednoduchšej zderivovanej dosiahli, musíme výslednú funkciu čo najviac zjednodušiť. Môžeme použiť dva možné prístupy. Za prvé sa môžeme pokúsiť o zjednodušenie priamo pri derivovaní. Ak napríklad pri derivovaní súčine zistíme, že jedna funkcia obsahuje iba konštanty, môžeme použiť iný vzorec ako pri derivovaní súčinu dvoch funkcií. Ďalšou možnosťou je rozdeliť deriváciu na dve časti. V prvej časti funkciu zderivujeme presne podľa pravidiel a v až v druhej časti sa budeme snažiť o jej zjednodušenie. Tento postup má svoje opodstatnenie, ak je derivácia súčasťou väčšieho programu, v ktorom sa zjednodušenie výrazu využívajú aj pre iné časti programu. Užívateľ napríklad môže požadovať aby sme mu zjednodušili aj ním zadanú vstupnú funkciu.

Náročnosť zjednodušovania funkcií a aritmetických výrazov obecne je veľmi individuálna a záleží hlavne na miere sotisfikovanosti programu. Ak budeme chcieť navrhnúť program, ktorý bude zvládať aj náročnejšie operácie ako vynímanie pred zátvorky alebo vyhľadávanie vzorcov, čaká nás pravdepodobne omnoho viac práce, než pri vyhodnocovaní výrazu popisovanom v prvej časti tejto práce.

Komerčné programy najčastejšie používajú na prvý pohľad veľmi jednoduchý algoritmus. Program pozná niekoľko pravidiel na úpravu aritmetických výrazov. Tie postupne prechádza a snaží sa ich aplikovať na zadaný výraz. Ak sa výraz po prejdení všetkých pravidiel nezmení, zjednodušovanie končí. Nie všetky pravidlá musia nutne výraz zjednodušiť. Môže nastať situácia, kedy jedno pravidlo výraz skomplikuje, aby iné pravidlo mohlo výraz zjednodušiť. Jeden príklad za všetky. Výraz $a^2 - b^2$, má ekvivalentný zápis *(a+b)*(a-b)*, ktorý je rozhodne komplikovanejší. Ak však nasleduje delenie výrazom *a-b* situácia sa rázom mení. Na druhej strane ak bude nasledovať výraz $+ b^2$, bude výhodnejší kratší zápis.

Jednotlivé pravidlá môžu byť rôzne komplikované v vzájomne medzi sebou prepojené. Tento postup je obzvlášť výhodný je program písaný v nejakom neprocedurálnom programovacom jazyku typu Prolog, kde je veľmi užitočný pomocník unifikácia.

Ďalšia možnosť ako zjednodušovať výrazy je použitie algoritmu podobajúceho sa algoritmu na vyhodnocovanie aritmetického výrazu. Pri vyhodnotení máme niekoľko možností ako reprezentovať práve zjednodušovaný výraz. Ten najjednoduchší, avšak najmenej účinný je založený na nasledujúcom princípe. Na vstupe algoritmu je postfixový výraz. Postupne sa spracovávajú jednotlivé operandy a operátory. Ak je na vstupe operand vloží sa na vrchol zásobníku, rovnako ako pri vyhodnocovaní. V prípade, že je na vstupe nejaká unárna funkcia, vyberie sa vrch zásobníku a spolu s funkciou vytvoria sú uložené na vrchol zásobníku. Ak je na vstupe binárna funkcia, vyberú sa zo zásobníku vrchné dva výrazy a podľa toho aká binárna operácia je práve spracovávaná okúsi sa o zjednodušenie. V prípade, že sa zjednodušenie nepodarí, vytvorí sa z oboch výrazov a binárnej funkcie nový výraz a ten je vložený na vrchol zásobníku. Toto riešenie má jednu podstatná nevýhodu a to tú, že možnosti zjednodušenia výrazov sú obmedzené na najjednoduchšie úpravy.

Jedná sa najmä o výrazy, ktoré vznikajú pri symbolickom derivovaní funkcií jednej premennej. Často krát sa stáva, že pri derivovaní vznikajú výrazy, v ktorých sa v nejakej časti výrazu násobí nulou. My by sme chceli aby program tento komplikovaný výraz nahradil nulou. Riešenie je celkom jednoduché. Pri operácii násobenia porovnáme obidva výrazy. Ak je aspoň jeden z nich nulový, bude nulový aj celkový výraz, ktorý vložíme na vrchol zásobníka. Podobný postup je pri násobení jedničkou alebo pripočítavaní nuly. V oboch prípadoch je postup rovnaký. Ak v prípade sčítania sa jeden z výrazov nulový, výsledný výraz

je rovný hodnote druhého výrazu. Podobne pri násobení jednotkou. Opäť ako výsledok použijeme hodnotu druhého výrazu. Medzi ďalšie možnosti, zjednodušenia spomeniem odčítanie rovnakých výrazov, kedy je výsledok nulový, delenie rovnakých výrazov s výsledkom rovným konštante jedna, alebo sčítanie, či násobenie rovnakých výrazov. Problém ale nastáva už v prípade, že máme na vstupe výraz typu *x + y - x (*v postfixe reprezentovaný ako: *x y + x -)*. Takýto výraz už zjednodušiť nevieme. Je to z toho dôvodu, že výraz *x + y* algoritmus zjednodušiť nevie a pri spracovávaní operácie mínus porovnáva výrazy *x+ y* a x. Pretože sa tieto dva výrazy nerovnajú nedôjde ani ku zjednodušeniu a výraz ostane v pôvodnom stave. Práve táto vlastnosť tohto návrhu spôsobuje, že architektúra je použiteľná iba na základné úpravy aritmetických výrazov.

Existuje niekoľko možností ako môžeme architektúru bez výrazných zásahov vylepšiť. Tým prvým je priradenie priority ku každému operadnu, či už štandartným ako premenné, konštanty, alebo parametre, rovnako ako operandom vytvoreným počas priebehu zjednodušovania. Pri spracovávaní binárnych operácií +,* (a pri miernych úpravách aj binárne mínus) porovnáme priority oboch operandov a ak má pravý operátor (bol na vrcholu zásobníku) nižšiu prioritu, oba operandy prehodíme. Výsledok si ukážeme na spracovaní výrazu x + y – x. Ako sme si ukázali v predchádzajúcom odstavci najskôr spracujeme výraz *x + y*. Ten sa nám zjednodušiť nepodarí. Dynamicky mu vytvoríme prioritu a pri porovnávaní výrazov *x + y* a *x* zistíme, že oba operandy vymeníme. Dostaneme tak výraz *–x + x + y*. Program porovnaním pôvodného a upraveného výrazu zistí, že došlo k zmene a pozmenený výraz sa pokúsi upraviť ešte raz. Tentoraz budeme najskôr porovnávať výrazy *–x* a *x*, ktoré dokážeme vďaka ďalšej miernej modifikácie architektúry eliminovať. Tou modifikáciu je pridanie znamienka k jednotlivým operandom. O každom operande si teda budeme pamätať samotná hodnotu , jeho prioritu a ešte aj znamienko. Vďaka tejto úprave sú operandy *x* a *–x* majú pri porovnaní rovnakú hodnotu samotného operátora a rôzne znamienka, čo pri sčítaní znamená, že sa odčítajú a nahradia sa nulou. Táto úprava nám umožní napríklad zmeniť výraz *sin(-x)* na *–sin (x)* alebo *cos(-x)* na *cos (x)*. Pri násobení pre zmenu z *(–x)*(-y)* na *x*y*.

Ešte lepších výsledkov dosiahneme ak logickú hodnotu znamienko nahradíme reálnym číslom, ktoré bude reprezentovať znamienko operandu ale aj násobnosť. Po tejto úprave budeme môcť jednoducho sčítavať už aj komplikovanejšie výrazy. Ako napríklad *x + 3*x*. Štruktúru môžeme ešte obohatiť aj u mocninu a podobne ako pri sčítaní rôznych násobností jednotlivých konštánt, môžeme ľahko násobiť a deliť rovnaké operandy s rôznymi

mocninami. Pri násobení ešte vynásobíme oba multiplikátory a výsledok použijeme ako multiplikátor nového operandu.

Možností ako upraviť architektúru je výrazne viac a obecne platí, že čím viac si budeme o jednotlivých výrazoch pamätať, tým lepšie výsledky budeme dosahovať. Netreba však zabúdať, že pridávaním stále nových atribútov k jednotlivým operandom sa zvyšuje priestorová i časová náročnosť na spracovanie výrazov. My sme si ukázali iba niekoľko najjednoduchších. Pre výrazne lepšie výsledky sú však nutné výraznejšie zásahy do architektúry a hlavne vynímanie pred zátvorky. Napríklad výraz *a*x + b*x* na *(a+b)*x* inak ako vynímaním pravdepodobne nedostaneme. Ďalšou možnosťou, používanou hlavne pri zjednodušovaniu polynómov, je tzv. faktorizácia, ktorý sa v matematike a jej aplikáciách označuje problém rozloženia čísla na súčin menších čísiel, v najčastejšej podobe potom rozklad celého čísla na súčin prvočísiel. Napríklad číslo 15 môžeme napísať ako súčin 3 * 5. Obecnejšie môžeme rozkladať aj iné algebraické objekty, napr. polynóm druhého stupňa $x^2 - 4$ môžeme vyjadriť ako súčin dvoch polynómov prvého stupňa *(x − 2)*(x + 2)*

5 Kreslenie grafov funkcií

V tejto kapitole sa zameriame na význam grafov a na problémy s nimi súvisiace. Zároveň sa pokúsime prísť na výhody a nevýhody kreslenia grafov na počítači.

Jeden z najvýznamnejších dôvodov, možno dokonca najvýznamnejším pre kreslenie grafov funkcií je naša vlastná predstavivosť. Väčšina z nás si totiž vie lepšie predstaviť obrázok ako tabuľku, či dokonca nejakú formulku, ktorá je v značnom počte prípadov navyše veľmi komplikovaná. To platí nielen u grafov funkcií jednej reálnej premennej, ale o grafoch obecne. Grafy sú výrazne prehľadnejšie napríklad aj oproti tabuľkám, ktoré však majú svoje vlastné výhody.

Graf funkcie nám poslúži aj v prípade, že chceme zistiť jej základné vlastnosti, ako body nespojitosti, lokálne extrémy, na ktorých úsekoch je funkcia rastúca a na ktorých klesajúca. Ďalšou oblasťou, k ktorej sú grafy veľmi užitočné je porovnanie viacerých funkcií. Je totiž výrazne jednoduchšie porovnať dve čiary ako pracne zisťovať ktorá funkcia, má v žiadaných bodoch vyššie hodnoty. Táto výhoda sa stráca v prípade, že

Rovnako ako v predchádzajúcich prípadoch má kreslenie grafov pomocou programu oproti kresleniu ručnému svoje výhody ale i nevýhody. Aby sme tieto dve metódy porovnali mali by sme vedieť akým spôsobom sa prevádzajú.

Začneme kreslením ručným. Najskôr určíme body nespojitosti a teda vlastne definičný obor funkcie. To sú body v ktorých nastáva delenie nulou, alebo je vo funkcií logaritmus ako argument záporne číslo. Funkciu zderivujeme a pomocou bodov v ktorých je derivácia nulová určíme úseky v ktorých je funkcia rastúca a kde klesajúca. Zároveň určíme lokálne extrémy. Podobným spôsobom postupujeme pri druhej derivácií, na základe ktorej určíme úseky kde je funkcia konvexná a kde konkávna. Na základe týchto údajov vykreslíme graf, pričom hodnoty v bodoch kde bola nulová prvá derivácia nulová a jedná sa teda o prípadné lokálne extrému nakreslíme presne. Zvyšok grafu doplníme podľa zvyšných údajov, ktoré sme zistili.

Oproti tomu kreslenie na počítači rieši problém iným smerom. Konkrétne hrubou silou. Program totiž vyráta hodnoty v obrovskom počte bodov a podľa ich hodnoty nakreslí príslušný graf. Rýchlosť vykreslenia a presnosť grafu závisia na počte vypočítaných bodov.

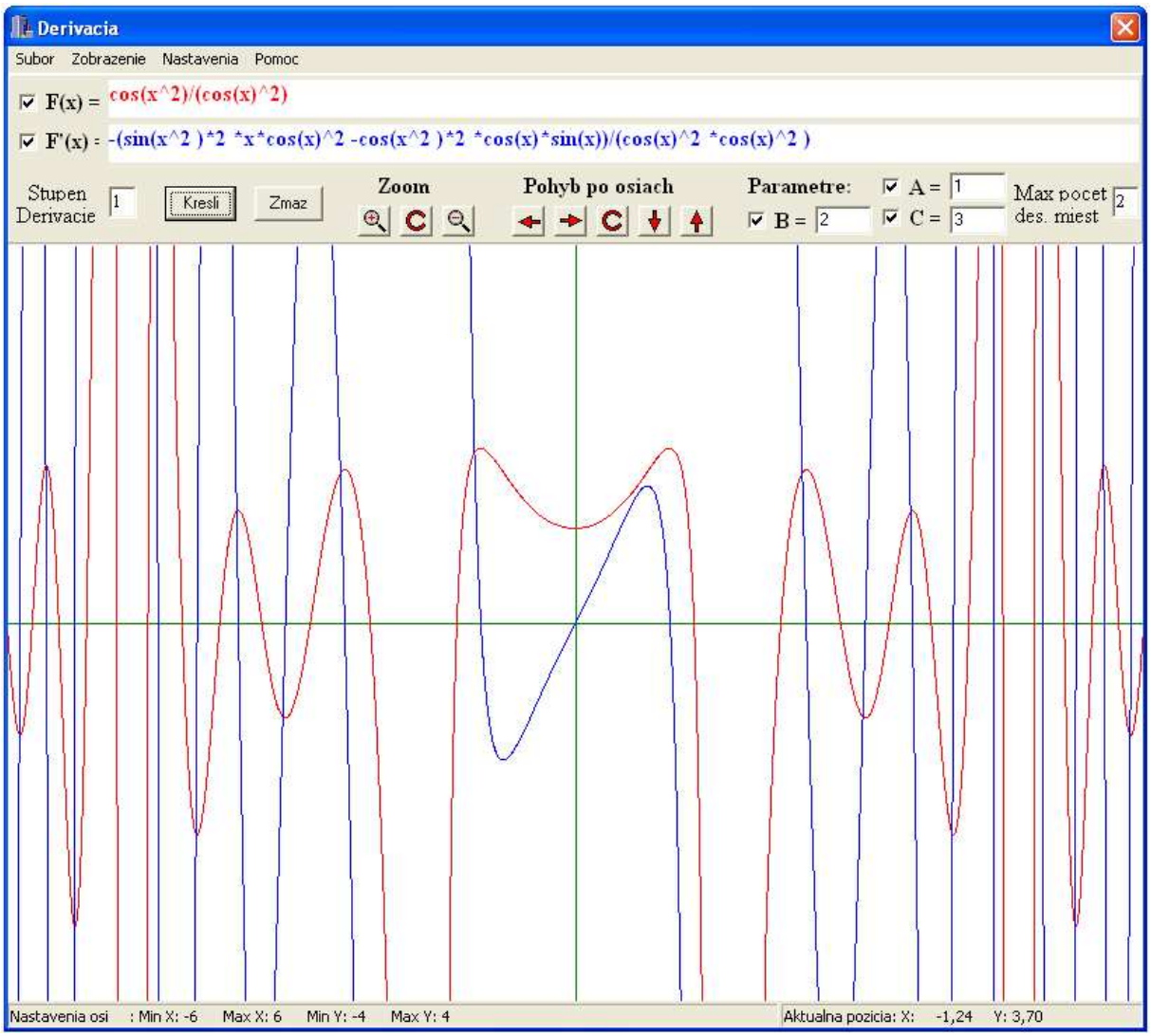
Čím vyššia presnosť tým presnejší graf, ale pomalší výpočet. Pri príliš malej presnosti sa v programe môže stať, že program, ktorý mal byť pôvodne spojitý, sa vykreslí ako zhluk bodiek, ktoré graf pripomínajú. Ak je na druhej strane presnosť príliš veľká, výpočet trvá neúmerne dlho. Tento problém je v jednotlivých programoch riešený rôzne. Niektoré programy nechajú na užívateľovi aby si sám určil presnosť vykresľovania, iné túto presnosť vyratávajú automaticky podľa aktuálneho rozdielu ypsylonovej osi posledných dvoch vykresľovaných bodov. Ak je vzdialenosť príliš veľká, program vyráta bližší bod. Podrobnejšie sa k metódam na rýchle vykresľovanie grafov na počítači vrátime v jednej z nasledujúcich kapitol.

Ďalšou nezanedbateľnou výhodou kreslenia grafov na počítači je, že všetky aspoň troška rozumné programy nám povolia zvoliť si aký výsek grafu chceme zobraziť a to do konca pre každú súradnicovú os si môžeme nastaviť vlastné hodnoty. Ako samozrejmosť môžeme v programoch možnosť graf ľubovoľne približovať alebo vzďaľovať.

Programov na kreslenie grafov je pomerne veľké množstvo a záleží na užívateľovi pre ktorý sa rozhodne. Obecne sa však dá povedať, že čím a komplexnejší program používame, tým náročnejšie a čím viac funkcií program zvláda, tým komplikovanejšie je jeho ovládanie.

6 Popis programu Derivácia

V tejto časti si bližšie predstavíme program Derivácia. Popíšeme si funkciu jednotlivých tlačítok, menu a nezabudneme sa ani pozrieť na možnosti nastavení. Najskôr ale ukážka samotného programu.



Ako je vidieť na obrázku, program sa delí na dve hlavné časti. A to časť grafickú, v ktorej sú vykreslené samotné grafy a časť ovládaciu, v ktorej sa nachádzajú ovládacie prvky programu. Pri popise programu začneme zoznamom podporovaných funkcií.

Podporovane funkcie:

unárne: sin, cos, tg, cotg, log, exp, unárne plus, unárne mínus

binárne: sčítanie, odčítanie, násobenie, delenie, umocnenie

Popis jednotlivých tlačítok:

Kresli: Funkciu môžeme čiastočne ovplyvniť nastaveniami programu, ktorým sa budeme venovať neskôr. Niektoré akcie však vykoná bez ohľadu na nastavenia. Hneď tou prvou je kontrola vstupných údajov. Tá pozostáva jednak z kontroly syntaktickej správnosti vstupného výrazu/funkcie, ďalej sú kontrolované hodnoty u stupňa derivácie a počtu desatinných miest, ktoré majú byť vo výsledku zobrazené. V prípade, že nebudú obsahovať kladné prirodzené čísla, program ohlási chybu. Ak užívateľ povolí používanie parametrov, je kontrolovaná aj ich hodnota. V tomto prípade sú to reálne čísla. V prípade, že Vám program hlási chybná zadanie a neviete nájsť chybu, uistite sa, že používate oddeľovač desatinných miest, ktorý máte nastavený vo vašom operačnom systéme. U anglickej verzie túto funkciu väčšinou plní kláves "." u českej a slovenskej pre zmenu kláves",". Po úspešnej kontrole program prevedie zadaný výraz do postfixového tvaru, kde ho zderivuje (v prípade, že výraz neobsahuje žiadnu premennú x, výsledok derivácie bude vždy nulový), výraz zjednoduší podľa a nastavení programu nakreslí grafy pôvodnej funkcie a jej derivácie. Ktoré grafy budú vykreslene, môžete ovplyvniť buď pomocou zaškrtávacích boxov na ľavo od zadania funkcie, alebo v menu Zobrazenia. K dispozíciu sú i klávesové skratky. V tejto verzií program počíta deriváciu bez ohľadu na nastavenie. Možnosť nastavenia, bude k dispozícií v niektorej z ďalších verzií programu.

Zmaž: Zmaže plochu, na ktorú sa vykresľujú grafy funkcií. Vo väčšine prípadov to pravdepodobne zmaže iba posledný graf. Výnimku tvorí situácia keď v menu alebo pomocou klávesovej skratky zakážete automatické mazanie. Ak je zobrazená časť grafu pri stredu súradníc a túto možnosť nezakážete, prekreslia sa súradnicové osi.

Zaškrtávacie tlačítka: Pomocou nich môžete nastaviť, či bude program povoľovať použitie parametrov. Ak ich povolíte, musia obsahovať reálne hodnoty. V prípade, že použitie parametrov zakážete a napriek tomu ich použijete v zadaní funkcie, program zahlási

syntaktickú chybu. Pomocou zaškrtávacích tlačítok môžeme ešte nastaviť, či sa budú vykresľovať graf pôvodnej funkcie, či graf jej derivácie.

Tlačítka pre pohyb po osiach a centrovanie: ako názov naznačuje, tieto tlačítka slúžia pre pohyb po grafe. Sú rozdelené na dve na dve skupiny. Prvá je určená pre pohyb po grafe. Ako náhradu za ne môžete používať numerickú klávesnicu. Hodnotu o ktorú sa výsek grafu posunie závisí na práve zobrazenom rozsahu, presnejšie na rozdiele najvyššej a najnižšej hodnoty zobrazenej na grafe. Druhá sada slúži na vzďaľovanie a približovanie grafu, pomocou ktorých si môžete zobraziť väčšiu časť grafu alebo naopak detailnejšie zobraziť nejakú jeho zaujímavú časť. K dispozícií sú upäť klávesové skratky. V tomto prípade kláves plus pre priblíženie, mínus pre vzdialenie a kláves „*“ pre reset nastavenia na štandardnú hodnotu. Obe sady sú navzájom nezávislé, takže napríklad reset jednej sady nijakým spôsobom neovplyvní tú druhú. Spomínané skratky sú uvedené i v menu Pomoc/Tipy a triky. Okrem tlačítok a klávesových skratiek môžeme zobrazený rozsah nastaviť aj pomocou menu Nastavenia. Táto možnosť je vhodná predovšetkým vtedy, ak chceme zmeniť pomer jednotlivých strán grafu alebo výrazne zmeniť veľkosť zobrazeného výseku.

V pravej dolnej časti okna je zobrazená aktuálna pozícia kurzoru myši na grafe. Presnosť jej zobrazenie, teda maximálny počet desatinných miest ktorý program zobrazí môžete nastaviť v pravej hornej časti grafu. Vľavo dole je pre zmenu zobrazený aktuálny rozsah grafu. Posledné nastavenie, ktorým môžete ovplyvniť chod programu mimo menu, je nastavenie stupňa derivácie. Čo je to stupeň derivácie, sa môžete dozvedieť v časti venovanej derivácií funkcie. Ostáva ešte opísať možnosti nastavení a funkcií, ktoré poskytuje menu programu.

Popis menu. Väčšina nastavení je duplicitná k tým ktoré môžete nájsť v hlavnom okne programu. Menu však obsahuje i niektoré nastavenia, ku ktorým sa inak ako cez menu alebo klávesovou skratkou nedostanete.

Súbor:
Uložiť: Uloží obrázok grafu funkcie do aktuálneho adresára do súboru typu .bmp
Uložiť ako: Podobne ako uložiť akurát užívateľ si sám zvoli meno súboru a kam súbor uložiť
Koniec: Ukončí program
Zobrazenie: Rôzne nastavenia a možnosti zobrazenia

Graf Funkcie: Určuje čí sa ma vykresliť graf pôvodnej funkcie

Graf derivácie: Určuje čí sa ma vykresliť graf zderivovanej funkcie

Súradnicové osi: Ak je zapnuté, budú sa vykresľovať súradnicové osi. Inak nie.

Automatické mazanie grafu: Ak zapnuté, pri každom novom vykreslení grafu sa starý zmaže.

Úprava zadaného výrazu: Pokúša sa o zjednodušenie zadanej funkcie, ak je zapnuté

Nastavenia:

Nastavenia osí: Rozsah grafu ktorý bude momentálne vykreslený

Nastavenia farieb: Nastavenie farby grafov pôvodnej funkcie a zderivovanej funkcie, farbu súradnicových osí a pozadia

Pomoc:

Tipy a Triky: Výpis podporovaných funkcii, preddefinovane konštanty, Klávesové skratky pre pohyb po osiach

O Programe: Základne informácie o programe a autorovi

7 Popis funkcií požitých v programe Derivácia

V tejto časti sa zameriame hlavne na rôzne špeciality algoritmov použitých v tomto programe. Kvôli kompletnosti uvediem algoritmy kompletne. Pre čitateľov, ktorí majú záujem skôr o všeobecné postupy používané pri navrhovaní algoritmov odporúčam skôr predchádzajúce kapitoly.

Poznámka k reprezentácií výrazu v postfixovom tvare: V kapitole venovanej reprezentácií aritmetického výrazu sme mali možnosť vidieť algoritmy, ktoré používajú prioritu operátorov. Z tohto dôvodu sú operátory, funkcie a zátvorky v postfixe reprezentované pomocou špeciálne definovaných konštánt s názvom odpovedajúcim danému operátoru či funkcií a ich hodnota, (dvojciferné číslo) reprezentuje ich prioritu. Čím vyššie číslo reprezentuje, tým vyššia je priorita. Toto priradenie je prevedené v špeciálnej funkcií „Dalsi“, ktorá má na starosti najbližší operand, alebo operátor zo vstupného infixového výrazu. Zároveň kontroluje, či vstupný výraz neobsahuje nejaký preklep alebo nepovolený parameter a výraz nieje syntaktický nesprávny.

Popis hlavných funkcii:

Kresli: Pomocou tejto funkcie program prevedie zadanú funkciu do jej postfixového tvaru, zderivuje, zjednoduší a prevedie späť do tvaru infixového. Pomocou funkcie Nakresli vykreslí obe funkcie. Funkcia kontroluje správnosť zadania vstupného výrazu, a to vrátane správnosti uzátvorkovania, hodnoty parametrov(čí obsahujú číslo), stupeň derivácie (čí obsahuje prirodzene čísla) a počet zobraziteľných desatinných miest (čí sa jedna o prirodzene číslo menšie ako šesť (tj. max počet desatinných miest)).

InfixToPostfix: Postupne pomocou funkcie „Dalsi“ načíta jednotlivé prvky v infixe. V prípade, že narazí na číslo, parameter alebo premennú, okamžité ju prevedie do postfixu. V prípade, že načíta operátor, tak zo zásobníka vyberie všetky operátory s rovnakou a vyššou prioritou a pridá ich na koniec postfixového výrazu. nakoniec samotný operátor vloží na vrch zásobníku. V prípade ľavej zátvorky ju iba pridá na vrch zásobníku. Ak načíta pravú zátvorku vyhadzuje všetky operátory zo zásobníku, až narazí na ľavú zátvorku. Pre prípad vyprázdnenia zásobníku je na spodku zásobníku bezpečnostná zarážka, ktorá zabraní čítaniu z prázdneho zásobníku. Ak ľavú zátvorku nenájde, výraz je zle uzátvorkovaný a tým pádom

syntakticky nesprávne. Keď dočíta vstup, presunie všetky operátory zo zásobníku do postfixu. Na konci algoritmu musí byť v zásobníku iba jeden výraz. Ak obsahuje ešte nejaké zátvorky, vyraz je zle uzátvorkovaný. Oproti algoritmu popísanému v časti venovanej reprezentácií aritmetického výrazu, obsahuje viac druhov kontroly zadaného aritmetického výrazu. Algoritmus kontroluje, či boli povolené parametre, nepovoľuje použitie binárneho operátora hneď za ľavou zátvorkou, nepovoľuje ani dva binárne operátory za sebou, okrem operátorov plus a mínus, ktoré v tomto prípade považuje za unárne. Rovnako povolí umiestnenie unárneho operátoru hneď za operátorom binárnym. Vďaka tejto funkcií nieje nutné používať zátvorky u výrazov ktoré začínajú unárnym mínus, alebo plus.

Derivation: Symbolicky (tabuľkovo) zderivuje postfixový výraz. Nekontroluje správnosť vstupu a nezjednodušuje výraz.
Napríklad výraz: 2 x * sin (sin(2*x) zderivuje na: 2 x * cos 0 x * 2 1 * + * (cos(2*x)*2)

PostfixToInfix: Prevedie postfix na infix. Takisto nekontroluje vstup. Algoritmus postupuje opačne ako prevod na postfix. Čísla dáva na zásobník a ak narazí na operátor vyberie jeden alebo dve čísla (parametre, premennú) zo zásobníku a prevedie do infixu. Poprípade uzátvorkuje. V tomto prípade algoritmus neobsahuje žiadne špeciality oproti algoritmu popísanému v kapitole 2.

Zjednodušenie Postfixu: Najkomplikovanejšia funkcia v programe. Pokúša sa o jednoduché zjednodušenie výrazu. Používa zásobník. Pre každý vyraz na zásobníku si pamätá hodnotu, či je to číslo alebo výraz (napr. sin) a znamienko výrazu (buď kladne alebo záporné). Zjednodušenia jednotlivých funkcii:
Sčítanie, Odčítanie: ak sú oba výrazy čísla, vyhodnotí ich
Vynímanie mínusu pri sčítaní a odčítaní: v prípade že druhý výraz je záporný, zmení znamienko operácie na mínus a znamienko druhého výrazu zmení na kladne. Ak boli oba výrazy rovnaké a líšili sa iba znamienkami výsledok je nula. Pri pripočítavaní nuly je výsledkom druhy vyraz. nevie zatiaľ spočítať dva nenulové výrazy.
Násobenie, Delenie: Ak sú oba výrazy čísla, vyhodnotí ich, ak je jedno nulové výsledok je nula, ak je jedno jednotka, výsledok je druhé, ďalej práca so znamienkami jednotlivých výrazov. Násobenie a delenie sa v jednotlivých detailoch líšia.

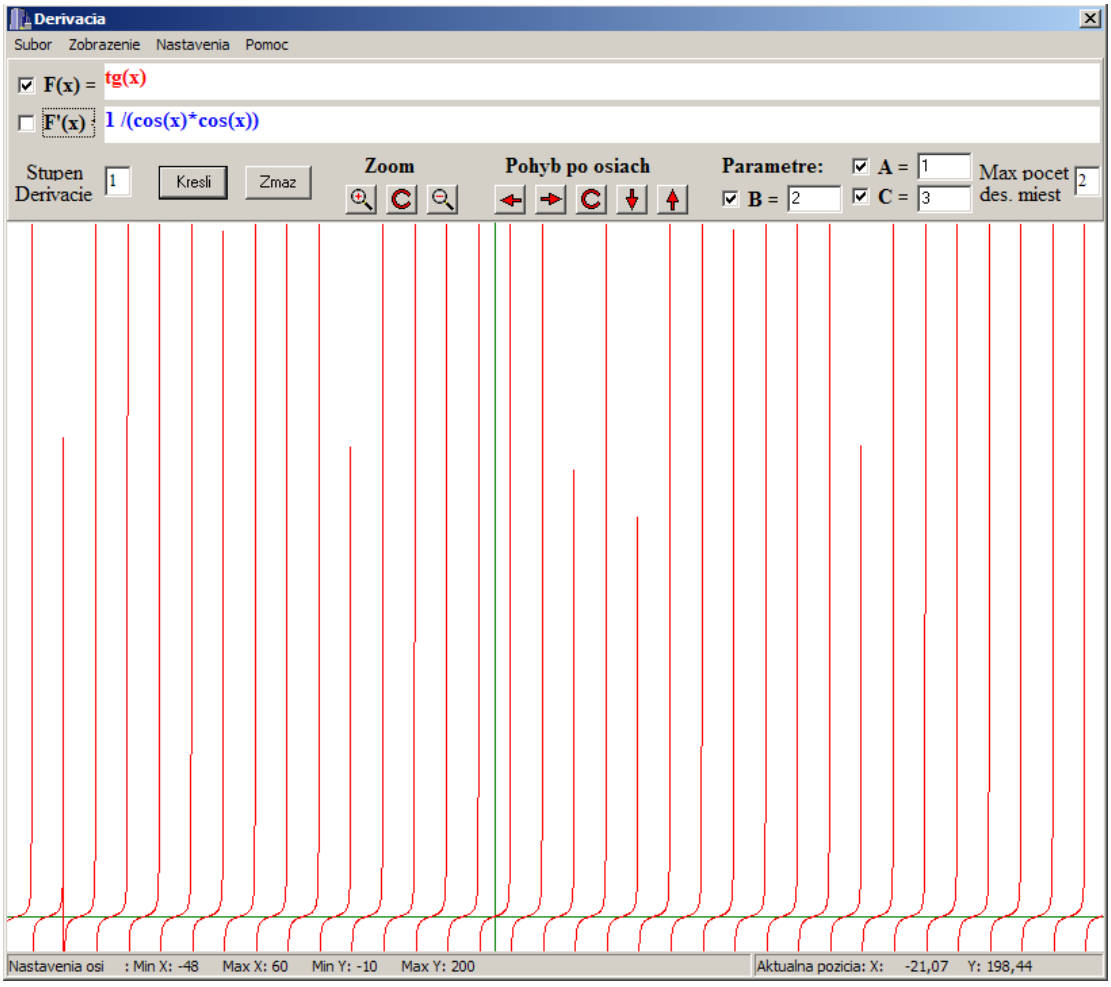
Ostatne funkcie: základne operácie. Napr. sin(-x) =-sin(x) atď.

Vyhodnotenie Postfixu: Vyhodnotí postfixový vyraz. Obsahuje ochranu proti deleniu nulou, pretečenie a pod.

Nakresli: Nakreslí graf funkcie zo zadaného postfixového výrazu. Funkcia nekontroluje syntaktickú správnosť výrazu. To je možné vďaka tomu, že vstup tejto funkcie je zároveň výstup iných časti programu, a je teda zaručené, že vstupná funkcia je syntakticky správna, Funkcia používa pokročilé algoritmy na vykresľovanie. Funkcia ráta hodnoty iba v určitých konkrétnych bodoch. Začína na spodnej časti zobrazeného intervalu a po intervaloch (Epsilon) sa stále posúvame smerom k maximálnej zobrazenej hodnote. Obe hodnoty X a Y sú následné prevedene z reálnej osi na os grafu, v ktorej je napríklad obrátená os Y (nula sa nachádza v hornej časti grafu). Interval, o ktorý sa zakaždým zvýši vykresľovaný bod, sa dynamicky mení podľa hodnoty vykresľovaných bodov.

Funkcia najskôr zistí hodnotu zadanej funkcie v určitom bode a podľa výslednej hodnoty a hodnoty v predchádzajúcom bode pokračuje ďalej. Ak boli oba body zobrazené v zobrazenom výseku grafu a zároveň od seba neboli príliš vzdialene, vykreslí sa medzi nimi čiara. Ak boli od seba príliš vzdialene, vyráta sa ďalší bod v strede medzi nimi a hodnota Epsilon sa zmenší na polovicu. Ak boli body zobrazene na rovnaké miesto, hodnota Epsilon sa zdvojnásobí a vyráta sa ďalší bod. Vďaka tomuto postupu sú zároveň funkcie vykreslené spojito, presne a hlavne veľmi rýchlo oproti pevnému Epsilon a vykresľovaniu iba pomocou bodov. Funkcia takisto rozoznáva rôzne prípady, keď sú jednotlivé body zobrazene pod alebo nad vykreslený úsek, prípadne funkcia v bode nie je definovaná. Každý prípad je samostatne ošetrený a je vykonávaná potrebná akcia. Napríklad ak sa hodnota (Y) predchádzajúceho bodu zobrazila pod rozsah a hodnota súčastného bodu v rozsahu a hodnota (X) je rovnaká, nakresli sa čiara zo spodku až po aktuálnu hodnotu (Y). Problém nastáva pri extrémne dlhých výrazoch, kde samotne vyhodnotenie trvá neúmerné dlho (radovo niekoľko desatín sekundy) čo pri vykreslení niekoľkých tisícok bodov na grafe môže spôsobiť, že celkove vykreslenie grafu je neúmerne dlhé. Ďalší problém nastáva pri funkciách, ktoré rastú/klesajú príliš prudko. Môže totiž nastať prípad, že jeden bod sa má zobraziť pod rozsah a ďalší bod nad. To sa čiastočne dá riešiť zmenšením Epsilonu a vyrátaním nového bodu, ale pri príliš prudkých funkciách ani to nepomáha. Po čase je totiž Epsilon natoľko malé, že samotný typ float s nim nie je schopný pracovať a neustálym delením sa Epsilon časom úplne vynuluje a program by

sa zacyklil. V tomto prípade musíme obnoviť Epsilon a čiaru nekresliť. Keby sme ju vykreslili mohli by nastať nežiadúce efekty ako napríklad u tg(x) alebo $\frac{1}{x}$. U funkcií podobných funkcií tg (x) nastáva ešte jeden problém. Ak program zobrazuje výsek grafu s vysokým hodnotami osi (Y), stáva sa, že predchádzajúci bod je zobrazený v zobrazenom úseku, ale nasledujúci bod je zobrazený pod intervalom. Opäť sa tento problém dá čiastočne vyriešiť zmenšením hodnoty Epsilon a teda vyrátaním bližšieho bodu. Ani tento postup nie je vždy efektívny a výsledok je označenie bodu za bod nespojitosti. Výsledný efekt na grafe je taký, že je zobrazený nekompletný graf, respektíve uťatá čiara. A to napriek tomu, že je zrejmé, že graf mal pokračovať ďalej. Dôvod, prečo tento graf nevykresliť do konca je rovnaký ako u predchádzajúceho problému a to ten, že nespojité funkcie by boli vykreslené nesprávne. U periodických funkcií nastáva ešte jeden problém. Posledný vykreslený bod v jednotlivých periódach nemá vždy rovnakú funkčnú hodnotu. To má za následok to, že čiara grafu nielenže nie je vykreslená celé, ale v rôznych periódach je rôzne vysoká. Tento problém je viditeľný najmä pri zobrazení vysokých hodnôt osi (Y). U štandartných a hlavne spojitých funkcii ale väčšinou problémy nenastávajú.

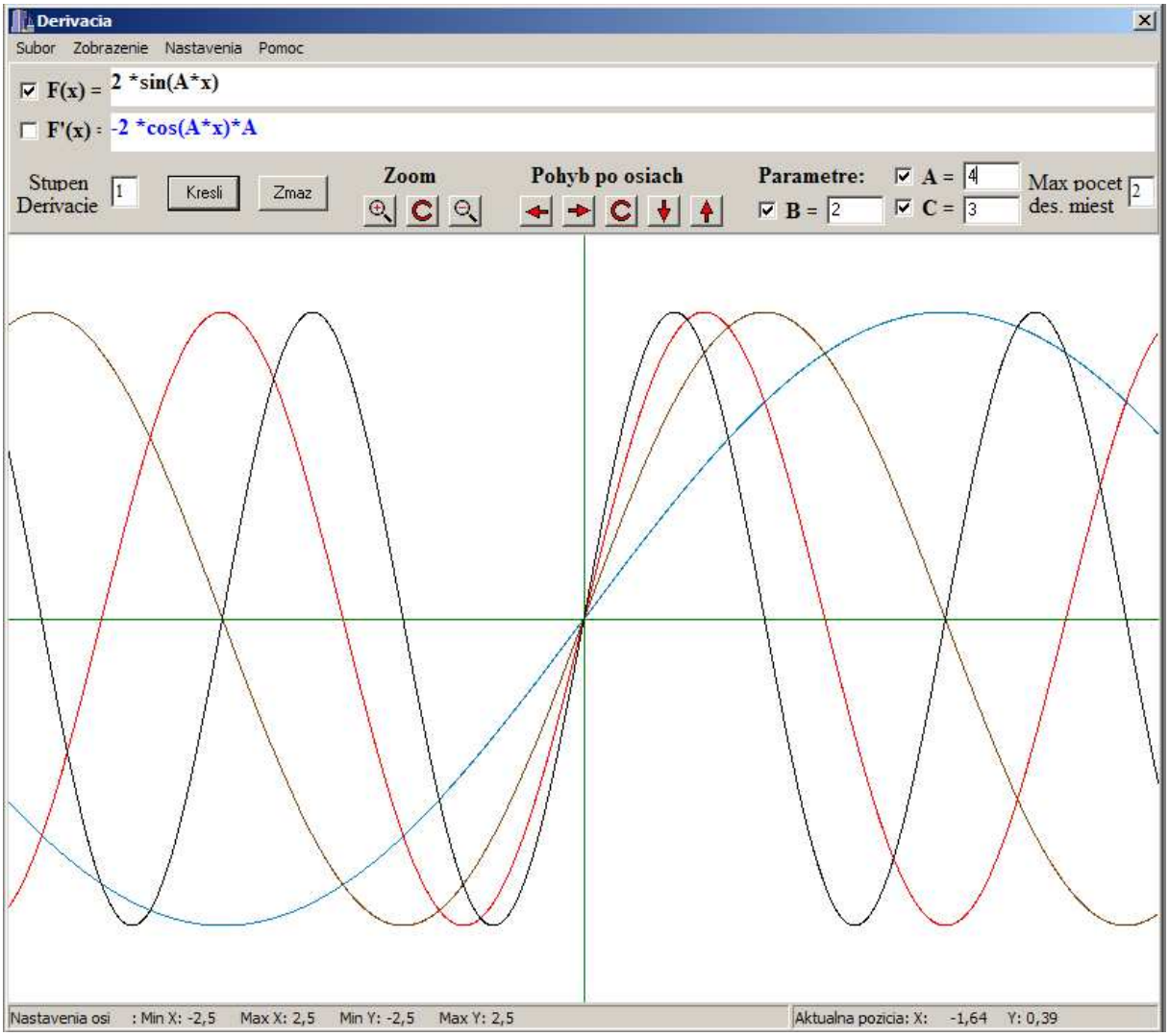


Zvyšné funkcie: Nastavenie Osi (nastavenie rozsahu zobrazenej funkcie), Nastavenie Farieb, O programe, Tipy a triky.

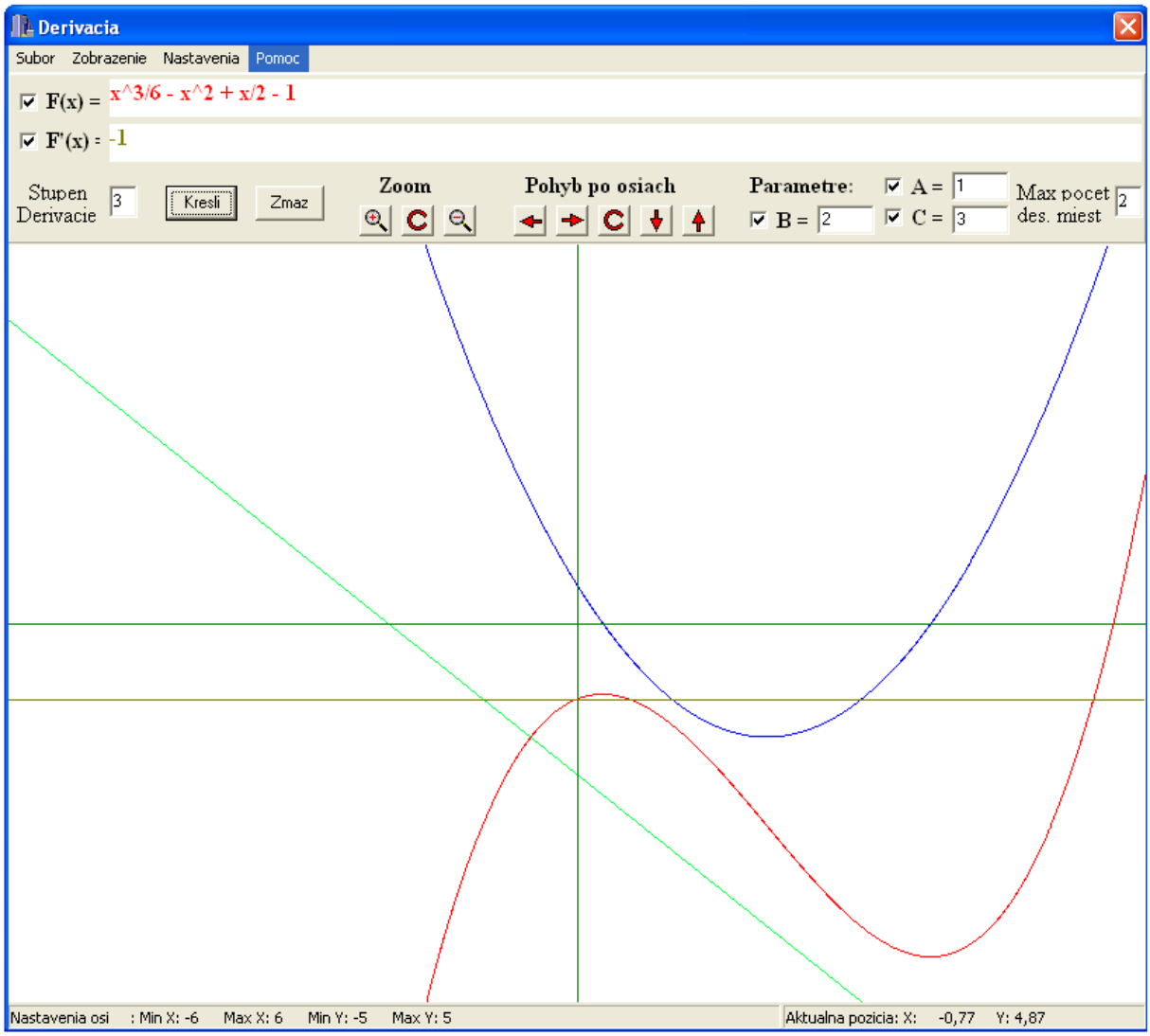
Program nepridelʼuje pamäť dynamicky ale staticky na začiatku programu, a to z toho dôvodu, že pri funkciách s radovo tisíckami výrazov je vyhodnotenie výrazu natoľko dlhé že pri kreslení niekoľkých Tisovcov bodov je dĺžka kreslenia grafu funkcie neuveriteľné dlha a program je nepoužiteľný. Niekoľko tisíc výrazov však možno dosiahnuť iba pri výpočte vysokého stupňa derivácie veľmi komplikovaných funkcii (napr. f(x)^g(x))

8 Ďalšie možnosti programu

Vďaka možnosti programu vypnúť automatické mazanie program umožňuje viaceré na prvý pohľad nenápadné možnosti použitia. Aby sme sa mohli dopracovať k výsledku, ktorý je vidieť na nasledujúcom obrázku, musíme najprv zrušiť automatické mazanie grafu pri jeho kreslení. Je to z toho dôvodu, že program povoľuje kresliť naraz iba jednu funkciu. Ak chceme jednotlivé grafy oddeliť i farebne, musíme zakaždým nastaviť novú farbu.



Na tomto príklade je vidieť možnosť použitia vykreslenia jednej funkcie avšak s rôznymi hodnotami parametrov.



Na ďalšom príklade je vidieť možnosť použitia vykreslenia jednej funkcie spolu so všetkými troma stupňami jej derivácie. Rovnako ako v predchádzajúcom prípade musíme kresliť každú deriváciu osobitne.

Veľmi užitočná je rovnako možnosť zobrazenia viacerých funkcií naraz. Táto alternatíva je výhodná hlavne ak chceme porovnať viaceré funkcie. Jednotlivé triky možeme medzi sebou rôzne kombinovať a napríklad kresliť naraz viacero funkcií spolu s ich deriváciami.

Nevýhoda tohto postupu je, že ak chceme zmeniť plochu zobrazeného grafu, musíme zadávať všetky funkcie odznova.

9 Použitá literatúra

1) P.Töpfer: Algoritmy a programovací techniky, Prometheus Praha 1995

2) Ladislav Bican: Lineární algebra a geometrie, Academia, Praha, 2000

3) Vojtěch Jarník: Diferenciální počet I, Academia, Praha, 1974

4) Jana Hoderová, Derivace, Brno, 2005