Masarykova univerzita
Přírodovědecká fakulta

Systém počítačové algebry Maxima

BAKALÁŘSKÁ PRÁCE

Zdena Trefilíková

Vedoucí práce: **RNDr. Roman Plch, Ph.D.**
Studijní program: **Matematika**
Studijní obor: **Matematika se zaměřením na vzdělávání**

Brno 2011

Masarykova univerzita

Přírodovědecká fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Zdena Trefilíková**

Studijní program - obor: **Matematika - Matematika se zaměřením na vzdělávání**

Ředitel Ústavu matematiky a statistiky PřF MU Vám ve smyslu Studijního a zkušebního řádu MU určuje bakalářskou práci s tématem:

Systém počítačové algebry Maxima

Computer Alegbra System Maxima

Oficiální zadání: Popište instalaci a použití volně šiřitelného systému počítačové algebry Maxima. Zaměřte se zejména na využití systému při výuce matematické analýzy.

Literatura:

Plch, Roman - Došlá, Zuzana - Sojka, Petr. Matematická analýza s programem Maple. Díl 1, Diferenciální počet funkcí více proměnných. prvni. Brno : Masarykova Universita, 1999. 80 s. ISBN 80-210-2203-5.

Vedoucí bakalářské práce: RNDr. Roman Plch, Ph.D.

Datum zadání bakalářské práce: červen 2009

Datum odevzdání bakalářské práce: dle harmonogramu ak. roku 2009/2010

V Brně dne 2.11.2009

v.z. prof. RNDr. Jiří Rosický, DrSc.
Ředitel Ústavu matematiky a statistiky

Zadání bakalářské práce převzal dne: 23.11.2009

Trefilíková
Podpis studenta

Ráda bych poděkovala RNDr. Romanu Plchovi, Ph.D. za vedení bakalářské práce, ochotu ke konzultacím, trpělivost a cenné rady při zpracování daného tématu.

Prohlašuji, že jsem svou bakalářskou práci napsala samostatně a výhradně s použitím citovaných pramenů.

V Brně, dne 4.1.2011 Zdena Trefilíková

Název práce: Systém počítačové algebry Maxima
Autor: Zdena Trefilíková
Ústav matematiky a statistiky Přírodovědecké fakulty, MU
Vedoucí bakalářské práce: RNDr. Roman Plch, Ph.D.
Abstrakt: Práce se zabývá systémem počítačové algebry Maxima a představuje stručný manuál k tomuto programu určený především studentům matematiky. Úvodní kapitoly popisují instalaci programu a jeho základní ovládání, poté se již práce zaměřuje na jednotlivé příkazy týkající se matematických výrazů a funkcí, matic, řešení soustav rovnic a tvorby grafů. Poslední kapitola se věnuje využití programu při výuce matematické analýzy, přičemž je kladen důraz zejména na sekci o vyšetřování průběhu funkce, v níž jsou aplikovány příkazy z předcházejících kapitol. Ke každému příkazu je vždy uveden krátký komentář a alespoň jeden příklad. Práce také poukazuje na některé problémy, s nimiž se uživatel programu může setkat, a navrhuje jejich řešení.
Klíčová slova: Maxima, systém počítačové algebry, matematická analýza, průběh funkce

Title: Computer Algebra System Maxima
Author: Zdena Trefilíková
Department of Mathematics and Statistics, Faculty of Science, MU
Supervisor: RNDr. Roman Plch, Ph.D.
Abstract: The thesis treats of the computer algebra system Maxima and represents a brief manual, primarily intended for students of mathematics. The introductory chapters describe the installation and basic control of the program, then the thesis focuses on particular commands related to mathematical expressions and functions, matrices, equation system solving and graph plotting. The last chapter deals with the use of the program in mathematical analysis. The emphasis is placed on the section about investigation of the course of function, in which commands from previous chapters are applied. A short commentary and at least one example are introduced to every command. The thesis also points out some of the problems that the user can come across and proposes their solutions.
Keywords: Maxima, computer algebra system, mathematical analysis, course of function

Obsah

Úvod

Systém počítačové algebry je software, který svému uživateli umožňuje řešit velké množství matematických úloh. Dokáže pracovat s reálnými i komplexními čísly a umí během okamžiku vyřešit i obtížnější úlohy jako jsou integrály, řešení soustavy rovnic nebo násobení matic.

První systémy počítačové algebry se objevují již v 60. letech 20. století. V roce 1963 představil Martinus J. G. Veltman, laureát Nobelovy ceny za fyziku, svůj program Schoonschip. Jen o rok později se na scéně objevuje Mathlab. V dalších letech spatří světlo světa také Mumath, Derive a Macsyma, předchůdce dnešního systému Maxima.

V dnešní době je na trhu řada systémů počítačové algebry. Některé z nich jsou komerční programy, jako například Maple nebo Mathematika, jiné jsou volně dostupné (tzv. open source programy), mezi které patří například Axiom, Sage a také systém Maxima, kterým se tato práce zabývá.

Systém Macsyma byl vytvořen v průběhu let 1968 až 1982 jako součást projektu MAC v MIT (Massachusetts Institute of Technology). V roce 1982 byl zdrojový kód systému Maxima předán Oddělení energie (Department of Energy). Tato verze je známa jako DOE Macsyma. Program byl poté udržován profesorem Williamem F. Schelterem z univerzity v Texasu, a to až do jeho smrti v roce 2001. V roce 1998 získal Schelter souhlas ke zveřejnění zdrojového kódu programu DOE MacSyma pod veřejnou licencí GNU a v roce 2000 inicializoval projekt Maxima na SourceForge, aby se program DOE Macsyma mohl nadále udržovat a vylepšovat pod svým novým názvem Maxima. Od té doby prochází program pravidelnými aktualizacemi. Nynější nejnovější verze je Maxima 5.22.1., která vyšla 13. srpna 2010. Největší výhoda této nové verze je, že její grafická nadstavba wxMaxima byla částečně přeložena do češtiny, což významně usnadní práci českému uživateli. Všechny potřebné informace o programu Maxima, jeho instalaci a aktualizacích se nacházejí na oficiálních internetových stránkách programu `http://maxima.sourceforge.net`.

Kapitola 1

První kroky

1.1 Stažení a instalace

Všechny informace potřebné k instalaci programu Maxima najdeme v sekci Download jeho oficiálních webových stránek. Zde si můžeme tyto soubory také stáhnout.

Instalace na OS Windows

Pro instalaci na OS Windows stáhneme soubor `maxima-x.y.z.exe`, kde x.y.z. je verze programu. Jakmile soubor spustíme, objeví se klasické instalační okno. Po klepnutí na tlačítko Next a akceptování licenčního ujednání můžeme začít s nastavením parametrů pro instalaci. Je možné vybrat úplnou instalaci, kompaktní instalaci ke spouštění programu pouze v příkazové řádce nebo vlastní instalaci, kde zaškrtneme ty komponenty, které chceme instalovat. Jednou z těchto komponent je také grafická nadstavba wxMaxima, kterou doporučuji nainstalovat, neboť s ní budeme pracovat. Pro úplnou instalaci je nutné mít cca 92 MB volného místa na disku. Nakonec zvolíme, kam chceme umístit zástupce programu, popř. vytvoříme ikonu na ploše. V dalším kroku již vidíme všechny informace o instalaci. V tomto okamžiku se ještě pořád můžeme vrátit kliknutím na Back a změnit nastavení. Pokud jsme s nastavením spokojeni, klikneme na Install a spustí se samotná instalace. Poté se zobrazí soubor Readme. Naposledy klikneme na Next a instalaci ukončíme klepnutím na tlačítko Finish.

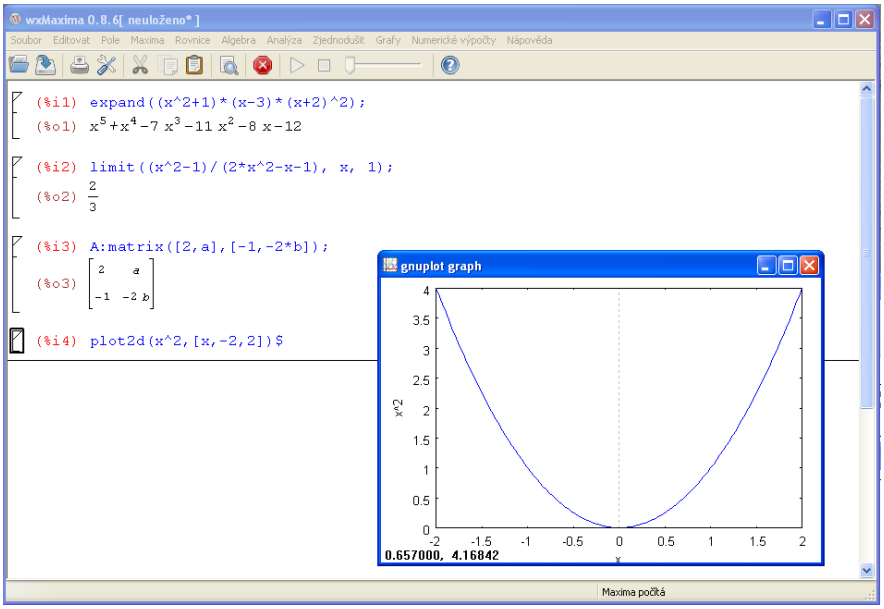
Instalace na OS Linux

Existuje několik možností, jak nainstalovat program Maxima na Linux, my si však ukážeme jen jednu z nich. Informace o dalších možnostech instalace najdeme na oficiálních webových stránkách systému Maxima. Nejprve stáhneme soubor `maxima-x.y.z.tar.gz`. Jestliže máme operační systém postavený na Debianu, po instalaci bychom nemohli spouštět grafickou nadstavbu wxMaxima.

Tento problém vyřešíme například tak, že si ještě před samotnou instalací programu nainstalujeme balík `cmucl` příkazem `sudo apt-get install cmucl` a poté program Maxima s tímto balíkem zkompilujeme. Instalační soubor programu nejprve rozbalíme příkazem `tar xzf maxima-x.y.z.tar.gz` a přejdeme do složky programu příkazem `cd maxima-x.y.z`. Program zkompilujeme s balíkem `cmucl` příkazem `./configure --enable-cmucl`. Nyní program nainstalujeme pomocí příkazu `make install` a nakonec nainstalujeme grafickou nadstavbu programu příkazem `apt-get install wxmaxima`.

1.2 Spuštění programu

Program Maxima lze spustit několika různými způsoby. Nejjednodušší možností je poklepání na ikonu `XMaxima` nebo `wxMaxima`. Program můžeme také spouštět z příkazové řádky, a to příkazy `maxima`, `xmaxima` nebo `wxmaxima`. Příkaz `maxima` spustí program v rámci příkazové řádky, `xmaxima` v samostatném okně a `wxmaxima` spustí grafickou nadstavbu programu. Ať použijeme kterýkoliv příkaz, spustíme tentýž program se stejnými funkcemi. Rozdíl uvidíme pouze v grafickém rozhraní. Grafická nadstavba wxMaxima nabízí velmi příjemné grafické prostředí a její velkou výhodou je její lokalizace do češtiny, i když ne úplná. Zatím můžeme využít české hlavní menu, které nám významně ulehčí práci, avšak na kompletní překlad programu nebo český manuál si budeme muset ještě počkat. Nadále bude tato práce používat právě grafickou nadstavbu wxMaxima.



Obrázek 1.1: Grafická nadstavba wxMaxima

Na Ústavu matematiky a statistiky Přírodovědecké fakulty Masarykovy Univerzity je systém Maxima nainstalován na serveru Bart, odkud se spouští příkazy `maxima`, `xmaxima` nebo `wxmaxima`.

1.3 Pravidla pro zadávání příkazů

Abychom mohli začít pracovat s programem Maxima, je nutné nejdříve uvést několik základních pravidel pro zadávání příkazů. Příkaz standardně ukončujeme středníkem. Aby byl příkaz následně proveden, stiskneme klávesy Shift+Enter.

```
(%i1) sqrt(4);
```

(%o1) 2

Pokud stiskneme pouze Enter, příkaz nebude proveden a program bude čekat na další příkaz. Tímto způsobem lze zadat několik příkazů současně. Po zadání posledního příkazu opět stiskneme Shift+Enter a program provede všechny příkazy současně.

```
(%i2) sqrt(4);
      sqrt(9);
```

(%o2) 2

(%o3) 3

Pokud chceme provádět příkazy pouze pomocí klávesy Enter, klikneme na možnost `Nastavení` v záložce `Editovat`. Zde zaškrtneme úplně poslední položku `Enter vyhodnocuje výraz`, čímž prohodíme funkce Enter a Shift+Enter.

Dále máme možnost provádět příkazy bez toho, aby se zobrazil výsledek. K takovému účelu slouží znak $, který napíšeme za příkaz místo středníku. Výsledek samozřejmě můžeme kdykoliv zobrazit nebo s ním dále pracovat.

```
(%i4) sqrt(4)$
```

Je tu také možnost zadaný příkaz neprovádět a pouze jej opsat. Využívá se k tomu apostrof ', který umístíme na začátek příkazu.

```
(%i5) 'limit(x^2/(x^3+2), x, inf);
```

(%o5) $$\lim_{x\to\infty} \frac{x^2}{x^3+2}$$

Maxima automaticky čísluje každý vstup i výstup. Vstupy jsou značeny `(%ix)` a výstupy `(%ox)`, kde x je jejich pořadové číslo. V dalších příkazech poté můžeme odkazovat na předešlé příkazy bez opětovného opisování. Pokud odkazujeme na poslední zadaný příkaz, používáme pouze znak %. Pro odkazování na starší příkazy musíme použít jejich označení. Obě možnosti jsou ukázány v následujícím příkladu:

```
(%i6) x^2-3;
```

(%o6) $$x^2-3$$

```
(%i7) sqrt(%);
```

(%o7) $$\sqrt{x^2-3}$$

```
(%i8) diff(%i6,x);
```

(%o8) $$2\,x$$

1.4 Ukládání, export a tisk

Po ukončení práce můžeme data v zápisníku uložit, exportovat nebo tisknout. Použijeme záložku `Soubor` hlavního menu, kde všechny tyto možnosti najdeme. Zvolíme-li položku `Uložit jako`, uložíme zápisník do `.wxm` souboru.

Soubor exportujeme pomocí položky `Export`. Exportujeme buď do formátu HTML nebo do zdrojového kódu programu TEX.

Pro tisk použijeme položku `Tisk`, kde následně vybereme, jestli chceme soubor přímo vytisknout na tiskárně nebo převést do formátu PDF či PS.

1.5 Ukončení programu

Program lze ukončit několika způsoby. První možností je ukončení pomocí příkazu `quit()`, přičemž prázdné závorky za quit jsou nutné. Pokud zadáme příkaz bez závorek, program se neukončí a pouze slovo quit opíše, jak je ukázáno níže. Tento postup použijeme, jestliže pracujeme v příkazové řádce.

```
(%i1) quit;
```

(%o1) *quit*

```
(%i2) quit();
CLIENT: Lost socket connection ...
Restart Maxima with 'Maxima->Restart Maxima'.
```

Pokud pracujeme s grafickou nadstavbou, ukončíme program v hlavním menu volbou `Ukončit` záložky `Soubor` nebo stisknutím kláves Ctrl + Q.

1.6 Nápověda

Velmi důležitou věcí nejen pro začínající uživatele je systém nápovědy, na který se zaměříme nejdříve. Vybereme-li v záložce `Nápověda` možnost `Nápověda programu Maxima` nebo stiskneme-li klávesu F1, objeví se rozsáhlý manuál programu, který obsahuje desítky kapitol s popisy příkazů a jejich příklady. Manuál najdeme také na webových stránkách Sourceforge[1]. K dispozici máme také další formy nápovědy a to příkazy `describe()`, `example()` a `apropos()`.

Příkaz `describe(příkaz)` popíše funkci zadaného příkazu.

```
(%i1) describe(float);

 -- Function: float (<expr>)  Converts integers, rational numbers
 and bigfloats in <expr> to floating point numbers.  It is also an
 'evflag', 'float' causes non-integral rational numbers and
 bigfloat numbers to be converted to floating point.
 There are also some inexact matches for 'float'.
 Try '?? float' to see them.
 (%o1) true
```

Příkaz `example(příkaz)` vypíše příklady pro zadaný příkaz. Tento příkaz je vhodné využít, pokud nevíme, jak se daný příkaz používá, např. pokud si nejsme jisti počtem a pořadím argumentů, které do příkazu dosazujeme. Ukažme si příklady pro příkaz `coeff`.

```
(%i2) example(coeff);
(%i3) coeff(b+tan(x)+2*a*tan(x) = 3+5*tan(x),tan(x))
(%o3) 2 a+1=5
(%i4)coeff(1+x*%e^x+y,x,0)
(%o4) y+1
(%o4) done
```

[1] `http://maxima.sourceforge.net/docs/manual/en/maxima.html`

Příkaz `apropos()` nám vypíše veškeré příkazy, které obsahují zadané seskupení písmen, což je užitečné, pokud si nejsme jisti, zda příkaz píšeme správně nebo hledáme podobný příkaz.

```
(%i5) apropos("sin");

(%o5) [asin,asinh,decreasing,increasing,maxpsinegint,maxpsiposint,
poisint,require\_posinteger,sin,sinh,sinsert,sinvertcase,
piargs-sin/cos]
```

Kapitola 2

Základy práce

V této kapitole se seznámíme se základními operacemi a příkazy, které se používají v systému počítačové algebry Maxima.

2.1 Jednoduché operace

Pro základní operace jako je sčítání, odčítání, násobení a dělení se používají obvyklé znaky $+$,$-$,$*$ a $/$. V případě násobení vkládáme hvězdičku také mezi číslo a proměnnou. Umocňování se provádí pomocí znaku ^, v případě odmocňování dosazujeme vhodný zlomek. Druhou odmocninu zadáváme příkazem `sqrt(výraz)`. U příkazů musíme sledovat prioritu operací a případně správně uzávorkovat. Ukažme si jednoduchý příklad.

```
(%i1) x^2/3;
```

$$(\%o1) \qquad \frac{x^2}{3}$$

```
(%i2) x^(2/3);
```

$$(\%o2) \qquad x^{\frac{2}{3}}$$

2.2 Možnosti zobrazování výstupu

V programu Maxima existuje několik možností pro zobrazování výstupu. Mezi jednotlivými formáty přepínáme pomocí příkazu `set_display()`. Implicitně je nastaven formát `xml`.

```
(%i1) x^x/sqrt(x^2+1);
```

(%o1) $$\frac{x^x}{\sqrt{x^2+1}}$$

Nyní zvolíme formát `ascii`.

```
(%i2) set_display(ascii)$

(%i3) %i1;

                                    x
                                   x
(%o3)                         ------------
                                     2
                              sqrt(x  + 1)
```

Pokud zadáme `none`, bude se příkaz zobrazovat pouze v jediném řádku.

```
(%i4) set_display(none)$

(%i5) %i1;

(%o5) x^x/sqrt(x^2+1)
```

Poslední možností je výstup v podobě zdrojového kódu programu TEX. Pro tento druh výstupu slouží příkaz `tex()`. Pro příklad uveďme zobrazení výrazu $\frac{x}{\sqrt{x^2-1}}$.

```
(%i6) tex(x/sqrt(x^2-1))$

$${{x}\over{\sqrt{x^2-1}}}$$
```

2.3 Známé konstanty a jejich zápis

Před konstanty e (Eulerovo číslo), i (imaginární jednotka) a π (Ludolfovo číslo) píšeme vždy znak %. Pokud tento znak vynecháme, program bude předpokládát, že se nejedná o žádnou speciální konstantu a nebude schopen ji vyčíslit. Ukážeme si jednoduché příklady, na kterých ihned vidíme rozdíl.

```
(%i1) i^2;
```

(%o1) $$i^2$$

```
(%i2) %i^2;
```

(%o2) -1

```
(%i3) log(e);
```

(%o3) $\log(e)$

```
(%i4) log(%e);
```

(%o4) 1

Program používá znak % také u výstupů s výjimkou Ludolfova čísla, které zapisuje přímo jako π.
Konstantu ∞, resp. $-\infty$ píšeme ve tvaru `inf`, resp. `minf`. V tomto případě znak % nepoužíváme.

2.4 Přiřazování hodnot a výrazů

Pro přiřazování používáme znak `:`. Tímto znakem můžeme proměnným přiřazovat hodnoty i výrazy s jinými proměnnými (ne však funkce).

```
(%i1) y:x^2+3;
```

(%o1) x^2+3

```
(%i2) sqrt(y);
```

(%o2) $\sqrt{x^2+3}$

Vidíme, že když jsme zadali do následujícího příkazu y, program automaticky pracoval s výrazem x^2+3. Jestliže potřebujeme zrušit některé z přiřazených hodnot, používáme k tomuto účelu příkaz `kill()`. V případě, že chceme zrušit všechna přiřazení, použijeme příkaz `kill(all)`.

```
(%i3) a:1;
```

(%o3) 1

```
(%i4) a;
```

(%o4) 1

```
(%i5) kill(a);
```

(%o5) *done*

```
(%i6) a;
```

(%o6) a

Nyní se podívejme na rozdílné výsledky u následujících dvou příkladů. V obou případech budeme přiřazovat stejné hodnoty, avšak přehodíme pořadí.

```
(%i7) b:3;
```

(%o7) 3

```
(%i8) a:b;
```

(%o8) 3

```
(%i9) a;
```

(%o9) 3

```
(%i10) kill(a,b);
```

(%o10) 1

```
(%i11) a:b;
```

(%o11) b

```
(%i12) b:3;
```

(%o12) 3

```
(%i13) a;
```

(%o13) b

V prvním případě jsme nejprve přiřadili proměnné b hodnotu 3. Když jsme pak následně přiřadili proměnné a hodnotu proměnné b, program ji automaticky vyhodnotil jako číslo 3. Ve druhém případě jsme postupovali opačně. Program by měl v tomto případě reagovat stejně, avšak Maxima bohužel nedokáže vyhodnotit dvě taková po sobě jdoucí přiřazení, a proto bere v úvahu pouze první přiřazení proměnné a.

2.5 Volba přesnosti výpočtu

Maxima pracuje s přesnými čísly, která zapisuje v symbolickém tvaru. Standardně tedy nezaokrouhluje čísla jako $\sqrt{2}$, e nebo $\frac{1}{3}$. Formát výpisu nastavujeme příkazy `numer:true` a `numer:false`.

```
(%i1) %pi;
```

(%o1) π

```
(%i2) numer:true;
```

(%o2) *true*

```
(%i3) %pi;
```

(%o3) 3.141592653589793

```
(%i4) numer:false;
```

(%o4) *true*

```
(%i5) %pi;
```

(%o5) π

Příkaz `float()` převede číslo zapsané v symbolickém tvaru na desetinné číslo. Zobrazuje vždy 16 cifer.

```
(%i6) float(%pi);
```

(%o6) 3.141592653589793

Pokud chceme zobrazovat hodnotu s jinou přesností, nastavíme ji nejprve příkazem `fpprec:` a následně použijeme `bfloat()` pro zobrazení hodnoty.

```
(%i7) fpprec:25;
```

(%o7) 25

```
(%i8) bfloat(%pi);
```

(%o8) 3.1415926535897932384626430*b*0

2.6 Komplexní čísla

Komplexní čísla zadáváme v algebraickém tvaru pomocí imaginární jednotky %i.

```
(%i1) k1:1+2*%i;
```

(%o1) $$2\,\%i + 1$$

```
(%i2) k2:4-3*%i;
```

(%o2) $$4 - 3\,\%i$$

Nyní budeme s těmito čísly dále pracovat. Vypočítáme jejich součin a podíl. Dostaneme však neupravené výsledky. Výsledky v základním tvaru získáme až po užití příkazu `rectform`.

```
(%i3) k1*k2;
```

(%o3) $$(4 - 3\,\%i)\,(2\,\%i + 1)$$

```
(%i4) rectform(k1*k2);
```

(%o4) $$5\,\%i + 10$$

```
(%i5) k1/k2;
```

(%o5) $$\frac{2\,\%i + 1}{4 - 3\,\%i}$$

```
(%i6) rectform(k1/k2);
```

(%o6) $$\frac{11\,\%i}{25} - \frac{2}{25}$$

Jestliže chceme zobrazit reálnou, resp. imaginární, část komplexního čísla, použijeme příkaz `realpart`, resp. `imagpart`. Samozřejmě tyto příkazy můžeme použít i u součinu nebo podílu a to i v případě, že součin ani podíl nevyčíslíme.

```
(%i7) y:1+%i;
```

(%o7) $$\%i + 1$$

```
(%i8) z:3+2*%i;
```

$$(\%o8) \qquad 2\,\%i + 3$$

```
(%i9) realpart(y*z);
```

$$(\%o9) \qquad 1$$

```
(%i10) imagpart(y/z);
```

$$(\%o10) \qquad \frac{1}{13}$$

2.7 Zadávání předpokladů

V programu Maxima existuje několik možností, jak zadat předpoklady, které budou platné pro další zadané příkazy. Ukážeme si, jak zadáváme předpoklady pomocí příkazů `ev`, `assume` a `declare`.

Příkaz ev

Příkaz `ev(výraz,p1,p2,...)` definuje podmínky v rámci jednoho příkazu. Výraz se vyhodnotí za předpokladu platnosti zadaných podmínek. Použijeme podmínku $a = \mathrm{e}$ u řešení následující rovnice[1].

```
(%i1) solve(2*x+a=4,x);
```

$$(\%o1) \qquad \left[x = -\frac{a-4}{2}\right]$$

```
(%i2) ev(solve(2*x+a=4),x,a=%e);
```

$$(\%o2) \qquad \left[x = -\frac{\%\mathrm{e}-4}{2}\right]$$

Zkusíme zadat více podmínek. Přidáme podmínku, aby se výstup zobrazil ve formě desetinného čísla.

[1] Popis příkazu solve najdeme v kapitole 5, která se zabývá řešenm rovnic.

```
(%i3) ev(solve(2*x+a=4),x,a=%e,numer);
rat: replaced -1.28171817154095 by -10951/8544 = -1.28171816479401
rat: replaced -1.28171816479401 by -9668/7543 = -1.28171814927748
rat: replaced 2.651464934376242E-4 by 2/7543 = 2.651464934376242E-4
rat: replaced -0.6408590746387 by -4834/7543 = -0.6408590746387
```

(%o3) $[x = 0.64085907463874]$

```
(%i4) a;
```

(%o4) a

Vidíme, že předpoklady opravdu platí pouze v rámci příkazů s prostředím `ev`, u dalšího příkazu je přiřazení již zrušeno.

Příkazy assume a forget

Příkazem `assume(p1,p2,...)` definujeme předpoklady pro další příkazy. Tyto podmínky, narozdíl od příkazu `ev`, jsou platné pro všechny následující příkazy až do doby, kdy je zrušíme příkazem `forget(p1,p2,...)`. Tyto předpoklady mohou být zadány pouze pomocí operátorů pro relaci, tzn. <,>,<=,>=, = a $\neq$.

```
(%i1) sqrt((a+b)^2);
```

(%o1) $|b + a|$

```
(%i2) assume(a+b>0);
```

(%o2) $[b + a > 0]$

```
(%i3) sqrt((a+b)^2);
```

(%o3) $b + a$

```
(%i4) assume(a+b<0);
```

(%o4) $[inconsistent]$

Na tomto místě program negativně reagoval na pokus o předefinování již daných podmínek. Abychom mohli změnit předpoklad $a+b > 0$ na předpoklad $a+b < 0$, musíme nejprve platný předpoklad zrušit a teprve potom definovat druhý.

```
(%i5) forget(a+b>0);
```

(%o5) $[b+a>0]$

```
(%i6) assume(a+b<0);
```

(%o6) $[b+a<0]$

```
(%i7) sqrt((a+b)^2);
```

(%o7) $-b-a$

V případě rovnosti a nerovnosti do příkazu nepíšeme = nebo $\neq$, ale používáme příkazy `equal` a `notequal`.

```
(%i8) assume(equal(x,0),notequal(y,1));
```

(%o8) $[equal\,(x,0)\,,notequal\,(y,1)]$

Pravidla pro tyto dva předpoklady jsou stejná jako u předchozích předpokladů.

Příkaz declare

Příkaz `declare(x1,p1,x2,p2,...)` definuje předpoklad p1 pro proměnnou x1, předpoklad p2 pro proměnnou x2, atd. Předpoklady rušíme příkazem `kill(xi)`, kde i je číslo proměnné. Příkazem `kill` můžeme zrušit libovolný počet proměnných najednou, např. `kill(x1,x4,x5)`. Nemusí se jednat pouze o proměnnou, můžeme zadat předpoklad i pro příkaz, funkci,... K příkazu `declare` se vážou různé druhy argumentů, které najdeme v manuálu. My si vezmeme na ukázku jednoduchý příklad.

```
(%i1) (-1)^(n+1);
```

(%o1) $(-1)^{n+1}$

Nyní zavedeme předpoklad, že n je liché číslo (anglicky odd).

```
(%i2) declare(n,odd);
```

(%o2) *done*

```
(%i3) (-1)^(n+1);
```

(%o3) 1

Stejně jako u příkazu `assume` musíme zrušit současný předpoklad, abychom mohli definovat nový.

```
(%i4) kill(n);
```

(%o4) *done*

```
(%i5) (-1)^(n+1);
```

(%o5) $(-1)^{n+1}$

Nyní pro změnu zavedeme předpoklad, že n je sudé (anglicky even).

```
(%i6) declare(n,even);
```

(%o6) *done*

```
(%i7) (-1)^(n+1);
```

(%o7) -1

Kapitola 3

Práce s výrazy

3.1 Příkazy expand, factor a gfactor

Příkazy `expand`, `factor` a `gfactor` využíváme při práci s polynomy. Mějme jednoduchý výraz ve tvaru součinu.

```
(%i1) a:(x^2+1)*(x-3)*(x+2)^2;
```

(%o1) $$(x-3)\,(x+2)^2\,\left(x^2+1\right)$$

Příkaz `expand()` roznásobí všechny závorky.

```
(%i2) expand(a);
```

(%o2) $$x^5+x^4-7\,x^3-11\,x^2-8\,x-12$$

Příkaz `factor()` převede roznásobený tvar opět na tvar součinu kořenových činitelů.

```
(%i3) factor(%);
```

(%o3) $$(x-3)\,(x+2)^2\,\left(x^2+1\right)$$

Dále můžeme použít příkaz `gfactor`, který má stejnou funkci jako factor, pracuje však s oborem komplexních čísel, a rozloží tedy i závorku (x^2+1).

```
(%i4) gfactor(%);
```

(%o4) $$(x-3)\,(x+2)^2\,(x-\%i)\,(x+\%i)$$

3.2 Příkazy num, denom a partfrac

Jestliže pracujeme s výrazy ve tvaru zlomku, můžeme pracovat zvlášť s čitatelem nebo jmenovatelem. Slouží k tomu příkazy `num` (z anglického numerator)a `denom` (z anglického denominator).

```
(%i1) v:((x+1)*(x+2))/((x^2+1)*(x-4));
```

(%o1) $$\frac{(x+1)\ (x+2)}{(x-4)\ (x^2+1)}$$

```
(%i2) num(v);
```

(%o2) $$(x+1)\ (x+2)$$

```
(%i3) denom(v);
```

(%o3) $$(x-4)\ \left(x^2+1\right)$$

Tyto dva příkazy můžeme také kombinovat s jinými.

```
(%i4) expand(num(v));
```

(%o4) $$x^2+3\,x+2$$

```
(%i5) gfactor(denom(v));
```

(%o5) $$(x-4)\ (x-\%i)\ (x+\%i)$$

Při práci s racionálními funkcemi používáme příkaz `partfrac()`, který rozloží funkci (v našem případě výraz v) na parciální zlomky.

```
(%i6) partfrac(v,x);
```

(%o6) $$\frac{30}{17\ (x-4)}-\frac{13\,x+1}{17\ (x^2+1)}$$

3.3 Příkazy rat, ratsimp a radcan

Příkazy `rat`, `ratsimp` a `radcan` slouží ke zjednodušování výrazů. Příkaz `rat` zjednoduší výrazy, které obsahují pouze základní operace.

```
(%i1) p:(x^2-1)/((x+1)*(x+2));
```

$$(\%o1) \qquad \frac{x^2-1}{(x+1)\,(x+2)}$$

```
(%i2) rat(p);
```

$$(\%o2) \qquad \frac{x-1}{x+2}$$

Se složitějšími výrazy si poradí příkaz `ratsimp`.

```
(%i3) q:sin(x/(x^2+x))=exp((log(x)+1)^2-log(x)^2);
```

$$(\%o3) \qquad \sin\left(\frac{x}{x^2+x}\right) = \%e^{(\log x+1)^2-\log^2 x}$$

```
(%i4) ratsimp(q);
```

$$(\%o4) \qquad \sin\left(\frac{1}{x+1}\right) = \%e\, x^2$$

Příkaz `radcan` má podobnou funkci, avšak umí zjednodušit také části výrazu, které obsahují exponenty, odmocniny nebo logaritmy.

```
(%i5) r:a+b*x+3*(a+b*x)+log(a/2)+sqrt((a+b*x)^(-1));
```

$$(\%o5) \qquad \sqrt{\frac{1}{b\,x+a}} + 3\,(b\,x+a) + b\,x + \log\left(\frac{a}{2}\right) + a$$

```
(%i6) radcan(r);
```

$$(\%o6) \qquad \frac{\sqrt{b\,x+a}\,(4\,b\,x+\log a+4\,a-\log 2)+1}{\sqrt{b\,x+a}}$$

3.4 Substituce

Substituce se zadává příkazem `subst()`, který má tři povinné parametry. Je důležité dodržovat určité pořadí. Podívejme se na rozdíl v substitucích ve výrazu $2\,y^2 + y + (x+1)^2$.

```
(%i1) subst(x+1,y,2*y^2+y+(x+1)^2);
```

(%o1) $$3\,(x+1)^2 + x + 1$$

```
(%i2) subst(y,x+1,2*y^2+y+(x+1)^2);
```

(%o2) $$3\,y^2 + y$$

V prvním případě jsme nahradili proměnnou y výrazem $x+1$, ve druhém případě naopak.

Můžeme také použít jiný způsob zápisu substituce. Ukažme si zápis ekvivalntní k prvnímu příkladu.

```
(%i3) subst(y=x+1,2*y^2+y+(x+1)^2);
```

(%o3) $$3\,(x+1)^2 + x + 1$$

Chceme-li, aby program provedl jednu substituci a ve výsledku další substituci, použijeme tzv. posloupnost substitucí. Substituce píšeme do hranatých závorek a oddělíme je čárkou. V tomto případě lze použít pouze zápis s rovnítkem. Můžeme samozřejmě provádět i více než dvě substituce za sebou.

```
(%i4) subst([y=x+1,(x+1)^2=z],2*y^2+y+(x+1)^2);
```

(%o4) $$3z + x + 1$$

Nyní si ukážeme další druh substituce. Vezměme si výraz $3\,(x+1)^2 + x + 1$ a zkusme nahradit $x+1$ proměnnou y.

```
(%i5) subst(x+1=y,3*(x+1)^2+x+1);
```

(%o5) $$3\,y^2 + x + 1$$

V tomto případě nám příkaz `subst` nepomohl, protože konec výrazu nenahradil, přestože se v něm vyskytuje $x+1$. K substituci tohoto výrazu proto použijeme příkaz `ratsubst`, který si všímá matematického významu výrazu, a proto nahradí i tuto část. V případě `ratsubst` lze použít pouze jeden způsob zápisu.

```
(%i6) ratsubst(y,x+1,3*(x+1)^2+x+1);
```

(%o6) $$3\,y^2 + y$$

Posledním druhem substituce, který si ukážeme, je substituce za operátory. K tomuto účelu nám slouží příkaz `opsubst`. Tuto funkci musíme nejprve načíst zadáním příkazu `load("opsubst")`.

```
(%i7) load("opsubst");
(%o7) "C:/PROGRA~1/MAXIMA~1.1/share/maxima/5.22.1/share/contrib/
      opsubst.lisp"

(%i8) opsubst(g,f,f(x));
```

(%o8) $$g\,(x)$$

Kapitola 4

Funkce

Následující kapitola popisuje, jak se v programu Maxima definují funkce a jaké operace u nich můžeme použít. Uvedeme si také zápisy několika často používaných funkcí.

4.1 Definování funkcí

Pro definování funkcí používáme operátor `:=`. Při zadávání musíme do závorky uvést vždy název proměnné, popř. proměnných, pokud zadáváme funkci dvou nebo více proměnných.

```
(%i1) f(x):=x^2+3;
```

(%o1) $$f(x) := x^2 + 3$$

Funkční hodnoty se počítají velmi jednoduše, zde vidíme funkční hodnotu pro $x = 1$.

```
(%i2) f(1);
```

(%o2) 4

Funkci je možné zadat také tak, že nadefinujeme, jak se bude funkce chovat v jednotlivých bodech nebo intervalech. Používáme k tomu větvení `if then`, `elseif then` a `else`.

```
(%i3) g(x):=if x<0 then -1 elseif x>1 then 1 else 0$
```

Funkce $g(x)$ je definována tak, že jestliže $x < 0$, pak $g(x) = -1$, jestliže $x > 1$, pak $g(x) = 1$ a ve všech ostatních bodech $g(x) = 0$. Pro kontrolu vypočítáme některé funkční hodnoty.

```
(%i4) g(0);
```

(%o4) 0

```
(%i5) g(1);
```

(%o5) 0

```
(%i6) g(-2);
```

(%o6) -1

```
(%i7) g(6/5);
```

(%o7) 1

4.2 Funkční operace

Funkce můžeme sčítat, odčítat, násobit, dělit a také skládat. Všechny tyto operace zadáváme obvyklými příkazy, a není tedy nutné tuto část rozsáhle popisovat. Uveďme si pouze jeden příklad na každý příkaz.

```
(%i1) f(x):=x^2+x-2;
```

(%o1) $f(x) := x^2 + x - 2$

```
(%i2) g(x):=x-1;
```

(%o2) $g(x) := x - 1$

```
(%i3) f(x)+g(x);
```

(%o3) $x^2 + 2\,x - 3$

```
(%i4) f(x)-g(x);
```

(%o4) $x^2 - 1$

```
(%i5) f(x)*g(x);
```

(%o5) $$(x-1)\left(x^2+x-2\right)$$

```
(%i6) f(x)/g(x);
```

(%o6) $$\frac{x^2+x-2}{x-1}$$

```
(%i7) f(g(x));
```

(%o7) $$x+(x-1)^2-3$$

4.3 Exponenciální a logaritmická funkce

Exponenciální funkci zadáváme pomocí příkazu `exp()`, případně můžeme použít konstantu e.

```
(%i1) f(x):=exp(x^3+2);
```

(%o1) $$f(x):=\exp\left(x^3+2\right)$$

```
(%i2) g(x):=%e^(x^3+2);
```

(%o2) $$g(x):=\%e^{x^3+2}$$

Logaritmickou funkci zadáváme netradičně příkazem `log()`, nikoliv `ln()`. Tento způsob zápisu se může plést s dekadickým logaritmem, avšak pro jiný než přirozený logaritmus Maxima nemá příkaz.

```
(%i3) h(x):=log(x);
```

(%o3) $$h(x):=\log(x)$$

Zda se opravdu jedná o přirozený logaritmus, můžeme ověřit zjištěním funkční hodnoty pro konstantu e, která musí být rovna 1.

```
(%i4) h(%e);
```

(%o4) 1

Pokud bychom přece jen chtěli počítat s dekadickým logaritmem, použijeme definici $\log_{10}(x)=\frac{\log(x)}{\log(10)}$.

4.4 Goniometrické a cyklometrické funkce

Funkce sinus a cosinus zadáváme podle očekávání `sin()` a `cos()`. Zápis funkcí tangens a cotangens se trochu liší od běžného zápisu, zadáváme je příkazy `tan()` a `cot()`. Jestliže známe tyto zápisy, můžeme jednoduše odvodit zápis cyklometrických funkcí, tedy funkcí inverzních ke goniometrickým. Tvary těchto funkcí jsou následující - `asin()`, `acos()`, `atan()` a `acot()`. Ukažme si příklad pro funkce `tan()` a `atan()`.

```
(%i1) tan(%pi/4);
```

$$(\%o1) \qquad 1$$

```
(%i2) atan(1);
```

$$(\%o2) \qquad \frac{\pi}{4}$$

Kapitola 5

Řešení rovnic a jejich soustav

V této kapitole si popíšeme příkazy, pomocí kterých řešíme rovnice a jejich soustavy. Příkazů existuje samozřejmě více, my si však uvedeme pouze tři z nich.

5.1 Příkaz solve

Příkazem `solve()` řešíme rovnice i jejich soustavy. V případě jedné rovnice píšeme `solve(rovnice, proměnná)`. Parametr proměnná nemusíme uvádět, pokud rovnice obsahuje pouze jednu proměnnou. Jestliže místo rovnice napíšeme jen výraz, program bude předpokládat, že chceme spočítat rovnici, kde je zadaný výraz roven nule. Následující dva příklady jsou dva různé způsoby zápisu téže rovnice.

```
(%i1) solve(x+2=0,x);
```

(%o1) $[x = -2]$

```
(%i2) solve(x+2);
```

(%o2) $[x = -2]$

Nyní si ukážeme příklad s funkcí $\sin(x)$.

```
(%i3) solve(sin(x)=0,x);

solve: using arc-trig functions to get a solution.
Some solutions will be lost.
```

(%o3) $[x = 0]$

V tomto případě nás program upozornil, že „některá řešení budou ztracena“. Tato rovnice má samozřejmě nekonečně mnoho řešení. Přičtením k-násobku konstanty π získáme další řešení. Maxima však zobrazuje pouze jedno z nich.

Jestliže chceme zadat soustavu rovnic, píšeme
`solve([rovnice 1,...,rovnice n],[proměnná 1,...,proměnná n])`.

```
(%i4) solve([y^2+x=1,x-2=3],[y,x]);
```

(%o4) $$[[y = -2\,\%i, x = 5], [y = 2\,\%i, x = 5]]$$

5.2 Příkaz linsolve

Příkaz `linsolve()` slouží pro řešení lineárních rovnic a jejich soustav.

```
(%i1) linsolve([y^2+x=1,x-2=3],[y,x]);
```

(%o1) $$[]$$

Zadali jsme soustavu z předcházejícího příkladu, první rovnice však nebyla lineární, a tudíž jsme neobdrželi žádný výsledek. Nyní změníme první rovnici na lineární.

```
(%i2) linsolve([y+x=1,x-2=3],[y,x]);
```

(%o2) $$[y = -4, x = 5]$$

V tomto případě jsme již soustavu pomocí `linsolve` vyřešili.

5.3 Příkaz find_root

Příkaz `find_root()` slouží pro přibližné řešení rovnic - zobrazí výsledek ve formě desetinného čísla. Příkaz má čtyři povinné parametry. První dva určují rovnici a proměnnou stejně jako u předchozích příkazů a další dva argumenty určují interval, na kterém se rovnice řeší. Pomocí tohoto příkazu můžeme vyřešit například rovnici $\cos(x) = \frac{x}{2}$ na intervalu $\langle 0, \pi \rangle$.

```
(%i1) find_root(cos(x)=x/2,x,0,%pi);
```

(%o1) 1.029866529322259

Kapitola 6

Matice

Tato kapitola se zabývá maticovou algebrou. Ukážeme si, jak se matice vytvářejí, jaké operace u nich můžeme použít, a dále také uvidíme, jak se zjišťuje determinant, hodnost, transponovaná matice a inverzní matice.

6.1 Vytváření matic

Pro vytvoření matice používáme příkaz `matrix([],[],...,[])`, přičemž v hranatých závorkách vypisujeme jednotlivé řádky matice. Matici

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix}$$

tedy zadáme tímto způsobem:

```
(%i1) A:matrix([a,b],[c,d],[e,f]);
```

(%o1)
$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix}$$

6.2 Operace s maticemi

Matice můžeme sčítat a odčítat velmi jednoduše, samozřejmě je nutné mít matice stejných rozměrů.

```
(%i1) A:matrix([2,a],[-1,-2*b]);
```

(%o1)
$$\begin{pmatrix} 2 & a \\ -1 & -2\,b \end{pmatrix}$$

```
(%i2) B:matrix([3*a,2],[1-b,1]);
```

(%o2)
$$\begin{pmatrix} 3\,a & 2 \\ 1-b & 1 \end{pmatrix}$$

```
(%i3) A+B;
```

(%o3)
$$\begin{pmatrix} 3\,a+2 & a+2 \\ -b & 1-2\,b \end{pmatrix}$$

```
(%i4) A-B;
```

(%o4)
$$\begin{pmatrix} 2-3\,a & a-2 \\ b-2 & -2\,b-1 \end{pmatrix}$$

Násobení matic již není tak jednoduché, neboť zde existují dva různé druhy násobení - pomocí znaků . a $*$.

```
(%i5) A.B;
```

(%o5)
$$\begin{pmatrix} a\ (1-b)+6\,a & a+4 \\ -2\ (1-b)\ b-3\,a & -2\,b-2 \end{pmatrix}$$

```
(%i6) A*B;
```

(%o6)
$$\begin{pmatrix} 6\,a & 2\,a \\ b-1 & -2\,b \end{pmatrix}$$

Vidíme, že násobení s použitím tečky je obvyklé násobení matic, kdy násobíme jednotlivé řádky a sloupce, zatímco použitím hvězdičky pouze vynásobíme prvky na odpovídajících si pozicích. Ukažme si ještě jeden příklad.

```
(%i7) C:matrix([1,b],[3,c-1],[2*a,-3]);
```

(%o7)
$$\begin{pmatrix} 1 & b \\ 3 & c-1 \\ 2\,a & -3 \end{pmatrix}$$

```
(%i8) D:matrix([2,c-3,4],[b+1,2,a]);
```

(%o8)
$$\begin{pmatrix} 2 & c-3 & 4 \\ b+1 & 2 & a \end{pmatrix}$$

```
(%i9) C*D;

fullmap: arguments must have same formal structure.
 -- an error. To debug this try: debugmode(true);
```

U těchto matic není možné násobit pouze jednotlivé prvky, jelikož matice nejsou stejného typu. Můžeme je však násobit pomocí tečky, protože matice C je typu 3×2 a matice D typu 2×3.

```
(%i10) C.D;
```

(%o10)
$$\begin{pmatrix} b\,(b+1)+2 & c+2\,b-3 & a\,b+4 \\ (b+1)\,(c-1)+6 & 2\,(c-1)+3\,(c-3) & a\,(c-1)+12 \\ 4\,a-3\,(b+1) & 2\,a\,(c-3)-6 & 5\,a \end{pmatrix}$$

Dělení matic provádíme pouze u matic stejného typu, kde se opět dělí prvky na stejných pozicích.

```
(%i11) A/B;
```

(%o11)
$$\begin{pmatrix} \frac{2}{3\,a} & \frac{a}{2} \\ -\frac{1}{1-b} & -2\,b \end{pmatrix}$$

6.3 Determinant

Pro výpočet determinantu čtvercových matic používáme příkaz `determinant()`.

```
(%i1) determinant(A);
```

(%o1)
$$a-4\,b$$

```
(%i2) determinant(D);

determinant: matrix must be square; found 2 rows, 3 columns.
 -- an error. To debug this try: debugmode(true);
```

V případě matice D nás program upozornil, že jsme použili příkaz na matici, která není čtvercová, a nelze tedy vypočítat její determinant.

6.4 Hodnost matice

Hodnost matice zjistíme zadáním příkazu `rank()`.

```
(%i1) rank(A);
```

$$(\%o1) \qquad 2$$

```
(%i2) rank(C);
```

$$(\%o2) \qquad 2$$

6.5 Transponovaná a inverzní matice

Pro transponování matic používáme příkaz `transpose()`. Inverzní matici získáme pomocí příkazu `invert()`.

```
(%i1) transpose(B);
```

$$(\%o1) \qquad \begin{pmatrix} 3\,a & 1-b \\ 2 & 1 \end{pmatrix}$$

```
(%i2) invert(B);
```

$$(\%o2) \qquad \begin{pmatrix} \frac{1}{2\,(b-1)+3\,a} & -\frac{2}{2\,(b-1)+3\,a} \\ \frac{b-1}{2\,(b-1)+3\,a} & \frac{3\,a}{2\,(b-1)+3\,a} \end{pmatrix}$$

```
(%i3) transpose(D);
```

$$(\%o3) \qquad \begin{pmatrix} 2 & b+1 \\ c-3 & 2 \\ 4 & a \end{pmatrix}$$

```
(%i4) invert(D);

determinant: matrix must be square; found 1 rows, 2 columns.
 -- an error. To debug this try: debugmode(true);
```

Inverzní matici jsme, podobně jako u determinantu, nemohli získat z matice, která není čtvercová.

Kapitola 7

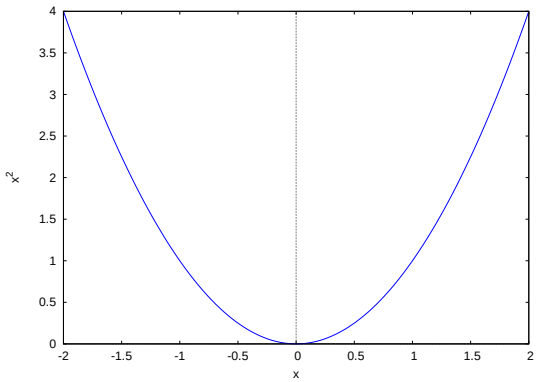
Tvorba grafů

V následující kapitole se budeme věnovat tvorbě 2D a 3D grafů. Uvedeme si příkazy, které se k tomuto účelu používají a zaměříme se také na řešení některých problémů, na které zde můžeme narazit.

7.1 2D grafy

2D grafy zobrazujeme pomocí příkazů `plot2d()` nebo `wxplot2d()`. Tyto dva příkazy plní stejnou funkci; rozdíl je pouze v tom, že příkaz `plot2d` ukáže graf funkce v novém okně, `wxplot2d` (použitelný pouze pro grafickou nadstavbu wxmaxima) graf zobrazí přímo pod příkaz. Dále budeme používat pouze příkaz `plot2d`. Abychom mohli zobrazit graf funkce, musíme uvést nejméně dva parametry. První parametr říká, kterou funkci chceme zobrazit, a druhý definuje interval na ose x, na kterém bude funkce zobrazena.

```
(%i1) plot2d(x^2,[x,-2,2]);
```

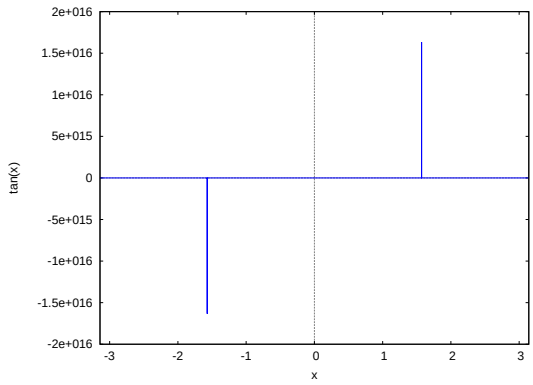


Obrázek 7.1: Graf funkce $f(x) = x^2$

Je však vhodné uvádět ještě třetí parametr, a to interval osy y, na kterém

chceme funkci zobrazit. Jestliže jej neuvedeme, program vybere interval sám podle funkčních hodnot. Podívejme se na graf funkce $\operatorname{tg} x$.

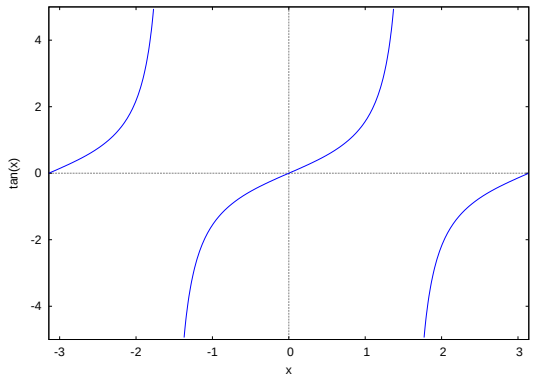
```
(%i2) plot2d(tan(x),[x,-%pi,%pi]);
```



Obrázek 7.2: Graf funkce $f(x) = \operatorname{tg} x$

Funkce $\operatorname{tg} x$ se v některých bodech blíží nekonečnu, a program tedy zobrazil graf na příliš velkém intervalu osy y. Z takového grafu nelze nic vyčíst. Proto přidáme parametr s mezemi intervalu na ose y.

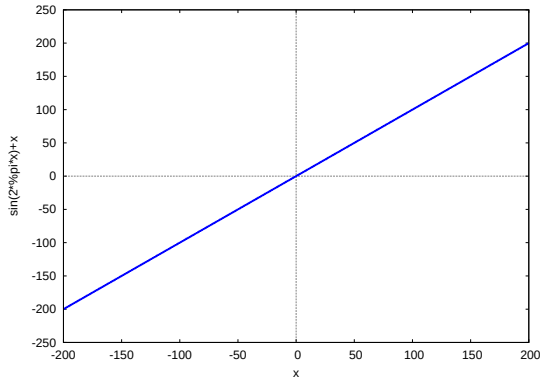
```
(%i3) plot2d(tan(x),[x,-%pi,%pi],[y,-5,5]);
```



Obrázek 7.3: Graf funkce $f(x) = \operatorname{tg} x$ upravený

Nyní můžeme dobře vidět průběh funkce $\operatorname{tg} x$ na intervalu $(-\pi, \pi)$. Ukažme si ještě jeden příklad.

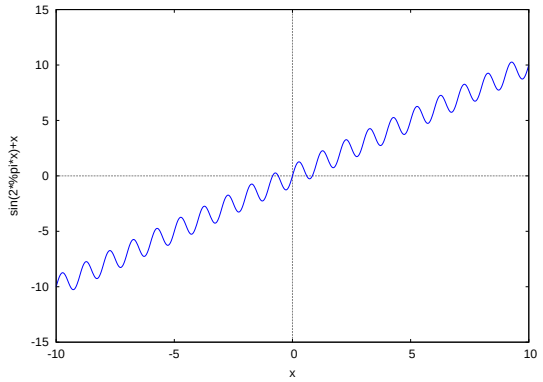
```
(%i4) plot2d(x+sin(2*%pi*x),[x,-200,200]);
```



Obrázek 7.4: Příliš velký interval

Na první pohled se zdá, že průběh této funkce vidíme velmi dobře. Není tomu tak. Jakmile zmenšíme interval na $(-10, 10)$, nastane velká změna.

```
(%i5) plot2d(x+sin(2*%pi*x),[x,-10,10]);
```

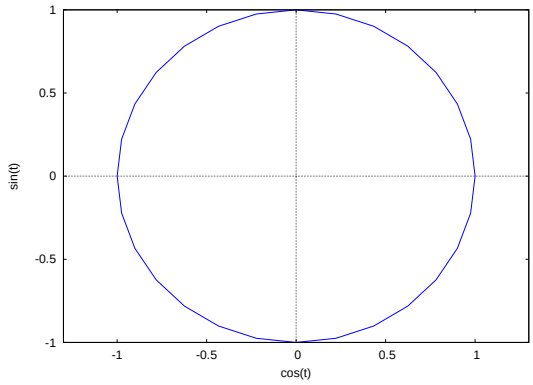


Obrázek 7.5: Vhodný interval

Na tomto grafu vidíme vlnky, které byly v předchozím případě malé a nahuštěné vedle sebe tak, že splynuly v přímku.

Funkce můžeme zadávat také parametricky příkazem `plot2d([parametric])`. Zkusíme zobrazit kružnici.

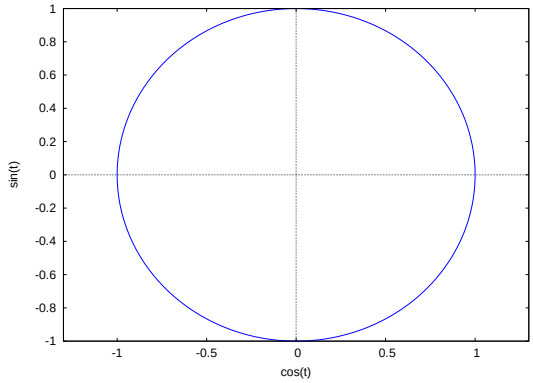
```
(%i6) plot2d([parametric,cos(t),sin(t),[t,-%pi,%pi]],
[x,-1.3,1.3]);
```



Obrázek 7.6: Kružnice 1

V tomto grafu se kružnice zdá být trochu „kostrbatá“. Tuto vadu odstraníme přidáním referenčních bodů. Implicitně je nastaveno 29 bodů, které program vyhodnotí a poté je spojí. Pomocí parametru `nticks` nyní nastavíme 500 referenčních bodů.

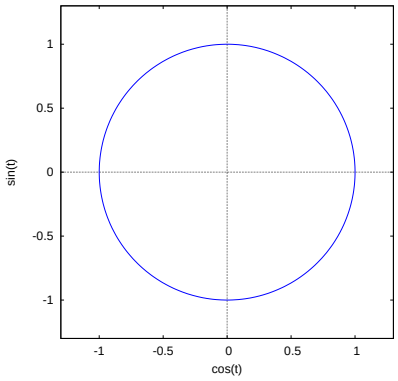
```
(%i7) plot2d([parametric,cos(t),sin(t),[t,-%pi,%pi]],
[x,-1.3,1.3],[nticks,500]);
```



Obrázek 7.7: Kružnice 2

Nyní je kostrbatost odstraněna. Stále však nemáme kružnici. Program si sám upravuje měřítko os, a proto místo kružnice dostáváme elipsu. I kdybychom zadali na ose x a y stejný interval, bude interval osy x na výstupu delší. V následujícím příkladě je ukázáno, jakým způsobem upravíme měřítka na osách.

```
(%i8) plot2d([parametric,cos(t),sin(t),[t,-%pi,%pi]],
[x,-1.3,1.3],[y,-1.3,1.3],[nticks,500],
[gnuplot_preamble,"set size ratio 1"]);
```

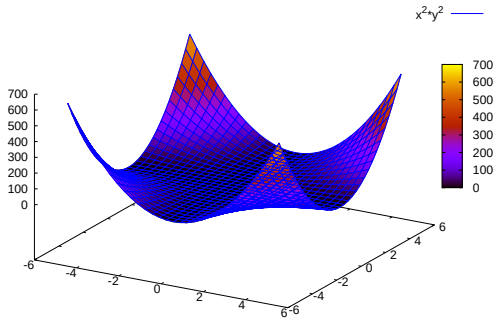


Obrázek 7.8: Kružnice 3

7.2 3D grafy

Tvorba 3D grafů je obdobou tvorby 2D grafů, a proto uvedeme jen několik příkladů. Pro zobrazení 3D grafů používáme příkazy `plot3d()` a `wxplot3d()`. V tomto případě musíme zadat tři povinné parametry - funkci, interval na ose x a interval na ose y.

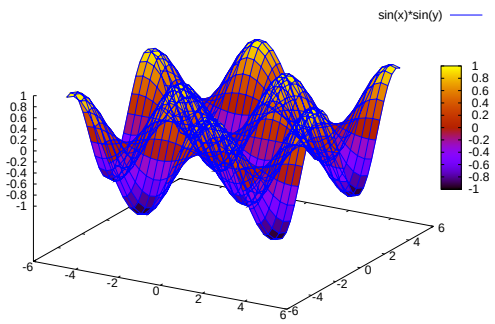
```
(%i1) plot3d(x^2*y^2,[x,-5,5],[y,-5,5]);
```



Obrázek 7.9: Graf funkce $f(x, y) = x^2y^2$

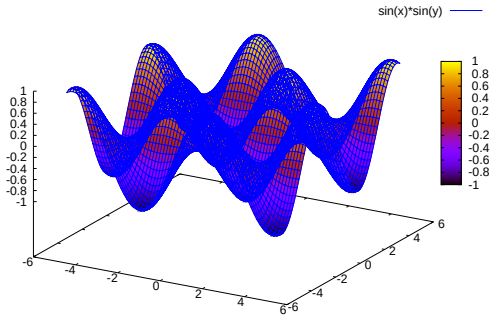
Podobnou funkci jako parametr `nticks` u 2D grafů plní u 3D grafů parametr `grid`, kterým zadáme množství referenčních bodů pro osu x a y. Implicitně je nastaveno 30 bodů na každé ose. Někdy je však vhodné počet bodů zvýšit. Ukažme si rozdíl na následující funkci.

```
(%i2) plot3d(sin(x)*sin(y),[x,-5,5],[y,-5,5]);
```



Obrázek 7.10: Funkce bez parametru grid

```
(%i3) plot3d(sin(x)*sin(y),[x,-5,5],[y,-5,5],[grid,70,70]);
```



Obrázek 7.11: Funkce s parametrem grid

7.3 Uložení grafu do souboru

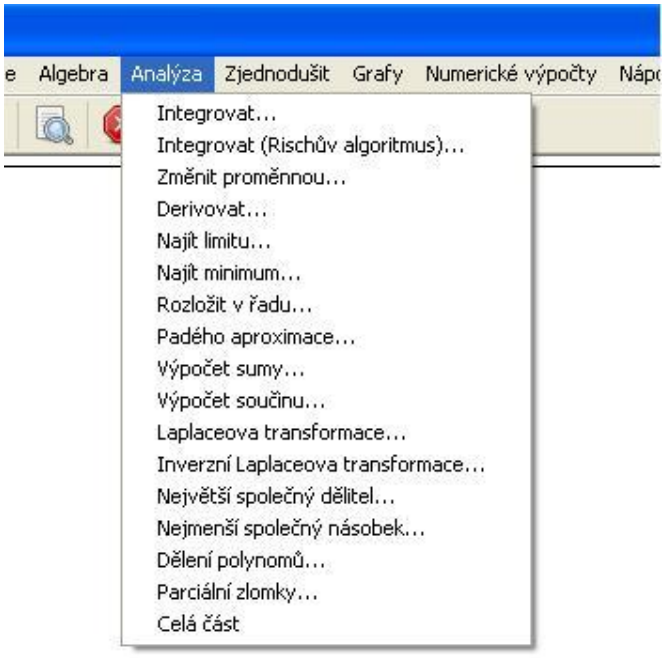
2D i 3D grafy můžeme ukládat do postscriptového souboru. U příkazu `plot2D` nebo `plot3D` nejprve nastavíme formát PS parametrem `[gnuplot_term, ps]` a následně určíme místo, kam výstupní soubor chceme uložit pomocí parametru `[gnuplot_out_file,"../jmenosouboru.eps"]`.

```
(%i1) plot3d(sin(x)*sin(y), [x,-5,5], [y,-5,5],[grid,70,70],
       [gnuplot_term, ps],
       [gnuplot_out_file, "C:/Documents and Settings/Admin/
       Dokumenty/graf.eps"])$
```

Kapitola 8

Matematická analýza

Následující kapitola se bude zabývat využitím programu Maxima při výuce matematické analýzy. Popíšeme si příkazy týkající se výpočtu limit, derivací, integrálů, Taylorova polynomu a průběhu funkce. Místo zadávání příkazů můžeme použít položky v záložce `Analýza` hlavního menu (viz obr. 8.1), my však budeme pracovat přímo s jednotlivými příkazy. Na příkladech si také ukážeme využití některých příkazů z předchozích kapitol. Většina zadání příkladů této kapitoly byla převzata z knihy [2].



Obrázek 8.1: Záložka Analýza

8.1 Limity

V programu Maxima používáme k výpočtu limit příkaz `limit()` se třemi povinnými argumenty. První argument je funkce, jejíž limitu počítáme, druhý argument je proměnná a třetí argument je bod, ke kterému se funkce blíží. Vypočítejme limitu

$$\lim_{x\to 1} \frac{x^2-1}{2\,x^2-x-1}.$$

```
(%i1) limit((x^2-1)/(2*x^2-x-1), x, 1);
```

$$(\%o1) \qquad \frac{2}{3}$$

Je nutné dávat pozor na výsledek následujícího příkladu.

```
(%i3) limit((x^2+1)/(x+1),x,-1);
```

$$(\%o3) \qquad infinity$$

Na první pohled se zdá, že limita je rovna nekonečnu (infinity=nekonečno), avšak není tomu tak. Tímto výstupem Maxima oznamuje, že daná limita neexistuje. Pokud by byla rovna nekonečnu, dostali bychom výsledek ∞. Správnost výsledku můžeme jednoduše ověřit porovnáním jednostranných limit. Limitu zleva, resp. zprava, najdeme přidáním volitelného argumentu `minus`, resp. `plus`.

```
(%i4) limit((x^2+1)/(x+1),x,-1,minus);
```

$$(\%o4) \qquad -\infty$$

```
(%i5) limit((x^2+1)/(x+1),x,-1,plus);
```

$$(\%o5) \qquad \infty$$

Jednostranné limity jsou různé, oboustranná limita tedy nemůže existovat.

8.2 Derivace

Pro výpočet využijeme příkazu `diff()` se dvěma argumenty. První argument nám udává funkci, jejíž derivaci počítáme, druhý argument je proměnná, podle které derivujeme. Vypočítejme derivaci funkce $y = \frac{1+x-x^2}{1-x+x^2}$.

```
(%i1) diff((2*x)/(1-x^2),x);
```

(%o1)
$$\frac{2}{1-x^2}+\frac{4\,x^2}{\left(1-x^2\right)^2}$$

Vidíme, že výsledek lze převést na společný jmenovatel, proto jej upravíme, například takto:

```
(%i2) subst(z,1-x^2,%);
```

(%o2)
$$\frac{2}{z}+\frac{4\,x^2}{z^2}$$

```
(%i3) factor(%);
```

(%o3)
$$\frac{2\,\left(z+2\,x^2\right)}{z^2}$$

```
(%i4) subst(1-x^2,z,%);
```

(%o4)
$$\frac{2\,\left(x^2+1\right)}{\left(1-x^2\right)^2}$$

Nyní si ukážeme výpočet derivace druhého řádu funkce $y = x \ln x$. Tuto derivaci můžeme spočítat dvěma následujícími způsoby:

```
(%i5) diff(diff(x*log(x),x),x);
```

(%o5)
$$\frac{1}{x}$$

```
(%i6) diff(x*log(x),x,2);
```

(%o6)
$$\frac{1}{x}$$

Vidíme, že výhodnější je druhá varianta, kde jsme přidali volitelný argument 2. Obzvláště u derivací řádu 3 a výše by bylo nepraktické opakovaně opisovat příkaz `diff` se všemi závorkami.

Na dalším příkladě si ukážeme, jak se v programu Maxima počítají derivace parciální. Vezměme funkci

$$u(x,y) = xy + \frac{x}{y}$$

a vypočítejme parciální derivace prvního a druhého řádu.

```
(%i7) u(x,y):=x*y+x/y;
```

(%o7) $$u\left(x,y\right):=x\,y+\frac{x}{y}$$

```
(%i8) diff(u(x,y),x);
```

(%o8) $$y+\frac{1}{y}$$

```
(%i9) diff(u(x,y),y);
```

(%o9) $$x-\frac{x}{y^2}$$

Abychom spočítali parciání derivace $\frac{\partial^2 u}{\partial x^2}$ a $\frac{\partial^2 u}{\partial y^2}$, stačí opět přidat volitelný argument 2.

```
(%i10) diff(u(x,y),x,2);
```

(%o10) $$0$$

```
(%i11) diff(u(x,y),y,2);
```

(%o11) $$\frac{2\,x}{y^3}$$

Na závěr spočítáme smíšenou parciální derivaci $\frac{\partial^2 u}{\partial x \partial y}$. Opět je možno použít dva způsoby zápisu.

```
(%i12) diff(diff(u(x,y),x),y);
```

(%o12) $$1-\frac{1}{y^2}$$

```
(%i13) diff(u(x,y),x,1,y,1);
```

(%o13) $$1-\frac{1}{y^2}$$

8.3 Integrály

Nyní se budeme věnovat výpočtům neurčitých a určitých integrálů. K výpočtu neurčitých integrálů slouží příkaz `integrate(funkce,proměnná)`. Jako příklad uveďme integrál $\int \frac{\mathrm{e}^x}{2+\mathrm{e}^x}\,dx$.

```
(%i1) integrate((%e^x)/(2+%e^x),x);
```

(%o1) $$\log\left(\%\mathrm{e}^x+2\right)$$

V případě určitých integrálů použijeme stejný způsob, pouze přidáme dva argumenty, které udávají příslušný interval, na němž integrál počítáme. Vypočtěme určitý integrál $\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}}\,dx$.

```
(%i2) integrate(1/sqrt(1-x^2),x,-1,1);
```

(%o2) $$\pi$$

V některých případech Maxima používá ve výsledku funkci erf (z anglického error function). Tato funkce je definována takto:

$$erf\left(z\right)=\frac{2}{\sqrt{\pi}}\int_0^z \mathrm{e}^{-t^2}\,dt$$

Jestliže obdržíme výstup obsahující funkci erf, musíme vypočítat výsledek jiným způsobem. Můžeme použít buď příkaz `float` nebo příkaz pro numerické integrování `romberg`.

```
(%i3) integrate(%e^(-x^2),x,1,3);
```

$$\frac{\sqrt{\pi}\,\mathrm{erf}\left(3\right)}{2}-\frac{\sqrt{\pi}\,\mathrm{erf}\left(1\right)}{2}$$

```
(%i4) float(%);
```

(%o4) $$0.13938321544709$$

```
(%i5) romberg(%e^(-x^2),x,1,3);
```

(%o5) $$0.13938329641992$$

8.4 Taylorův polynom

K výpočtu Taylorova polynomu používáme příkaz `taylor()` se čtyřmi argumenty. První z nich je zadaná funkce, následuje proměnná, poté střed a nakonec stupeň polynomu. Určíme Taylorův polynom stupně 6 se středem v bodě 0 pro funkci

$$f(x) = \ln \frac{\sin x}{x}.$$

```
(%i1) taylor(log(sin(x)/x),x,0,6);
```

(%o1) $$-\frac{x^2}{6} - \frac{x^4}{180} - \frac{x^6}{2835} + \cdots$$

Na konci výsledku Maxima vždy zobrazuje tři tečky, což může vyvolat dojem, že se jedná o nekonečnou řadu. Pomocí příkazu `expand` se přesvědčíme, že se skutečně jedná o polynom.

```
(%i2) expand(%);
```

(%o2) $$-\frac{x^6}{2835} - \frac{x^4}{180} - \frac{x^2}{6}$$

8.5 Vyšetřování průběhu funkce

Vyšetřování průběhu funkce patří mezi jednu z nejdůležitějších úloh matematické analýzy, které řeší studenti na vysokých školách, proto se tomuto tématu budeme věnovat trochu podrobněji. Vyšetřeme průběh funkce

$$f\left(x\right) = \frac{x^2\ \left(x-1\right)}{\left(x+1\right)^2}.$$

```
(%i1) f(x):=(x^2*(x-1))/(x+1)^2;
```

(%o1) $$f\left(x\right) := \frac{x^2\ \left(x-1\right)}{\left(x+1\right)^2}$$

Definiční obor

Určíme definiční obor funkce. Zadaná funkce je ve tvaru zlomku, a proto musíme vyloučit body, v nichž je jmenovatel roven nule.

```
(%i2) j:denom(f(x));
```

(%o2) $$(x+1)^2$$

```
(%i3) solve(j=0);
```

(%o3) $$[x=-1]$$

Vidíme, že bod $x=-1$ nepatří do definičního oboru.

Průsečíky se souřadnicovými osami

Dále najdeme průsečíky grafu funkce se souřadnicovými osami. Abychom nalezli průsečíky s osou x, položíme $f(x)=0$.

```
(%i4) solve(f(x)=0);
```

(%o4) $$[x=0, x=1]$$

Průsečíky s osou y získáme dosazením 0 za proměnnou x.

```
(%i5) f(0);
```

(%o5) $$0$$

Lokální extrémy

Nyní určíme stacionární body funkce, tzn. body, v nichž je první derivace funkce rovna nule. V těchto bodech může funkce nabývat lokálních extrémů. Spočítáme tedy nejprve první derivaci funkce $f(x)$.

```
(%i6) d1:diff(f(x),x);
```

(%o6) $$\frac{x^2}{(x+1)^2}+\frac{2\,(x-1)\,x}{(x+1)^2}-\frac{2\,(x-1)\,x^2}{(x+1)^3}$$

Získaný výsledek zjednodušíme (není to však nutné).

```
(%i7) factor(d1);
```

(%o7) $$\frac{x\,(x^2+3\,x-2)}{(x+1)^3}$$

Nyní derivaci položíme rovnu nule.

```
(%i8) solve(d1=0);
```

$$(\%o8) \qquad [x = -\frac{\sqrt{17}+3}{2}, x = \frac{\sqrt{17}-3}{2}, x = 0]$$

Vidíme, že existují tři stacionární body. Abychom zjistili, zda má funkce $f(x)$ v těchto bodech lokální maximum (resp. minimum), spočítáme její druhou derivaci.

```
(%i9) d2:diff(f(x),x,2);
```

$$(\%o9) \qquad \frac{4\,x}{(x+1)^2} + \frac{2\,(x-1)}{(x+1)^2} - \frac{4\,x^2}{(x+1)^3} - \frac{8\,(x-1)\,x}{(x+1)^3} + \frac{6\,(x-1)\,x^2}{(x+1)^4}$$

```
(%i10) h:factor(d2);
```

$$(\%o10) \qquad \frac{2\,(5\,x-1)}{(x+1)^4}$$

Pokud je druhá derivace funkce ve stacionárním bodě kladná (resp. záporná), pak funkce má v tomto bodě minimum (resp. maximum). Pomocí substituce nyní spočítáme hodnoty druhé derivace ve stacionárních bodech a také hodnoty funkce $f(x)$ v těchto bodech. Ve výrazech se vyskytují odmocniny, a proto přepneme na výstup ve formě desetinných čísel, abychom mohli výsledky lépe porovnat.

```
(%i11) numer:true;
```

$$(\%o11) \qquad true$$

```
(%i12) subst(0,x,h);
```

$$(\%o12) \qquad -2$$

```
(%i13) f(0);
```

$$(\%o13) \qquad 0$$

```
(%i14) subst(-((sqrt(17)+3)/2),x,h);
```

$$(\%o14) \qquad -0.87368304210317$$

```
(%i15) f(-((sqrt(17)+3)/2));
```

(%o15) -8.818299727218765

```
(%i16) subst((sqrt(17)-3)/2,x,h);
```

(%o16) 0.60805804210317

```
(%i17) f((sqrt(17)-3)/2);
```

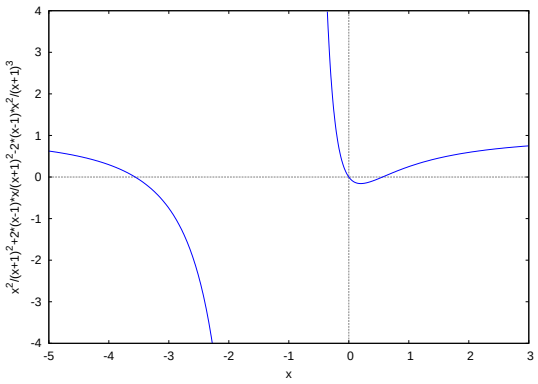
(%o17) -0.056700272781236

```
(%i18) numer:false;
```

(%o18) $false$

Výsledky ukázaly, že funkce nabývá v bodech $x = 0$, $(f(0) = 0)$ a $x = -\frac{\sqrt{17}+3}{2}$, $(f(-\frac{\sqrt{17}+3}{2}) = -8{,}82)$ lokálního maxima a v bodě $x = \frac{\sqrt{17}-3}{2}$, $(f(\frac{\sqrt{17}-3}{2}) = -0{,}06)$ lokálního minima. S pomocí grafu první derivace funkce $f(x)$ si výsledek ověříme.

```
(%i19)plot2d(d1,[x,-5,3],[y,-4,4]);
```



Obrázek 8.2: Graf první derivace funkce $f(x)$

Vidíme, že první derivace funkce $f(x)$ je kladná na intervalech $(-\infty, -\frac{\sqrt{17}+3}{2})$, $(-1, 0)$ a $(\frac{\sqrt{17}-3}{2}, \infty)$ (funkce $f(x)$ je tedy na těchto intervalech rostoucí) a záporná na intervalech $(-\frac{\sqrt{17}+3}{2}, -1)$ a $(0, \frac{\sqrt{17}-3}{2})$ (na těchto intervalech je funkce $f(x)$ klesající). Funkce $f(x)$ tedy skutečně nabývá v bodech $x = 0$, $(f(0) = 0)$ a $x = -\frac{\sqrt{17}+3}{2}$, $(f(-\frac{\sqrt{17}+3}{2}) = -8{,}82)$ lokálního maxima a v bodě $x = \frac{\sqrt{17}-3}{2}$, $(f(\frac{\sqrt{17}-3}{2}) = -0{,}06)$ nabývá lokálního minima.

Inflexní body, konvexnost, konkávnost

Nyní zjistíme, zda má funkce nějaké inflexní body, tzn. body, ve kterých se mění konvexita funkce na konkávnost nebo naopak. Nejprve najdeme body „podezřelé“ z inflexe, tedy body, ve kterých je druhá derivace funkce $f(x)$ rovna nule.

```
(%i20) solve(d2=0);
```

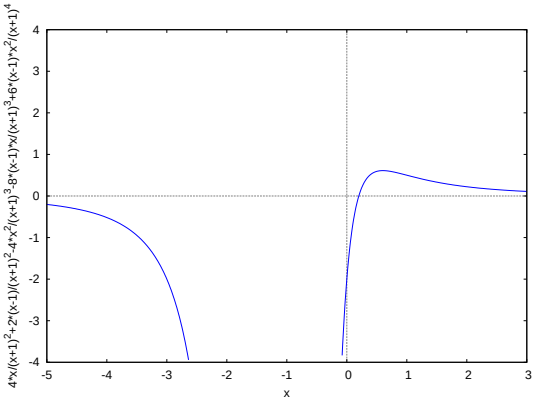
(%o20) $$[x = \frac{1}{5}]$$

```
(%i21) f(1/5);
```

(%o21) $$-\frac{1}{45}$$

Našli jsme jediný takový bod $x = \frac{1}{5}$, $(f(\frac{1}{5}) = -\frac{1}{45})$. Z grafu druhé derivace funkce vyčteme, na kterých intervalech je funkce konvexní a konkávní a jestli je nalezený bod skutečně inflexním bodem.

```
(%i22)plot2d(d2,[x,-5,3],[y,-4,4]);
```



Obrázek 8.3: Graf druhé derivace funkce $f(x)$

Vidíme, že druhá derivace je na intervalech $(-\infty, -1)$ a $(-1, \frac{1}{5})$ záporná a na intervalu $(\frac{1}{5}, \infty)$ kladná. V prvních dvou intervalech je tedy funkce konkávní a ve třetím intervalu je funkce konvexní. Bod $x = \frac{1}{5}$ je inflexním bodem a mění se v něm konkávnost funkce na konvexnost.

Asymptoty bez směrnice

Již dříve jsme zjistili, že do definičního oboru nepatří bod -1. Je tedy možné, že jím prochází asymptota bez směrnice. Vyšetříme chování funkce $f(x)$ v okolí tohoto bodu pomocí příkazu `limit`.

```
(%i23) limit(f(x),x,-1);
```

(%o23) $-\infty$

V bodě $x = -1$ existuje nevlastní limita rovna $-\infty$, a proto přímka $x = -1$ je asymptotou bez směrnice.

Asymptoty se směrnicí

K výpočtu asymptot se směrnicí ve tvaru $y = kx+q$ se užívá následujících vzorců:

$$k = \lim_{x\to\pm\infty} \frac{f(x)}{x} \qquad q = \lim_{x\to\pm\infty} [f(x) - kx]$$

Opět tedy použijeme příkaz `limit`.

```
(%i24) k1:limit(f(x)/x, x, inf);
```

(%o24) 1

```
(%i25) limit(f(x)-k1*x, x, inf);
```

(%o25) -3

```
(%i26) k2:limit(f(x)/x, x, minf);
```

(%o26) 1

```
(%i27) limit(f(x)-k2*x, x, minf);
```

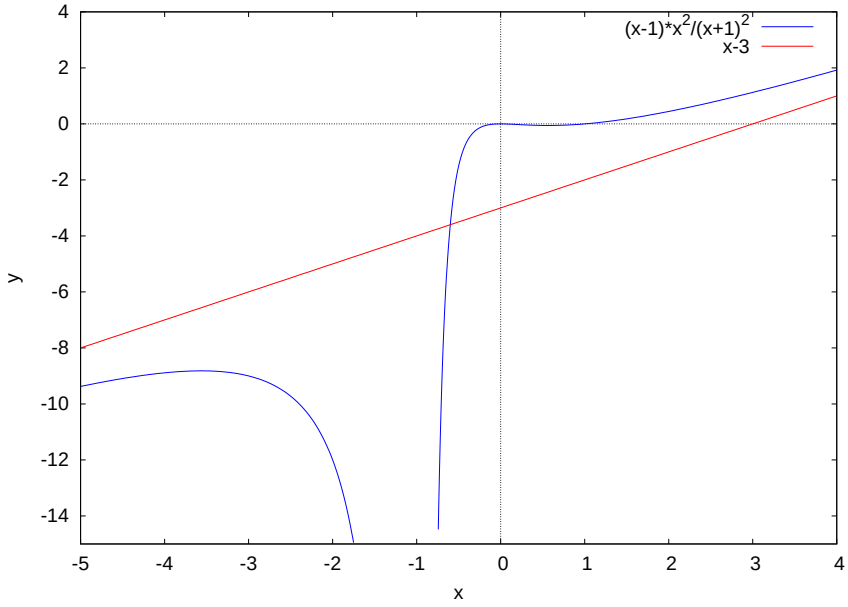
(%o27) -3

V obou případech nám vyšla jediná asymptota se směrnicí daná vzorcem $y = x-3$.

Zobrazení grafu funkce

Na závěr zobrazíme graf funkce $f(x)$. Rozhodneme o vhodných intervalech os x a y. Na ose x nám bude stačit interval $(-5, 4)$, v němž se nacházejí všechny důležité body, a na ose y interval $(-15, 4)$, abychom viděli všechny jejich funkční hodnoty. Spolu s grafem funkce zobrazíme také asymptotu se směrnicí $y = x - 3$. Graf je znázorněn na obrázku 8.4.

```
(%i28) plot2d([f(x),x-3], [x,-5,4], [y,-15,4]);
```



Obrázek 8.4: Graf funkce $f(x)$

Závěr

Systém počítačové algebry Maxima je v dnešní době, kdy klademe velký důraz na využívání počítačové techniky, velmi užitečným matematickým programem. Grafické prostředí programu je přehledné, ovládání relativně jednoduché, manuál programu je obsáhlý a velmi pěkně zpracovaný.

Nedávná lokalizace programu do českého jazyka nabízí pohodlnější ovládání pro českého uživatele. V rámci komentovaných výstupů však program stále komunikuje s uživatelem v anglickém jazyce a také manuál zatím není dostupný v češtině, a proto je k užívání programu nutná znalost odborného anglického jazyka. Program však prochází neustálým vývojem a je tedy pravděpodobné, že se úplného překladu do češtiny brzy dočkáme.

Velkou předností systému Maxima je jeho ovládání, které se moc neliší od ovládání komerčního systému Maple. Právě Maple je na Masarykově univerzitě poměrně rozšířeným matematickým programem. Systém Maxima má však nespornou výhodu v tom, že jde o open source program, a lze jej tedy získat za nulové náklady a používat na jakémkoliv pracovišti i po ukončení studia. Program proto doporučuji jako vhodnou alternativu systému Maple, kterou je možné používat bez jakéhokoliv omezení.

Na přiloženém CD se nacházejí zápisníky ve formátu `.wxm` se všemi použitými příkazy. Zápisníky, které jsou rozčleněny podle jednotlivých kapitol, mohou být vhodnou pomůckou pro začínající uživatele.

Seznam použité literatury

[1] BUŠA, J. *MAXIMA. Open source systém počítačovej algebry* [online]. Košice: Východoslovenské tlačiarne, 2006 [cit. 12. prosince 2010]. Dostupné z WWW: <http://people.tuke.sk/jan.busa/kega/maxima/maxima_brozura.pdf>. ISBN 80-8073-640-5.

[2] DĚMIDOVIČ, B. P. *Sbírka úloh a cvičení z matematické analýzy*. 1. vydání. Havlíčkův Brod: Fragment, 2003. ISBN 80-7200-587-1.

[3] DODIER, R. *Maxima 5.22.1 Manual* [online]. August 2010 [cit. 2. ledna 2011]. Dostupné z WWW:<http://maxima.sourceforge.net/docs/manual/en/maxima.html>.

[4] DOŠLÁ, Z., PLCH, R., SOJKA, P. *Matematická analýza s programem Maple. Díl 1, Diferenciální počet funkcí více proměnných*. Brno: Masarykova Univerzita, 1999. ISBN 80-210-2203-5.

[5] *Maxima, a Computer Algebra System* [online]. [cit. 12. prosince 2010]. Dostupné z WWW:<http://maxima.sourceforge.net/>.