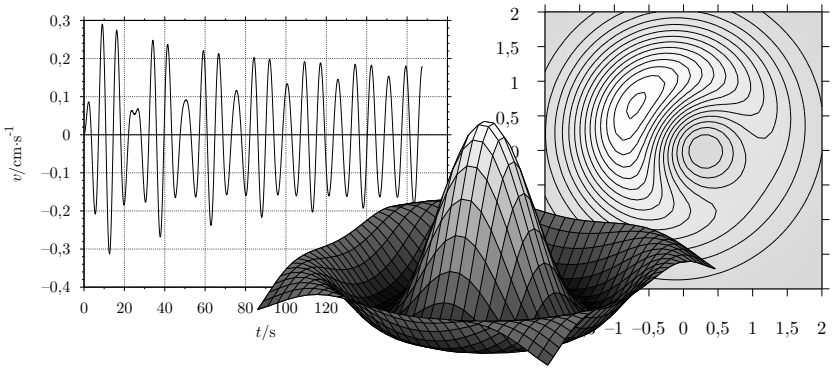
# DYNAMICKÉ MODELOVÁNÍ V PROGRAMU GNU OCTAVE

Lukáš Richterek
Katedra experimentální fyziky PřF UP Olomouc




Poslední úpravy: 18. září 2007

# Obsah

# Úvod

Řešení fyzikální úlohy má obvykle dvě samostatné části – obecnou a numerickou. V obecné části sestavíme na základě vhodných fyzikálních zákonů potřebné rovnice a z nich matematickými úpravami dojdeme ke vztahům, které vyjadřují hledané veličiny pomocí veličin daných a různých konstant. U jednodušších úloh pak v numerické části do těchto vztahů dosadíme číselné hodnoty daných veličin a konstant a získáme číselné hodnoty hledaných veličin, které doplníme příslušnými jednotkami, u složitějších někdy provádíme rozměrovou zkoušku.

V praxi se však často setkáváme se zdánlivě jednoduchými úlohami, jejichž řešení vede k transcendentním nebo diferenciálním rovnicím, jež buď neumíme řešit elementárními prostředky (analyticky), nebo je toto řešení – zejména na úrovni střední školy – velmi obtížné. Takové úlohy pak musíme řešit přibližnými numerickými metodami. Dnes, v době širokého rozšíření osobních počítačů a programovatelných kalkulátorů a bohaté nabídky využitelného software, se otevírají možnosti pro využití těchto metod prakticky všem zájemcům.

Problematika numerického řešení fyzikálních úloh na úrovni střední školy není zcela nová. Tento text proto navazuje na materiály [12, 18, 19] využívající český program *FAMULUS*, který si v minulých letech získal relativně velkou popularitu na školách. Jeho vazba na operační systém MS DOS a hlavně řada programů podobného zaměření vedly k tomu, že již není dále vyvíjen, i když zájemci si jej stále mohou stáhnout a nainstalovat např. z adresy `http://kdf.mff.cuni.cz/veletrh/sbornik/software/software.html`. Cílem toho textu je nabídnout jednu z možných volně dostupných alternativ tohoto programu, konkrétně *GNU Octave* (domovská stránka `http://www.octave.org/`).

Vzhledem k omezenému rozsahu zde nemůžeme nahradit podrobný uživatelský manuál k programu, namísto systematického výkladu se s ním budeme seznamovat prostřednictvím konkrétních příkladů. Zaměříme se na základní přehled zahrnující nejdůležitější vlastnosti programu, jeho výhody a případné nevýhody (na většinu z nich a možná i na další přijdou jistě uživatelé sami) a na konkrétních úlohách si ukážeme několik možností řešení transcendentních i jednoduchých diferenciálních rovnic. Více úloh spojených většinou s fyzikální olympiádou najdou zájemci v diplomové práci [9].

# Část 1

# Program GNU Octave

Jak napovídá sám název programu, je GNU Octave součástí *projektu GNU* (`http://www.gnu.cz/`), který patří k symbolům svobodného software. Zahrnuje celou škálu programů s různým využitím. Připomeňme nejprve základní principy projektu a s ním spojené licenční podmínky, které se vztahují i na software *GNU Octave*.

## 1.1 GNU a GPL

### 1.1.1 Projekt GNU

Projekt GNU byl založen v roce 1984 s cílem vytvořit klon operačního systému UNIX, který by byl šířen jako svobodný software (angl. free software). Odtud pochází i název – GNU je rekurzivní akronym pro „GNU's Not UNIX“ a vyslovuje se [gnů].

Obr. 1.1: Logo GNU projektu

V návaznosti na tento projekt byla v roce 1985 založena *Nadace pro svobodný software* (Free Software Foundation) s cílem podporovat práva uživatelů počítačů používat, studovat, kopírovat, modifikovat a redistribuovat počítačové programy. Nadace podporuje vývoj svobodného softwaru, jmenovitě pak operačního systému GNU. Dnes jsou hojně používány různé varianty operačního systému GNU s kernelem Linux; ačkoliv se těmto systémům často říká Linux, přesnější pojmenování pro ně je GNU/Linux.

Obr. 1.2: Richard Matthew Stallman (převzato z [1])

Základem filosofie projektu GNU je přesvědčení, že svobodný software je záležitost svobody: lidé by měli mít svobodu využívat software všemi způsoby, které přinášejí nějaký společenský užitek. Software se liší od hmotných objektů (židle, sendviče nebo benzínu) v tom, že může být mnohem snadněji kopírován a modifikován a uživatelům se dává možnost – u programů patřících do GNU projektu – tuto výhodu maximálně využít. Zakladatelem a ústřední postavou hnutí je *Richard Matthew Stallman* (obr. 1.2, osobní stránka `http://http://www.stallman.org/`); původně absolvent ba-

kalářského studia fyziky na Harvardově univerzitě je znám především jako autor editoru *GNU Emacs* a překladače gcc.

FSF/UNESCO adresář svobodného softwaru (FSF/UNESCO Free Software Directory, `http://directory.fsf.org/`) dnes zahrnuje více než 4000 balíčků svobodného softwaru poskytuje. Pro podporu šíření svobodného softwaru, GNU projekty i non-GNU projekty, poskytuje FSF FTP servery (`ftp://ftp.gnu.org/`) které jsou zrcadleny po celém světě. Pro vývojáře mnoha GNU i non-GNU projektů poskytuje Free Software Foundation CVS server Savannah (`http://savannah.gnu.org/`).

### 1.1.2 GPL – General Public Licence

Na všechny softwarové balíčky zařazené do projektu GNU (včetně programu *GNU Octave*) se vztahuje *všeobecná veřejná licence GNU* (v originále GNU General Public Licence) označovaná nejčastěji zkratkou *GNU GPL*[6] popřípadě některá z jejích slabších verzí (např. GNU LGPL). Podle svých principů a v kontrastu s běžně známým copyrightem bývá tato licence označována jako *copyleft*. Jejím cílem je zaručit:

- svobodu ke sdílení a úpravám svobodného softwaru;
- právo spouštět program za jakýmkoliv účelem;
- právo studovat, jak program pracuje, a přizpůsobit ho svým potřebám (předpokladem k tomu je přístup ke zdrojovému kódu, nejen k k programu samotnému);
- právo redistribuovat kopie dle svobodné vůle;
- právo vylepšovat program a zveřejňovat zlepšení, aby z nich mohla mít prospěch celá komunita (předpokladem je opět přístup ke zdrojovému kódu).

V tomto smyslu je svobodný (free) software takový, k němuž je k dispozici také zdrojový kód, spolu s právem tento software používat, modifikovat a distribuovat. Naprostá většina svobodného software je zdarma, ačkoliv to podle licence není podmínkou. Svobodný software neznamená nekomerční, komerční vývoj svobodného software není ničím neobvyklým. Za získání kopií svobodného software můžete platit, nebo je obdržet zdarma, ovšem bez ohledu na způsob, jak jste je získali, máte vždy svobodu kopírovat a měnit software, dokonce prodávat nebo darovat jeho kopie nebo pozměněné verze. Copyleft licence tak říká, že pokud redistribuujete originální nebo pozměněnou verzi programu, musíte tuto verzi redistribuovat pod stejnou licencí pod jakou jste získali původní program, nesmíte přidat žádná omezení, abyste tak neodepřeli zmíněné základní svobody ostatním.

Kromě toho bývá zvykem, že jednotlivé programové balíčky mají své vlastní internetové stránky, odkud je nejen možné si program stáhnout, ale uživatelé tam najdou i manuály (standardní bývá angličtina, překlady do dalších jazyků závisí většinou na množství uživatelů a dobrovolných překladatelů v dané jazykové

oblasti) a diskusní mailové skupiny, kam je možné posílat dotazy ohledně používání programu a poučit se na konkrétních problémech u zkušenějších uživatelů či přímo autorů programu. Pro zdatné programátory se také otevírá možnost přímo se na vývoji či vylepšování programu podílet (což dělá i běžný uživatel, pokud hlásí autorům chyby, s nimiž se při své práci setkal – každý, i sebedokonalejší program je koneckonců jen lidským výtvorem).

S pojem svobodného software volně souvisejí (a někdy jsou ne zcela správně zaměňovány) také *freeware* a *open source*. Termín „freeware“ nemá jasnou a přijatelnou definici, ale je běžně používán pro balíky programů, u kterých je dovolena distribuce a používání, ale ne modifikace (zdrojové kódy nejsou dostupné), mezi svobodný software proto nepatří. Termín „open source“ je často používán k označení víceméně týchž věcí jako svobodný software, i když jeho licence nemusí zaručovat tolik svobod jako GPL. Podrobnější popis jednotlivých kategorií najdeme např. na stránkách `http://www.gnu.org/philosophy/categories.cs.html`.

Jak již bylo zmíněno, do kategorie svobodného software dnes patří celá řada programu, které právě díky své volné dostupnosti mohou být právě ve školství vhodnou alternativou komerčně prodávaných proprietárních programů. Zmiňme např. program pro vykreslování a znázorňování dat *GNUplot* (domovská stránka `http://www.gnuplot.info/`, ukázky grafů `http://gnuplot.sourceforge.net/demo/`), který používá ke grafickému znázornění i *GNU Octave*, balík pro symbolické výpočty (tzv. počítačová algebra) *GNUMaxima*[1] (domovská stránka `http://maxima.sourceforge.net/`) nebo program pro grafické úpravy obrázků a fotografií *Gimp* (GNU Image Manipulation Program; domovská stránka `http://www.gimp.org/`, české stránky `http://www.gimp.cz`).

## 1.2 Program GNU Octave

*GNU Octave* představuje vyšší programovací jazyk („jazyk pro neprogramátory“) zaměřený na numerické operace podobný poměrně rozšířenému *Matlabu*® firmy The Mathworks (`http://www.mathworks.com/`). Jeho autor *John W. Eaton* – počítačový administrátor skupiny chemického inženýrství na University of Wisconsin, která se mimo jiné zabývá i návrhy chemických reaktorů

[1] *Maxima* vychází z projektu *Macsyma*, jenž byl vyvíjen v MIT (Massachusetts Institute of Technology) a financován United States Department of Energy a dalšími vládními organizacemi. O vývoj jedné z verzí Macsyma se staral od roku 1982 až do své smrti v roce 2001 Bill Schelter, jenž v roce 1998 získal svolení uveřejnit svou verzi pod GPL. Tuto verzi nazývanou *Maxima* nyní udržuje nezávislá komunita vývojářů a uživatelů.

– tento jazyk vytvořil roce 1988 jako pomůcku pro studenty, aby nemuseli zvládnout běžné programovací jazyky Fortran nebo C++. Samotné jméno „Octave“ prý pochází ze jména autorova bývalého učitele, který proslul svou schopností dělat rychlé kalkulace na kouscích papírků. Program je podobně jako *Matlab*® založen na efektivním počítání s maticemi, pracovat můžeme buďto interaktivně (postupné zadávání příkazů) nebo spouštěním delších předem připravených skriptů. Historie zadaných příkazů se zaznamenává a je možné se k nim postupně vracet nebo si historii uložit jako hotový skript. Zadávat lze i komplexní čísla, k dispozici jsou dále goniometrické, hyperbolické funkce, logaritmy, logické funkce, cykly, statistické funkce, polynomy, algoritmy pro numerické řešení diferenciálních rovnic, numerickou integraci, konverzi (a rozklad) obrazových i audio dat, zpracování signálů atd. Jak již bylo řečeno, pro zobrazování grafických výsledků se automaticky volá program *GNUplot*, z něhož lze obrázek uložit v různých formátech (postscript, png apod.).

V současné době (září 2007) jsou k dispozici dvě verze programu – stabilní (číslo 2.1.xx) používaná v tomto textu a beta-verze (číslo 2.9.xx), která je zejména v oblasti práce s 2D i 3D grafikou bližší *Matlabu*® a lze ji v omezené podobě spouštět přímo z CDROMů nebo flashdisků. Na důležité rozdíly v textu upozorníme.

## 1.3 Proč (ne)používat GNU Octave

Volba programu, který bude používat, patří vždy k výsostným záležitostem uživatele. V oblasti matematicky orientovaných programů je dnes k dispozici velké množství balíků, od komerčních proprietárních (např. *Matlab*®, *Maple*® nebo *Mathematica*®) až pro freeware nebo svobodný software (např. *Scilab* nebo *GNU Maxima*). Nabízí se proto otázka, jaké výhody či nevýhody jsou spojeny s používáním programu *GNU Octave* a proč si (ne)vybrat právě tento balík. Kromě ceny (je zdarma, takže oproti výše zmíněným proprietárním programům vychází jedna licence asi o 30 000,- Kč levněji) bychom měli připomenout a zvážit následující klady („+“) a zápory („−“):

- − K efektivnímu používání musíme zvládnout alespoň základy programovacího jazyka, což ale platí pro většinu matematického software.
- − Základem výpočtů jsou manipulace s maticemi, což v některých situacích vyžaduje zvláštní syntax (více v následující kapitole).
- − Program nemá propracovaná grafické prostředí a proto „není moc o klikání“; většinu úkonů ovládáme pomocí klávesnice z příkazové řádky.
- ± *GNU Octave* není výukový program, proto se nezaměřuje ani na grafické ovládání či pohyblivé simulace. Tím se na druhé straně více blíží programům často užívaným v reálné technické praxi.

± Využití programu se neomezuje zdaleka jen na dynamické modelování, jemuž je převážně věnován tento text. Díky tomu se může zdát pro jeden daný účel použití příliš robustní a složitý, na druhou stranu – pokud si na program už jednou zvykneme – můžeme v jednou prostředí řešit hodně problémů.

± Jde o mezinárodní projekt, většina materiálů a manuálů i diskusní skupina používá angličtinu, jejíž znalost je bezesporu výhodou. Na druhou stranu je velmi pravděpodobné, že v široké komunitě někdo řešil a už vyřešil podobný problém jako my a bude nám umět poradit. Větší počet uživatelů také rychleji odhalí případné chyby a nedostatky programu.

\+ Programovací jazyk používaný *GNU Octave* má syntaxi velmi podobnou (i když ne 100% kompatibilní) v praxi široce používanému *Matlabu*®. Pokud si náš student zvykne na základní principy a filozofii, případný pozdější přechod mezi programy není obtížný.

\+ *GNU Octave* je multiplatformním programem, lze ho používat jak v operačním systému Windows, tak ve většině verzí Linuxu (v řadě distribucí jsou i předkompilovány instalační balíčky). Přiznejme, že používání pod Linuxem je v současné době pohodlnější.

\+ Základní manuály – na rozdíl např. od Scilabu (pokud je nám známo) – jsou k dispozici i v češtině ([11, 15]).

\+ S použitím *GNU Octave* se setkáme na řadě míst ve Wikipedii [1] a pokud s programem umíme pracovat, můžeme si přímo spustit skripty, které tam nalezneme.

V současné době roste popularita java-appletů a také simulačních programů typu *Interactive Physics*™ (domovská stránka v angličtině má adresu `http://www.design-simulation.com/IP/index.php`, informace v češtině najdeme např. na `http://www.ictphysics.upol.cz/` nebo `http://www.gjwprostejov.cz/projekty/sipvz03/`). Zatímco pro tvorbu vlastních appletů musíme zvládnout objektově orientovaný programovací jazyk *Java* vyvinutý firmou Sun Microsystems se vším, co k tomu patří, výhodou „interaktivní fyziky“ je bezesporu propracované grafické prostředí umožňující poměrně snadno vytvářet velmi působivé simulace reálných fyzikálních dějů. Jak již bylo řečeno v první části textu, řada předdefinovaných faktorů přitom jednak částečně omezuje oblast použití a jednak část fyzikálních vlastností zůstává před zvídavým žákem skryta. Proto jsme přesvědčeni, že i dnes má smysl se dynamickým modelováním zabývat, neboť při něm máme pod kontrolou všechny parametry a procedury. A hlavně – radost z dobře napsaného a fungujícího skriptu určitě stojí za to!

# Část 2

# První kroky s programem

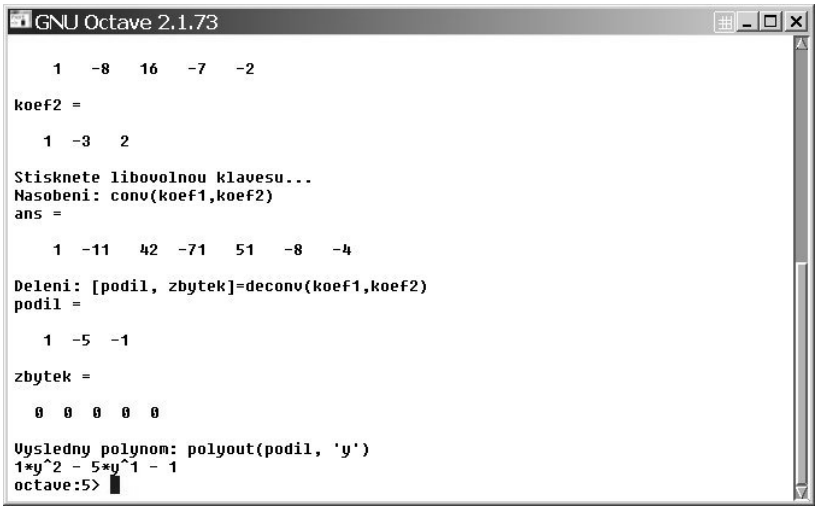
## 2.1 Příkazový řádek a terminál

Již jsme zmínili, že základním pracovním prostředím programu je terminál s příkazovým řádkem (obr. 2.1), pomocí něhož zadáváme programu jednotlivé příkazy. K jednou zadaným příkazům se můžeme vracet pomocí klávesy „se šipkou nahoru“, takže je není nutné vypisovat znovu. Terminál je také schopen doplňovat jména příkazů, definovaných funkcí a jmen souboru v aktuálním adresáři pomocí klávesy Tab (tabulátor); pokud např. napíšeme

```
octave:1> ta
```

a stiskneme tabulátor, program nabídne možná doplnění

```
table       tabulate     tan          tanh        taylorcoef
```




Obr. 2.1: Terminálové okno s příkazovým řádkem ve Windows XP

I když je možné historii příkazů ukládat, delší posloupnosti příkazů se bezesporu vyplatí uložit do souboru a volat všechny příkazy v tomto souboru (tj.

celý uložený skript) jedním příkazem, odpovídajícím jménu uloženého souboru. Podobně jako u *Matlabu*® soubory ukládáme s příponou `*.m`, seznam souboru v aktuálním adresáři zjistíme příkazem[1]

```
octave:3> ls
```

jehož odpověď může vypadat např. následovně

```
baliste.m     grafika.m   liss.m      micek.m      pruzkyv.m
balist.m      grafika2.m  lyzarfv.m   oscilb.m     raketa.m
foucault.m    haleboop.m  matice.m    parasut.m    skladkmitf.m
```

Soubory s příponou `*.m` můžeme přímo spustit, z výše uvedeného výpisu třeba

```
octave:4> foucault
```

spustí model kmitů Foucaultova kyvadla popsaný v části 4.3.2. Aktuální adresář, v němž momentálně *GNU Octave* pracuje, získáme příkazem

```
octave:5> pwd
```

a do jiného adresáře (složky) se přepneme příkazem[2]

```
octave:6> cd
```

K psaní či následné editaci vlastních skriptů lze využít většinu textových editorů, např. *Poznámkový blok* (Notepad), který je součástí instalace operačního systému Windows. Píšeme-li však skriptů více, patrně oceníme možnosti pokročilejších editorů, které umí číslovat řádky, barevně odlišovat komentáře, příkazy, funkce i číselné konstanty (tzv. syntax highlighting neboli zvýrazňování syntaxe). většinu těchto editorů lze využít i pro jiné programovací jazyky, psaní dokumentů v TEXu nebo editaci kódu www-stránek. I v této kategorii je nabídka freewarových programů poměrně široká – přímo s instalací *GNU Octave* pro Windows se nainstaluje editor *Scite* (`http://www.scintilla.org/SciTE.html`), z dalších stojí za zmínku např. původem český *PSPad* autora Jana Fialy (`http://www.pspad.com/cz/`), který lze do systému i integrovat jako náhradu až příliš jednoduchého Poznámkového bloku. U obou zmíněných programů lze nastavit řadu parametrů, včetně fontů a barev tak, jak to uživateli subjektivně nejlépe vyhovuje.[3] Nastavíme-li, aby se daný typ souboru otevíral vždy ve vybraném editoru, můžeme se pustit do psaní prvních programů.

[1] Modifikací `ls -l` získáme podrobný výpis souborů s jejich velikostí a datem poslední změny.

[2] I k němu existuje řada možných parametrů např. `cd /` přepíná do hlavního adresáře (složky), jimž v OS Windows většinou bývá disk `C:`. Obsahují-li jména adresářů mezery, musíme dát název do uvozovek, tj. `cd "/Documents and Settings"`.

[3] V současné verzi PSPadu používáme stejný zvýrazňovač pro *GNU Octave* i *Matlab*® označený jako MATLAB13.

## 2.2 Základní funkce a příkazy

Nyní jsme připraveni vyzkoušet první kroky s programem. Podobně jako *Matlab*® vypisuje *GNU Octave* čísla v různých formátech. V základním nastavení vypisuje dvě notoricky známé základní konstanty ve tvaru

```
octave:1> pi, e
pi = 3.1416
e = 2.7183
```

Chceme-li si výstup na více desetinných míst, použijeme

```
octave:2> format long
octave:3> pi, e
pi = 3.14159265358979
e = 2.71828182845905
```

k původnímu nastavení se vrátíme pomocí `format short`. Zdůrazněme, že tím *neovlivňujeme* přesnost, s jakou *GNU Octave* používá čísla při výpočtech. Maximální počet desetinných míst, která můžeme zobrazit je 42[4]; dosáhneme toho příkazem.

```
octave:4> printf ("%.42f\n", pi);
```

kde `\n` značí přechod na nový řádek po vykonání příkazu. Podobně si můžeme nechat vypsat např. $\sqrt{10}$

```
octave:5> printf ("%.42f\n", sqrt(10));
3.162277660168379522787063251598738133907318
```

Základními objekty, s nimiž program efektivně pracuje, jsou matice, i číselné konstanty chápe jako matice s jedním prvkem. Jednořádkovou matici (řádkový vektor) zadáme ve tvaru

```
octave:6> m=[1 2 3 4 5 6]
m =

  1  2  3  4  5  6
```

[4]Chceme-li např. Ludolfovo číslo $\pi$ na 1 000 desetinných míst, musíme použít program pro symbolické výpočty (např. *GNU Maxima*) nebo alespoň nainstalovat k programu *GNU Octave* knihovnu pro symbolické výpočty, s níž bude v takových případech spolupracovat (např. *GiNaC*). V beta-verzi *GNU Octave* 2.9.xx tuto knihovnu najdeme, v linuxovských distribucích většinou stačí doinstalovat balíček `octave-forge`. Potom lze použít: `digits(1000);` a `Pi`.

matici k ní transponovanou (tj. sloupcový vektor)

```
octave:7> m'
ans =

  1
  2
  3
  4
  5
  6
```

což lze dosáhnout i přímo zadáním (řádky matice ukončujeme středníkem)

```
octave:8>  m=[1; 2; 3; 4; 5; 6]
m =

  1
  2
  3
  4
  5
  6
```

Ukončíme-li řádek středníkem, matice se vytvoří, ale nevypíše na obrazovce, což je užitečné u velkých matic s mnoha prvky

```
octave:9>  m=[1; 2; 3; 4; 5; 6];
```

Vyjdeme-li z původní řádkové matice `m=[1 2 3 4 5 6]`, můžeme násobení řádkové a sloupcové matice zapsat

```
octave:10> m*m'
ans = 91
```

z matematického hlediska jde o součet $\sum_i m_i^2$. Protože u násobení matic závisí na pořadí (není komutativní), záměnou matice $m$ a matice transponované získáme matici

```
octave:11> A=m'*m
A =

   1   2   3   4   5   6
   2   4   6   8  10  12
```

```
   3   6   9  12  15  18
   4   8  12  16  20  24
   5  10  15  20  25  30
   6  12  18  24  30  36
```

neboli matici s prvky $A_{ij} = m_i m_j$. *Při násobení a umocňování matic proto musíme dát pozor!* Chceme-li umocnit prvky matice např. na druhou, *nemůžeme napsat* pouze `m^2` – chybová hláška nám napovídá, že *GNU Octave* se snaží umocnit matici jako celek a to lze jen v případě čtvercové matice

```
octave:12> m^2
error: for A^b, A must be square
error: evaluating binary operator '^' near line 11, column 2
```

Chceme-li získat matici s jednotlivými prvky $m_i^2$ (tj. umocněnou „člen po členu“), musíme napsat

```
octave:12> m.^2
ans =

   1   4   9  16  25  36
```

nebo pomocí násobení `m.*m`. *Toto opomenutí bývá příčinou mnoha chybových hlášek*, u čtvercových matic, kdy se chybová hláška neobjeví, může vést ke špatnému výsledku. Dodejme, že na stejný zápis a „nepohodlí“ si musí zvyknout i uživatelé *Matlabu*®.

Kromě tištěných manuálů či návodů na internetu můžeme základní návody a informace získat přímo pomocí programu příkazem

```
octave:13> help
```

hledáme-li pomoc ke konkrétnímu příkazu, připojíme jeho jméno

```
octave:14> help plot
```

V terminálu se nám objeví popis, jímž můžeme postupně listovat, opustíme ho klávesou „q“, po jejímž stisknutí můžeme zadávat další příkazy. V případě potřeby můžeme výpis terminálového okna vymazat příkazem

```
octave:15> clc
```

nadefinované konstanty a proměnné vymažeme příkazem

```
octave:16> clear
```

pouze matici `m` vymažeme příkazem `clear m`. Hlášky vypisované na obrazovku a komunikující s uživatelem zadáváme pomocí příkazu `printf` nebo `fprintf`, např.

```
octave:17> fprintf ("Stisknete libovolnou klavesu...\n");
```

Význam to má zejména, pokud připravujeme programy, jež bude spouštět i někdo jiný (třeba naši studenti). Chceme-li v duchu předchozí ukázky do skriptu zařadit přerušení běhu programu s čekáním na stisk libovolné klávesy uživatelem, použijeme `pause`.

## 2.3 Některé matematické funkce

Z hlediska středoškolské matematiky a fyziky má *GNU Octave* předdefinováno několik zajímavých funkcí a procedur. Např. řešení soustavy lineárních rovnic

$$\begin{aligned} 4x + 3y - 2z &= 40 \\ 6x - 5y - 3z &= 50 \\ 3x + 2y + 5z &= 220 \end{aligned}$$

snadno najdeme pomocí matice soustavy

```
octave:1> A=[4 3 -2;6 -5 3;3 2 5]
A =

   4   3  -2
   6  -5   3
   3   2   5
```

a matice sestavené z pravých stran

```
octave:2> B=[40;50;220]
B =

   40
   50
  220
```

podílem matic

```
octave:3> A\B
ans =

  10.000
  20.000
  30.000
```

určujícím řešení $x = 10$, $y = 20$ a $z = 30$. Na tomto lze demonstrovat dvojí typ dělení matic. Výše použitá operace `A\B`[5] matematicky odpovídá násobení inverzní maticí zleva $A^{-1}B$. Běžně užívaný znak dělení `A/B` pak reprezentuje výraz $AB^{-1}$, který ovšem pro naše konkrétní matice nelze vypočítat pro nekompatibilní počet řádků a sloupců matic $A$ a $B$. Rozdíl mezi oběma děleními pak ukážeme na pomocí matice $B$, srovnáme-li výrazy[6]

```
octave:4> B\B, B/B
ans = 1.0000
ans =

  0.030476  0.038095  0.167619
  0.038095  0.047619  0.209524
  0.167619  0.209524  0.921905
```

*Podobně jako při násobení musíme dávat pozor, chceme-li dělit jednotlivé prvky matice, kde je opět třeba vložit tečku*

```
octave:5> B./B
ans =

  1
  1
  1
```

Na matici $A$ si můžeme ukázat výběr jejích jednotlivých prvků. Např. prvek $A_{23}$ vyvoláme příkazem,

```
octave:6> A(2,3)
ans = 3
```

[5]Znak \ používáme také v jiném významu – rozdělujeme jím příkaz na více řádků, pokud je z nějakých důvodů jejich délka omezena jako např. při sazbě tohoto textu. Potom za ním ale následuje přechod na nový řádek.

[6]Dodejme, že pojem inverzní matice zavádíme přesně vzato pouze u matic čtvercových, z ukázky vidíme, že dvojí typ dělení umožňuje *GNU Octave* i pro některé matice jiného typu.

3. řádek a 2. sloupec pak postupně

```
octave:7> A(3,:), A(:,2)
ans =

  3  2  5

ans =

   3
  -5
   2
```

Součet řádkové nebo sloupcové matice (vektoru) získáme jednoduše příkazem

```
octave:8> sum(B)
ans = 310
```

podobně determinant matice $A$

```
octave:9> det(A)
ans = -241
```

Na příkladu kvadratické $4x^2 + x - 3 = 0$ rovnice můžeme demonstrovat schopnost hledat kořeny polynomu. Z koeficientů polynomu sestavíme matici v pořadí od nejvyššího k nejnižšímu

```
octave:10> koeficienty=[4 1 -3]
koeficienty =

   4   1  -3
```

Kvadratický polynom na levé straně můžeme pro kontrolu nechat vypsat ve zvolené proměnné (v tomto případě $x$)

```
octave:11> polyout(koeficienty, 'x')
4*x^2 + 1*x^1 - 3
```

Hledané řešení – kořen polynomu – vrací funkce matice koeficientů

```
octave:12> roots(koeficienty)
ans =

  -1.00000
   0.75000
```

Získáváme tak řešení $x_1 = -1$ a $x_2 = 0{,}75$.

Řešit můžeme i úlohu opačnou – najít mnohočlen ke známým kořenům. Např. pro hodnoty $x_1 = -0{,}7$, $x_2 = 0{,}9$ jsou kořeny polynomu s koeficienty

```
octave:13> poly([-.7, .9])
ans =

   1.00000  -0.20000  -0.63000
```

tj. $x^2 - 0{,}2x - 0{,}63$.

Program umožňuje nalézt jak součin, tak podíl dvou polynomů opět pomocí matic jejich koeficientů. Např. pro polynomy $x^4 - 8x^3 + 16x^2 - 7x - 2$ a $x^2 - -3x + 2$ vytvoříme matice `koef1=[1 -8 16 -7 -2]; koef2=[1 -3 2];`. Koeficienty součinu obou polynomů získáme příkazem

```
octave:15> conv(koef1,koef2)
ans =

    1  -11   42  -71   51   -8   -4
```

a koeficienty podílu

```
octave:16> podil=deconv(koef1,koef2)
podil =

   1  -5  -1
```

Poté můžeme vypsat i odpovídající polynom např. v proměnné $y$

```
octave:17>  polyout(podil,'y')
1*y^2 - 5*y^1 - 1
```

neboli $y^2 - 5y - 1$.

## 2.4 Práce se soubory

Nutnou podmínku práce s jakýmkoli programem představuje možnost ukládat informace do souborů na pevném disku, síti či přenosném médiu. Některé základní funkce jako zjištění aktuálního adresáře či výpis souborů v něm již byly zmíněny v části 2.1. Nyní zadefinujeme matici prvků po jedné od 1 do 10 (tj. s krokem rovným 1); protože pro nás bude výhodnější sloupcová matice namísto řádkové, použijeme operaci transponování matice `'`

```
octave:1> mereni=[1:1:10]'
mereni =

   1
   2
   3
   4
   5
   6
   7
   8
   9
  10
```

Nyní k téže matici přidejme sloupec s (pseudo)náhodnými hodnotami okolo 5. Použijeme k tomu generátor (pseudo)náhodných čísel `rand(i,j)`, který vrací matici $i \times j$ pseudonáhodných čísel v intervalu $\langle 0{,}1 \rangle$, vynásobením číslem $0{,}1$ docílíme toho, že čísla v druhém sloupci se budou od 5 lišit až na druhém desetinném čísle. Výsledná dvousloupcová matice tak připomíná výsledek deseti měření nějaké veličiny (tím jsme se inspirovali i pro název proměnné)

```
octave:2> mereni=[mereni [5+.1*rand(10,1)]]
mereni =

   1.0000   5.0124
   2.0000   5.0323
   3.0000   5.0559
   4.0000   5.0607
   5.0000   5.0124
   6.0000   5.0832
   7.0000   5.0341
   8.0000   5.0759
   9.0000   5.0659
  10.0000   5.0149
```

K téže matici můžeme přidat další sloupec (dvěma náhodnými čísly zvětšujeme náhodné „rozmazání“ hodnot ve třetím sloupci)

```
octave:3> mereni=[mereni [9.81*12.5+0.05*rand(10,1)-0.07*rand(10,1)]]

mereni =

    1.0000    5.0124  122.6048
```

```
    2.0000    5.0323  122.6408
    3.0000    5.0559  122.5813
    4.0000    5.0607  122.6648
    5.0000    5.0124  122.6527
    6.0000    5.0832  122.6507
    7.0000    5.0341  122.5876
    8.0000    5.0759  122.6040
    9.0000    5.0659  122.6187
   10.0000    5.0149  122.6318
```

Získanou matici `mereni` nyní uložíme do souboru `mereni.txt` jako čistý text (ne v binárním formátu), aby ji bylo možné načíst i jiným programem (např. tabulkovým kalkulátorem *Excel*)

```
octave:4> save -text mereni.txt mereni
```

Po vymazání proměnné z paměti

```
octave:5> clear mereni
```

ji můžeme načíst ze souboru (jméno načtené proměnné bude odpovídat jménu souboru bez přípony)

```
octave:6> load mereni.txt
```

Zkonstruovanou matici využijeme k ukázce možnosti, jak s pomocí *GNU Octave* zpracovat výsledky měření. Podíváme-li se na hodnoty v matici, vidíme, že první sloupec by mohl odpovídat číslu měření, druhý času $t \approx 5\,\mathrm{s}$ a třetí dráze volného pádu $s \approx gt^2/2$. Vybereme-li z načtené matice jednotlivé sloupce a uložíme je do proměnných

```
octave:7> pokus=mereni(:,1); cas=mereni(:,2); draha=mereni(:,3);
```

lze vypočítat aritmetické průměry $\bar{x} = 1/n \sum_{i=1}^{n} x_i$

```
octave:8> [mean(cas) mean(draha)]
ans =

    5.0448  122.6237
```

odchylky $x - \bar{x}$

```
octave:9> [center(cas) center(draha)]
ans =
```

```
  -0.0323554  -0.0189170
  -0.0124431   0.0171056
   0.0111785  -0.0424311
   0.0159147   0.0410511
  -0.0323656   0.0289959
   0.0384046   0.0269348
  -0.0106577  -0.0361146
   0.0310966  -0.0197190
   0.0211222  -0.0050090
  -0.0298949   0.0081032
```

a standardní odchylky $\sqrt{\sum_{i=1}^{n}(x-\bar{x})^2/(n-1)}$

```
octave:10> [var(cas) var(draha)]
ans =

  0.00072476  0.00083179
```

S problematikou měření souvisí i regrese – hledání závislosti mezi veličinami. Zatímco např. *Excel* standardně umožňuje pouze regresi lineární, *GNU Octave* umožňuje veličinami proložit polynom požadovaného stupně, v naší ukázce zvolíme 2. Zadefinujme nejprve veličiny $x$ a $y$ jako jednořádkové matice (vektory)

```
octave:11> x=1:1:10, y=x.^2+x+3+0.5*rand(1,10)-0.3*rand(1,10)
x =

   1   2   3   4   5   6   7   8   9  10

y =

 Columns 1 through 8:

    5.1096    9.1916   15.0361   23.0593   33.1781
                                  44.8745   59.1035   75.3419

 Columns 9 and 10:

   92.9817  112.8734
```

Koeficienty kvadratického polynomu vyhovující podmínce nejmenších čtverců najdeme příkazem

```
octave:12> polyfit(x,y,2)
```

```
ans =

  0.99685  1.02112  3.08015
```

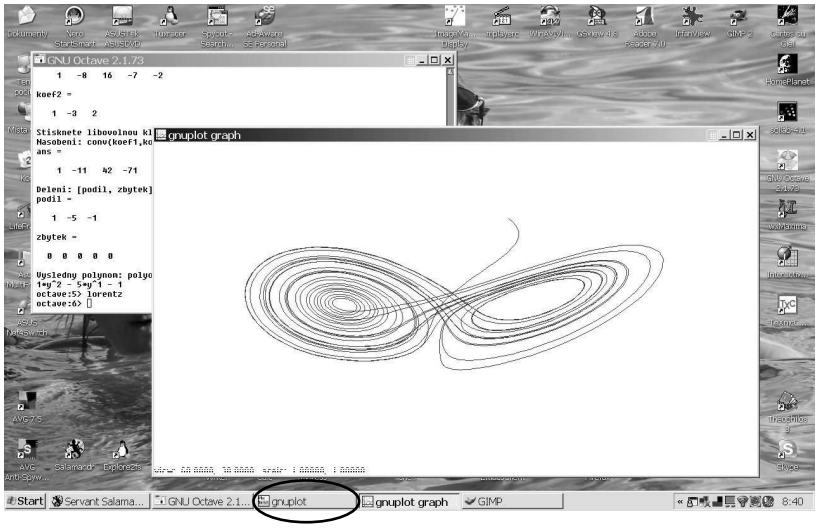
Nejlepší aproximací 2. stupně je tedy polynom $y = 0{,}99685x^2 + 1{,}02112x + + 3{,}08015$.

Na závěr uveďme, jak lze vypsat např. posledních 10 zadaných příkazů

```
octave:13> history 10
```

## 2.5 Kreslíme obrázky

Nezbytnou součástí dynamického modelování je i vykreslení vypočtených hodnot a funkčních závislostí. Věnujme se proto na závěr kapitoly základním příkazům dvourozměrné a třírozměrné grafiky.




Obr. 2.2: Okno s obrázkem vykresleným programem GNUplot

Jak již bylo řečeno, *GNU Octave* pro vykreslování grafů automaticky volá *GNUplot*, v OS Windows se na panelu ve spodní části obrazovky objeví dvě nové položky (viz. obr. 2.2). Zatímco okno s vlastním grafem můžeme bez problémů zavřít a při dalším požadavku se nám vykreslí nový graf (nebo překreslí stávající

s novými parametry), druhé příkazové okno samotného *GNUplotu* (na obr. 2.2 je vyznačeno) *zavřít nesmíme*; pokud se to stane, přeruší se komunikace (pipeline) mezi *GNU Octave* a *GNUplotem* a žádný další obrázek se již nevykreslí – je nutno *GNU Octave restartovat*.[7]

### 2.5.1 Dvourozměrná grafika

Jednoduchý graf funkce $y = \sin x$ pro $x \in \langle 0{,}10 \rangle$ vykreslíme pomocí příkazů[8]

```
octave:1>  x=0:pi/4:10; plot(x,sin(x))
```

Nejprve definujeme hodnoty nezávisle proměnné $x$ od 0 do 10 s krokem $\pi/4$ a poté vykreslíme příslušnou závislost. Pokud se nám jako v tomto případě zdá graf příliš hranatý (program spojuje jednotlivé body úsečkami, i když lze použít i splajny), musíme zjemnit dělení např. zjemněním kroku na $\pi/100$. Rozsah nezávisle proměnné můžeme zadat i volbou počáteční a konečné hodnoty spolu s počtem dělicích bodů. Chceme-li např. výše použitý interval rozdělit na sto intervalů (mezi 101 body), použijeme

```
octave:2>  x=linspace(0,10,101); plot(x,sin(x))
```

K příkazu plot můžeme zadat i další nepovinné parametry, např. vykreslit i body se spočítanými funkčními hodnotami parametrem `-o` (což může být užitečné při znázorňování diskrétních dat)

```
octave:3>  x=linspace(0,10,11); plot(x,sin(x),'-o')
```

nebo změnit barvu

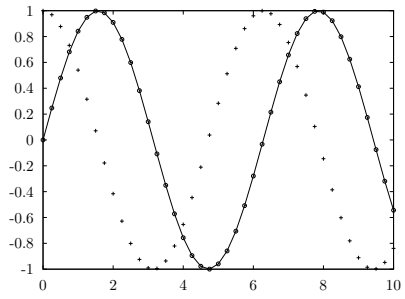
```
octave:4>  x=linspace(0,10,11); plot(x,sin(x),'-b')
```

popř. vykreslit několik grafů najednou s různými parametry pro každý z nich

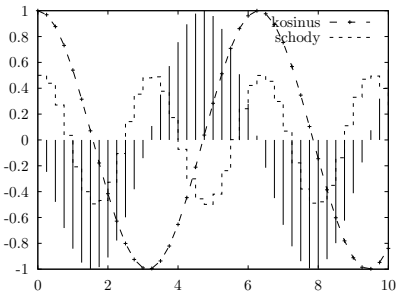
```
octave:5>  x=linspace(0,10,41);
octave:6>  plot(x,sin(x),'-*',x,cos(x),'ob')
```

výsledek vidíme na obr. 2.3, jinou volbu

```
octave:7> plot(x,-sin(x),'^',x,cos(x),'-ob;kosinus;', \
x,-sin(x).^2+.5,'mL;schody;')
```




Obr. 2.3: Dva grafy v jednom obrázku s různými parametry




Obr. 2.4: Totéž s jinou volbou parametrů příkazu plot

pak na obr. 2.4. Nezadáme-li parametry žádné, budou grafy vykresleny čarami (úsečkami spojujícími jednotlivé body), jejichž barva, resp. typ přerušované čáry u černobílého výstupu, se budou automaticky pro jednotlivé čáry lišit v závislosti na typu výstupu (obrazovka, postscriptový soubor, obrázek png apod.)[9] Každému lze určitě doporučit, aby se nebál s volbami experimentovat a sám si našel kombinaci parametrů, která jemu nejvíce vyhovuje. Z předchozích ukázek vidíme, že jimi pro každý z grafů můžeme definovat:

- *typ grafu* čárový (výchozí, `-`), tečkovaný (`.`), schodovitý (`L`) nebo vynášecí (`^`).
- *barvu* a to buď písmenem `r`, `g`, `b`, `m`, `c` či `w` nebo čísly 1–6, tj. v uvedeném pořadí červená (red), zelená (green), modrá (blue), purpurová (magenta), azurová (cyan) a bílá (white). Vyzkoušením se přesvědčíme, že i číslům 7–9 jsou barvy přiřazeny ovšem bez komentáře v manuálu.
- *bodový graf* neboli vykreslení bodů buď přímo uvedením symbolu, který se má v daném bodě vykreslit jako `*`, `+`, `o`, `x`, nebo dvojciferným číslem, v němž první cifra určuje barvu a druhá symbol, opět lze experimentovat s čísly 1–9). Z uvedených ukázek vidíme, že vyznačení bodů lze kombinovat s vykreslením čar, např. `-*`.

[7] V operačním systému Linux se toto druhé okno vůbec neotevírá, pouze samotný graf, takže se s tímto problémem nesetkáme.

[8] Z důvodu úspory místa zde nebudeme reprodukovat všechny grafy, předpokládáme, že čtenář má možnost si výstup jednotlivých příkazů bezprostředně ověřit přímým zadáním do programu.

[9] Nastavení terminálu, tj. typu čar a bodů lze zjistit v programu *GNUplot* příkazem `test`.

- *popisek* uvádíme mezi středníky, např. `;kosinus;`, středník na konci popisku nesmí chybět. Výchozí popisky mají tvar „line 1“, „line 2“ atd.

Pokud je navíc zapnut mód `mouse` (v OS Windows automaticky ve výchozím nastavení), vidíme aktuální hodnoty souřadnic bodů, nad nimiž se pohybujeme myší, v levém dolním rohu obrazovky. Stisknutím prostředního tlačítka můžeme značku zvoleného bodu s těmito hodnotami přidat do grafu, tahem při stisknutém pravém tlačítku můžeme vybrat detailní výběr nějaké části grafu (pak ale před dalším obrázkem musíme anulovat veškerá nastavení obrázku). Tento doplněk, dostupný pouze v novějších verzích, lze kdykoli zapnout příkazem `__gnuplot_set__ mouse` a naopak inaktivovat `__gnuplot_set__ nomouse`.

Chceme-li vykreslené obrázky vytisknout popř. zařadit do textu, uvítáme možnost měnit vzhled os, jejich popisek, poměru stran obrázku apod. U řady obrázků může být kód (posloupnost příkazů) pro tyto změny stejně dlouhý jako kód pro samotný výpočet hodnot. Uživatelé *Matlabu*® mají k dispozici některé příkazy typu `axis`, nejvíce možností prozatím skýtá zadávání příkazu samotného *GNUplotu*, které bývají uvozeny `__gnuplot_set__` nebo `__gnuplot_raw__`. V našem textu se omezíme jen na několik nejdůležitějších možností, podrobný popis by nutně suploval manuál k programu GNUplot (viz např. [25]). Dodejme, že ve starších verzích programu se namísto uvedených příkazů používal tvar `gset` resp. `graw`, které sice zůstávají stále implementovány, ale při jejich prvním volání nás varovná hláška upozorní, že se s nimi v dalším vývoji již nepočítá a pro začínajícího uživatele nemá tudíž smysl si na ně zvykat. Naopak v beta-verzi 2.9.xx již nejsou k dispozici ani příkazy typu `__gnuplot_set__` a případné omezené úpravy lze provézt výhradně pomocí funkcí `axis`, `grid` apod. analogických jejich matlabovským protějškům (ale ne s nimi zcela kompatibilních).

Ukažme si je na příkladu funkce přirozeného logaritmu $\ln x$.[10] Graf logaritmické funkce pro $x \in \langle 0{,}10\rangle$ vykreslíme příkazem

```
octave:8> x=linspace(0,10,101); plot(x,log(x));
```

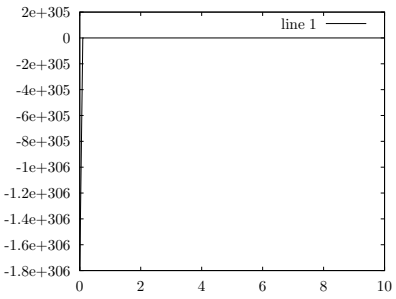
jehož výsledek na obr 2.5 nás určitě neuspokojí. Zavřeme proto okno s obrázkem (ale ne s programem GNUplot!) a změníme rozsah vykreslených hodnot na ose $y$ na hodnoty $y \geqq -10$ (to musíme udělat většinou, pokud funkce má ve zobrazovaném intervalu singulární bod, v němž hodnoty rostou nebo klesají k $\pm\infty$) nastavením parametru `yrange`. Dále přidáme souřadnicovou síť (`grid`), vypneme popisek v pravém horním rohu (`nokey`) a změníme směr značek na osách směrem

[10] Musíme dát pozor, že pro přirozený logaritmus o základu $e$ čísla $b$, tj. pro $\ln b$ použijeme příkaz `log(b)`, zatímco pro dekadický logaritmus o základu 10 $\log b$ příkaz `log10(b)`. Logaritmus o libovolném základě pak získáme podílem logaritmů jednoho z těchto typů podle známých vztahů $\log_a b = \ln b / \ln a = \log b / \log a$.

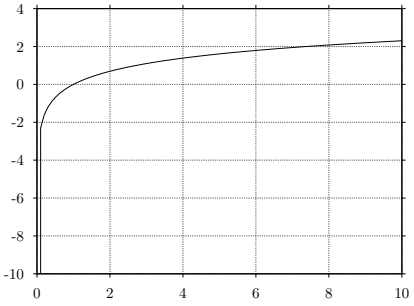
ven z obrázku (`tics out`). Nakonec obrázek překreslíme s novým nastavením pomocí příkazu `replot`. Celá posloupnost příkazů pak vypadá následovně[11]

```
octave:9> __gnuplot_set__ yrange [-10:]
octave:10> __gnuplot_set__ grid
octave:11> __gnuplot_set__ nokey
octave:12> __gnuplot_set__ tics out
octave:13> replot
```

její výstup je na obr. 2.6. K dispozici jsou i logaritmická měřítka na osách, buď




Obr. 2.5: Graf logaritmické funkce v základním nastavení




Obr. 2.6: Upravený graf logaritmické funkce

na ose $x$ nebo $y$ příp. na osách obou

```
octave:14>  semilogx(x,log(x))
octave:15>  semilogy(x,log(x))
octave:16>  loglog(x,log(x))
```

Dále můžeme pochopitelně upravit rozsah hodnot zobrazených na ose $x$[12] nastavením parametru `xrange` a použít desetinné čárky namísto tečky parametrem `decimalsign`

```
octave:17>  __gnuplot_set__ xrange [0.1:]
octave:18>  __gnuplot_set__ decimalsign ','
```

[11] V beta-verzi 2.9.xx souřadnicovou síť aktivujeme pomocí **`grid on`**, deaktivujeme pomocí **`grid off`**, rozsahy na osách zadáváme jako parametr funkce **`axis`**.

[12] Označení **`xrange`** resp. **`yrange`** či v trojrozměrném případě **`zrange`** se nezmění, ani když pracujeme s jinými proměnnými než $x$,$y$,$z$.

Všechna nastavení anulujeme na výchozí příkazem `__gnuplot_raw__ ("reset;")`. Ve většině případů se tím aktivuje vykreslování pouhých bodů (ne čar), které musíme opět zapnout. Celá dvojice příkazů pak vypadá

```
octave:19> __gnuplot_raw__ ("reset;")
octave:20> __gnuplot_set__ data style lines;
```

Vykreslit lze i parametricky definované křivky, *musíme však dát pozor, aby matice definující parametry měly stejný rozměr, tj. stejný počet prvků*, v našem případě 361. Pro $\varphi \in \langle 0{,}8\pi \rangle$ a $r \in \langle 0{,}2 \rangle$ nakreslíme spirálu s rostoucím poloměrem o rovnicích $x = r\cos\varphi$, $y = r\sin\varphi$ a také křivku popsanou v polárních souřadnicích rovnicí $r = 2\sin(2\varphi)$, tj. $x = 2\sin(2\varphi)\cos\varphi$, $y = 2\sin(2\varphi)\sin\varphi$. Použijeme postupně příkazy

```
octave:21> phi=linspace(0,8*pi,361);
octave:22> r=linspace(0,2,361);
octave:23> plot(r.*cos(phi),r.*sin(phi),\
               2*sin(2*phi).*cos(phi),2*sin(2*phi).*sin(phi));
```

Správný tvar křivek vyžaduje stejné měřítko na obou osách

```
octave:24> axis('equal'), replot
```

zobrazení os můžeme i vypnout

```
octave:25> axis('off'), replot
```

nebo zapnout zobrazení os procházejících počátkem souřadnic příslušným typem čáry (v našem případě plná tenká čára linetype -1[13])

```
octave:26> __gnuplot_set__ xzeroaxis lt -1
octave:27> __gnuplot_set__ yzeroaxis lt -1
octave:28> replot
```

a nevykreslovat rám okolo grafu

```
octave:29> __gnuplot_set__ noborder
octave:30> replot
```

V poslední ukázce vyplníme barvou oblast mezi grafem funkce a osou $x$ o rovnici $y = 0$. Nejprve opět pro jistotu vymažeme předchozí nastavení grafů

[13] Číslování čar a dalších parametrů nastíněno spolu s parametry příkazu `plot` na str. 25, hodnotě $-1$ odpovídá tenká plná čára.

```
octave:31> __gnuplot_raw__ ("reset;")
octave:32> __gnuplot_set__ data style lines;
```

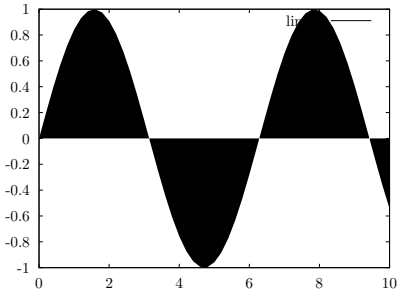
Pro hodnoty $x \in \langle 0{,}10 \rangle$ opět vykreslíme graf funkce $\sin x$. Změníme styl vykreslování na vyplňování ohraničené osou $x$ (`y1=0`)

```
octave:33> __gnuplot_set__ style data filledcurves y1=0
octave:34> x=linspace(0,10,51)'; plot(x,sin(x))
```

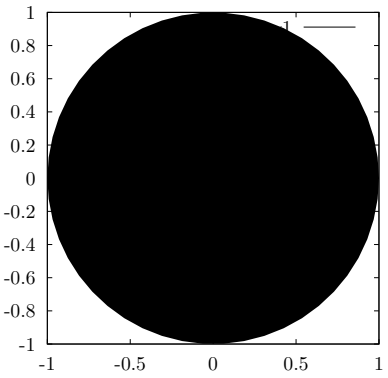
Vyplnit můžeme i uzavřené křivky, pro jednoduchost zvolíme jednotkový kruh s hranicí $x = \cos\varphi$, $y = \sin\varphi$, přitom musíme opět nastavit stejné měřítko na obou osách

```
octave:35> phi=linspace(0,2*pi,51);
octave:36> __gnuplot_set__ style data filledcurves closed
octave:37> axis('equal')
octave:38> plot(cos(phi)', sin(phi)');
```

Výsledky pak vidíme na obr. 2.7 a 2.8. Je zřejmé, že implementace této možnosti




Obr. 2.7: Vyplněná oblast ohraničená grafem funkce




Obr. 2.8: Vyplněný kruh

je prozatím spíše v počátcích (více je rozvinuta již v beta-verzi *GNUplotu*). Uživatelé, kteří by chtěli vyplňovat oblasti zvolenými barvami a kombinovat vyplňování s kreslením křivek mohou doinstalovat balík *EpsTk* (`http://www.epstk.de/`). Ten využívá možností postscriptu a je poměrně dobře a snadno použitelný v operačním systému Linux.

Před dalším kreslením nesmíme zapomenout vrátit se k výchozímu nastavení

```
octave:39> __gnuplot_raw__ ("reset;")
octave:40> __gnuplot_set__ style data lines;
octave:41> __gnuplot_set__ data style lines;
```

nebo restartovat *GNU Octave*.

### 2.5.2 3D grafika

K snadné ukázce trojrozměrných grafů můžeme použít demonstračních funkcí `sombrero(n)` a `peaks(n)`, kde $n$ určuje hustotu sítě bodů, v nichž budou spočítány funkční hodnoty (množství intervalů podél os $x$ a $y$, tj. poslední argument funkce `linspace`). Funkce odpovídají závislostem

$$\begin{aligned}
\texttt{sombrero}\,(x,y) &= \frac{\sin\left(\sqrt{x^2+y^2}\right)}{\sqrt{x^2+y^2}}, \quad x,y \in \langle -8,8\rangle \\
\texttt{peaks}\,(x,y) &= 3\,(1-x)^2 \exp\left[-x^2-(y+1)^2\right] - \\
&\quad -10\left(\frac{x}{5}-x^3-y^5\right)\exp\left(-x^2-y^2\right) - \\
&\quad -\frac{1}{3}\exp\left[-(x+1)^2-y^2\right], \quad x,y \in \langle -3,3\rangle
\end{aligned}$$

Výstup příkazů

```
octave:1> sombrero(51)
octave:2> peaks(51)
```

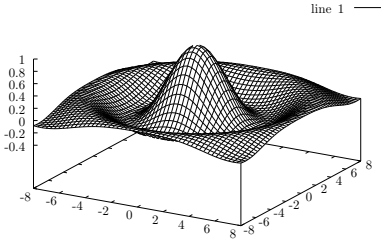
vidíme na obrázcích 2.9 a 2.10.

Zadefinujme si nyní postupně vlastní funkci dvou proměnných. Jak již bylo naznačeno, pro výpočet jejích funkčních hodnot musíme vytvořit „síť“ bodů (meshgrid) se všemi možnými kombinacemi hodnot nezávisle proměnných, v našem případě $x$ a $y$. Pro $x,y \in \langle -2,2\rangle$ k tomu použijeme příkaz `meshgrid` s tím, že dělení na osách samotných definujeme podobně jako u dvourozměrných grafů pomocí `linspace` (nebo pomocí kroku ve zvoleném intervalu, např. `-2:.1:2` – snadno ověříme, v čem nejsou definice stejné). Pro funkci

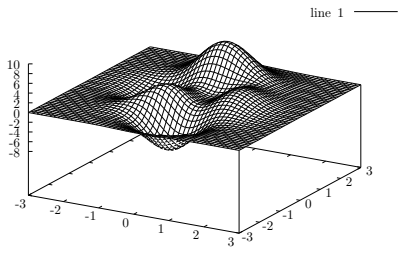
$$z = \left[(x-0{,}3)^2+y^2\right]\exp\left[-x^2-(y-0{,}2)^2\right]$$

zadáme

```
octave:3> [x,y]=meshgrid(linspace(-2,2,41),linspace(-2,2,41));
octave:4> z=((x-.3).^2+y.^2).*exp(-x.^2-(y-.2).^2);
octave:5> mesh(x,y,z);
```




Obr. 2.9: `sombrero(51)`




Obr. 2.10: `peaks(51)`

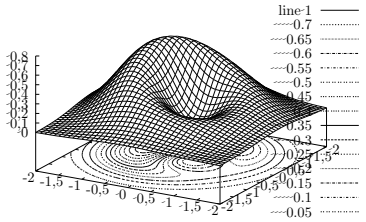
Připomeňme, že středníkem na konci řádku jsme zakázali výpis delších matic na obrazovku a vlastní vykreslení provedl příkaz `mesh`, jehož argumentem jsou matice jednotlivých proměnných ($x$ a $y$ přitom musí tvořit síť „meshgrid“). Pro zachování kompatibility s *Matlabem*® je téhož výsledku dosáhnout také příkazem `surf(x,y,z)`, ale plocha se – na rozdíl od *Matlabu*® – většinou nevyplní barvami (viz níže).

Nyní můžeme opět graf upravovat změnou patřičných parametrů. Zobrazení „vrstevnic“ tj. čar spojujících body se stejnou hodnotou $z$ aktivujeme parametrem `contour`, jejich hustotu zvýšíme na 30 vrstevnic mezi minimem a maximem parametrem `cntrparam`, poté již výše zmíněným způsobem nahradíme desetinnou tečku čárkou parametrem `decimalsign` a nakonec obrázek překreslíme v novém nastavení pomocí `replot`

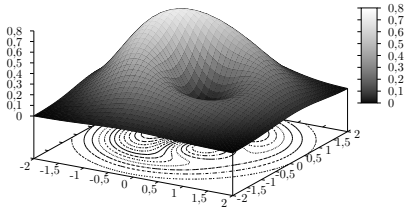
```
octave:6> __gnuplot_set__ decimalsign ","
octave:7> __gnuplot_set__ contour
octave:8> __gnuplot_set__ cntrparam level auto 30
octave:9> replot
```

Výsledek vidíme na 2.11. Pokud bychom chtěli vykreslit vrstevnice odpovídající konkrétním hodnotám, použijeme např. `__gnuplot_set__ cntrparam levels discrete .1, .5, .9` pro hodnoty $z = 0{,}1$, $z = 0{,}5$ a $z = 0{,}9$. Před dalšími úpravami zopakujeme nastavení vyzkoušená již u dvourozměrných obrázků – deaktivujeme popisky parametrem `nokey` a změníme směr značek na osách `tics out`. Potom naopak zapneme barevné stínování plochy parametrem `pm3d` a změníme použité barvy pomocí `palette`.[14]

[14] Doporučujeme za každou změnou provést překreslení příkazem `replot`, aby se čtenář sám přesvědčil, jaké změny jednotlivé kroky s grafem způsobí. Funkce spojené se stínováním jsou dostupné až v novějších verzích *GNUplotu*.




Obr. 2.11: Graf spolu s vrstevnicemi



Obr. 2.12: Vrstevnice spolu s barevným stínováním plochy

```
octave:10> __gnuplot_set__ nokey
octave:11> __gnuplot_set__ tics out
octave:12> __gnuplot_set__ pm3d
octave:13> __gnuplot_set__ palette defined \
             ( 0 "blue", 3 "green", 6 "yellow", 10 "red" )
octave:14> replot
```

výsledek znázorňuje obr. 2.12. Pokud nechceme vykreslovat barevný obdélník v pravé části znázorňující přiřazení barev hodnotám $z$, stačí zadat

```
octave:15> __gnuplot_set__ nocolorbox
octave:16> replot
```

K dispozici je mnoho barevných „palet“, jež si může uživatel nadefinovat použitím nejrůznějších kombinací, z ukázky vidíme, že přiřazujeme barvy relativním pozicím hodnot $z$ mezi minimem (0) a maximem (10) na desetistupňové škále.[15] Spektrum od modré po červenou lze získat např. příkazy

```
octave:17> __gnuplot_set__ palette defined \
             ( 0 "dark-blue", 1 "blue", 2 "cyan", \
               3 "yellow", 4 "red" , 5 "dark-red" )
octave:18> replot
```

nebo

```
octave:19> __gnuplot_set__ palette rgbformulae 33,13,10;
octave:20> replot
```

[15] Hodnoty 0 a 10 neodpovídají skutečným hodnotám $z$, jde o relativní stupnici.

Pro černobílé obrázky vystačíme pouze se stupni šedé, např.

```
octave:21> __gnuplot_set__ palette defined \
                ( 0 "dark-grey", 1 "white")
octave:22> replot
```

nebo

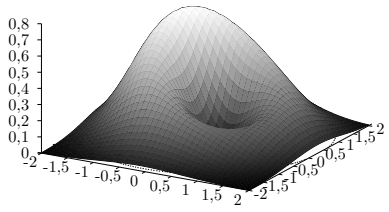
```
octave:23> __gnuplot_set__ palette defined \
                  ( 0.5 0.5 0.5 0.5, 1 1 1 1)
octave:24> replot
```

K dalším na internetu doporučovaným paletám patří také kombinace `rgbformulae 21,22,23` (odpovídá „hot“ přechodu černá-červená-žlutá-bílá).

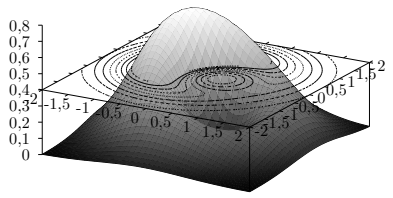
Pokud chceme změnit polohu plochy vůči základní rovině $xy$, musíme změnit parametr `ticslevel`, udávající polohu nejnižších bodů plochy relativně k celému rozsahu osy $z$. Volbou

```
octave:25> __gnuplot_set__ ticslevel 0
```

„přilepíme“ plochu na základnu (obr. 2.13). Záporná čísla mají za následek vyzdvižení základny nad graf (nebo jeho část), jak vidíme z obr. 2.14.




Obr. 2.13: Volba `ticslevel 0`




Obr. 2.14: Volba `ticslevel -0.5`

Další proměnnou, kterou lze nastavit, je pohled na vykreslenou plochu prostřednictvím parametru `view`. Má celkem čtyři argumenty – úhel $\vartheta \in \langle 0{,}180^\circ \rangle$ rotace okolo osy $x$, úhel $\varphi \in \langle 0{,}360^\circ \rangle$ rotace okolo osy $z$, měřítko os v základní rovině a konečně měřítko svislé osy $z$. Přednastavené hodnoty jsou `60, 30, 1, 1`. Na náš obrázek zkusme aplikovat např.

```
octave:26> __gnuplot_set__ view 15, 135, 1, 4
```

Pokud je zapnut mód `mouse` (v OS Windows ve výchozím nastavení), vidíme aktuální hodnoty parametru view v levém dolním rohu obrazovky a měnit je můžeme pomocí myši – při stisknutém levém tlačítku obrázkem otáčíme, při stisknutém prostředním měníme měřítko ve směru osy $z$. Jak bylo řečeno, tento doplněk, dostupný podobně jako barevné stínovaní v novějších verzích, lze kdykoli zapnout příkazem `__gnuplot_set__ mouse`.

Speciálním a hodně užívaným je pohled „ze shora“ s vrstevnicemi vykreslenými přes barevné stínování. Dosáhneme toho nastavením

```
octave:27> __gnuplot_set__ view map
```

Protože hodnoty $x$ i $y$ leží ve stejném intervalu, nastavíme opět stejné měřítko na osách `axis('equal')`. Ukažme si také změnu popisu os, k čemuž slouží postupně parametry `xtics`, `mxtics` a `xlabel` pro osu $x$ a obdobné parametry (např. `mytics` nebo `zlabel` pro osy zbývající)[16]. V naší ukázce umístíme podél osy $x$ hlavní značky automaticky od $-2$ s krokem $0{,}5$ a na ose $y$ konkretní popisky do zvolených bodů

```
octave:28>  __gnuplot_set__ xtics -2, 0.5
octave:29> __gnuplot_set__ ytics \
                 ("1. bod" -2, "2. bod" 0, "3. bod" 1)
```

Dále aktivujeme menší značky tak, že určíme jejich počet mezi značkami hlavními (což jde, pokud se opakují podle nějakého algoritmu, jako v případě naší osy $x$)

```
octave:30>  __gnuplot_set__ mxtics 10
```

Nyní ještě doplníme označení os, u něhož můžeme zvolit i jiný font. V našem případě použijeme pro osu $y$ standardní font Symbol pro psaní symbolů a řeckých písmen, pro osu $x$ skloněné písmo Times-Italic o velikost 30 bodů.[17]

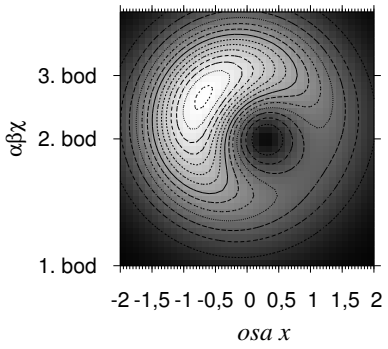
```
octave:31>  __gnuplot_set__ xlabel "{/Times-Italic=30 osa x}"
octave:32>  __gnuplot_set__ ylabel "{/Symbol abc}"
octave:33>  replot
```

Výsledek posloupnosti příkazů je uveden na obr. 2.15.

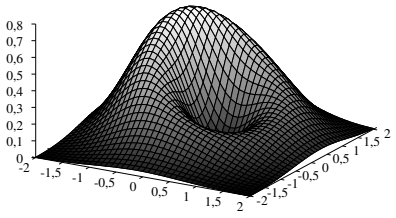
Chceme-li podobně jako v *Matlabu*® vykreslit stínovanou plochu spolu s čarami spojujícími body na ní, použijeme posloupnost příkazů

[16] Opět platí, že názvy se nezmění, i když pracujeme s jinými proměnnými.

[17] Změna fontů se nemusí projevit v každém prostředí, např. na obrazovce počítače nebo v obrázku png. V postscriptových souborech ji ale lze použít téměř vždy, užití speciálních, třeba TEX-ovských fontů (většinou v tomto textu), závisí na konkrétní instalaci prohlížeče postscriptu.




Obr. 2.15: Pohled ze shora s upravenými popisky




Obr. 2.16: Barevné stínování spolu s vykreslením čar.

```
octave:1> [x,y]=meshgrid(linspace(-2,2,41),linspace(-2,2,41));
octave:2>  z=((x-.3).^2+y.^2).*exp(-x.^2-(y-.2).^2);
octave:3> global gnuplot_has_pm3d=1;
octave:4> gnuplot_has_pm3d=1;
octave:5> surf(x,y,z)
```

Protože funkce `surf` nastavuje příslušné parametry pro `mesh`, budou se poté opět obě funkce chovat stejně. Stejného efektu docílíme i postupnými kroky

```
octave:1> [x,y]=meshgrid(linspace(-2,2,41),linspace(-2,2,41));
octave:2>  z=((x-.3).^2+y.^2).*exp(-x.^2-(y-.2).^2);
octave:3> __gnuplot_set__ pm3d at s hidden3d 1
octave:4> __gnuplot_set__ nohidden3d
octave:5> __gnuplot_set__ nosurf
octave:6> __gnuplot_set__ line style 1 lt 1 lw 1
octave:7> mesh(x,y,z)
```

Výsledek po úpravě dalších známých parametrů

```
octave:8> __gnuplot_set__ nokey
octave:9> __gnuplot_set__ nocolorbox
octave:10> __gnuplot_set__ tics out
octave:11> __gnuplot_set__ decimalsign ","
octave:12> __gnuplot_set__ palette rgbformulae 33,13,10
octave:13> __gnuplot_set__ ticslevel 0
```

vidíme na obr. 2.16.

Zbývá uvést, jak vykreslené grafy uložit pro další použití. Z mnoha dostupných formátů, z nichž mnohé jsou velmi speciální se zaměříme na tři. Jako první uveďme PNG (Portable Network Graphics), který postupně i na internetu nahrazuje formát GIF dostupný ve starších verzích při zachování jeho vlastností. K exportu našeho obrázku do souboru `pokus.png` postupně zadáme:

```
octave:14> __gnuplot_set__ term png large
octave:15> __gnuplot_set__ output 'pokus.png'
octave:16> replot
```

přičemž nepovinný údaj `large` v prvním řádku požaduje větší písmo popisek os. Obr. 2.16 byl získán exportem do postscriptového souboru `pokus.eps` (přesněji jde Encapsulated PostScript vyvinutý firmou Adobe Systems), u něhož navíc požadujeme pro všechny popisky font „Times-Roman“ o velikosti 25 bodů

```
octave:17> __gnuplot_set__ term post eps enh "Times-Roman" 25
octave:18> __gnuplot_set__ output 'pokus.eps'
octave:19> replot
```

Z dalších možností uveďme formát SVG (Scalable Vector Graphics – škálovatelná vektorová grafika neboli značkovací jazyk popisující dvojrozměrnou vektorovou grafiku pomocí XML), který by se měl v budoucnu stát základním otevřeným formátem pro vektorovou grafiku na internetu (většina používaných formátů jako GIF, JPG nebo výše zmíněné PNG jsou formáty pro rastrovou grafiku). Čtenář jistě sám doplní, že v takovém případě by předcházející příkazy obsahovaly `term svg` a např. `output "pokus.svg"`. Tento formát je také doporučován pro náčrtky a grafy v příspěvcích do Wikipedie [1].

Protože nastavení pro jeden obrázek mohou být zcela nežádoucí pro obrázky jiné, bude v ukázkách v příští kapitole vždy na počátku vymazání veškerých proměnných příkazem `clear`, vymazán terminál příkazem `clc` a zrušeno nastavení grafického výstupu

```
octave:1> clear;
octave:2> clc;
octave:3> __gnuplot_raw__ ("reset;")
octave:4> __gnuplot_set__ data style lines;
```

Náš stručný přehled základních funkcí není zdaleka vyčerpávající a slouží především k lepšímu pochopení programů a procedur uvedených v následujících kapitolách. Více informací najde čtenář buď v [11, 15] popř. v jiných internetových zdrojích.

# Část 3

# Pokročilejší funkce

Ke studiu a sestavení dynamických modelů potřebujeme ještě cykly a logické proměnné (rozhodování), jež patří k nezbytnému arzenálu každého programovacího jazyka. Jejich použití si postupně vysvětlíme na algoritmech pro numerické řešení algebraických rovnic v této kapitole a diferenciálních rovnic v kapitole následující. Popsané skripty budou jednoduchou alternativou k předdefinovaným procedurám, které k těmto účelům program přímo nabízí a jež by také rozhodně neměly uniknout naší pozornosti.

## 3.1 Numerické řešení algebraických rovnic

Pod numerickým řešením algebraických rovnic rozumíme přibližný výpočet reálného kořene (popř. kořenů) rovnice

$$f(x) = 0$$

na určitém intervalu $(a,b)$. Základem všech metod je *Bolzanova věta*[1], zaručující existenci hledaného řešení:

*Bolzanova věta*: Nechť $f$ je funkce spojitá na intervalu $\langle a,b\rangle$. Mají-li čísla $f(a)$ a $f(b)$ různá znaménka, tj. $f(a)\cdot f(b) < 0$, pak existuje alespoň jeden bod $x^* \in (a,b)$ v němž platí $f(x^*) = 0$.

Pokud bude zaručeno, že daná funkce je na intervalu $\langle a,b\rangle$ rostoucí nebo klesající, existuje takový bod $x^*$ právě jeden. Používané metody se liší způsobem sestavení tzv. iterační posloupnosti $\{x_n\}$ takové, že $\lim x_n = x^*$. Absolutně přesný výpočet této limity však většinou není možný a proto se spokojujeme s $k$-tým členem iterační posloupnosti $|x^* - x_k| < \varepsilon$, kde $\varepsilon$ je předem dané kladné číslo – požadovaná přesnost řešení rovnice. Výběr konkrétní metody závisí na vlastnostech funkce $f$ na intervalu $(a,b)$. Jedním z důležitých kriterií je i rychlost konvergence – kolik členů posloupnosti je zapotřebí k dosažení požadované přesnosti.

K nejpoužívanějším metodám patří:

1. *Metoda půlení intervalu (metoda bisekce)*, jejíž výhodou je jednoduchá konstrukce zmíněné posloupnosti (středy intervalů) a minimální požadavky na funkci $f$, nevýhodou je v některých případech pomalá konvergence.

[1] Zatímco na internetu najdeme Bolzanovu větu poměrně snadno (viz např. [13]), klasické učebnice matematické analýzy jako [10] popř. známá přehledná publikace [17] ji uvádí jen jako jednu z bezejmenných vět o vlastnostech spojitých funkcí.

2. *Metoda tětiv (regula falsi, lineární interpolace)*, kde princip vytváření iterační posloupnosti spočívá postupném nahrazování funkce $f(x)$ na příslušném intervalu úsečkami a hledání průsečíků těchto úseček s osou $x$.
3. *Metoda tečen (Newtonova metoda)*, v níž jsou kladeny větší požadavky na funkci $f$ (existence spojité první i druhé derivace), ale její výhodou je větší rychlost konvergence iterační posloupnosti, důležité je však správná volba intervalu $(a,b)$, jinak se dokonce může stát, že posloupnost k hledané limitě $x^*$ nebude vůbec konvergovat.

Mohli bychom zmínit i např. *metodu prosté iterace*, z hlediska studentů středních škol nám však připadají nejschůdnější metoda půlení intervalu a metoda regula falsi. Program *GNU Octave* má k tomuto účelu i nadefinovanou proceduru `fsolve` využívající modifikovanou Newtonovu iterační metodu. Podrobnější informace včetně odvození potřebných vztahů lze najít např. v [3, 9, 14, 18]. Zde si ukažme jejich použití na konkrétní úloze převzaté z [20], kde je použit program *FAMULUS*.

### 3.1.1 Úloha: kometa Hale-Bopp

Kometa Hale-Bopp objevená 23. července 1995 prolétla 1. dubna 1997 periheliem ve vzdálenosti $r_\mathrm{p} = 0{,}9141$ AU od Slunce. Hlavní poloosa její trajektorie měří $a = 187{,}8$ AU. Naším úkolem je určit, v jaké vzdálenosti od Slunce se nacházela v době objevení a jakou rychlostí se přitom pohybovala?

Určování poloh astronomických těles v konkrétním čase patří v astronomii k základním úlohám. Pro pohyb po eliptické trajektorii lze odvodit *Keplerovu rovnici* (viz. např. [9, 20])

$$E - \frac{e}{a}\sin E - \sqrt{\frac{\varkappa M}{a^3}}\,t = 0, \tag{3.1}$$

kde $E$ tzv. je excentrická anomálie udávaná v radiánech, $\varkappa = 6{,}67{\cdot}10^{-11}\,\mathrm{m^3{\cdot}s^{-2}{\cdot}kg^{-1}}$ gravitační konstanta, $M \approx 1{,}989{\cdot}10^{30}\,\mathrm{kg}$ hmotnost Slunce, $a = 2{,}809\,5{\cdot}10^{13}\,\mathrm{m}$ a $e$ hlavní poloosa a excentricita eliptické trajektorie. Snadno vypočítáme relativní excentricitu $\varepsilon = e/a = (a - r_\mathrm{p})/a = 0{,}995\,13$ a délku vedlejší poloosy trajektorie $b = \sqrt{a^2 - e^2} = 18{,}51$ AU. Pro kartézské souřadnice polohy komety s počátkem ve středu Slunce pak platí

$$x = a\cos E - e, \qquad y = \frac{b}{a}\,a\sin E = b\sin E, \tag{3.2}$$

pro vzdálenost komety od Slunce $r = \sqrt{x^2 + y^2}$. Podle zadání od objevení komety do průletu periheliem uplynulo $618\,\mathrm{dnů}$ neboli $618{\cdot}24{\cdot}3\,600 =$

= 53 395 200 s. Tento čas a zbývající hodnoty dosadíme do Keplerovy rovnice (3.1) a upravíme ji na tvar

$$E - 0{,}99513 \sin E - 0{,}00413 = 0 \tag{3.3}$$

Jde o transcendentní rovnici, kterou neumíme vyřešit elementárními prostředky (analyticky), Keplerovou rovnicí (3.3) je excentrická anomálie určena implicitně. Jedná se o rovnici typu $f(t,E) = 0$ a její řešení pro zvolené $t$ musíme provést některou z přibližných numerických metod. Pro $t \in \langle 0,T \rangle$, kde $T$ odpovídá periodě oběhu komety, je výraz $\sqrt{\varkappa M/a^3}\, t$ z intervalu $\langle 0,2\pi \rangle$ a také $E$ je v intervalu $\langle 0,2\pi \rangle$. Jakmile známe $E$, vypočítáme souřadnice bodů trajektorie v čase $t$ podle vztahů (3.2).

Jak již bylo řešeno, k získání numerického řešení se nabízí několik způsobů.

### Řešení pomocí předdefinované funkce `fsolve`

Nejprve musíme zadefinovat vlastní funkci (označme ji `f`) odpovídající levé straně rovnice (3.3).[2] Definice bude ohraničena příkazy `function` a `endfunction`. Poté do argumentu příkazu `fsolve` uvedeme název funkce a počáteční bod, tj. hodnotu, od níž má algoritmus řešení hledat; v našem případě 0. Celý výpis použitých kroků pak vypadá následujícím způsobem

```
1 octave:1> function y=f(E)
2 > y=E-0.99513*sin(E)-0.0041307;
3 > endfunction
4 octave:2> E=fsolve("f",0)
```

Předcházející řádky lze samozřejmě uložit, např. do souboru `halebp.m` a posloupnost příkazů volat příkazem `halebp`. Volaná funkce `fsolve` vypíše hledané řešení

```
5 E = 0.25891
```

Hledaná hodnota je $E = 0{,}25891\,\mathrm{rad}$. Uvedený postup je sice velmi jednoduchý (zadání zabírá pouhé 4 řádky), ale neumožňuje nám zcela nahlédnout do algoritmu, jakým *GNU Octave* k tomuto výsledku došel a mít pod kontrolou všechny parametry.[3] Chceme-li použít konkrétní metodu, musíme zadat přesný postup výpočtu. Máme-li zájem o výpis většího počtu desetinných míst, zapneme

[2] Na rozdíl od *Matlabu*® nemusí být definice uživatelské funkce uložena v samostatném souboru.

[3] K dispozici je ještě analogická funkce `fzero`, která má více parametrů a umožňuje zadat i požadovanou přesnost.

`format long`, `fsolve` pak vrací `E = 0.258914801661345`. Abychom nemuseli vypisovat celý algoritmus pokaždé znova, je vhodné si na každou metodu vytvořit samostatný soubor. Připomeňme, že ho stačí napsat v libovolném textovém editoru a uložit s příponou `*.m` do adresáře, v němž budeme s *GNU Octave* pracovat (přepneme se do něj pomocí příkazu `cd`). Celý program pak voláme jménem souboru bez přípony.

### Řešení metodou půlení intervalu

Možná podoba programu uložená ve zvláštním souboru (např. `halboppi.m`) je následující

```
# --- Metoda puleni intervalu pro reseni Keplerovy rovnice
# --- Vymazeme hodnoty promennych z~predchazejicich vypoctu
clear;
# --- a obsah  terminalu
clc;
# --- zadani konstant
a=187.8*1.4959787e11;
rp=0.9141*1.4959787e11;
kappa=6.672e-11;
M=1.9891e30;
t=618*24*3600;
# --- Definice funkce
function y=f(E)
   y=E-0.99513*sin(E)-0.00413;
endfunction
# --- Pozadovana presnost
presnost=1e-6;
# --- Zaciname merit dobu vypoctu
t0=clock();
# --- Krajni body a stred intervalu
x1=0;           # levy okraj
x2=pi;          # pravy okraj
f1=f(x1);       # leva funkcni hodnota
f2=f(x2);       # prava funkcni hodnota
k=1;            # krok vypoctu
xk=0.5*(x1+x2); # stred intervalu
fk=f(xk);       # hodnota funkce ve stredu intervalu
# --- Formatovany vypis hodnot
printf("k=%f xk=%f f(xk)=%.20f\n", [k,xk,fk]);
# --- Cyklus, ktery se opakuje do dosazeni pozadovane presnosti
while abs(x1-xk)>2*presnost
if (f1*fk<0)
  x2=xk;
else
```

```
35   x1=xk;
36 endif
37 k=k+1;
38 xk=0.5*(x1+x2);
39 fk=f(xk);
40 # --- Formatovany vypis hodnot
41 printf("k=%f xk=%f f(xk)=%.20f\n", [k,xk,fk]);
42 endwhile
43 # --- Ukoncime mereni trvani vypoctu a vypiseme cas potrebny na beh cyklu
44 casvypoctu=etime(clock(),t0)
45 # --- Vypocteme polohu a rychlost v~pozadovanem okamziku
46 r=sqrt((a*cos(xk)-(a-rp))^2+(sqrt(a^2-(a-rp)^2)*sin(xk))^2)/1.5e11
47 v=sqrt(kappa*M*(2/(r*1.5e11)-1/a))/1e3
```

Protože jednotlivé části jsou okomentovány a mělo by být zřejmé, jaký účel jednotlivé části mají, doplňme jen pár poznámek. Komentáře, tj. řádky, které nebude *GNU Octave* provádět a interpretovat, mohou být uvozeny buď znakem `#`, nebo `%` (z důvodu kompatibility s *Matlabem*®). Psaní komentářů lze podobně jako u jiných programovacích jazyků vřele doporučit. Činí kód přehlednějším a srozumitelnějším jak pro jiné uživatele, tak pro samotného autora, pokud se k programu vrací po delší době. Otázkou volby je použití diakritiky v komentářích, zde se jí z důvodu přenositelnosti mezi různými operačními systémy s různým kódováním češtiny vyhýbáme.

Zadáváme-li vstupní hodnoty (řádky 7–11) většinou nepožadujeme jejich vypsání na obrazovku, proto jsou ukončeny znakem `;`. Naopak poslední řádky tento ukončovací znak nemají, neboť chceme vypočtené hodnoty

$$\begin{aligned} r &= \sqrt{\left[a\cos x_\mathrm{k} - (a - r_\mathrm{p})\right]^2 + \left[\sqrt{a^2 - (a - r_\mathrm{p})^2}\,\sin x_\mathrm{k}\right]^2} \\ v &= \sqrt{\varkappa M\left(\frac{2}{r} - \frac{1}{a}\right)} \end{aligned}$$

vypsat, přitom je přímo převádíme na astronomické jednotky AU resp. na $\mathrm{km{\cdot}s^{-1}}$. Při běhu cyklu chceme, aby program postupně vypisoval jak číslo kroku $k$ iterační posloupnosti, střed intervalu v tomto kroku $x_\mathrm{k}$ (momentální aproximaci hledaného řešení) a hodnotu funkce v tomto bodě $f(x_\mathrm{k})$ (posledně jmenovanou na 20 desetinných míst) příkazem `printf`.

V programu se setkáváme s třemi novými prvky, o nichž ještě nebyla v textu řeč. Jde jednak o měření času, který počítač spotřebuje na provedení výpočtu nebo některé jeho části. Slouží k tomu příkaz `etime`, který spočte rozdíl dvou časových údajů zadaných jako jeho argumenty. Systémový čas pak vrací funkce `clock()` jako vektor se složkami odpovídajícími roku, měsíci, dni, hodině, minutám a

sekundám. Čas potřebný na provedení výpočtu je orientační, neboť závisí nejen na hardwarové konfiguraci (procesor, operační paměť apod.), ale i na procesech běžících momentálně v počítači na pozadí. Proto se údaj i u dvou bezprostředně následujících stejných výpočtech může o něco lišit (až v řádu sekund).

Dalším novým prvkem je rozhodovací podmínka `if` (32.–36. řádek) větvící výpočet na základě vyhodnocení nějakého logického výrazu, větvení je uzavřeno výrazem `endif`, obě možnosti odděluje `else`. První část se provede, pokud je podmínka větvení splněna, druhá pokud splněna není. V našem případě jde o rozhodutí, v kterém z intervalů $\langle x_1,x_\mathrm{k}\rangle$ nebo $\langle x_\mathrm{k},x_2\rangle$ bude výpočet pokračovat. Vybrat musíme ten, v němž leží nulový bod; v něm musí mít funkční hodnoty opačná znaménka, tj. jejich součin bude záporný. Leží-li v $\langle x_1,x_\mathrm{k}\rangle$ bude v dalším kroku $x_\mathrm{k}$ horní, v opačném případě dolní hranicí studovaného intervalu.

Posledním prvkem, o němž se zmíníme, je cyklus typu `while` (31.–42. řádek) probíhající tak dlouho, dokud je splněna zadaná podmínka, a ukončený výrazem `endwhile`. V našem případě jde o dosažení přesnosti výsledku – chceme, aby se poslední nalezené $x_\mathrm{k}$ od skutečného řešení lišilo méně než o zvolené $\varepsilon$. Protože při posledním kroku najdeme ještě střed intervalu, stačí testovat, zda je $x_\mathrm{k}$ od jednoho z krajních bodů (zde $x_1$) vzdáleno více než $2\varepsilon$.

Po zadání příkazu `halboppi` se na obrazovce objeví výpis

```
k=1.000000 xk=1.570796 f(xk)=0.57153632679489663193
k=2.000000 xk=0.785398 f(xk)=0.07760499223527936308
k=3.000000 xk=0.392699 f(xk)=0.00774931764925237444
k=4.000000 xk=0.196350 f(xk)=-0.00192069129854763841
k=5.000000 xk=0.294524 f(xk)=0.00152332039781002439
k=6.000000 xk=0.245437 f(xk)=-0.00048994036543237827
k=7.000000 xk=0.269981 f(xk)=0.00043675232187719883
k=8.000000 xk=0.257709 f(xk)=-0.00004569147745284684
k=9.000000 xk=0.263845 f(xk)=0.00019064495530215532
k=10.000000 xk=0.260777 f(xk)=0.00007126924592972851
k=11.000000 xk=0.259243 f(xk)=0.00001248874645650745
k=12.000000 xk=0.258476 f(xk)=-0.00001667618293280625
k=13.000000 xk=0.258859 f(xk)=-0.00000211244972662082
k=14.000000 xk=0.259051 f(xk)=0.00000518346210189903
k=15.000000 xk=0.258955 f(xk)=0.00000153433504570567
k=16.000000 xk=0.258907 f(xk)=-0.00000028935007293472
k=17.000000 xk=0.258931 f(xk)=0.00000062241929662567
k=18.000000 xk=0.258919 f(xk)=0.00000016651631525900
k=19.000000 xk=0.258913 f(xk)=-0.00000006142145290121
k=20.000000 xk=0.258916 f(xk)=0.00000005254628767694
k=21.000000 xk=0.258915 f(xk)=-0.00000000443786849456
casvypoctu = 0.012676
r = 7.1242
v = 15.610
```

Vidíme, že funkční hodnota $f(x_\mathrm{k})$ v nalezených bodech se ve 3. sloupci stále více blíží 0, hledanou číselnou aproximací řešení je poslední hodnota $E = x_k =$ $= (0{,}258\,915 \pm 10^{-6})\,\mathrm{rad}$. K dosažení této přesnosti aproximace $x^*$ z Bolzanovy věty (viz s. 37) jsme potřebovali $k = 21$ kroků.

### Řešení metodou tětiv

Výpis programu pro řešení Keplerovy rovnice metodou tětiv může být třeba v souboru `halbopte.m` následující

```
# --- Metoda tetiv pro reseni Keplerovy rovnice
# --- Vymazeme hodnoty promennych z~predchazejicich vypoctu
clear;
# --- a obsah terminalu
clc;
# --- Definice funkce
function y=f(E)
   y=E-0.99513*sin(E)-0.00413;
endfunction
# --- Pozadovana presnost
presnost=1e-8;
# --- Zaciname merit dobu vypoctu
t0=clock();
# --- Krajní body
a=0;            # levy okraj
b=pi;           # pravy okraj
fa=f(a);        # leva funkcni hodnota
fb=f(b);        # prava funkcni hodnota
k=1;            # krok vypoctu
x(k)=(a*fb-b*fa)/(fb-fa); # aproximace řešení
fk=f(x(k));       # hodnota funkce ve stredu intervalu
# --- Formátovaný výstup hodnot
printf("k=%f x(k)=%f f(xk)=%.20f\n", [k,x(k),fk]);
# --- Cyklus
while abs(f(x(k)))>presnost
k=k+1;
if (f(a)*f(x(k-1))<0)
  x(k)=(a*f(x(k-1))-x(k-1)*fa)/(f(x(k-1))-fa);
else
  x(k)=(x(k-1)*fb-b*f(x(k-1)))/(fb-f(x(k-1)));
endif
fk=f(x(k));
# --- Formátovaný výstup hodnot
printf("k=%f xk=%f f(xk)=%.20f\n", [k,x(k),fk]);
endwhile
casvypoctu=etime(clock(),t0)
```

Zde v testovací podmínce cyklu `while` na 25. řádku testujeme, zda se hodnota funkce v hledaném bodě $f(x_\mathrm{k})$ liší od 0 více než je požadovaná přesnost. Vidíme, že od předcházející metody se program skutečně liší jen konstrukcí bodu $x_\mathrm{k}$.

Příkaz `halbopte` pak vypíše

```
k=1.000000 x(k)=0.004130 f(xk)=-0.00410987521635498669
k=2.000000 xk=0.008234 f(xk)=-0.00408980538600391998
...
k=416.000000 xk=0.258915 f(xk)=-0.00000001033584787723
k=417.000000 xk=0.258915 f(xk)=-0.00000000997460833894
casvypoctu = 0.26952
```

Pro danou rovnici je metoda regula falsi evidentně méně efektivní, neboť při stejné přesnosti potřebujeme přes 400 kroků.

Vidíme, že všemi způsoby dostáváme $E \approx (0{,}258\,915 \pm 10^{-6})\,\mathrm{rad}$. Dosazením do vztahů (3.2) dostaneme vzdálenost komety od Slunce v době jejího objevení e vzdálenost komety od Slunce $r = 7{,}12\,\mathrm{AU}$ a rychlost v tomto okamžiku $v = 15{,}6\,\mathrm{km{\cdot}s^{-1}}$.

Zájemce o další středoškolské úlohy řešené pomocí *GNU Octave* lze odkázat na práci [9], podobné úlohy řešené programem *FAMULUS* najde v textu [18].

## 3.2 Numerické řešení diferenciálních rovnic

Numerické řešení soustavy diferenciálních rovnic je jádrem dynamického modelování. V programu *GNU Octave* máme na výběr, zda sestavíme cyklus odpovídající některé z Eulerových metod či metod Rungova-Kuttova typu nebo zda využijeme předdefinované funkce `lsode`.[4] Její použití ilustrujeme na dvou konkrétních příkladech.

[4] Jde také o algoritmus patřící mezi Rungovy-Kuttovy metody nazvané původně po německých matematicích Carlu Davidu Tolmé Rungovi (1856–1927) a Martinu Wilhelmu Kuttovi (1867-1944). Na nějaké variantě Rungových-Kutových metod jsou vesměs založeny procedury pro řešení obyčejných diferenciálních rovnic ve většině matematických programů *FAMULUS* nevyjímaje. Pro uživatele znalé Matlabu jsou v *GNU Octave* navíc funkce `ode23`, `ode45`, `ode78` a `rk2fixed`, `rk4fixed`, `rk8fixed` pro Rungovy-Kutovy metody naznačeného řádu s proměnným, resp. fixovaným (pevným) krokem, jejichž autorem je Marc Compere. Zde se přidržíme pouze funkce `lsode`. Název samotný je akronymem k „Livermore Solver for ODEs“ a vychází z fortranovského balíčku Alana Hindmarshe (`http://www.netlib.org/odepack/`). V podstatě jde opět o variantu Rungovy-Kuttovy metody 4. řádu s tím, že konkrétní podmetodu (Adamsovu apod.) lze volit parametrem funkce `lsode_options`.

### 3.2.1 Van der Polův oscilátor

Van der Polův oscilátor sehrál významnou úlohu ve vývoji nelineární dynamiky. Z matematického hlediska je popsán van der Polovou rovnicí

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} + \mu \left(y^2 - 1\right) \frac{\mathrm{d}y}{\mathrm{d}t} + y = 0, \tag{3.4}$$

kde $\mu \geqq 0$ je konstantní parametr. Rovnice tohoto typu se objevila při studiu nelineárních elektrických obvodů v prvních radiopřijímačích v roce 1926 holandským inženýrem B. van der Polem, podobnou rovnicí se zabýval už lord Rayleigh okolo roku 1880 v souvislosti s nelineárními vibracemi [7, 21]. Rovnice (3.4) se od rovnice harmonického oscilátoru liší prostředním členem odpovídajícím nelineárnímu tlumení $\mu \left(y^2 - 1\right) \mathrm{d}y/\mathrm{d}t$. Pro $|y| > 1$ vede ke skutečnému kladnému tlumení, naopak pro $|y| < 1$ představuje negativní tlumení (buzení) systému. Jinými slovy, tlumí velké a budí malé výchylky. Lze odhadnout, že systém může přejít do stavu, kdy se oba procesy vyrovnají a z obecnějších vět lze ukázat [21], že pro každé $\mu > 0$ má rovnice jednoznačně určený limitní periodický režim. Jak ukážeme, s rostoucím $\mu$ se tyto limitní kmity víc a více liší od kmitů harmonických.

Pro použití funkce `lsode` rozdělíme diferenciální rovnici 2. řádu na soustavu rovnic 1. řádu

$$\begin{aligned} \frac{\mathrm{d}y}{\mathrm{d}t} &= x_2, \\ \frac{\mathrm{d}^2 y}{\mathrm{d}t^2} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}y}{\mathrm{d}t}\right) &= -\mu \left(y^2 - 1\right) \frac{\mathrm{d}y}{\mathrm{d}t} - y = -\mu \left(x_1^2 - 1\right) x_2 - x_1. \end{aligned}$$

Takto zapsaná nám umožňuje zavést dvě matice (vektory). Jedna, již označíme `x` obsahuje v prvním řádku `x(1)` hodnoty $y$ a ve druhém `x(2)` hodnoty 1. derivace $\mathrm{d}y/\mathrm{d}t$. Druhá matice `xdot` se skládá z derivací v matice první, tj. z první a druhé derivace $y$. Soustavu proto lze přepsat ve tvaru

$$\begin{aligned} \mathrm{xdot}(1) &= x(2), \\ \mathrm{xdot}(2) &= -\mu \left(x(1)^2 - 1\right) x(2) - x(1). \end{aligned}$$

Soubor `vanderpol.m` pak může mít následující podobu

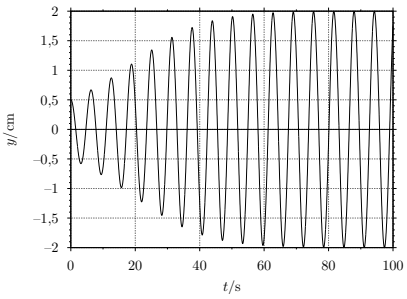
```
1 # --- van der Poluv oscillator
2 # --- Vymazeme hodnoty promennych z~predchazejicich vypoctu
3 clear
4 # --- a obsah terminalu
5 clc;
```

```
6  global mu
7  mu=.1;
8  # --- pocatecni podminky
9  x0=[0.5; 0];
10 # --- definice diferencialni rovnice
11 function xdot = vdpol(x,t)
12 global mu;
13   xdot = [x(2); -mu*(x(1).^2-1).*x(2)-x(1)];
14 endfunction
15 # --- oblast integrace
16 t=linspace(0,100,5000);
17 # --- reseni soustavy rovnic
18 vdp=lsode( "vdpol",x0, t);
19 % vykresleni zavislosti
20 plot(t,vdp(:,1))
21 % vykresleni rychlosti v zavislosti na poloze ("fazovy prostor")
22 plot(vdp(:,1),vdp(:,2))
```

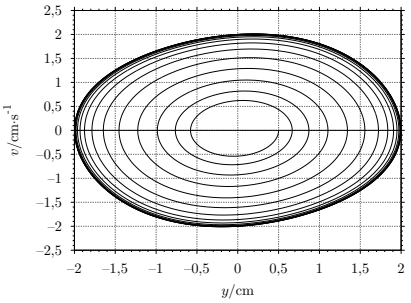
Všimněme si, že pokud přiřazujeme hodnotu nějaké konstantě, kterou potom využíváme v definici funkce (v našem případě jde o parametr $\mu = 0{,}1$), musíme ji deklarovat jako globální konstantu příkazem `global` na počátku před přiřazením hodnoty (6. řádek) i v těle funkce (12. řádek). Pokud se vyskytuje pouze v definici funkce, mohli bychom se tomu vyhnout tak, že bychom pouze v definici funkce nahradili 12. řádek za `mu=0.1`.

Vzhledem k tomu, že jde o rovnici 2. řádu (nebo po provedené úpravě soustavu dvou rovnic 1. řádu), zadáváme na 9. řádku počáteční podmínky jako vektor `x0`, jehož složky odpovídají volbě $y_0 = 0{,}5$ a $v_0 = \mathrm{d}y/\mathrm{d}t|_0 = 0$. Časový interval, v němž bude provedena integrace pak zadáváme na 16. řádku pomocí nám již známého příkazu `linspace`; zde konkrétně $t \in \langle 0{,}100 \rangle$ s a interval je rozdělen na 5000 bodů. Tímto dělením určujeme integrační krok a tím i přesnost výsledku popř. hladkost vykreslených křivek – pokud je dělení jemné, trvá výpočet déle, pokud je ale příliš hrubé, namísto křivek dostaneme lomené čáry a výsledek nebude dostatečně přesný.

Samotná procedura `lsode`, která provádí vlastní numerickou integraci, vrací matici (zde pojmenovanou `vdp`), jejíž sloupce postupně odpovídají veličinám `x(1)` a `x(2)`, tj. v našem případě hodnotám $y$ a první derivace $\mathrm{d}y/\mathrm{d}t$. Chceme-li potom pouze hodnoty $y$, vybíráme na 20. řádku pouze první sloupec této matice `vdp(:,1)`, chceme-li druhý, použijeme `vdp(:,2)`. Závislosti $y = y(t)$ a $v = \mathrm{d}y/\mathrm{d}t = v(y)$ jsou pro $\mu = 0{,}1$ a uvedené počáteční podmínky znázorněny na obr. 3.1 a 3.2. Dodejme, že ve výpisu programu nejsou uvedeny příkazy ovlivňující vlastnosti grafu popsané v části 2.5. Vidíme, že průběh kmitů připomíná harmonický pohyb. Z trajektorie pohybu ve fázovém prostoru $y - v$ je zřejmá odlišnost, neboť harmonickým kmitům by odpovídala elipsa, zatímco v našem případě limitní fázová trajektorie připomíná elipsu poněkud deformovanou.

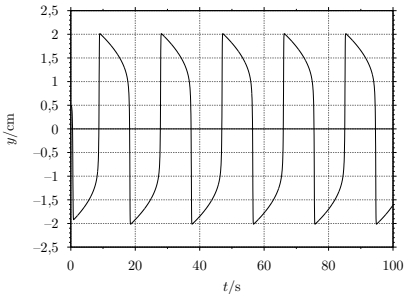



Obr. 3.1: Závislost $y = y(t)$ pro $\mu = 0{,}1$

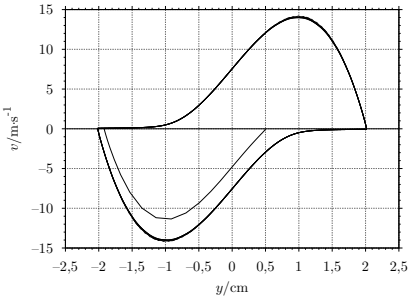



Obr. 3.2: Závislost $v = v(y)$ pro $\mu = 0{,}1$

Pro vyšší hodnoty parametru $\mu$ vynikne odlišnost od harmonického pohybu ještě více, což obrážejí obr. 3.3 a 3.4 vykreslené pro stejné počáteční podmínky. Čtenář si může sám ověřit, že limitní kmity oscilátoru závisejí pouze na parametru




Obr. 3.3: Závislost $y = y(t)$ pro $\mu = 10$




Obr. 3.4: Závislost $v = v(y)$ pro $\mu = 10$

$\mu$ a nikoli na zvolených počátečních podmínkách (s výjimkou případů, kdy ke kmitům vůbec nedojde jako u triviální volby $y_0 = v_0 = 0$).

### 3.2.2 Lorenzův atraktor

Na závěr kapitoly si ukažme použití procedury `\lsode` u soustavy 3 obyčejných diferenciálních rovnic. Jako model takové soustavy použijeme známý

Lorenzův atraktor, který se stal jedním ze symbolů nelineární dynamiky a teorie chaosu.

Je nazván po meteorologovi Edwardu Nortonu Lorenzovi, jenž v roce 1963 z hydrodynamických rovnic vynucené konvekce v atmosféře za silně zjednodušených předpokladů odvodil systém tří provázaných nelineárních diferenciálních rovnic. Numerické řešení pak odhalilo typicky chaotické chování – silnou závislost na počátečních podmínkách, která komplikuje předpověď vývoje systému. Není jistě překvapením, že příklad takových systémů poskytuje právě meteorologie a ne náhodou je možné počasí spolehlivě předpovídat jen na několik následujících dnů. Samotnému Lorenzovi se podařilo zpopularizovat podobné jevy pomocí „efektu mávnutí motýlího křídla“.[5]

Lorenzovy rovnice lze zapsat ve tvaru [21]

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sigma\,(y - x)\,, \tag{3.5a}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = x\,(r - z) - y, \tag{3.5b}$$

$$\frac{\mathrm{d}z}{\mathrm{d}t} = xy - bz, \tag{3.5c}$$

kde $\sigma$, $r$ a $b$ jsou nezáporné hydrodynamické parametry – $\sigma$ tzv. Prandtlovo číslo, $r$ tzv. Rayleighovo číslo ($b$ pojmenování nemá).

Chceme-li vykreslit známý obrázek odpovídající jednomu z řešení Lorenzových rovnic, může soubor `lorentz.m` obsahovat následující kód

```
# --- Lorenz atractor
# --- Vymazeme hodnoty promennych z~predchazejicich vypoctu
clear
# --- a obsah terminalu
clc;
x=zeros(3,1);
# --- definice diferencialni rovnice
function xdot = loren(x,t)
sigma=10;
r=28;
b=8/3;
xdot = [sigma * (x(2) - x(1));
        x(1) * (r - x(3))-x(2);
        x(1) * x(2) - b * x(3)];
```

[5] Citlivost na změnu počátečních podmínek je ilustrována tvrzeními typu, že mávnutí motýlích křídel v Brazílii může rozhodnout o tom, zda texaské pláně budou zasaženy tornádem či nikoli. Dodejme, že Lorenzova práce zůstala až do počátku 70.-ých let 20. století bez větší odezvy [7, 21].

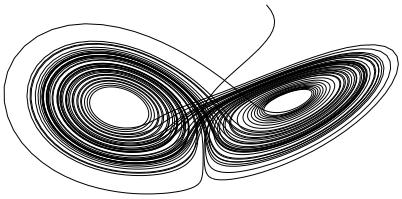
```
15 endfunction
16 # --- oblast integrace
17 t=(0:0.01:60)'
18 # --- reseni soustavy rovnic
19 lr = lsode("loren", [3;14;50], t);
20 # --- vykresleni reseni - trojrozmerna krivka
21 __gnuplot_set__ noborder
22 __gnuplot_set__ noxtics
23 __gnuplot_set__ noytics
24 __gnuplot_set__ noztics
25 __gnuplot_set__ nokey
26 plot3(lr(:,1),lr(:,2),lr(:,3));
```

Upozorněme opět na dosud nezmíněné příkazy a odlišnosti od programu pro van der Polův oscilátor. Na první pohled vidíme, že zde chybí globální deklarace konstant příkazem `global`, jejich hodnoty jsou uvedeny přímo v definici funkce definující soustavu Lorenzových diferenciálních rovnic na 9.–11. řádku. Počáteční podmínky tentokrát nevypisujeme na zvláštní řádek, ale zapisujeme přímo jako druhý argument procedury `lsode`. V tomto případě jde o počáteční podmínky $x_0 = 3$, $y_0 = 14$ a $z_0 = 50$ (matice `[3;14;50]`). Oblast integrace je dána na 17. řádku intervalem $t \in \langle 0{,}60 \rangle$ s, hodnoty v něm měníme s krokem $\Delta t = 0{,}01\,\mathrm{s}$.

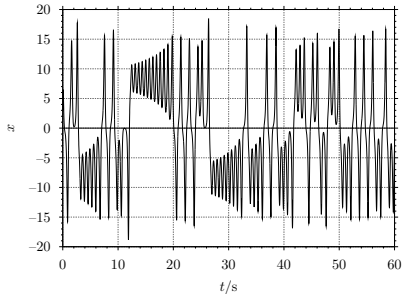
Upozorněme také, že označení funkce (zde `loren`) musí být odlišné od jména celého souboru (bez přípony, zde `lorentz`). Protože řešíme soustavu rovnic prvního řádu, není nutné ji nijak přeskupovat, ale lze ji přímo přepsat do definice funkce na 12.–14. řádku. Pracujeme s maticemi `x` o složkách $x_1 = x$, $x_2 = y$, $x_3 = z$ a `xdot` obsahující derivace `x` podle času, tj. $\mathrm{xdot}_1 = \mathrm{d}x/\mathrm{d}t$, $\mathrm{xdot}_2 = \mathrm{d}y/\mathrm{d}t$ a $\mathrm{xdot}_3 = \mathrm{d}z/\mathrm{d}t$. Procedura `lsode` pak vrací matici (zde označenou `lr`) se sloupci odpovídajícími složkám matice `x`, tj. souřadnicím $x$, $y$, $z$.

Chceme-li vykreslit prostorovou křivku, jejíž body musí být určeny třemi souřadnicemi, použijeme příkaz `plot3`, jehož argumentem jsou právě hodnoty jednotlivých souřadnic. Nepovinnou, ale často užívanou částí je deklarace nulové matice `x` na 6. řádku pomocí funkce `zeros`. Konkrétně `zeros(3,1)` vytvoří matici $3 \times 1$ obsahující samé 0, je ekvivalentní zápisu `[0;0;0]`. Konečně příkazy na řádcích 21–25 ovlivňují vzhled obrázku – postupně zakazují vykreslování os, značek na nich a popisu obrázku. Výsledek je pak znázorněn na obr. 3.5a. Trajektorie se protíná jen zdánlivě (jde o důsledek projekce na rovinu) a překlápí se zprava doleva a nazpátek vždy po několika oběžích na každé straně, přičemž počet oběhů na každé straně odpovídá náhodné posloupnosti. V prostoru je však trajektorie lokalizována v poměrně úzké oblasti nazývané podle Ruelleho a Takense od roku 1971 *podivným atraktorem*. S podobnými rovnicemi se setkáme u laserů, dynam a některých typů vodních kol.

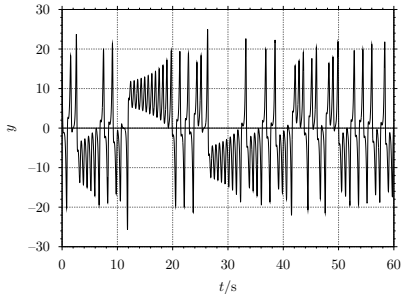
Poučné je i vykreslení závislostí jednotlivých souřadnic na čase příkazy typu `plot(t,lr(:,1))` pro $x = x(t)$ a analogicky pro $y$ i $z$; vidíme je na obr. 3.5b–d. Řešení odpovídá aperiodickým oscilacím, jež se nikdy zcela přesně neopakují [21]. Čtenáři doporučujeme vyzkoušet i jiné počáteční hodnoty popř. hodnoty hydrodynamických parametrů (např. hodnotu $r = 99{,}96$).



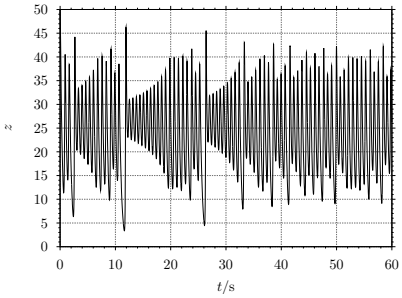
(a) Prostorová křivka




(b) Závislost $x = x(t)$




(c) Závislost $y = y(t)$




(d) Závislost $z = z(t)$

Obr. 3.5: Řešení Lorenzových rovnic pro $\sigma = 10$, $r = 28$, $b = 8/3$ a počáteční podmínky $x_0 = 3$, $y_0 = 14$ a $z_0 = 50$

# Část 4

# Příklady jednoduchých dynamických modelů

V poslední kapitole stručně uvedeme příklady skriptů pro *GNU Octave*, jimiž lze řešit diferenciální rovnice popisující konkrétní fyzikální systémy a jevy. Převážná většina z nich byla navíc teoreticky popsána v první části textu a modelována pomocí programu *Coach*. Nebudeme proto opakovat odvození či sestavení pohybových rovnic, zaměříme se na programování v *GNU Octave* a prezentaci grafických výsledků. Význam téměř všech použitých příkazů byl popsán v předcházejících kapitolách (k rychlé orientaci lze využít i rejstřík použitých příkazů na s. 74). Pro přehlednost nebudeme uvádět příkazy definující pouze vzhled obrázku, popisky os apod., jež byly shrnuty v části 2.5.

Modely jsou rozděleny podle použité integrační metody, tj. Eulerovy, Rungovy-Kuttovy popř. pomocí předdefinované funkce `lsode`. Dodejme, že další zajímavé úlohy řešené pomocí *GNU Octave* najde čtenář v diplomové práci [9], řešení pomocí programu *FAMULUS* v [12, 18].

## 4.1 Modely využívající Eulerovu metodu

### 4.1.1 Balistická křivka

Modelujeme šikmý vrh v odporujícím prostředí popsaný v části 1.4, odpovídající program pro *FAMULUS* najde čtenář v [24]. Uvažujeme těleso tvaru koule o hmotnosti $m$ a poloměru $R$ vržené počáteční rychlostí $v_0$ pod úhlem $\alpha$ k horizontu. Při pohybu na něj působí konstantní tíhová síla $m\boldsymbol{g}$ svisle dolů a proti pohybu síla odporu prostředí úměrná druhé mocnině velikosti okamžité rychlosti podle Newtonova vztahu $\boldsymbol{F}_\mathrm{o} = -kv\boldsymbol{v} = -C\varrho Sv\boldsymbol{v}/2$, kde $C$ je součinitel odporu závisející na tvaru tělesa, $\varrho$ hustota prostředí (uvažujeme vzduch) a $S = \pi R^2$ plocha příčného průřezu střely. Zvolíme-li standardně osu $x$ vodorovně a osu $y$ svisle vzhůru, musíme řešit soustavu diferenciálních rovnic

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t} = -kvv_x \tag{4.1a}$$

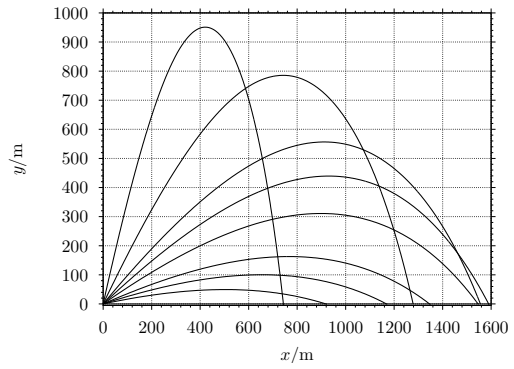
$$m\frac{\mathrm{d}^2y}{\mathrm{d}t} = -mg - kvv_y, \tag{4.1b}$$

kde parametr $k$ postupně vypočteme uvedeným způsobem. Výsledné trajektorie – balistické křivky – pro několik různých elevačních úhlů lze vykreslit např. pomocí souboru `baliste.m` s obsahem

```
# --- numericke reseni - balisticka krivka
# --- pro nekolik elevacnich uhlu
clear;
clc;
# --- definice pocatecnich podminek (v~zakladnich jednotkach SI)
t0=0;
x0=0;
y0=0;
v0=200         # pocatecni rychlost
eluhel=[10 15 20 30 37.9 45 60 75]     % elevacni uhly ve stupnich
alpha=eluhel*pi/180; # prevedeni na radiany
# --- krok integrace
dt = 0.05;
# --- parametry prostredi
R=0.04;         # polomer strely
m=2.5;          # hmotnost strely
ro=1.3;         # hustota vzduchu
C=0.48;         # koeficient odporu
g=9.81;         # tihove zrychleni
S=pi*R^2;
k=C*ro*S/2;
hold on # pro vice obrazku do jednoho grafu
# --- cyklus pro hodnoty elevacnich uhlu
for j=1:columns(alpha)
clear t
clear x
clear y
vx=v0*cos(alpha(j));
vy=v0*sin(alpha(j));
v=sqrt(vx^2+vy^2);
# --- vlastni numericka integrace - klasicka Eulerova metoda
t(1)=t0;
x(1)=x0;
y(1)=y0;
i=1;
 while (y(i)>=0)  # testujeme dopad na rovinu y=0
   i=i+1;
   x(i)=x(i-1)+vx*dt;
   y(i)=y(i-1)+vy*dt;
   ax=-k*v*vx/m;
   ay=-g-k*v*vy/m;
   vx=vx+ax*dt;
   vy=vy+ay*dt;
   v=sqrt(vx^2+vy^2);
   t(i)=t(i-1)+dt;
 endwhile
# --- vykresleni zavislosti
```

```
48 plot(x,y);
49 endfor
50 hold off
```

Ve výpisu vidíme několik dosud nezmíněných příkazů a postupů. V testovací podmínce cyklu `while` na 36. řádku nyní ověřujeme, zda těleso nedopadlo na rovinu $y = 0$, po dopadu se výpočet ukončí. V rámci jednoho běhu programu vykreslujeme balistické křivky pro několik hodnot elevačních úhlů zadaných ve stupních maticí `eluhel`(10. řádek), která se hned na následujícím řádku přepočítává na matici `alpha` v radiánech.




Obr. 4.1: Balistické křivky pro různé hodnoty elevačních úhlů

K využití všech hodnot pak používáme cyklus `for` mezi 24–49 řádkem ukončený příkazem `endfor`. Podobně jako v jiných programovacích jazycích, mění se v průběhu cyklu celočíselná proměnná (zde $j$) od jedničky do množství prvků matice `alpha`. Počet sloupců resp. řádků matice zjistíme snadno pomocí funkce `columns` (24. řádek) resp. `rows`. Takto můžeme hodnoty elevačních úhlů libovolně dopisovat nebo umazávat a program si vždy sám zjistí jejich počet.

V každém běhu cyklu, tj, pro každou hodnotu elevačního úhlu se vypočtené `t`, `x` a `y` ukládají do matic toho jména a pro jiný úhel budou jiné. Nechceme-li vytvářet vícerozměrné matice, musíme vždy na konci cyklu hodnoty vykreslit (48. řádek) a na začátku vynulovat příkazem `clear` (25.–27. řádek). Takto `t`, `x` a `y` zůstávají vektory, jejichž prvky voláme jednoduše `x(0)`,`y(j)` apod.

Uvedený postup vyžaduje také použití příkazů `hold on` a `hold off` (22. a 50. řádek). Protože hodnoty $x$ a $y$ jsou počítány postupně vždy v každém cyklu, nelze počkat na konec a použít několik dvojic argumentů funkce `plot` jako v části 2.5.1. Příkaz `hold on` zajistí, že další vykreslování bude provedeno

do již otevřeného obrázku a to tak dlouho, dokud volbu nevypneme voláním `hold off`. Výstup z popsaného programu je znázorněn na obr. 4.1, zvolené počáteční hodnoty jsou uvedeny a okomentovány ve výpisu programu. Vidíme, že za daných podmínek nedosáhneme největší délky vrhu pro elevační úhel $45°$, ale o něco menší (asi $37{,}9°$).

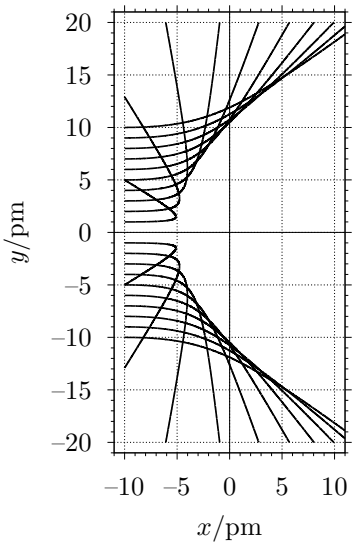
### 4.1.2 Rutherfordův rozptyl

Ostřelování zlaté fólie o tloušťce několika atomů jádry helia patří k nejznámějším historickým experimentům. Experiment samotný byl poprvé uskutečněn Hansem Geigerem a Ernestem Marsdenem roku 1909 pod vedením Ernesta Rutherforda, který jej v roce 1911 vysvětlil a zformuloval známý planetární model atomu. Fyzikální rozbor problému je nastíněn v části 5.2.

Na nabité $\alpha$-částice s kladným nábojem $Q_\mathrm{a} = 2e$ a hmotností $m_\mathrm{a} \approx 4u$ působí jádra atomů zlata s nábojem $Q_\mathrm{j} = 79e$ odpudivou coulombovskou silou. Pohybové rovnice mají tvar

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t} = \frac{1}{4\pi\varepsilon_0}\,\frac{Q_\mathrm{a}Q_\mathrm{j}}{r^3}\,x \qquad (4.2\mathrm{a})$$

$$m\frac{\mathrm{d}^2y}{\mathrm{d}t} = \frac{1}{4\pi\varepsilon_0}\,\frac{Q_\mathrm{a}Q_\mathrm{j}}{r^3}\,y, \qquad (4.2\mathrm{b})$$




Obr. 4.2: Trajektorie částic při Rutherfordově rozptylu

kde $\varepsilon_0 = 8{,}854{\cdot}10^{-12}\,\mathrm{F{\cdot}m^{-1}}$ je permitivita vakua, ve vztazích pro náboje a hmotnost vystupují elementární náboj $e = 1{,}602{\cdot}10^{-19}\,\mathrm{C}$ a hmotnostní atomová jednotka $u = 1{,}66{\cdot}10^{-27}\,\mathrm{kg}$. Trajektoriemi částic budou hyperboly, jejichž příklad je znázorněn na obr. 4.2. Vykresleny byly pomocí souboru `ruther.m` s obsahem

```
1 # --- numericke reseni pro rozptyl na odpuzujícím centru
2 clear;
3 clc;
4 # --- definice pocatecnich podminek a konstant
5 u=1.66e-27;     # hmotnostni atomova jednotka
6 q=1.602e-19;    # elementarni naboj
7 eps0=8.854e-12; # premitivita vakua
```

```
ma=4*u;         # hmotnost alpha-castice
Qa=2*q;         # naboj alpha-castice
Qj=79*q;        # naboj jadra Au
kQQ=1/4/pi/eps0*Qa*Qj; # koeficient v~Coulombove zakone
ymax=2e-11;     # maximalni hodnota y pro vykreslovani
t0=0;
x0=-1e-11;
y0=[-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 4 5 6 7 8 9 10]*1e-12;
vx0=1e6;
vy0=0;
# ------------------------------------------------
dt=1e-20;       # integracni krok
# --- nastavenni os
axis([-11 11 -21 21],'equal');
hold on         # pro vice kreivek v~jednom obrazku
# --- cyklus pro nekolik ruznych trajektorii
cas_spusteni = clock();
for j=1:columns(y0)
# --- vlastni numericka integrace - Eulerova metoda
clear x
clear y
t(1)=t0;
x(1)=x0;
y(1)=y0(j);
vx=vx0;
vy=vy0;
i=1;
# --- cyklus while s~integraci
 while ((abs(y(i))<=ymax) & (x(i)>=x0))
   i=i+1;
   x(i)=x(i-1)+vx*dt;
   y(i)=y(i-1)+vy*dt;
   r=sqrt(x(i)^2+y(i)^2);
   ax=kQQ/r^3*x(i-1)/ma;
   ay=kQQ/r^3*y(i-1)/ma;
   vx=vx+ax*dt;
   vy=vy+ay*dt;
   t(i)=t(i-1)+dt;
 endwhile
# --- vykresleni zavislosti
plot(x/1e-12,y/1e-12,'1');
endfor
hold off
# --- udaje o~case spotrebovanem na vypocet
celkovy_cas = etime(clock(), cas_spusteni);
disp(['Program bezel ',num2str(celkovy_cas),' sekund.']);
```

Opět zadáváme jedním příkazem na 15. řádku více počátečních podmínek maticí, neboť nás zajímají trajektorie s různým momentem hybnosti $\alpha$-částic, který se při pohybu zachovává a který při počátečním pohybu ve směru osy $x$ závisí na hodnotě $y_0$. Různé počáteční podmínky opět procházíme v cyklu `for...endfor` mezi 25.–49. řádkem, opakované vynášení do stejného obrázku aktivujeme a deaktivujeme pomocí `hold on... hold off`. Vypočítané hodnoty souřadnic vynášíme přímo v pikometrech, proto je v rámci příkazu `plot` na 48. řádku příslušně přeškálujeme.

Z příkazů, s nimiž jsme se v uvedené podobě v textu dosud nesetkali, upozorněme na `axis` (21. řádek), který nemá pouze argument `'equal'` požadující stejné měřítko na obou osách, ale také nastavení rozsahu vykreslovaných hodnot parametrem `[-11 11 -21 21]`, jež je ekvivalentní dvojici `__gnuplot_set__ xrange [-11:11]` a `__gnuplot_set__ yrange [-21:21]` ; omezujeme tím vykreslování na hodnoty $x \in \langle -11{,}11\rangle$, $y \in \langle -21{,}21\rangle$. V podmínce cyklu `while` na 36. řádku pak testujeme, zda hodnota $x$ nesklesla pod $x_0$ a hodnota $y$ neleží mimo interval $\langle -y_{\max}, y_{\max}\rangle$. Ke zobrazení času spotřebovaného na výpočet je pak na posledním řádku využita funkce `disp`, díky níž se nevypíše pouze hlášení `celkovycas=7.963`, ale celé hlášení `Program bezel 7.963 sekund`. Protože argumentem `disp` mohou být pouze textové řetězce, převádíme číselný údaj o čase na řetězec („string“) pomocí funkce `num2str`.

### 4.1.3 Pohyb lyžaře s odporem prostředí

Uveďme jednu z úloh zpracovaných pro systém *FAMULUS* v textu [23] se zadáním: Lyžař o hmotnosti $m = 90\,\mathrm{kg}$ získá po určité době jízdy na velmi dlouhém rovném svahu se sklonem $\alpha = 15^\circ$ stálou rychlost $v_\mathrm{m} = 18\,\mathrm{m{\cdot}s^{-1}}$. Součinitel smykového tření mezi lyžemi a sněhem je $f = 0{,}055$, tíhové zrychlení $g = 9{,}81\,\mathrm{m{\cdot}s^{-2}}$. Velikost síly odporu vzduchu je přímo úměrná druhé mocnině rychlosti: $F_\mathrm{o} = Kv^2$. Úkolem je určit velikost koeficientu $K$ a modelovat závislost rychlosti a dráhy lyžaře v závislosti na čase.

Při pohybu lyžaře se uplatní pohybová složka tíhové síly, smykové tření a odpor vzduchu. Výslednice těchto sil má velikost

$$F = mg\,(\sin\alpha - f\cos\alpha) - Kv^2.$$

Na velmi dlouhém svahu dosáhne rychlost lyžaře mezní hodnoty $v_\mathrm{m}$, při které je výslednice všech sil nulová a platí

$$\begin{aligned} mg\,(\sin\alpha - f\cos\alpha) - Kv_\mathrm{m}^2 &= 0 \\ K = \frac{mg\,(\sin\alpha - f\cos\alpha)}{v_\mathrm{m}^2} &= 0{,}5605\,\mathrm{N{\cdot}m^{-2}{\cdot}s^{-2}}. \end{aligned}$$

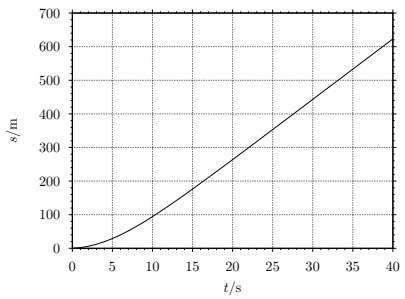
Okamžité zrychlení lyžaře v čase $t$ potom bude

$$a = \frac{F}{m} = g\,(\sin\alpha - f\cos\alpha) - \frac{K}{m}\,v^2.$$

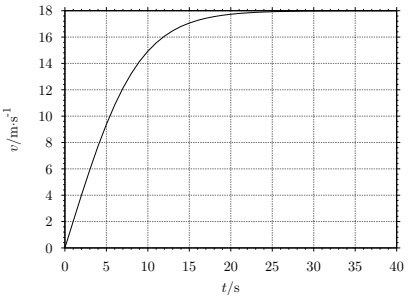
a program pro modelování závislosti rychlosti lyžaře na čase může vypadat následovně:

```
1  # --- numericke reseni rychlosti lyzare na svahu s~odporem prostredi
2  clear;
3  clc;
4  # --- definice pocatecních podminek
5  t(1)=0;
6  s(1)=0;
7  v(1)=0;          # pocatecni rychlost
8  m=90;            # hmotnost lyzare
9  uhel=15;         # uhel svahu ve stupnich
10 alpha=uhel*pi/180;
11 f=0.055;         # soucinitel smykoveho treni
12 vm=18;           # mezni rychlost
13 g=9.81;          # tihove zrychleni
14 K=m*g*(sin(alpha)-f*cos(alpha))/vm^2;
15 dt = 1;          # krok integrace
16 # --- vlastní numerická integrace - metoda Eulerova
17    i=1;
18    a=g*(sin(alpha)-f*cos(alpha))-K*v^2/m;
19 # --- vlastni cyklus
20 while (v<=0.9999*vm)  %
21    i=i+1;
22    a=g*(sin(alpha)-f*cos(alpha))-K*v(i-1)^2/m;
23    v(i)=v(i-1)+a*dt;
24    s(i)=s(i-1)+v(i)*dt;
25    t(i)=t(i-1)+dt;
26 endwhile
27 # --- vykreslení závislosti v~na t
28 plot(t,v);
29 pause
30 # --- vykreslení závislosti s~na t
31 plot(t,s);
```

Výsledný graf je na obrázcích 4.1.3a,b. Připomeňme, že rychlost lyžaře se mezní rychlosti $v_{\mathrm{m}}$ sice blíží, ale z matematického hlediska ji přesně nikdy nedosáhne. Chceme-li na vyhnout nekonečnému cyklu `while`, musíme do podmínky zadat rychlost o něco nižší (na 20. řádku volíme konkrétně $0{,}9999\,v_{\mathrm{m}}$).




(a) Závislost $s = s(t)$ pro pohyb lyžaře




(b) Závislost $v = v(t)$ pro pohyb lyžaře

## 4.2 Modely využívající Rungovy-Kuttovy metody

I když ve většině matematických programů se používají Rungovy-Kuttovy metody 4. řádu, pro naše účely je pro svou jednoduchost zcela postačující metoda 2. řádu s pevným (fixním) krokem, použitá v následujícím příkladu.

### 4.2.1 Matematické kyvadlo s libovolnou počáteční výchylkou

Matematické kyvadlo patří k nejznámějším modelům. Zde se podobně jako v části 2.5 budeme zabývat pohybem s libovolně velkou počáteční výchylkou. Pohybová rovnice má známý tvar (viz např. [2, 4])

$$\frac{\mathrm{d}^2\varphi}{\mathrm{d}t^2} = -\frac{g}{l}\,\sin\varphi,$$

kde $g$ je tíhové zrychlení $l$ délka závěsu. Pro jednoduchost uvažujme sekundové kyvadlo s dobou kmitu 1 s, pro jeho délku musí platit

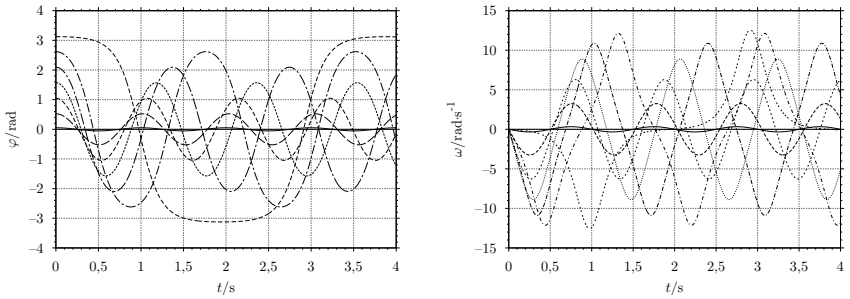
$$l = \frac{g}{4\pi^2} \approx 0{,}25\,\mathrm{m}.$$

Lépe tak uvidíme prodloužení doby kmitu při větších počátečních výchylkách. Výpis programu `kyvrk2.m` může mít podobu

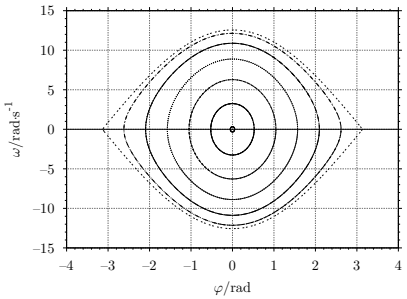
```
1 # --- matematicke kyvadlo s~libovolnou pocatecni vychylkou
2 clear;
3 clc;
4 # --- definice počátečních podmínek (v~základních jednotkách SI)
5 phi0=[3 30 60 90 120 150 179]*pi/180;
```

```
6  # --- vlastni cyklus
7  x=v=[];
8  hold on
9  for j=1:columns(phi0)
10 clear t;
11 clear ph;
12 t(1)=0;
13 ph(1)=phi0(j);
14 dph(1)=0;      # počateční derivace (úhlová rychlost)
15 mez=4;         # mez integrace - 4 male kmity
16 h = mez/500;   # krok integrace
17 g=9.81;
18 l=g/(4*pi^2);  # delka sekundoveho kyvadla
19 # --- vlastní numerická integrace - RK metoda 2. řádu
20 i=2;
21   while t<mez
22     k1=-g/l*sin(ph(i-1));
23     php=ph(i-1)+dph(i-1)*2*h/3+k1*2*h^2/9;
24     k2=-g/l*sin(php);
25     ph(i)=ph(i-1)+dph(i-1)*h+(k1+k2)*h^2/4;
26     dph(i)=dph(i-1)+(k1+3*k2)*h/4;
27     t(i)=t(i-1)+h;
28     i=i+1;
29   endwhile
30 # --- ukladame vychylku a rychlost do matice pro pozdejsi pouziti...
31 x=[x;ph];
32 v=[v;dph];
33 # --- Vykresleni zavislosti
34 plot(t,ph);
35 endfor
36 hold off
37 # --- vykresleni zavislosti uhlove rychlosti na case
38 for j=1:columns(phi0)
39 hold on
40 plot(t,v(j,:))
41 hold off
42 endfor
43 # --- vykresleni zavislosti uhlove rychlosti na vychylce
44 for j=1:columns(phi0)
45 hold on
46 plot(x(j,:),v(j,:))
47 hold off
48 endfor
```

Podobně jako v části 4.1.1 zde zadáváme několik různých počátečních výchylek v jediné matici `phi0` (5. řádek), počet jejích prvků (tj. zadaných hodnot) opět zjišťujme pomocí funkce `columns`. Podobně jako v části 4.1.1 užíváme




(a) $\varphi = \varphi(t)$ (b) $\omega = \omega(t)$




(c) $\omega = \omega(\varphi)$

Obr. 4.3: Kmity matematického kyvadla s počátečními výchylkami $\varphi_0 = 3°$, $30°$, $60°$, $90°$, $120°$, $150°$ a $179°$

i cyklu `for`...`endfor` a opakovaného vykreslování do stejného obrázku pomocí `hold on`... `hold off`. Chceme-li vykreslit kromě závislosti výchylky na čase $\varphi = \varphi(t)$ také závislosti úhlové frekvence na čase $\omega = \omega(t)$ a úhlové frekvence na výchylce $\omega = \mathrm{d}\varphi/\mathrm{d}t = \omega(\varphi)$ (tj. znázornění kmitů ve „fázovém prostoru“), musíme cyklus s vykreslováním opakovat třikrát (9.–35. řádek včetně výpočtu, 38.–42. řádek a 44.–48. řádek pro vykreslení). Abychom třikrát neopakovali celou integraci, ukládáme hodnoty postupně do matic `x` a `v`, které již nejsou vektory – každý jejich řádek odpovídá řešení pro jednu počáteční výchylku. Proto tyto matice na počátku zavádíme jako prázdné (`x=v=[] ;` na 7. řádku), abychom k nim mohli přidávat další řádky z hodnot `ph` a `dph` příkazy `x=[x;ph]` resp. `v=[v;dph]`

(31.–32. řádek). Integrační krok na 16. řádku je pak volen tak, aby uvažovaný časový interval $\langle 0{,}4\rangle$ byl rozdělen na 500 dílků.

Dodejme, že není možné zadat limitní hodnotu počáteční výchylky $\varphi_0 =$ $= 180^\circ$, neboť v tomto případě při nulové počáteční rychlosti těleso začne padat volným pádem a rovnice matematického kyvadla pro něj nebude platit. Všechny tři zmíněné závislosti jsou postupně vykresleny na obrázcích 4.3a–c. Vidíme, že s rostoucí počáteční výchylkou se pohyb více a více liší od harmonického, pro $\varphi_0 = 179^\circ$ je doba kmitu téměř $4\,\mathrm{s}$, tj. asi $4\times$ větší než pro malé kmity kyvadla.

### 4.2.2 Pohyb družice v gravitačním poli Země a Měsíce

Problém je po fyzikální stránce popsán v části 1.8, jde o příklad tzv. omezeného problému tří těles, kdy předpokládáme, že třetí těleso (v našem případě družice) neovlivňuje svou gravitací zbývající dvě tělesa (Zemi a Měsíc), která se pohybují po kruhových trajektoriích okolo společného hmotného středu. Pokud úlohu navíc řešíme v soustavě, která se otáčí spolu se Zemí a Měsícem, můžeme obě tělesa považovat za nehybná, počátek vztažné soustavy volíme ve středu Země. Úlohu budeme řešit v přiblížení, kdy zanedbáváme setrvačné síly spojené s neinerciálností rotující soustavy oproti gravitačnímu působení. Dodejme, že obecný problém tří těles je úloha, která v důsledku nelineárnosti rovnic nemá obecné analytické řešení a zabývala se jím řada význačných matematiků (Euler, Lagrange, Jacobi, Hill, Poincaré, Levi-Civita, Birkhoff).

V naší úloze použijeme několik astronomických konstant – hmotnost Země $M_\mathrm{Z} = 5{,}983{\cdot}10^{24}\,\mathrm{kg}$, hmotnost Měsíce $M_\mathrm{M} = 7{,}374{\cdot}10^{22}\,\mathrm{kg}$, jež se nacházejí ve vzájemné vzdálenosti $x_\mathrm{M} = 60{,}13\,R_\mathrm{Z}$. Poloměr Země $R_\mathrm{Z} = 6{,}371{\cdot}10^{6}\,\mathrm{m}$ použijeme jako jednotku vzdálenosti, souřadnice polohy družice budeme uvádět v násobcích $R_\mathrm{Z}$. Dále budeme předpokládat, že družice se pohybuje výhradně pod vlivem gravitačního pole s vypnutými motory.

Pro určení gravitačních sil, které působí na kosmickou sondu, musíme určit nejen souřadnice polohy vzhledem k Zemi ($x_\mathrm{SZ}$, $y_\mathrm{SZ}$), ale i relativní souřadnice sondy vzhledem k Měsíci $x_\mathrm{SM} = x_\mathrm{SZ} - x_\mathrm{M}$, $y_\mathrm{SM} = y_\mathrm{SZ} - y_\mathrm{M}$, kde $x_\mathrm{M}$, $y_\mathrm{M}$ jsou souřadnice Měsíce ve vztažné soustavě spojené se Zemí. Měsíc umístíme na osu $x$ naší soustavy, takže klademe $x_\mathrm{M} = 60{,}13R_\mathrm{Z}$, $y_\mathrm{M} = 0$. Současně je také zřejmé, že $y_\mathrm{SM} = y_\mathrm{SZ}$. Pro vzdálenost družice od Země platí $r = \sqrt{x_\mathrm{SZ}^2 + y_\mathrm{SZ}^2}$, pro její vzdálenost od Měsíce $r_\mathrm{M} = \sqrt{x_\mathrm{SM}^2 + y_\mathrm{SM}^2} = \sqrt{\left(x_\mathrm{SZ} - x_\mathrm{M}\right)^2 + y_\mathrm{SZ}^2}$.

Složky zrychlení sondy budou výslednicí gravitačního působení Země a Měsíce, můžeme pro ně proto psát

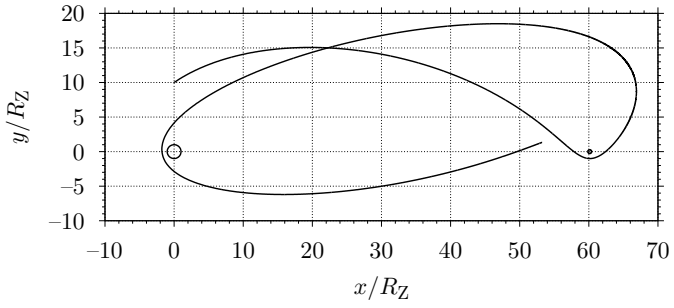
$$\begin{aligned} a_x &= -\varkappa\frac{x_{\mathrm{SZ}}M_{\mathrm{Z}}}{r^3} - \varkappa\frac{x_{\mathrm{SM}}M_{\mathrm{M}}}{r_{\mathrm{M}}^3} = -\varkappa\frac{x_{\mathrm{SZ}}M_{\mathrm{Z}}}{r^3} - \varkappa\frac{(x_{\mathrm{SZ}}-x_{\mathrm{M}})\,M_{\mathrm{M}}}{r_{\mathrm{M}}^3}, \\ a_y &= -\varkappa\frac{y_{\mathrm{SZ}}M_{\mathrm{Z}}}{r^3} - \varkappa\frac{y_{\mathrm{SM}}M_{\mathrm{M}}}{r_{\mathrm{M}}^3}. = -\varkappa\frac{y_{\mathrm{SZ}}M_{\mathrm{Z}}}{r^3} - \varkappa\frac{y_{\mathrm{SZ}}M_{\mathrm{M}}}{r_{\mathrm{M}}^3}. \end{aligned}$$

Program `sondark2` pak může vypadat následovně

```
# --- numericke reseni pohybu druzice
clear;
clc;
# --- pocatecni podminky a konstanty ---
RZ=6.378e6;     % polomer Zeme
t(1)=0;
xsZ(1)=0;
ysZ(1)=10*RZ;
vx=2.7e3;       % x-ova slozka startovni rychlosti
vy=1.84e3;      % y-ova slozka startovni rychlosti
xM=60.13*RZ;
yM=0;
mez= 1.2e6;     % mez integrace
h = mez/6000;   % krok integrace
# --- parametry prostredi, konstanty ---
k=6.67259e-11;  % gravitacni konstanta
MZ=5.98e24;     % hmotnost Zeme
MM=7.374e22;    % hmotnost Mesice-
# --- vlastni numericka integrace - RG metoda 2. radu
i=2;
while (t<mez & xsZ(i-1)^2+ysZ(i-1)^2>RZ^2)
   k1x=-k*xsZ(i-1)*MZ/(xsZ(i-1)^2+ysZ(i-1)^2)^1.5 \
    -k*MM*(xsZ(i-1)-xM)/((xsZ(i-1)-xM)^2+ysZ(i-1)^2)^1.5;
   k1y=-k*ysZ(i-1)*MZ/(xsZ(i-1)^2+ysZ(i-1)^2)^1.5 \
    -k*MM*ysZ(i-1)/((xsZ(i-1)-xM)^2+ysZ(i-1)^2)^1.5;
   xp=xsZ(i-1)+vx*2*h/3+k1x*2*h^2/9;
   yp=ysZ(i-1)+vy*2*h/3+k1y*2*h^2/9;
   k2x=-k*xp*MZ/(xp^2+yp^2)^1.5-k*MM*(xp-xM)/((xp-xM)^2+yp^2)^1.5;
   k2y=-k*yp*MZ/(xp^2+yp^2)^1.5-k*MM*yp/((xp-xM)^2+yp^2)^1.5;
   xsZ(i)=xsZ(i-1)+vx*h+(k1x+k2x)*h^2/4;
   ysZ(i)=ysZ(i-1)+vy*h+(k1y+k2y)*h^2/4;
   vx=vx+(k1x+3*k2x)*h/4;
   vy=vy+(k1y+3*k2y)*h/4;
   t(i)=t(i-1)+h;
   i=i+1;
endwhile
# --- vykreslení zavislosti
```

```
38 axis('equal');
39 hold on
40 # --- Vykresleni Zeme
41 u=linspace(0,2*pi,100);
42 plot(cos(u),sin(u),'b')
43 # --- Vykresleni Mesice
44 plot(1738000/RZ*cos(u)+xM/RZ,1738000/RZ*sin(u),'g')
45 # --- vykresleni trajektorie
46 plot(xsZ/RZ,ysZ/RZ,'r');
47 hold off
```

Počáteční podmínky odpovídají hodnotám $x_{\mathrm{SZ0}} = 0$ (`xsZ(1)`), $y_{SZ0} = 10R_{\mathrm{Z}}$ (`ysZ(1)`), $v_{x0} = 2{,}7\,\mathrm{km{\cdot}s^{-1}}$ a $v_{y0} = 1{,}84\,\mathrm{km{\cdot}s^{-1}}$. Výsledná podoba trajektorie je přitom na změnu počátečních podmínek velmi citlivá. V podmínce cyklu `while` na 21. řádku testujeme nejenom zda čas $t$ nepřekročil stanovenou horní mez, ale také dopad na zemský povrch (tj. zda $r > R_{\mathrm{Z}}$). Kromě samotné trajektorie (příkazem `plot` na 46. řádku) jsou vykresleny i kružnice znázorňující povrch Země i Měsíce (42. resp. 44. řádek), opět nastavujeme stejné měřítko na obou osách příkazem `axis('equal')`. Výsledek znázorňuje obr. 4.4. Dodejme, že pokud bychom použili Eulerovu namísto Rungovy-Kuttovy metody, k dosažení stejné přesnosti by se citelně prodloužila doba výpočtu, zejména na pomalejších počítačích.




Obr. 4.4: Příklad pohybu družice v soustavě Země–Měsíc

### 4.2.3 Pružné kyvadlo

Model popsaný již v části 2.3 je tvořen kyvadlem, u něhož je závěs realizován pružinou o tuhosti $k$ a délce $l$ (v nezatíženém stavu). Jde o poměrně jednoduchý a zároveň velmi zajímavý model, na němž lze studovat řadu jevů spojených

s oscilacemi. Svědčí o tom mimo jiné dva články z nedávné doby [5, 8], jež můžeme vřele doporučit zájemcům o hlubší seznámení s problematikou i o odkazy na další zdroje informací.

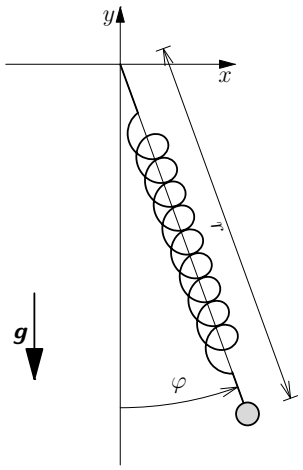
Pro naše numerické řešení vystačíme s pohybovou rovnicí

$$m\frac{\mathrm{d}^2\boldsymbol{r}}{\mathrm{d}t^2} = \boldsymbol{G} - k\,(r-l)\;\frac{\boldsymbol{r}}{r},$$

kterou lze v kartézských souřadicích $x = r\sin\varphi$, $y = r\cos\varphi$ rozepsat do složek

$$\begin{aligned} m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} &= -k\,(r-l)\;\frac{x}{r}, \\ m\frac{\mathrm{d}^2y}{\mathrm{d}t^2} &= -g - k\,(r-l)\;\frac{y}{r}. \end{aligned}$$

Počátek souřadnic přitom volíme v místě závěsu podle obr. 4.5.




Obr. 4.5: K zavedení souřadnic pro pružné kyvadlo

Jedná se o systém s dvěma význačnými módy (kmity kyvadla spojené se změnou souřadnice $\varphi$ a kmity pružiny spojené se změnou $r$), mezi nimiž je nelineární vazba, která je zodpovědná za specifické vlastnosti pohybu. Rovnovážná poloha, kdy je závaží po zavěšení na pružinu v klidu ve svislé ose $y$ je určena vzdáleností od bodu závěsu $r = l_{\mathrm{g}} = l + mg/k$, vůči této hodnotě budeme v souladu s [5] vykreslovat změny vzdálenosti závaží od bodu závěsu. Pro snadnější porovnání s uvedenými prameny [5, 8] volíme tytéž parametry – nezatíženou délku pružiny $l = 0{,}28\,\mathrm{m}$, tuhost pružiny $k = 12{,}5\,\mathrm{N\cdot m^{-1}}$.

Jak je odvozeno v [5, 8], předávání energie mezi zmíněnými módy je nejvýraznější při rezonanci, kdy platí

$$\omega_r = \sqrt{\frac{k}{m}} = 2\omega_\varphi = \sqrt{\frac{g}{l_{\mathrm{g}}}}.$$

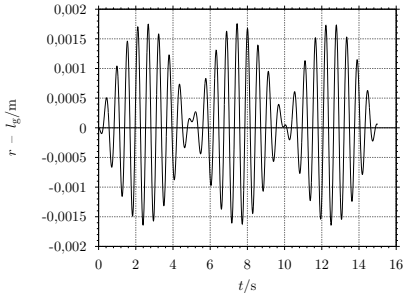
Po dosazení za $l_{\mathrm{g}}$ dostaneme hmotnost závaží, které musíme zavěsit, aby rezonance nastala

$$m = \frac{lk}{3g} \approx 0{,}12\,\mathrm{kg}. \tag{4.3}$$

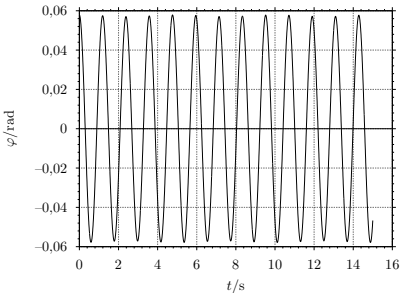
Ukažme nejprve případ dominantních kyvů, kdy kývání kyvadla bude minimálně ovlivněno kmity pružiny. Předpokládáme, že na počátku závaží o hmotnosti $m = 0{,}09\,\mathrm{kg}$ vychýlíme o úhel $\varphi_0 = 0{,}05785\,\mathrm{rad}$, aniž by se pružina prodloužila, tj. $r_0 = l_\mathrm{g}$. Výpis programu `pruzkrk.m` pak může být následující:

```
# --- numericke reseni pruzneho kyvadla
clear;
clc;
# --- konstanty
l=0.28;       # delka nezatizene pruziny
k=12.5;       # tuhost pruziny
g=9.81;       # tihove zrychleni
m=0.09;       # hmotnost zavazi
lg=l+m*g/k;   # rovnovazna delka pruziny
# --- pocatecni podminky
d0=0.00;     # pocatecni prodlouzeni pruziny
phi0=0,05785; # pocatecni vychylka od svisleho smeru
x0=(lg+d0)*sin(phi0);    # prepoctena soradnice x0
y0=-(lg+d0)*cos(phi0);   # prepoctena soradnice y0
vx0=0;        # pocatecni rychlost ve smeru x
vy0=0;        # pocatecni rychlost ve smeru x
# --- oblast integrace
t0=0;         # cas merime od 0
mez=15;       # horni mez integrace
h=5e-3;       # integracni krok
# --- vlastni numericka integrace - RK metoda 2. radu
t(1)=t0;
x(1)=x0;
y(1)=y0;
vx=vx0;
vy=vy0;
i=1;
# --- cyklus probiha pro t<= horni mez
 while (t(i)<=mez)
   i=i+1;
   r=sqrt(x(i-1)^2+y(i-1)^2);
   k1x=-k*(r-l)/r*x(i-1)/m;
   k1y=-g-k*(r-l)/r*y(i-1)/m;
   xp=x(i-1)+vx*2*h/3+k1x*2*h^2/9;
   yp=y(i-1)+vy*2*h/3+k1x*2*h^2/9;
   rp=sqrt(xp^2+yp^2);
   k2x=-k*(rp-l)/rp*xp/m;
   k2y=-g-k*(rp-l)/rp*yp/m;
   x(i)=x(i-1)+vx*h+(k1x+k2x)*h^2/4;
   y(i)=y(i-1)+vy*h+(k1y+k2y)*h^2/4;
   vx=vx+(k1x+3*k2x)*h/4;
   vy=vy+(k1y+3*k2y)*h/4;
```

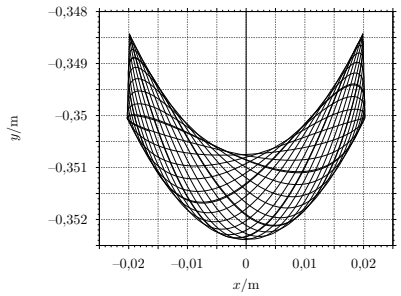
```
43    t(i)=t(i-1)+h;
44  endwhile
45 # --- vykresleni trajektorie
46 plot(x,y);
47 # --- vykresleni závislosti r-lg na case
48 plot(t,sqrt(x.^2+y.^2)-lg);
49 # --- vykresleni závislosti phi na case
50 plot(t,atan2(x,abs(y)))
```




(a) Závislost prodloužení $r(t) - l_\mathrm{g}$ na čase



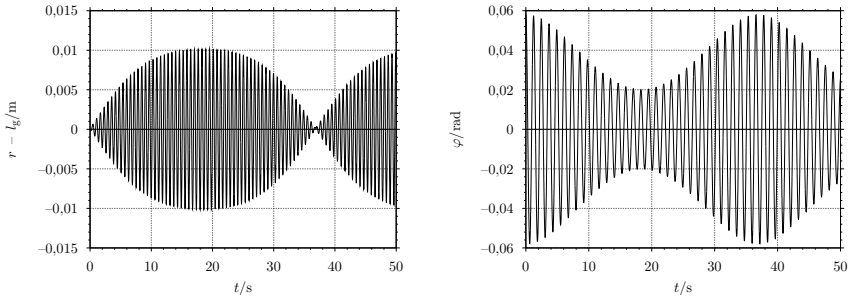

(b) Závislost výchylky $\varphi$ na čase




(c) Výsledný pohyb v rovině $xy$

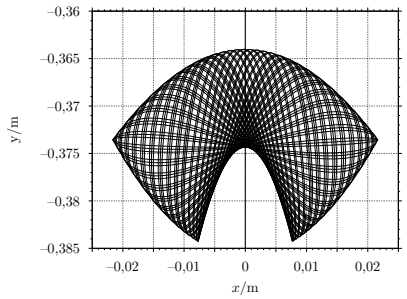
Obr. 4.6: Kmity pružného kyvadla v režimu dominantních kyvů

Vidíme, že počáteční délku pružiny zadáváme v programu na 11. řádku rozdílem $d_0 = r_0 - l_\mathrm{g}$, jež je v tomto případě nulový, počáteční hodnotu kartézských souřadnic $x_0$ a $y_0$ pak dopočteme podle již uvedených převodních vztahů. Naopak, z hodnot $x$ a $y$ získaných integrací získáme $r = \sqrt{x^2 + y^2}$ a $\varphi = \operatorname{arctg} x/|y|$, neboť v naší volbě osa $y$ směřuje nahoru a pod závěsem je $y$ záporné. Použili jsme




(a) Závislost prodloužení $r(t) - l_\mathrm{g}$ na čase

(b) Závislost výchylky $\varphi$ na čase




(c) Výsledný pohyb v rovině $xy$

Obr. 4.7: Kmity pružného kyvadla v blízkosti rezonance

tu dvě dosud nezmíněné funkce programu `atan2(a,b)` pro výpočet $\mathrm{arctg}\,(a/b)$ a `abs` pro výpočet absolutní hodnoty. Výsledné kmity jsou znázorněny na obr. 4.6. Je zřejmé, že změny prodloužení pružiny jsou relativně velmi malé a výslednému pohybu skutečně dominují kyvy v souřadnici $\varphi$. Toto tvrzení je jen zdánlivě v rozporu s výslednou trajektorií na obr 4.6c, neboť změny v souřadnici $y$ jsou mnohem menší než v souřadnici $x$.

Změníme-li hmotnost závaží na $m = 0{,}12\,\mathrm{kg}$ a prodloužíme časový interval pro integraci, abychom zachytili delší průběh pohybu, mělo by podle rovnice (4.3) docházet k rezonanci. Na obr. 4.7 je skutečně patrné postupné předávání energie mezi kyvy kyvadla a kmity pružiny. I zde platí, že tvar trajektorie na obr. 4.7 je deformován nestejným měřítkem, změny ve svislém směru jsou ve skutečnosti mnohem menší než ve směru vodorovném. Pokud bychom ale nastavili na obou

osách stejné měřítko, trajektorie by se nám redukovala ve „tlustě rozmazanou čáru“ kyvů.

## 4.3 Modely využívající funkce `lsode`

Modely využívající procedury `lsode` jsou vesměs stručnější a kompaktnější, naopak je u nich obtížnější testovat případnou dodatečnou podmínku (např. dopad na rovinu $y = 0$ jako v části 4.1.1) a při jejím nesplnění výpočet zastavit. Mezi tyto modely patří pochopitelně i příklady uvedené v částech 3.2.1 a 3.2.2.

### 4.3.1 Balistická křivka podruhé

Vraťme se ještě jednou k balistické křivce popsané v části 4.1.1. Pokud vyjdeme z rovnic (4.1a) a rozhodneme se použít funkci `lsode`, může soubor `balist.m` obsahovat příkazy

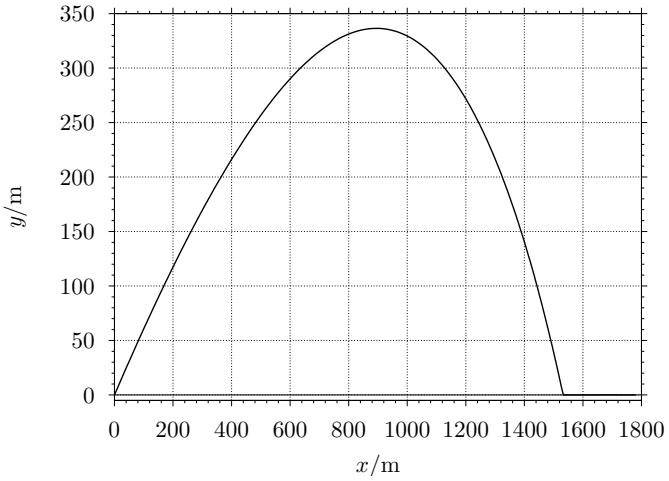
```
# --- numericke reseni balistickeho problemu
clear;
clc;
# --- definice pocatecnich podminek
t0=0;
x0=0;
y0=0;
v=input('zadej pocatecni rychlost v~m/s: ');
alpha=input('elevacni uhel ve stupnich: ')*pi/180;
vx=v*cos(alpha);
vy=v*sin(alpha);
# --- parametry prostredi a strely
R=0.04;                 # polomer strely
global m=2.5;           # hmotnost strely
ro=1.3;                 # hustota vzduchu
                        # koeficient odporu
C=input('zadej koeficient odporu C (<0,1>): ');
global g=9.81;          # tihove zrychleni
S=pi*R^2;
global k=C*ro*S/2;
# --- definice funkce pro integraci
function xdot=fce(xp,t)
 global k~m g # prebira globalne definovane konstanty
  xdot=zeros(4,1);
  xdot(1)=xp(2);
  xdot(2)=-k/m*sqrt(xp(2).^2+xp(4).^2).*xp(2);
  xdot(3)=xp(4);
  xdot(4)=-g-k/m*sqrt(xp(2).^2+xp(4).^2).*xp(4);
endfunction
# --- meze integrace a cas jako nezavisle promenna
```

```
31 dm=0;
32 hm=2*vy/g;
33 t=linspace(dm,hm,200);
34 # --- vlastni numericka integrace pomoci lsode
35 yp=lsode("fce",[x0,vx,y0,vy],t);
36 x=yp(:,1);
37 y=yp(:,3).*(yp(:,3)>=0);
38 # --- vykresleni reseni
39 plot(x,y);
```

Novým, v tomto textu dosud nepoužitým příkazem je ze `input` na řádcích 8, 9 a 17. Poslouží nám tehdy, pokud chceme vytvořit interaktivní program, který po spuštění požádá uživatele (našeho studenta) o vložení hodnot příslušných proměnných. Konkrétně na základě 8. řádku vypíše program

```
zadej pocatecni rychlost v m/s:
```

a čeká na vložení číselné hodnoty ukončené klávesou Enter. Jak vidíme, máme možnost zadat text výzvy, která se objeví na obrazovce. V sofistikovanějších programech bychom mohli ošetřit, zda zadané hodnoty splňují další dodatečné podmínky (nezápornost apod.).




Obr. 4.8: Balistická křivka pro $v_0 = 200\,\mathrm{m{\cdot}s^{-1}}$, $\alpha = 32°$ a $C = 0{,}48$

Jak již bylo uvedeno výše, na rozdíl od Eulerovy metody s cyklem `while` použité v části 4.1.1 zde neukončíme integraci po dopadu na rovinu $y = 0$,

proto musíme záporné hodnoty souřadnice $y$ odfiltrovat jinak. Nechceme-li nějakým vedlejším výpočtem omezovat horní mez integrace (zde, jak je zřejmé ze 32. řádku, je zvolena jako doba šikmého vrhu v neodporujícím prostředí), využijeme jedné z dalších metod efektivní práce s maticemi. Na 37. řádku se setkáváme se sloupcovou maticí `yp(:,3)>=0`, kterou vytváříme ze 3. sloupce matice `yp` pomocí logické podmínky `>=0`. Výsledkem bude vektor (sloupcová matice), v níž budou buď jedničky (na místech, kde prvky `yp` podmínky splňují, tj. zde jsou nezáporné), nebo nuly (na těch řádcích, kde hodnoty `yp` podmínku nesplňují). Vynásobíme-li poté prvky této matice původní vektor `yp(:,3)` (pozor, ne matice jako celek, ale odpovídající prvky mezi sebou, takže musíme použít `.*`), násobíme její nezáporné prvky 1 a záporné 0, takže získáme matici obsahující nezáporné hodnoty `yp(:,3)` a na místech záporných hodnot 0. Výsledná křivka pro interaktivně zadané hodnoty $v_0 = 200\,\mathrm{m{\cdot}s^{-1}}$, $\alpha = 32^\circ$ a $C = 0{,}48$ je znázorněna na obr. 4.8.

### 4.3.2 Foucaultovo kyvadlo

Stáčení roviny kmitu Foucaultova kyvadla vlivem Coriolisovy síly patří od svého prvního předvedení Jeanem Bernardem Léonem Foucaultem v pařížském Pantheonu v roce 1851 k základním důkazům rotace Země. Foucaultův model prý nesl hmotnost $28\,\mathrm{kg}$ na laně dlouhém $68\,\mathrm{m}$ [1].

Zvolíme-li vztažnou soustavu spojenou s rotujícím povrchem Země tak, že osa $x$ míří v daném místě ve směru poledníku na jih, osa $y$ ve směru rovnoběžky na východ a osa $z$ svisle vzhůru, omezíme-li se na malé kmity vzhledem k délce kyvadla, lze pro průmět kmitů do roviny $xy$ v místě se zeměpisnou šířkou $\lambda$ odvodit přibližné rovnice (viz např. [22])

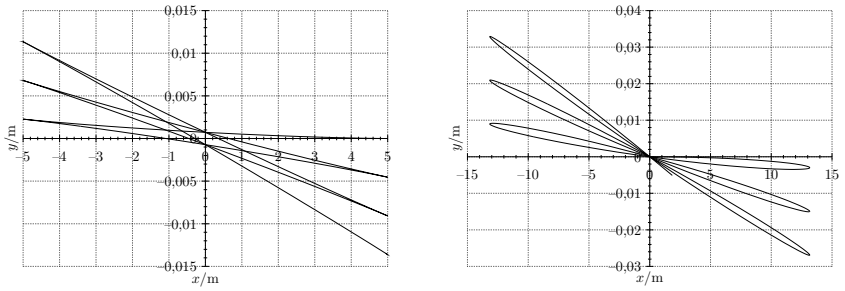
$$\begin{aligned} \ddot{x} &= -\omega^2 x + 2\dot{y}\Omega\sin\lambda, \\ \ddot{y} &= -\omega^2 y - 2\dot{x}\Omega\sin\lambda, \end{aligned}$$

v nichž $\omega = \sqrt{g/l}$ je úhlová frekvence matematického kyvadla téže délky a $\Omega \approx 7{,}3{\cdot}10^{-5}\,\mathrm{s^{-1}}$ úhlová frekvence otáčení Země okolo vlastní osy. Do souboru `foucaoult.m` můžeme napsat

```
1 # --- numericke reseni Foucaultovo kyvadlo
2 clear;
3 clc;
4 # --- definice pocatecnich podminek
5 t0=0;
6 x0=5;
7 y0=0;
8 vx0=0;
```

```
vy0=0;
global phi=50/180*pi;       # zemepisna sirka
global om=2*pi/(86400);     # uhlova rychlost rotace Zeme
global l=68;                # delka kyvadla
global g=9.81;
# --- vlastni numericka integrace - preddefinovana procedura lsode
function xdot=fce(xp,t)
global phi om g l           # prebira globalne definovane konstanty
  xdot=zeros(4,1);
  xdot(1)=xp(2);
  xdot(2)=-g/l*xp(1)+2*om*sin(phi)*xp(4);
  xdot(3)=xp(4);
  xdot(4)=-g/l*xp(3)-2*om*sin(phi)*xp(2);
endfunction
# --- meze integrace
dm=0;
hm=50;
# --- cas jako nezavisla promenna, oblast integrace
t=linspace(dm,hm,500);
# --- reseni soustavy diferencialnich rovnic
yp=lsode("fce",[x0,vx0,y0,vy0],t);
# --- vykresleni
plot(yp(:,1),yp(:,3));
```

Řešení vidíme na obr. 4.9a. Kyvadlo bylo v tomto případě uvedeno do pohybu puštěním z maximální výchylky. Pokud změníme počáteční podmínky tak, že kyvadlo rozhoupeme udělením počáteční rychlosti z rovnovážné polohy, bude mít průmět trajektorie do roviny $xy$ podobu znázorněnou na obr. 4.9b. Připomeňme, že měřítko na osách není stejné, za uvažovaných 50 s nemůže být výchylka ve směru osy $y$ velká (naopak je řádově skoro 1000-krát menší).




(a) Počáteční podmínky $x_0 = 5\,\mathrm{m}$, $v_{x0} = 0\,\mathrm{m{\cdot}s^{-1}}$

(b) Počáteční podmínky $x_0 = 0\,\mathrm{m}$, $v_{x0} = 5\,\mathrm{m{\cdot}s^{-1}}$

Obr. 4.9: Kmity Foucaultova kyvadla

# Literatura


[1] Wikipedie, otevřená encyklopedie, URL: `http://cs.wikipedia.org/`.

[2] *Bajer J.*: Mechanika 3. PřF UP Olomouc 2006.

[3] *Bartsch H.J.*: Matematické vzorce. SNTL, Praha 1984.

[4] *Brdička M. – Hladík A.*: Teoretická mechanika. Academia, Praha 1987.

[5] *Dvořák L.*: Pružné kyvadlo: od teoretické mechaniky k pokusům a zase zpátky. PMFA 51 (2006), č. 4, s. 312–327.

[6] *GNU project*: GNU General Public License v. 2 (český překlad), URL: `http://gnu.cz/article/30/pdf/gpl-cz.pdf`.

[7] *Greiner W.*: Classical mechanics: systems od particles and Hamiltonian dynamics. Springer-Verlag, New York 2003.

[8] *Havránek A. – Čertík O.*: Pružné kyvadlo. PMFA 51 (2006), č. 3, s. 198–216.

[9] *Holíková L.*: Použití numerických metod v úlohách středoškolské fyziky. Dipl. práce, UP Olomouc 2006. URL: `http://optics.upol.cz/~richterek/files.html`.

[10] *Jarník V.*: Diferenciální počet I. Academia, Praha 1984.

[11] *Just M.*: Octave – český průvodce programem, URL: `http://www.octave.cz/`.

[12] *Lepil O.*: Dynamické modelování. Studijní text, UP, Olomouc 2001.

[13] *Mařík R.*: Průběh funkce (Matematika (nejen pro) krajináře a nábytkáře), URL: `http://user.mendelu.cz/marik/mat-web/mat-webse4.html`.

[14] *Míka S.*: Numerické metody algebry. Matematika pro vysoké školy technické sešit IV, SNTL, Praha 1982.

[15] *Poláček J.*: GNU Octave, URL: `http://www.abclinuxu.cz/software/veda/gnu-octave`.

[16] *Přikryl P.*: Numerické metody matematické analýzy. Matematika pro vysoké školy technické sešit XXIV, SNTL, Praha 1985.

[17] *Rektorys K. a kol.*: Přehled užité matematiky I, II. SNTL, Praha 1988.

[18] *Šedivý P.*: Užití numerických metod při řešení rovnic ve fyzikálních úlohách. Studijní text pro řešitele 35. ročníku FO, MAFY, Hradec Králové 1996.

[19] *Šedivý P.*: Modelování pohybů numerickými metodami. Knihovnička Fyzikální olympiády č. 38, Gaudeamus, Hradec Králové 1999. URL: `http://fo.cuni.cz/texty/modelov.pdf`.

[20] *Šedivý P. – Volf I.*: Pohyb tělesa po eliptické trajektorii v radiálním gravitačním poli. Knihovnička FO č. 43, MAFY, Hradec Králové 2000. URL: `http://fo.cuni.cz/texty/druzice.pdf`.
[21] *Strogatz S.H.*: Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering. Westview Press 2000.
[22] *Trkal V.*: Mechanika hmotných bodů a tuhého tělesa. ČSAV, Praha 1956.
[23] *Volf I. – Šedivý P.*: Pohyby tělesa v odporujícím prostředí. Hradec Králové 1992. URL: `http://fo.cuni.cz/texty/odpor.pdf`.
[24] *Vybíral B. – Zdeborová L.*: Pohyb těles s vlivem odporových sil. Knihovnička FO č. 55, MAFY, Hradec Králové 2002. URL: `http://fo.cuni.cz/texty/odpory.pdf`.
[25] *Williams T. Kelley C. – others.*: Gnuplot. An Interactive Plotting Program, URL: `http://www.gnuplot.info/docs/gnuplot.html`.

# Seznam příkazů použitých v textu